THE MISOSYS QUARTERLY

In this issue:

- A cc for MC
- Add SETEOF to EnhComp
- Change baud rates with SETBAUD
- Unlock protected BASIC programs
- WinCalc: a PRO-WAM application

Volume I, Issue I

Summer 1986

THE MISOSYS QUARTERLY

Volume I, Issue i

Summer 1986

Table of Contents

PO Box 239, Sterling, VA, 22170-0239
703-450-4181

The BLURB by Roy Soltoff

This is the first edition of THE MISOSYS QUARTERLY. Somewhere between 400 and 500 MISOSYS customers have recognized the value of this publication to their computing interests by the support of their subscriptions. Considering that the notification of its impending publication was mailed out to 16,000 recognized users of our products, I am a little discouraged at what I consider to be the poor response to our subscription request mailing. On the other hand, maybe many of the recepients didn't realize that when we said, "Each issue of THE MISOSYS QUARTERLY will have a significant special available only to subscribers", we really meant it! In fact, to reward those early birds who subscribed for this first issue, they will find at least $19.95 in savings available to them: $14.95 off the purchase of one of a list of software items, and $5 off of the price for DISK NOTES 5. We really don't intend to overprint these issues as the experience we learned from Logical Systems is that past issues are a liability - we have thousands of old LSI QUARTERLYs and JOURNALS from Volume II. The subscriber specials are noted on the bound-in card which must be returned to order any special.

More on the acquisition of LSI

KRB was taken aback by our recent acquisition of the Logical System's retail operation. Anyone who has followed the TRS-80 history should feel quite at ease with this event; however, since there may be some readers unfamiliar with our company, let me share my response to KRB.

To begin with, TRSDOS 6 does not enter into the retail acquisition as TRSDOS 6 is not a retail product of LSI. Next, be aware that I was one of the co-founders of Logical Systems and owned one third of that company up until the reorganization which occurred shortly before TRSDOS 6 was being developed. I was also the system designer of TRSDOS 6.0.

There is no technical support line in Milwaukee. MISOSYS is located in Sterling Virginia and that is where we will be offering telephone support. Our number is prominently displayed in our ads as well as listed on our literature. We will continue to support what was the LSI product line. I say that in the past tense as MISOSYS has acquired the copyrights to the products as well.

Since an LSI JOURNAL has not been published by LSI for quite some time, I am rather taken aback by the question as to its continuance. On the other hand, to demonstrate our willingness and desire to support the TRS-80 user community, we intend to initiate a new publication for us to be called THE MISOSYS QUARTERLY. This will be a cross between our NOTES FROM MISOSYS and the old LDOS QUARTERLY/LSI JOURNAL. This will be the primary vehicle for written support [this is the first issue -ed].

We have done away with updates to LDOS. We sell replacement LDOS 5.1.4 disks for $14.95 + S&H. So far, this has had overwhelming approval from those customers already taking advantage of this offer. This arrangement saves the LDOS user from mailing back the old disk; we provide a brand new disk(s) in return.

Our registered customers will receive flyers from us at intervals. The LSI database is ours now. In fact, the machine where the database resides is in our premises. We are in the process of merging the LSI base with our base. [This has already been done -ed]. We sent out 16,000 flyers a few months back.

Since LDOS is our system now, we can best answer questions concerning continued development with the statement that any development which is considered to be prudent and which makes economic sense will be considered. We intend to offer a 5.3 release which will at a minimum extend the system date to beyond 1987. Other enhancements will be considered.

What happened to THE GUIDE?

The following letter addressed to 80 Micro's Feedback Loop discusses why we decided to stop publishing THE PROGRAMMERS GUIDE TO LDOS/TRSDOS VERSION 6.

"It appears that one of your readers wrote

to you on the 5th of December berating MISOSYS for not having any more copies of THE PROGRAMMER'S GUIDE TO TRSDOS 6 - especially complaining about our lack of advertising of THE GUIDE. Since 80 Micro surprised us with a review of THE GUIDE which appeared in the November issue, the resulting bubble in sales of that book caused us to sell out much sooner than expected. Since a few folks may have gotten miffed that we chose not to reprint the book, one of which wrote to you, perhaps your readers may also be interested in knowing why we reached that decision. We would hope that you would consider printing this entire letter.

THE GUIDE was first published during August of 1983; it was priced at $20 retail. Since the focus of the book was directed to the assembly language programmer, the publication was devoted more to the hacker than the general Model 4 user. Because of this, it was marketed directly to the core of hackers frequenting the LSI Special Interest Group (SIG) on Compuserve. THE GUIDE soon gained notoriety on the PowerSOFT SIG, also on Compuserve, and also frequented by the TRS-80 hacker. It appeared in our 83-2 catalog available in November of 1983. Two pages were devoted to THE GUIDE in our second issue of NOTES FROM MISOSYS which was mailed to 2000 folks in December of 1983. Its first ad in 80 Micro was in the June 1984 issue (which, of course, was received in May).

In November of 1984, MISOSYS did another direct mailing to our customer base and placed THE GUIDE on sale for $15.00. Effective January 1985, the price was reduced to $14.95. Our ad in the February 1985 issue of 80 Micro (which was received by subscribers in January) reflected the price reduction. THE GUIDE continued in our ads until the July 85 issue (we did not place an 80 Micro ad in the August 1985 issue). Meanwhile, other vendors continued to advertise THE GUIDE in 80 Micro as recent as the December 1985 issue (JMG Software International and DiskCount Data). PowerSOFT has had THE GUIDE listed in their catalogs for quite some time. Thus, for someone to say that THE PROGRAMMER'S GUIDE TO TRSDOS 6 has not been advertised means they are talking out of their hat.

Let's examine just how many copies of THE PROGRAMMER'S GUIDE TO TRSDOS VERSION 6 have been sold since it was first published. Here are the monthly figures for book sales since its publication in August 1983:

Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec	Tot
							26	46	57	38	65	232
83	42	42	29	41	27	44	20	49	48	99	85	609
50	24	21	38	65	7	20	24	11	72	75		407

The bottom line total is 1248 books in over two years - a dismal showing! The peak in November and December of 1984 reflect our direct mailing with a sale price of $14.95 versus the original $20. The November issue of 80 Micro had a little review of this book way back on page 119 of the "Express Checkouts" which caused a jump in sales of the book in October and November of 1985. That's really why we ran out of books. Of course, since THE GUIDE was published starting in August 1983, I consider that review about two YEARS late!

We just don't expect the market to present significant quantities of new Model 4 owners to justify a reprint of 500 books - the minimum quantity necessary for reprinting. Thus, we can't justify reprinting this book since we consider its unit sales level insufficient to capture 500 more sales nor do we expect the TRS-80 Model 4 marketplace to have a considerable increase in the number of assembly language programmers who also wish to purchase this book. Perhaps we're wrong to discontinue this book; however, we certainly cannot afford to lose money over it. If the TRS-80 community of users wish us to continue publishing this book, they need to get more serious in their purchase of it. Products need to be purchased with sufficient volume to justify their contin-

uation. I don't think we are alone in this feeling. What ever happened to IJG? What ever happened to Dilithium Press? I am still stuck with about 300 copies of THE BOOK, Volume II. Since that's about the Model I, no one wants it. Sorry, but not enough people have purchased THE GUIDE for us to continue it."

Since that was written, we have licensed DiskCount Data to reprint THE GUIDE. Thus, anyone needing a copy of that book may obtain a new copy from them. We still stand behind our statements, however.

Where's MISOSYS been?

David B. Lamkins inquired as to our whereabouts during the lean months in 1985. No, we haven't dropped off the face of the earth. In fact, we have been busy with software development. Unfortunately, the company has not made enough to be able to put out another issue of NOTES. We expect to get a letter out soon announcing the new compiler, MC 2.x.

MRAS was released in August 85 and has been advertised in issues of 80-Micro. We hope to finish up MC for release by the end of the month, although with documentation left to do, it will probably slip to November [the final release of MC was released in July of 1986 -ed]. MC is a very big job for Rich Deglin and myself. Rich has done an outstanding job and I am proud of what he has accomplished.

As far as future focus, my crystal ball is so fuzzy, that I don't really have any plans for other machines. I doubt the Amiga will be a success since the home consumer is so confused by now that another razzle dazzle computer is not about to make them spend big bucks. Although the ST-520 Atari is less expensive, what home consumer needs it? Business these days went IBM compatible since business needs stability. Even though IBM is not leading edge technology, it nevertheless keeps things stable. The micro business is definitely not stable.

How would you like it if phonograph record format changed every few years. At most, over the span of tens of decades, I have seen 78s, 45s (still around), 33s (still around), and now compact disks (CDs). I expect CDs to take over entirely within 10 years. I expect that the CD-ROM drive will be the one thing to galvanize the home computer industry since I see that as a solution to a problem that is evident in every family. The CD-ROM definitely is a viable solution to an existing need. I can't wait to get my hands on Grolier's CD-ROM encyclopedia at a reasonable price. So much for rambling.

Daniel Fox inquired about the possibilities of tying two TRS-80s together via the RS-232 port. We had considered such a facility last in 1984. It would have been modeled after our ADE floppy emulator except that the "emulated" drive would be a drive on the remote system. We were considering it as a hardwired facility via the RS-232 for enabling a higher speed transfer - which would really be necessary.

The one thing which stopped us from this project was manpower resources. You see, nothing else was able to stand in the way of other projects such as PRO-NTO, MRAS - our latest relocating macro assembler, and MC - our full K&R compatible C compiler. We also needed time to implement a driver for our Cipher FloppyTape cartridge drive which stores about 28 Megs on a DC-600 tape cartridge (which still has not gotten done yet).

We also had to get software implemented for other machines. Thus, time was too critical for that project's development. A fully functional "TWOCLAN" (one name we picked standing for two computer local area network) would probably take about three man months of development time. That would probably encompass six months of clock time. We just didn't see that kind of time materializing; however, the idea was a good one. It would be fairly elementary to connect the "remote" computer so as to be able to share all its peripherals.

To add a more current note to the previous paragraphs, we have recently released a powerful BASIC compiler for the Models I/III and Model 4 machines. We still expect to release a RATFOR translator some time this year. We will be working on the

LDOS 5.3 release to be out in early 1987. We also expect to port some of the LSI Model I/III utilities to the Model 4 environment. We also expect to retail a hard disk driver package for the Radio Shack hard drives, soon. Rich Deglin is working on a powerful disk editor for MS-DOS. Karl has been hacking at an IFC for MS-DOS. Also on the drawing board is a port of Disk Sort Merge (DSM) to MS-DOS (we may even look at porting BSORT). We thusly have been busy and will continue to be busy generating and supporting products for the TRS line of computers.

Letters to the editor

Ed Clapp comments on NOTES: "In my opinion you provide through that publication the best support for your products of any software supplier, including some of the best known and I sincerely hope you will be able to continue publishing it. If things become too tight in the present market squeeze I would suggest that you offer it on a subscription basis. I am sure that most of your customers would be willing to pay for it - I certainly would.

Tom Wyckoff reports that, "A recent magazine article stated that although Commodore invented software incompatibility, Tandy raised it to an art form. That gave me quite a laugh. I wrote this using Mod I Scripsit, with Mod III LDOS on my Mod IV. Granted they had some outside help, but that's a long way from incompatable."

DOS 5.3/6.3 coming soon

Logical Systems reports that they are testing the upcoming release [early 1987] of TRSDOS 6.3. Complete details on this release will be in the next issue of THE MISOSYS QUARTERLY. Suffice it to say that the primary enhancements will be the directory date extension to about 1999 and the introduction of time stamping. There will also be additions to BASIC such as the BEEP facilities, a cross reference facility, and a facility to invoke SVCs from BASIC. MISOSYS is providing a full-screen text editor modeled after TED - the editor which is part of our Mister ED application pac.

MISOSYS will also be releasing LDOS 5.3 early next year. It too will have the date extension and time stamping, as well as some improvements to BASIC. I expect to also provide a 64x16 version of TED. Further details will be announced in the next QUARTERLY.

Contributions to THE MISOSYS QUARTERLY

MISOSYS is accepting articles for consideration of their use in future issues of TMQ. Articles should be submitted on both floppy disk media and paper. At this time, the compensation for articles selected for publication is a MISOSYS Gift Certificate(s). Any articles already submitted but not yet published may still be considered for future publication. Thanks for your sincere efforts.

Current LDOS release

The current release of LDOS is 5.1.4. Files are dated August 5, 1984. If your existing user manual is from an LDOS 5.1.3 or LDOS 5.1.3R release, be advised that we have a set of ducumentation updates which can be purchased to bring your 5.1.3 manual up to date. The charge is $2 IF YOU ARE PLACING SOME OTHER MERCHANDISE ORDER. If ordered separately, the charge is $2.50. These prices are for United States, Canada, and Mexico only! Please specify 514 docs or 514R docs; whichever is appropriate for you.

Software Submissions

MISOSYS is looking for quality application software submissions for publication. The primary focus is on applications and utilities for use with MS-DOS compatible machines - primarily Tandy compatibles. Submissions should be accompanied by complete documentation for the targeted end user. We still maintain an interest in software for the TRS-80 Model 4, where the software product is targeted to a wide audience. Our forte is language products and DOS enhancements. Submissions should be directed to the attention of "Software Submissions".

Family Update

It's been a while since the last report. Stacey turned 3 in June. Stefanie's second

birthday is this October. No further arrivals are scheduled. Over at the Kids R Us opening, Stacey measured in at 3 feet 3 inches against Stefanie's 3 feet. Stef's vocabulary has gotten her to the position of being able to vocalize her wants (i.e. more Juicy Juice, more Cheerios, etc.). Now Stacey, on the other hand, amazes me with her breadth of vocabulary and understanding of events. I guess its been quite some time since I was three. Part of our "training" is no candy, no cartoons, and generous doses of Sesame Street. Since getting on cable, the Disney channel also offers some good viewing material - especially for Brenda. There's also plenty of Dr Seuss books around.

This Fall, we are planning to schedule Stacey into a three-day-a-week three-hour-a-day pre-school that's targeted for fun and games. Stacey up to this point was going to a fun-for-two-year-olds class which met two days a week for an hour each. Stefanie advances to that class now. The big question is when do I pop for their own Model 1000? Until next time...

= =

Our LDOS Support Group on Compuserve

MISOSYS is the FORUM MANAGER of a Special Interest Group (SIG) on Compuserve. This SIG is properly entitled the LDOS/TRSDOS6 SIG. It is one of the oldest SIGs on Compuserve and has been a strategic facility for technical discussions concerning LDOS, TRSDOS6, the MAX-80, and the TRS-80 in general. The System Operator, or SYSOP, is Joe Kyle-DiPietropaola [known as jjkd]. Joe is a former employee of Logical Systems, knows the DOS well as well as most of the combined LSI/MISOSYS product line, and serves as SYSOP for us.

The SIG is frequented by many knowledgable folks; not to mention many just getting their feet wet. Therefore, the SIG is a handy way to rapidly get an answer to a problem which you may be having - be it a new one or an old one.

The SIG is also host to a number of data bases - each containing lots of "goodies" for your use. Some of the files are shareware, some are freeware, and some are public domain. Check out our SIG. All you need is a Compuserve USERID, a modem for your machine, and time to logon. From the Compuserve main menu, you only need to GO PCS49! What follows are some recent messages concerning the use of our SIG.

Becoming a SIG Member

Fm: LDOS Support 76703,437 -> To join the LDOS/TRSDOS 6 Forum, all you need do is pick the "JOIN" option on the non-member menu. Welcome aboard! --jjkd--

Reading SIG Messages

Fm: LDOS Support 76703,437 -> The SB command will give you descriptive names for the sections. This command is valid at both the Function: and DL n: prompts. To set your user options, issue the OP command at the Function: prompt. You will see a number of options, and you wish to turn off Brief prompts, and also turn on menu mode. That's "BRE OFF" and "MEN ON" Finally, make sure that you save the changes. --jjkd--

Fm: LDOS Support 76703,437 -> To read messages in the message base that are new since you last logged on, do a "RN" at the Forum main menu. To find out what is available to download for the Model 4/TRSDOS 6, do the following: "DL 3" will get you to the Model 4 download area. Follow that with a "CAT /DES" to get a catalog of all available files with descriptions. This is a long list, and I recommend that you hardcopy, or capture to a disk file. You can then examine the list at will, and come back later to download the files you'd like to have. To download, use "DOW FILNAM.EXT" at the DL prompt. If presented with a protocol menu (CIS "A", "B", "XMODEM", etc.), choose the one that your terminal program supports. Remember to give the file the proper name on your system, e.g. BINHEX.TXT would become BINHEX/TXT, MOOSE.BIN would become MOOSE/CMD, etc. Feel absolutely free to

yell with more questions at any time. Oh yeah, to get out of the DL area and back here to the "main Forum" use the "EXIT" command. --jjkd--

Fm: LDOS Support 76703,437 -> Not all commands update your last message read pointer. In general, the "mass read commands" (RF, RN) do, and the "individual read commands" (RI, RM) don't. Note also that if you read messages, go to the DL area and then logoff or bye from there, your message pointer will not be updated even if you used RF or RN. You must use EXIT to return to the Forum, and can then Logoff or Bye from here. --jjkd--

Fm: H. Brothers <OLT/WESIG> 70007,1150 -> Logging on with *tty does, indeed, make a major difference. If you want/need more than one set of defaults per baud rate, the process is not too difficult to do, but you'll need to muck around in PRO a bit. SU = store unformatted. Instead of a normal <s>tore at the Leave Action prompt, you can type SU (or SPU) to tell the Sig formatter "no matter what screen parameters the readers of this message have set, **don't** reformat this message to make it easy for them to read." Another way of interpreting SU is "keep your hands off the separate lines and tab columns which I worked so hard to create."

Editing Your messages

Fm: LDOS Support 76703,437 -> FILGE doesn't get you any more message formatting capability than the normal SIG editor. It merely allows you to put in blank lines, not necessarily keep them. To get a blank line, you need to enter a line as a single period, then start the next line with a period or a space. The space will cause the line to also be indented, the period will not. In neither case will the period show. You also have the option of using SU Store Unformatted instead of the normal S. This will prevent the SIG software from doing this kind of reformatting, and what you type will be exactly what you get. This is not a universal panacea, as it will also prevent the SIG software from reformatting your message to the proper width for the receiving terminal. SU should only be used as a last resort. For tabbing, ".>n" will indent "n" spaces from the left margin. Note that this formatting is specific to the SIG display software, and has nothing particular to do with FILGE or SED (the "other" SIG editor). --jjkd--

Fm: LDOS Support 76703,437 -> The screen width in the Forums is set independently of your screen width in the "normal" (DISPLA) area of CIS. That is what is set through DEFALT. Your screen width in the Forum is set via the OP command. --jjkd--

Fm: H. Brothers <OLT/WESIG> 70007,1150 -> When you enter C at the leave action prompt, you are returned to the Filge editor and the top of the document. You can start a line with /B to move to the bottom, or /TYPE to see the entire message and move to the bottom. Filge, even in its emasculated SIG version, has a great deal of power and flexibility -- a lot more than the "SIG Editor" called SED. However, it takes some practice to get used to it. If you are interested, I can point you to some instructional files that explain both the normal and arcane commands available in Filge. - hardin

Uploading, Downloading, and MNETA

The following series of messages relate primarily to the access of files in the various DownLoad sections of our SIG. The DLs encompass the storehouse of programs, fixes, and technical information which has been provided over the years by our users. There's lots of freeware, shareware, and "stuff". Here are some hints on how to get ahold of these files and how to add to the storehouse.

Fm: Les Mikesell 70010,266 -> The DL's are the download areas of the SIGS. From the SIG function prompt type SN to get a list of the section numbers and their topics. The DL areas contain files related to the same topics (DL 3 is the mod 4 area here although a re-organization is planned). To get there, just type DL 3 at the function prompt. The commands there work just like the public ACCESS area. - Les

Fm: jeff brenton [CLMFORUM] 76703,1065 -> Type "DL 3<cr>" - the XA's are now called

Data Libraries, or DL's, although "XA3<cr>" might still work. I haven't tried it recently, and it has been over 18 months since "XA" was officially supported.

Fm: LDOS Support 76703,437 -> Do you have anything to use for download other than COMM? If not, turn on your printer and enter the command, X8, at the main Forum prompt. This will give you step by step instructions for simple downloading with COMM. Use this info to get MNETA.JCL from DL3. Download this to a file called MNETA/JCL, and DO it. You will be left with a file MNETA/CMD that will upload and download here using CIS "A" protocol, which is much more reliable than using COMM (which has no error checking). Make sure that you SETCOM (W=8,P=N,BREAK=Ø) before using MNETA. Any questions or problems, just yell. --jjkd--

A 26K file is fine, but we do not recommend the use of BINHEX for files of that length. Instead, do this: Go into BASIC. Load the program. Save it in ASCII (via SAVE"FILESPEC/EXT",A). You may now upload the file using one of the supported protocols (CIS "A", CIS "B" or XMODEM) and the command "UPL PROGRAM.BAS/TYPE:ASCII". That should do it. --jjkd--

CIS filenames are limited to six letters, so some abbreviation is usually called for. Any filename greater than 6 characters will be truncated. Thus, if you upload two files [e.g. TELCOM1.BIN and TELCOM2.BIN], the second will overwrite the first. You can use a shorter name and then use the description field for more detailed info. --jjkd--

SQueezed files are binary, and must be uploaded via commands like "UPL XT4168.LQR/TYPE:BINARY". "/TYPE:BINARY" is what you were missing. --jjkd--

OK, for packed strings (all eight bits) and ML, etc., leave the program in compressed format, but upload it via BINARY, ala "UPL filnam.BAS/TYPE:BINARY" Note that you must use CIS "A", CIS "B" or "XMODEM" to upload with this option. --jjkd--

Do this: (1) SET *CL COM/DVR; (2) SETCOM (W=8,P=N,BREAK=Ø) [insert baud rate selection if necessary]; (3) COMM *CL; (4) When you get to the point that you wish to download a file, *before* downloading do a <CLEAR><SHIFT><Ø> to execute a DOS command, execute the DOS command "MNETA *CL"; (5) Now you may issue the BRO or DOW command to CIS, and when asked for a name for your computer make sure that you use a slash (/) instead of a period (.) as necessary. The key points are that BREAK=Ø and the fact that MNETA must be running before you indicate that you wish to download a file. With XMODEM it's generally the other way 'round, you start the host before you start your end. Note also that you pick CIS "A" protocol as your transfer option. If you don't get asked, go to CIS-9 and set your terminal type to "other". --jjkd--

Fm: Les Mikesell 70010,266 -> There are two things you must do when using MNETA: (1) If you have TRSDOS 6.2 use SETCOM (W=8,P=N,BREAK=Ø), otherwise SETCOM(W=8,P=N,BREAK=128) - these are OK for COMM also. (2) Be sure to invoke MNETA *before* choosing the A protocol in the download procedure. CIS immediately sends a query to your program when you choose A or B protocol which will be missed if you are still in COMM at this point. - Les

Fm: jeff brenton [CLMFORUM] 76703,1065 -> That means that you have already selected protocol before loading MNETA, and CIS is waiting for a reply from it's test. The secret is to invoke MNETA, *then* go into the DL. If MNETA is *not* running when you answer the "Select protocol" prompt, you're out of luck. Right after you hit <ENTER>, CIS sends out the "OK, who are you?" inquiry to your terminal. While it is doing that, you are off loading MNETA. MNETA never sees the question, so it can't answer it. CIS never gets a reply, so it aborts the download, since you don't support A protocol. Don't feel bad - I used to do it all the time!

Message Limits

Fm: Bill Evans 70160,436 -> According to a Compuserve reference card for Forums the message limit is 96 lines or 2500 characters, which ever comes first. - Bill

Help for the DAK-ADC 'Duck' modem

Fm: Nate Salsbury 72167,1750 -> I haven't picked up GETTIME/CMD [used to read the time from the DAK modem and set the DOS time -ed]. Try scanning in DL 3: "S GET???.???/DES". Also, try "CAT [PPN of the author of GETTIME/CMD]/DES". This will give you ALL the files that person has put into the DL.

What's in the DL sections?

Fm: BOB KAYE 73047,2422 -> Can anyone tell me how to have the description of a DL program written to the screen if you don't know any of the keywords; the program name is apparently not one of them. Thanks in advance - Bob Kaye

Fm: Jim Kyle (CLM SIG) 76703,762 -> If you know the name, type S name /DES and hit ENTER. If you know just the first letter, make it S X????? /DES (assuming that X was the first letter). For all files in the DL, S/DES will do it. You'll get the same display as for BROwse, but it won't have the pause and prompt; instead, (if you used a wildcard or all) it keeps going until all are shown and then you get the DOS prompt. This is a nice way to make your own catalog. just open the capture buffer and do S/DES. Then when it's done save the list and examine it offline. If you want a full list of all keywords used in a DL, the command KEY will give it to you. I use this when i don't know either the filename or the keywords used for it; usually I get enough hints frrom the KEY listing to be able to do a BRO /KEY:keyword and find it. ..jk..

Fm: LDOS Support 76703,437 -> If you know the program name, or part of it, you can do either a BROwse or CATalog based on that name, or a wildcard. For example, for a filename that you think begins with FOO: "BRO FOO???.???" or "CAT FOO???.???/DES". The first will show you the files one at a time, with descriptions, and the second will give you a continuous listing with descriptions. --jjkd--

Fm: Jim Kyle (CLM SIG) 76703,762 -> I learned a new trick since leaving the message: KEY D* will give you all keywords in that DL that start with D. Used it last night to find all of the files (both of them) with keyword DISASSEMBLER in all 5 IBM SIGs, and it took only a few minutes to scan all 45 DL's involved. MUCH faster than using BRO!!! ..jk..

Information on Communications

Here's some information concerning our X-FTS package. The XMODEM protocol is not a CP/M protocol - it is a protocol for the transfer of files. A file is a file. There is no requirement for the file to be text or binary. The protocol dictates a transmission of 128-byte blocks with handshaking on each block. Thus, there should never be any action on the part of any program which adheres to the XMODEM protocol to alter or modify the data within a block. Since the X-MODEM protocol was ORIGINALLY developed on a CP/M system, it has been populated across all sorts of systems. Our implementation on the TRS-80 satisfies the standard. If you are trying to transmit a text file which incorporates some line ending sequence to a system which uses another line ending sequence, the proper procedure would be to either preprocess the file or postprocess the file. I would never recommend processing the file during transmission.

Our LCOPY and PRO-CURE/CONVCPM products relate specifically to the transfer of files from LDOS to CP/M and from CP/M to TRSDOS/LDOS respectively. Since the products directly relate to two systems which use a different line ending sequence for text files, each product includes a program which is used to post process the file. CVTEXT supplied with LCOPY adds linefeed to follow carriage return. CVTEXT supplied with PRO-CURE/CONVCPM removes linefeeds which follow a carriage return. A program to add line feeds is extremely simplistic. It could be written in C in about 2 minutes. I recommend that you explore implementing such a preprocessing program to convert TEXT files which you want to send to the VAX to adhere to the line ending sequence used by the VAX. Preprocess your text files then use X-FTS on the processed file.

Here are some communications' tips gleaned from our Compuserve SIG.

Fm: WINSTON BARROWS 73537,1366 -> How do I turn off the reverse video frequently effected when I'm on-line?

(RE): If you are using COMM, then do a CLEAR SCREEN command (which is <CLEAR><SHIFT 8>). -Roy

Fm: LDOS Support 76703,437 -> 9600 baud is awful fast for that poor Model 100. It won't even keep with receiving at 1200 baud in many cases. Sending from the Model 100 to the Model 4 is ok at up to 1200 baud with DTD on, and up to at least 4800 baud ok with DTD off. Do this:

(1) Set up with the correct baud rate, parity, etc. the same at both ends. How about 37ele at the Model 100 end, and SETCOM (B=1200) at the Model 4 end.

(2) Go into COMM at the Model 4 end, and do a <clear><6> <clear><9> which is Receive File ID, and enter a file name.

(3) Do a <clear><6> <clear><:> which is file receive on.

(4) Do a <clear><7> <clear><-> which is DTD (dump to disk) off.

(5) Start the transfer at the Model 100 end.

(6) When the xfer is done, do a <clear><6> <clear><-> which is file receive off, followed by <clear><7> <clear><:> which is DTD on.

(7) When the drive stops running, do a <clear><6> <clear><Ø> which is file receive reset to close the file. --jjkd--

Fm: LDOS Support 76703,437 -> Ok, [for file transfer between LDOS LCOMM on the MAX80 and the Model 100] let's take it a step at a time:

(1) Get a null modem cable to hook the two machines together. If you want to make your own, that is not very difficult. Say so and we'll go through that.

(2) Connect them together via the cable obtained in #1.

(3) On the MAX, enter the following commands: SET *CL RS232M and LCOMM *CL

(4) On the M100, go into TELCOM and set your stats as 37I1E.

(5) On the M100, go into terminal mode. You should now be able to type back and forth between the two machines. Hitting enter will not necessarily move the cursor to the next line, thus each line may overwite the previous line on the screen. Don't worry about this yet.

(6) To send a file from the Model 100 to the Max:

(a) on the MAX, do the keystrokes in boldface which mean the text in brackets: <clear><6> [File Receive]; <clear><9> [identify the file]; filespec/ext:d [this is where you specify what file on the Max80 will receive the data]; <clear><7> [dump to disk]; <clear><-> [off (don't store to disk now)]; <clear><6> [file receive]; <clear><:> [on]. The Max is now ready to receive the file

(b) on the 100, do: <upload> [hit whatever that function key is, I forget]; <enter> [just hit enter for the width prompt]. The 100 is now sending the file. When it has finished:

c) On the Max, do: <clear><6> [file receive]; <clear><-> [off]; <clear><7> [dump to disk]; <clear><:> [on] the file is now written to the disk. <Clear><6> [file receive]; <clear><0> [reset] the file is now closed. That's it! When you've got that working, we'll cover the reverse direction. Then, we'll see about getting more speed. --jjkd--

The DOS Column

This DOS section will be covering a pot pourri of items concerning primarily LDOS; but also including some topics of LS-DOS 6.x and its most popular release, TRSDOS 6.x.

The Programmer's Guide

Victor McClung caught an error in THE PROGRAMMER'S GUIDE TO LDOS/TRSDOS VERSION 6. On page A-183, it states that you can end a string parameter with an <ENTER>. THE GUIDE is wrong. Victor was right about the required closing quotes on a string entry using an @PARAM SVC. Page A-183 of "THE GUIDE" is indeed incorrect. Checking back into earlier releases, this requirement was not added in 6.2 - it was always there. If memory serves me correctly, I was going to add this facility when I did 6.0; apparently, it never got in.

Mod4 port control under LDOS

Ronald Wick had a problem with the INVBEL3/FLT which appeared in NOTES, Issue IV. Any filters or drivers which make use of the Model 4 hardware features that are switched via a Z-80 port when operating in Model III mode generally maintain an image of the port in RAM. This is similar to the image of the MODOUT port kept in the Model III DOS products. INVBEL3 keeps an image of the memory management port in RAM address 40ADH. I discovered this by disassembling the filter. Since filters such as this do NOT initialize this mask image, they assume that something has initialized it. Well, unless you have patched your DOS, 40ADH is not initialized by LDOS. When operating in the Model III mode, the value must be initialized to zero (hex). You can accomplish this via a patch to SYS0/SYS with the command: PATCH SYSØ/SYS.RSØLTØFF (X'4ØAD'=ØØ). Note that the password shows numeric zeroes, not letter ohs. I tested INVBEL3/FLT both before and after setting that byte to zero. The filter does indeed cause a crash which is the result of the value previously contained in 40ADH. After the initialization, the filter works as advertised. I believe that I will code LDOS 5.3 to standardize X'40AD' for that use and to initialize it to a value of zero.

Model 4 Inverse Video

Here is some information which will clarify the behaviour of the Model 4's inverse video. The non-programming reader may find the discussion on inverse video enlightening. Programmer's should read it to avoid falling into a trap. The text relates to a HELP facility and the keyboard interaction.

The Model 4 hardware has a primitive inverse video. What is done is that it uses the high-order bit of the displayed byte (bit 7) to indicate that a particular character is to be inverted when the hardware is set for inverse video ON. I say that it is primitive because bit 7 is used for other things when inverse video is OFF. Thus, if the display driver accepts character values whose bit-7 is on when under normal video, these will be interpreted as inverted characters when the hardware is switched to inverse video.

The standard ASCII codes range from 1 through 127 (X'01' through X'7F' hexadecimal). Thus, the standard ASCII codes are denoted by character values using bits 0-6; bit-7 is always off (a zero, to speak). When the display driver senses the code to turn on inverse video (16d), it does two things: it switches the hardware to inverse video and flags a routine to turn on bit-7 of all characters received by the driver until the inverse video off code is received. What then happens to character values already represented by a code with bit-7 on while inverse video is ON? They will be displayed as some standard ASCII character - but in inverse video. That's why the solid block cursor which has a character value of X'B0' appears as an inverse zero which has a character value of X'30'. The difference between X'B0' and X'30' is that the former has bit-7 set.

Once the driver is in inverse video, when the code to turn OFF inverse video is sensed (17d), the driver just "un-flags"

the bit-7 setting routine. The driver cannot tell the hardware to switch to normal video because then any characters left on the screen in inverse video will be transformed into some other non-ASCII character. Only when a home-cursor (28d) is sensed, will the display driver switch the inverse video hardware to normal.

Okay, a HELP screen, which uses inverse video, accepts a response to a prompt to return to BASIC. If a function key is entered as "the key", you get an inverse video "something" if @KEYIN is used for the input or if the entered character is displayed. The @KEYIN SVC is great for accepting input lines; however, since it was designed for a "line input", it automatically displays the entries prior to returning to what invoked it. The function keys F1-F2 generate codes X'81'-X'83'. The display of the function key code entered will then look like an inverse video character. Check the appendix (page A-70) in the TRSDOS 6 manual and note that the "something" will be that shown for codes 1, 2, or 3 (shifted function keys will produce characters for codes 17, 18, or 19 if inverse video is on). The solution is to not use @KEYIN but use @KEY. The @KEY SVC is a single key entry - it does not display the character entered.

(Extended Command Interpreter)

ECI's: Hints and Kinks

Here's a few technical comments concerning ECI's used under TRSDOS 6. As far as the statements concerning SYS13/SYS on page A-180, the GUIDE is correct. The GUIDE does NOT state that control is passed to SYS13 after execution of a program invoked by @CMNDR. The SYS13 module gains control from SYS1. Now with an additional understanding of the system, one would know that @ABORT and @EXIT processing request a command entry from the keyboard (or from JCL if it's active). The @CMNDI and @CMNDR requests have had HL pointing to the user's command string. If the first character is '*', control goes to SYS13 (the ECI). The @ABORT, @EXIT, and @CMNDI functions have placed the return address of @EXIT on the stack prior to going to SYS13. The @CMNDR has not touched the stack - the return address on the stack in effect is the address following the @CMNDR SVC.

The discussion of SYS13 states nothing about register HL. The PROGRAM ENTRY CONDITIONS on page 6-100 do not pertain to SYSnn/SYS entries. However, you are correct if you observe that HL points to the character following the '*' (a kludge on the part of the the DOS in an attempt to have it follow the program entry conditions). As an aside, BC points to the '*' - which is always going to be the start of the command line buffer. BC does follow program entry conditions, DE does not.

RET opcodes are not subject to the value of EFLAG$ - they are subject to the correct coding and honoring of the stack as the ECI should follow! All programs should exit with a RET instruction if they are capable of being invoked via @CMNDR. All programs that change the value of SP should restore the SP to its initial value prior to RET'ing. That's true of the ECI, also.

LS-DOS 6.2 for the Model II/12

The LS-DOS 6.x Operating System for the TRS-80 Models 2 and 12 provides an environment compatible with the TRS-80 Model 4 running under TRSDOS 6.x. All software that uses only documented system resources of TRSDOS 6.x should operate without modification on the Model 2/12 under LS-DOS 6.x.

LS-DOS is distributed entirely on an AS-IS basis. MISOSYS will answer no technical questions regarding the use of this system, programming under this system, or on the technical interface to the system. This policy is due to the very limited distribution of this system. Because of this condition, we recommend that LS-DOS 6.x be purchased only by the technically advanced user. If you are not familiar with programming and running under TRSDOS 6 (or very familiar with LDOS 5.x), don't buy LS-DOS 6.x for the 2/12. TRSDOS 6.x Owner/Technical manuals should be purchased from Radio Shack for use with this system.

Most programs that run under TRSDOS 6.x will run under LS-DOS 6.x. Of course, any programs (such as superSCRIPSIT) which directly access the hardware of the Model 4, or otherwise violate the DOS interface conventions will not run. LS-DOS 6.x is intended to provide a Model 4 environment only. There is no support for any Model 2/12 TRSDOS functions (such as TRSDOS 1.2a, 2.0a, or 4.x). There are slight differences in keyboard and video handling, due to hardware differences on the Model 2/12. Model 4 graphics characters are not available. Reverse video is supported. Details regarding keyboard mapping are supplied with the system.

If you want to purchase a copy of LS-DOS 6.x Operating System for your Model 2/12, remit $49.95 + $3S&H (US). If you enclose an original TRSDOS 6.2 (or later) diskette as proof of purchase, we will place a copy of Model 4 TRSDOS 6 BASIC 01.01.00 onto your LS-DOS master diskette. For an additional $39.95 per system, we will include the driver and formatter for the Radio/Shack 12 and/or 15 Meg hard disk drives - the Radio Shack 8 Meg drive is not supported.

DOSPLUS4 Incompatibility note

We have received a few queries as to why some of our Model 4 products cause DOSPLUS-IV to "hang" upon program exit. This occurs in programs such as LC, MC, and most of our utilities designed to run like DOS LIBrary programs. Here's the answer. First let me say that the problem experienced using LC under DOS PLUS IV is certainly NOT a bug in LC but rather a bug in DOSPLUS. Page 227 of the Model 4 Technical Reference Manual sold by Tandy Corp (this is the manual which defines the Model 4 hardware and the technical interfacing to TRSDOS 6) states that a program may exit via one of three procedures; SVC-@ABORT, SVC-@EXIT, or by a RET instruction. The latter is used by PRO-LC since the system stack is properly maintained.

There is a reason for the use of RET over @EXIT. Any program can invoke any other program via the @CMNDR SVC provided they honor each other's memory utilization. If a program wants to invoke another and have the subsidiary program return to the invoking program, the subsidiary must use RET to exit (or the primary must take control over the @EXIT SVC). DOSPLUS does not support a RET from a program because it doesn't place the @EXIT address on the stack prior to passing control to a program invoked via @CMD or @CMNDI.

Moving a Hard Drive's directory

One LDOS MAX80 user with a hard drive had a problem when the default directory track was bad on his drive. The HDCHECK utility provided with the M80HD package can lock out a bad track only after the directory file has been written to the disk - sort of a catch-22. Here's a solution.

It is true that HDCHECK cannot lockout a bad directory track since locking out is achieved by setting allocation bits in the GAT - which is in the directory itself. If the directory track is bad, it is too late for HDCHECK.

You can establish the directory on a different cylinder by changing the "directory cylinder" byte in the Drive Control Table (DCT) for the drive in question after FORMATing with M80FORM but prior to adding the system information via FORMAT. The DCT's start at address X'4700' and occupy 10 bytes per logical drive (i.e. logical drive 0 = X'4700', logical drive 1 = X'470A', etc.) The tenth byte of each DCT points to the directory cylinder for its respective logical drive. For the drives which you are having problems with, decrease the value by 2. The DEBUG module can be used for this. This operation would need to be done anytime that you reformat the hard drive.

SYSGEN the CLOCK under LDOS 5.1.4

If you wish to be able to SYSGEN the CLOCK under LDOS 5.1.4, apply the following patch:

```
PATCH SYS7/SYS.SYSTEM (D08,C8=01 00 10 42)
```

As an aside, LSI's patch to SYS0/SYS which

allowed LDOS to always BOOT in FAST speed on a Model 4 provided no way to SYSGEN SLOW. If you wish to restore the capability of SYSGENing SLOW/FAST, apply the patch:

```
PATCH SYS0/SYS.SYSTEM (D0D,20=38).
```

The LDOS and TRSDOS6 systems inhibit the generation of a configuration file if the spooler is active. This is by design, not oversight.

LDOS 5.x SVCs and whatnot

All SVC's supported under LDOS 5 work under TRSDOS 6 with the exception of HIGH$ - which has a slightly different protocol. TRSDOS 6 includes many SVC's not included under LDOS 5. I would not recommend one program to run under both versions unless the program took great pains to adapt itself to the DOS it was running under. That, of course, limits its execution to >5200H - not too useful on the Model 4.

While we're on the subject of SVCs, Rob Healey recently reported the following. "My copy of LDOS 5.1.4 dated 8/5/84 has a bug in the SVC table. The SVC dealing with setting HIGH$ is set up incorrectly. I patched SYS7/SYS with FED to fix the problem. The address was off by one. I don't know if this bug has been recorded or not but I figured since you will be making a new update soon, it would be a good time to fix it if it hasn't been already." Rob went on to supply the Model III patch. I checked out his report and he was right! Just shows you how many people are using the optional SVC table under LDOS 5.1! Anyway, here's Rob's patch as well as one for LDOS 5.1.4 Model I.

```
I   -> PATCH SYS7/SYS.SYSTEM (DØF,56=4A)
III -> PATCH SYS7/SYS.SYSTEM (D1Ø,88=4A)
```

Caution for Directory tinkerers

Here's a programming note which addresses an obscure quirk of both LDOS and TRSDOS. It's associated with a rare form of directory access. I resolved an obscure problem with my DESCRIBE package that I thought I would share with you in case the issue ever comes up with another LDOS or TRSDOS 6 user. My DESCRIBE program adds a descriptor field to the directory. I have introduced a few tricks that permit it to function without conflict with other programs that may access the directory. You will need to understand a little of how I interface DESCRIBE before you can appreciate the "bug".

DESCRIBE associates a 64-byte record to each usable HIT entry in the directory. I extend the directory by first calculating the proper ERN for DIR/SYS and update the directory - filling extents 2-4 with X'FF's. I do this because the ERN is miscalculated by FORMAT in all cases where the cylinder exceeds 34 sectors and is grossly in error when a cylinder is 256 sectors (the ERN shows up as zero). I next set the CREATE bit and show the remainder of the directory cylinder allocated in the first extent of the DIR/SYS directory record. Thus, on a drive with a cylinder larger than the actual directory, I can make use of the wasted space. I proceed to open the DIR/SYS file for reading. I use the FORCE-TO-READ flag and the "master open" scheme under TRSDOS 6.x and the master password under 5.x. I then force write access by adjusting the open FCB. I then proceed to write out the 64-byte extensions via sector writes (@WRITE). One byte is storage for a copy of the DEC for a file. The remaining 63 bytes are for the description information. This then allocates the extra space needed by the directory descriptors. This process is performed for the CREATE command of DESCRIBE which then cycles to read the descriptor extension. When the user wishes to physically update changes to the descriptors, the extension is written back to disk but the DIR/SYS entry's ERN is reset to the value for the directory records only. Thus, other programs which access the directory get an EOF when they read the last system sector. Also, the DOS doesn't deallocate my space since the create bit is set. I don't CLOSE the directory so no date is added. The only visible indication of the extension is the "C" showing up in a DIR (I).

I had received a problem report from two customers trying to use DESCRIBE with a Tandy 15 Meg hard drive. In both a one-

head logical drive and a two-head logical drive, the customer gets an "Attempt to read system data record" error. I checked my code and could not understand where this error could materialize. My 5-meg VR Data drive would be the same configuration as a one or two head Tandy 15-Meg partition. I checked out DESCRIBE on my drive. Thus, my drive was one head with 306 physical cylinders. This is organized exactly like the Tandy 15-Meg with a one head partition. It worked flawlessly. I had one of my customers debug the problem with me on the phone reporting where to put breakpoints in the DESCRIBE's creation routine. The error 6 was returned after the first @WRITE (SVC 75). I began to suspect something funny in the hard disk driver for the Tandy drive. If the driver made the comparison on directory cylinder for both read and write, then an error 6 would result on write.

I disassembled the driver and could not see anything funny with it. I double checked the @READ system code and specifically saw the trap for passing an error-6 as no error. After spending some time thinking over the problem, I pinpointed where I was going wrong. I then checked the system code for @WRITE with VERIFY turned on. This was the culprit. If VERIFY is on, an @WRITE is followed up by a VERSEC. The VERSEC does not pass through any code to trap an error-6. Thus, when the hard disk driver gets the VERSEC function, it makes the directory cylinder comparison and will return the error-6 if the @VERSEC was to the directory. My flaw was in forgetting that a hard drive with a cylinder larger than the directory would then have @WRITEs performed on the directory cylinder and if VERIFY was on which is the default of TRSDOS 6, it would return the error. This could not happen with a floppy since the @WRITE would generate the data sector DAM. This is not the case with @WRITE an a hard drive. I didn't experience the problem since I don't run with VERIFY on.

I'm not sure that I can fault the DOS since the directory DAM is simulated on the hard drive. Also, it would be unusual to expect a system data record to be accessed via @WRITE and not @WRSSC. Thus, this quirk should not indicate any request for future changes to the DOS. I worked up a patch to DESCRIBE to deal with the possible error-6 return from @WRITE. I thought that the scenario may prove educational for you.

Am I at DOS Ready?

Bob Kaye asked, "Is there any FLAG that I can check to see if the CPU is at TRSDOS Ready?" Bob, As a matter of fact, if you have a filter in the keyboard device, when the command interpreter is awaiting an input, the CFLAG$ will have bit-2 set. This will be an indicator that the DOS is at DOS Ready. -Roy

FORMAT bug with 2-sided drives

There is a bug in all current releases of LDOS 5.1 and TRSDOS 6.2 which causes the formatting of a 2-sided disk to fail under one situation. If you attempt to format both sides of a NEVER-FORMATTED disk (in a 2-headed drive) and the last access of that drive was on the second side of the previously loaded diskette, FORMAT will lockout track Ø and won't permit the disk to be used (can't put BOOT/SYS anywhere else). The reason is due to a bug in FORMAT which does not reset the side select bit in the Drive Control Table for that drive prior to initiating the formatting. Thus, the formatting information for side Ø of track Ø actually gets written to the second side; result, the first side of track Ø is not formatted (unless, of course, the disk had been previously formatted). The fix for LDOS 5.1.4 (Model I and III) is in the PATCH section. The fix for TRSDOS 6.2.1 is also there.

Add @VDPRT SVC to TRSDOS 6.2

Here's something that a few folks have asked for over the past years (?). You all know that you can do a screen print with <CTL-:> under TRSDOS 6.2; however, a few wanted that capability as an SVC so that it could be invoked under program control. Well, here is a little patch to do just that. This alteration to TRSDOS 6.2 adds SVC #107, @VDPRT. It is official as I

requested Logical Systems to include it with Release 6.3 to be out early next year. Here's the patch:

```
PATCH SYSØ/SYS.LSIDOS
     (DØ8,78=35 Ø9:FØ8,78=F4 1A)
```

DIRectory command defaults

This comes under the department of you can't please all of the people all of the time. The TRSDOS 6 DIR command was designed to have the "A" parameter default to ON so that you would see all of the available information on files. This default was changed from OFF to ON with the 5.1 release of LDOS. Now some folks complained about that change. Here is the little patch to change Model I AND MODEL III LDOS 5.1.4's default back to OFF:

```
PATCH SYS6/SYS.SYSTEM
     (DØ5,D9=ØØ ØØ)
```

Under TRSDOS 6, the CAT command will be somewhat equivalent in/ operation; however, if you still care to alter the default of "A", it can be done. An appropriate patch to TRSDOS 6.2.X for the same change is

```
PATCH SYS6/SYS.LSIDOS
     (DØ7,DB=ØØ ØØ:FØ7,DB=FF FF)
```

Add 2-Side prompt to TRSDOS 6.2.x FORMAT

If you are using 2-sided floppy drives and wish to have FORMAT prompt you for the number of sides (as is done in LDOS) rather than have to specify (SIDES=2) on the command line, just apply the following patch to SYS0/SYS of TRSDOS 6.2:

```
PATCH SYSØ/SYS.LSIDOS
     (DØØ,81=11:FØØ,81=31)
```

DOS Notes from our Compuserve SIG

#: 64158 11-Apr-86 02:42:09
Fm: Harmon Ruble 70150,310
To: LDOS support

Problem.... Harddisk crash... Reading in from floppies and get parity error during read and all stops. I need those files and can fix or delete the one that is bad. How do I make it keep on dumping in the rest of the files. Harmon

#: 64159 11-Apr-86 04:29:53
Fm: Gary Bender 70375,1070
To: Harmon Ruble 70150,310

Harmon, if the error is in the file (and not the directory) you might try making the file invisible with the ATTRIB command before doing the backup. That way you do not have to delete it, so you can go back later and maybe try to fix it. Do a $/$ backup so BACKUP does not try to copy INV and SYS files. Don't be surprised if other files show up bad as you go. Disk faults often cover several tracks, and LDOS/TRSDOS could have extents from several files in the bad area.

#: 64381 19-Apr-86 00:25:55
Sb: RTC speed fix
Fm: Joe Sare 72667,3124

For those of you who are running model 3 software on a model 4, and use the 4 meg clock speed there is a program I uploaded to DLØ that will correct the speed of the real time clock when running at 4 meg in model 3 mode. This program also allows you to disable the RTC, or the system task routines individually or in combination. There is an LDOS patch required to make SYSTEM(FAST) work on the model 4 hardware. If anyone needs this patch, or more information on fixclock, contact me on E-mail - 72667,3124.

#: 64457 21-Apr-86 16:32:36
Sb: #64300-#Interrupt headaches!
Fm: LDOS Support 76703,437

[TRSDOS 6.2 incorporated a "SYSTEM (SMOOTH)" command which forced the floppy disk driver to turn off interrupts earlier. This resulted in less throughput problems when a disk drive's rotational speed was precisely 300 RPM. The side effect was that you could no longer type during disk I/O. -ed] The equivalent of the SYSTEM (SMOOTH) function for LDOS 5.1.4 (Model III) was published in the LSI Journal. Here it is: PATCH SYS0/SYS.SYSTEM (D05,5B=F3) --jjkd--

#: 65093 09-May-86 15:09:24
Sb: #65074-Format problems
Fm: LDOS Support 76703,437

FORMAT under LDOS and TRSDOS 6 has traditionally used the "worst-case" 6D B6 format data pattern. This is the hardest bit pattern to write and read in double density. Some folks use the pattern E5, which is worst case for single density, but is very easy in double density.

I can't count the number of times that people have called me with problems under LDOS or TRSDOS 6, but "knew" that their hardware was OK 'cause it worked with "another" OS. In the vast majority of the cases the errors didn't occur either because the "working" software "coddled" the hardware by using an "easy" format pattern (leaving the errors to blast real data that is "tougher" later), or just plain didn't catch the error!

#: 65085 09-May-86 11:07:12
Sb: #65054-Format problems
Fm: LDOS Support 76703,437

If it is significantly easier to re-format an already formatted disk, as compared to a bulk-erased disk, and this is repeatable for a given disk, it generally indicates one of two problems: (1) Poor head write amplitude, (2) insufficient head compliance. The first is a hardware problem that very little can be done about, as the most common problem is a bad head. Fixable electronic causes exist also, but are much less common. The second is generally due to insufficient pressure of the head on the diskette. This can be caused by a bad/weak head spring, warped carriage assembly or drive, bad, worn or missing pressure pad in a single sided drive, etc.

For years, I've fixed these things in addition to building them and programming them. I've never seen a software only disk analyzer that was worth the media it's supplied on, with the exception of measuring drive rotational speed. That can be done via software only with good reliability if care is taken in its use. The new Dysan Digital alignment disk and supporting software is not as bad as it's predecessors, but I don't believe that it's available for the TRS-80. It still can't match a good tech equipped with 'scope, DVM and brains. --jjkd--

#: 65370 16-May-86 01:00:30
Sb: #DOS error message mixup
Fm: Nate Salsbury 72167,1750

Joe, I'm running MC. I have a SYSTEM/JCL on Drive 0 and the 'working' JCL (called MC/JCL) on drive 1. For obscure reasons, I put a write-protect patch on my Drive 0 disk. When I tried, 'DO MC (N=progname)', I got an error message saying, "Illegal access attempted to protected file!" After a LOT of fiddling with LS-FED-II, I stumbled on the write-protect tab and, as soon as I removed it, all went smoothly. Question: Why did I get THAT message instead of "Write protected disk"?

#: 65376 16-May-86 21:15:16
Sb: #65370-DOS error message mixup
Fm: LDOS Support 76703,437

SYSTEM/JCL was successfully opened on drive zero, even though it was write-protected. Since the drive was write-protected, the file was given read-only status. A further attempt to write to this file results in the "Illegal access attempted to protected file" error. To distinguish *why* the file is protected requires additonal processing, and most utilities don't. The file could have "protected" status due to password protection, being open for write access by another process, being on a write protected disk (in this case), etc. --jjkd--

#: 65482 20-May-86 18:42:44
Sb: TRSDOS 6.2 vs LDOS 5.1.4
Fm: Bruce Travers 72215,1272

I have converted several model 3 LDOS programs to model 4 TRSDOS by replacing the CALLed routines with the appropriate SVCs. There seems to be quite a few programs that run MUCH slower under 6.2 than they do under 5.1.4., most noticably during disk I/O. A good example of this is the CMDFILE program that comes with LDOS. I find that it takes 5-10 seconds between block loads from a HARD DISK, yet it takes less than a second per block on 5.1.4. Any ideas ?? Is there a patch that I am

missing? It's a shame to go to all the work of converting some useful programs only to find that it would be faster to boot up LDOS, use the program and return to TRSDOS. Thanks in advance for any suggestions. - Bruce

re: A given program running under LDOS (FAST mode) on a Model 4 will ALWAYS run much faster than the same program running in Model 4 mode. That's because of the huge amount of background tasking being done in the Model 4 mode. LDOS doesn't have to do memory management like TRSDOS 6 does. My guess is that 25% of the Model 4 processor is spent on things such as keyboard scanning, cursor blinking, time clock counting, bank switching, video/keyboard memory management, etc. The faster clock speed of the Model 4 was supposed to account for this extra memory management as well as up the processing speed; however, when you go back to the Model III mode and use the faster speed, you have essentially boosted the performance of the Model III. Note also that there are differences in the sector interleave formatted under LDOS versus TRSDOS6. Using a disk formatted by LDOS but used by TRSDOS6 may in fact require more rotations to read an entire track of sectors. -Roy

#: 65633 25-May-86 12:50:06
Sb: directory errors
Fm: ROBERT L. BABER 74076,1236

Using LDOS 5.1.4 on the trs-80 model 1, I have experienced directory errors. The only manifestation I have observed is error messages when checking the directory with the DIRCHECK program. I suspect that the trouble might be caused by the RENAME command. Apparently, it does not update the HIT entry for extended directory records. Please advise if a correction is necessary and/or desirable.

#: 65651 25-May-86 23:01:27
Sb: #65634-message no. 65633
Fm: LDOS Support 76703,437
To: ROBERT L. BABER 74076,1236 (X)

What is the exact error message? I am not aware of any HIT problems with FXDE's. If the problem is a bad backwards FXDE pointer/link, this is a known problem with BACKUP clearing a non-existent MOD flag on an FXDE. Since this link is never used by anything other than DIR checking programs, it is not a bad problem. It will likely be fixed in the next go-round. --jjkd--

#: 65680 26-May-86 10:41:16
Sb: message 65651
Fm: ROBERT L. BABER 74076,1236

The exact error message from DIRCHECK was: Incorrect hash code at HIT X'17', should be X'60' When I rename a file covering 5 or more extents and then run DIRCHECK, this type of error message appears. If I respond to DIRCHECK that it should not fix the directory and then subsequently rename the file back to its original name, and then run DIRCHECK again, the error message does not appear. I suspect that rename stores the new hash code in the HIT entry for the primary directory record, but not in the HIT entry/entries for extended directory record(s). [It turns out that @RENAME does indeed NOT alter the hash index table position for any FXDE in use by a renamed file. This in no way inhibits any access of the file. That entry can just as well be any non-zero value. Of course, since it is set to the original hashed file name/ext, it would be nice to have it altered consistently by @RENAME. Looking at the TRSDOS6 overlay which handles @RENAME, there is no room to add such "beautifying" but functionally unnecessary code. -Roy]

LDOS obscure overlay problem

The next few messages relate to an obscure problem in LDOS 5.x. The problem arises under any condition where device output has been routed to a disk file which must be extended in size by SYS8 while the character output is coming from another system overlay. Since the requsting overlay has been overwritten by SYS8, when the device handler returns, it returns to SYS8 rather than to the overlay which was in the overlay region prior to SYS8's being invoked. The problem is obscure because you rarely build an environment which creates this scenario. Routing the device to a CREATEd file, will eliminate the possibility for error as the disk file would then need no size extension. The problem does not exist in TRSDOS 6.x as

the file I/O extender routine was moved to the resident portion of SYS0/SYS, SYSRES.

#: 65819 S3/Mod 4/4P TRSDOS 6 28-May-86 23:27:58 Sb: DOS Overlay problem Fm: Les Mikesell 70010,266 To: Adam Rubin 74126,2244 (X)

The overlay problem in LDOS would show up if output generated from an overlay (LDOS Ready from SYS1 or perhaps a DEBUG screen) is redirected to disk and comes at a time when SYS8 must be invoked to allocate disk space. The fact that no one has noticed this says something about how often device redirection is used....

#: 65914 S3/Mod 4/4P TRSDOS 6
31-May-86 16:02:09
Sb: DOS Overlay problem
Fm: Tim Mann 70040,504

Grumph. Unless you've fixed it since I worked for LSI, what Les described is a real bug too. Try linking *DO to a disk file and running the debugger for a while. When the disk file needs another extent, SYS8 will be pulled in and clobber SYS5. Then when the i/o routines try to return to the code in SYS5 that called them, the SYS8 code will be there instead and the system will crash. --Tim [Note, this bug is inherent in LDOS, not TRSDOS 6 -ed]

(RE) Yes, Tim. That sounds like a bug, too. Truth is, I can't remember doing much routing of the *DO device to a disk file then using the debugger. Actually, when I had to do some heavy debugging, I usually used my special debugger which was a SYS5/SYS9 module which relocated itself to high memory and did not operate from the overlay area. I don't think that Bill will fix that bug in 6.3. I can't get too excited over it for 5.3, either since low-memory is too critical for other things. - Roy

#: 65942 01-Jun-86 15:22:17
Sb: DOS Overlay problem
Fm: T. Lee Horne, III 70115,207

When the IRS wanted log time for my computer I tried *JL and trashed my system several times. I wrote LDOS then and they sent me a letter with this same information. Essentially, don't use *JL or you risk trashing your directory and loosing data. - Lee

(RE) Ah, but if you route it to a CREATEd file, then you should be home free. Of course, that issue is moot now, isn't it? -Roy

#: 66012 02-Jun-86 22:47:32
Sb: DOS Overlay problem
Fm: Les Mikesell 70010,266
To: T. Lee Horne, III 70115,207 (X)

Not quite; You can use *JL safely as long as it is sent to a DEVICE, not a disk file. Also, this should not be a problem under 6.x since the function of LDOS SYS8 was moved to SYS0 and is always resident. -Les

#: 65965 02-Jun-86 00:48:15
Sb: DOS Overlay problem
Fm: John Garner 72457,1613

Don't forget that this can also happen with ZSHELL as described in Issue 3 of "Notes"... - John

(RE) I didn't forget about ZSHELL. I just didn't want to stir up too many waters. Why do you think I spent the effort to make the "SYS8" LDOS code resident in TRSDOS 6? And somebody even suggested that TRSDOS 6.3 could gain resident memory space by shifting that code back into an overlay!!!!! -Roy

#: 66009 02-Jun-86 22:46:56
Sb: DOS Overlay problem
Fm: Les Mikesell 70010,266

No, I'm sure that I have run into the SYS file conflict without involving JOBLOG at all. I think the first time it happened, I had ROUTED *PR to a disk file and was using DEBUG, trying to use screen prints to capture certain areas of a program. It probably will happen anytime output from any overlay is redirected to disk and invokes SYS8. Or, as in my example, when INPUT to a overlay invokes something that generates the output. - Les

#: 66285 09-Jun-86 01:18:20
Sb: Using HD on LDOS & TRSDOS 6
Fm: John Garner 72457,1613

Yes, it can be done. I presume your hard disk does in fact have four heads. You also must have the appropriate drivers for both LDOS and TRSDOS. If you have a Radio Shack HD, the TRSDOS drivers come with it. At any rate, the general procedure would be:

(1) Boot with TRSDOS. Use a command like: SYSTEM (DRIVE=1,DISABLE,DRIVER="TRSHD6") NOTE: The driver must be on the disk in drive :0.

(2) Answer the questions the driver asks. Answer "How many heads.." with "1" and "Starting head..." with 2 (assuming heads are numbered 1-4).

(3) Repeat this for DRIVE=2 and use a starting head of 3.

(4) Repeat for DRIVE=3 and use a starting head of 1.

(5) Use SYSTEM (DRIVE=4,DRIVER="FLOPPY") to access the other floppy drive. Its physical driver number is 2.

(6) Use the hard disk formatter to format each of the the 3 partitions (drives 1, 2, and 3).

(7) BACKUP /SYS:0 :3 (SYS) to copy TRSDOS onto the hard disk.

(8) Make the hard disk the system disk with SYSTEM (SYSTEM=3). This will also make the left-hand (or lower, forgot what model) floppy drive :3.

(9) SYSGEN (DRIVE=3) to save the configuration. (Forgot to mention: disk in floppy drive should be a backup of the hard disk init disk.)

(10) Boot LDOS. Use a command like: SYSTEM (DRIVE=3,DISABLE,DRIVER="TRSHDx") Answer the questions as before, except the starting head will be 4.

(11) Format the hard disk partition with the LDOS hard disk formatter.

(12) Install LDOS on the hard disk with BACKUP :0 :3 (SYS,INV).

(13) Issue the following commands:

```
SYSTEM (DRIVE=1,DISABLE)
SYSTEM (DRIVE=2,DISABLE)
SYSTEM (DRIVE=4,DRIVER="MOD3")
```

and answer the prompt with "2";

```
SYSTEM (SYSTEM=3)
```

The effect of all this is to make the floppies drives 3 and 4 (just like under TRSDOS). It has been my personal experience that less confusion will result if the floppies are always the same drive number. The fact that drives 1 and 2 are disabled under LDOS is of no problem.

(14) Save the configuration by typing the following:

```
SYSTEM (SYSGEN)
COPY CONFIG/SYS.CCC:0 :3.
```

I have left out a few details, like copy the rest of the TRSDOS stuff onto the hard disk, etc. Also, if you are using something other than R/S hard disk and drivers, some of the steps may be somewhat different. Hope this was of some help to you. - John

#: 67126 04-Jul-86 11:51:15
Sb: #67097-MOVING FILES
Fm: LDOS Support 76703,437

Files may be moved from NEWDOS to LDOS in the following manner:

1) Format a disk under LDOS with the parameters necessary to make a thirty-five track, single density, single sided disk with a single track directory (this is also the Model 1 TRSDOS 2.3 structure).

2) Copy the files to this disk under NEWDOS.

3) Re-boot under LDOS. LDOS should now be able to directly read this disk. If also going from Model 1 to Model 3, you may need to use the LDOS "REPAIR :n (ALIEN)" command on the disk before reading.

Note that if you have PowerSoft's Super Utility Plus, it will move files directly between some NEWDOS formats and all LDOS formats. --jjkd--

(RE): MISOSYS also has a CONVDOS program available on DISKNOTES3 (available for $10 +S&H) which can "convert" files off of a NEWDOS80 double density system disk. -Roy

Assembly Language: EDAS & MRAS

On page 42, line 18340, of THE SOURCE, Volume I - The System, there is an instruction which cannot be assembled by EDAS 4.3 or MRAS. The "LD HL,CRTEND+1<" is an error in THE SOURCE. The reason that EDAS 4.1 did not flag it as an error is that it never flagged errors where an assumed term from an expression was omitted. Thus, it passed such things as "LD A,(IX+)" where the right hand term of the "+" was omitted. In THE SOURCE's example, the ending "<" is a shift operator which should have a right hand term. The EDAS 4.3 assembler flags this as an expression error since it does not want to assume that you meant to either drop the "<" or add a "0" shift operand. In other words, it flagged a legitimate error in the source code.

Certain coding conventions permitted in absolute code generating assemblers (such as EDAS and PRO-CREATE) will not be acceptable in relocatable code generating assemblers (such as MRAS and PRO-MRAS). In particular, it is important to pay close attention to the side effects of expressions. When assembled by MRAS, a code fragment such as:

```
SLASH   JR START
OLDHI   DW $-$
        DB MODDCB-SLASH-5
        DB 'SLASH0'
MODDCB  DW $-$
```

will produce many "Multiple definition" errors which are difficult to understand by most folks. It is caused by a phase error. A phase error occurs when the address of a label differs between assembler passes. The reason for it in this case may not be too obvious.

Here is the reason for the "strange" behaviour. Let's look at the above code on assembler pass one. At the evaluation of the expression in the third statement, the MODDCB symbol is not yet defined; thus, it evaluates to absolute zero - all undefined symbols are considered absolute! The "SLASH" symbol is code relative. On page 2-15 under the rules for evaluation of expressions involving subtraction, "absolute - relative" is invalid. That means that the assembler generated a relocation reference error and did not assemble the DB instruction; however, error messages are only displayed during the listing pass, pass two.

On pass two, the MODDCB symbol is defined as code relative; the subtraction rules permit the expression to be evaluated without error as both of the symbols are defined as code relative. The subtraction of two code relatives results in an absolute - which is acceptable. Therefore, no error exists on the second pass for that statement and it assembles to a byte value. Unfortunately, since the DB instruction had not been assembled on the first pass, the location counter differs on the second pass. The difference between pass 1 and 2 is one byte which thusly multiply defines all symbols which follow that statement.

The solution is to NEVER code an expression which involves a relative symbol unless it adheres to the rules noted on page 2-15. The alternative, which is one that should be followed when no "relocatable" facilities are needed, is to use the '-GC' command line switch to direct MRAS to directly generate an executable command file or use the ASEG and ORG assembler pseudo-OPs to specify the module as being an absolute segment. The code fragment came from a filter which is not one which requires relocation nor was it coded to permit it to be assembled by a relocatable code generating assembler. With the "-GC" switch, MRAS correctly assembles a file coded to be assembled by an absolute code generating assembler (such as our PRO-CREATE product).

I investigated the result of modifying MRAS to keep a "relocation reference error" as a "warning" error and not a "fatal" error. This would have the effect of keeping the byte assembled so a phase error would not result - in this case. On the other hand, a negative side effect would generate a bad link file if there was a legitimate relocation reference error.

Let me clear up some confusion about our MRAS product and its advertised compatibility with Microsoft's M-80. According to our catalog, MRAS generates M-80 compatible link files. There is no claim that MRAS is 100% cross compatible with M80/L80. MRAS cannot link the output of BASCOM because MRAS does not support the "chain address" special link item (SLI). That SLI is 12, which is excluded from our list. Such an SLI is only needed in connection with a one-pass compiler which directly generates relocatable object modules (such as FORTRAN, BASCOM, and COBOL). On the other hand, I have been contemplating upgrading our linker to support "chain address" in response to requests from some of our customers who desire to use our linker with Microsoft's language products. When we enhance our MLINK linker to support chain address, we will notify our customers as to the upgrade policy.

Some users want to use our MLIB librarian with Microsoft's libraries and run into trouble. The problem with the BASRUN/REL library was that it did not have a proper end of file pointer in the directory. Once I corrected that, I was able to properly read the library into MLIB. The same thing was true of GRPLIB/REL; it's directory entry EOF pointer was wrong and needed correction. The problem with those libraries is that the EOF pointer must be pointing to a byte which is of the value, X'9E'. BASCOM/REL is too large to fit into MLIB's buffer; thus, it cannot be operated on by MLIB.

BASIC: EnhComp

As you may be aware from both our advertising in 80 MICROCOMPUTING and the last miniNOTES flyer, MISOSYS released a new BASIC compiler. We have recently upgraded it to version 2.4 to correct some bugs which slipped by in the 2.3 release. We're pretty excited about this compiler. It was originally written by Phil Oliver who some may know as the author of EnhBas - an add-on extension to the Model III BASIC interpreter as well as that old favorite of gamesters, Scarfman. We spent a lot of time cleaning up the code, fixing bugs, and porting it over to the Model 4 native mode (i.e. a Model 4 running as a Model 4 under TRSDOS 6). We also added a few enhancements.

Okay, why come out with a new BASIC compiler for the Model I/III and 4 at this late date? The reason was that its unique operating environment and supported assembler interface demanded that it be made available to the BASIC programming community. The first "feature" is one of the programming environment. The typical compiler environment requires this scenario. Load an editor then load or input source code. Save the code to disk then exit the editor. Invoke the compiler to produce the "/CMD" program [some compiling environments utilize separate compilation, assembling, and linking phases]. Run the program to test it. Then repeat this cycle. The EnhComp environment starts with you invoking a supervisor program, S/CMD. S loads the editor. You then either load or input the source. Type "RUN" and the supervisor then takes control. It first saves your source to a temporary file, TEMP/BAS. It instructs the compiler to compile your source program to a temporary file, "TEMP/CMD". Next, this CMD file is run for testing. When it completes, S regains control, reloads the editor, which reloads your source file. If anything went wrong during the compile or program run (short of a program CRASH), S would still regain control and complete its cycle. Note how this saves you from having to type in all of the iterative program development commands.

The second "feature" is the convenience and flexibility of a built-in Z80 assembler. This is not just a rudimentary facility of allowing you to set up DATA values which represent Z80 machine instructions, this is a complete assembler. For instance, check out this little Model 4 BASIC program!

```
DIM VIDEO%(960)
FOR X = 0 TO 255
%FILL(X,VARPTR(VIDEO%(0)))
NEXT: END
COMMAND FILL(X%,V%)
Z80-MODE
LD A,(&(X%)):LD HL,(&(V%)):LD (HL),A
PUSH HL:LD D,H:LD E,L:INC DE:LD BC,1919
```

```
LDIR: POP HL:LD A,15:LD B,5:RST 40:RET
HIGH-MODE: ENDCOM
```

First, note the absence of line numbers. With EnhComp, they're not needed except when you need to reference a line (i.e. GOTO, GOSUB, etc.). You can also use labels in lieu of line numbers if you so choose. Second, note the assembler code intermixed with the "high-level" BASIC. Note in this assembler routine that it can access BASIC's variables and you can also put more than one assembler instruction on a physical line - just like in BASIC! The EnhComp "COMMAND" reserved word allows you to add to BASIC's repertoire of statements - somewhat akin to FORTH's extensibility.

Now for those BASIC types who just want to code in BASIC, how does this compiler's dialect differ from that you may be used to? To begin with, let me very clearly state that there is no intention that this compiler can directly compile any existing BASIC program designed for your BASIC interpreter. What the author of EnhComp has tried to do is to provide you a set of features that will mimic most of the features of your interpretive BASIC. In some cases, the mimicry has been perfect. Well, the only legitimate basis of comparison would be to use the BASIC interpreter as a benchmark. On the Model I and III environments, you compound the comparison because of the preponderance of different DOS vendors and differences in the disk BASIC they provided. On the Model 4, there is only one - Microsoft's BASIC as provided with TRSDOS 6. So lets take a look at that one.

EnhComp supports unlimited length variable names versus a maximum of 40 for MS. String variables, on the other hand, can be up to 32767 bytes in length with EnhComp. EnhComp does not support EQV or IMP, all other numeric and string operations are supported. For sequential files, EnhComp supports OPEN, PRINT#, PRINT# USING, WRITE#, INPUT#, LINE INPUT#, EOF, and CLOSE identically as Microsoft. We consider LOC to be superfluous for sequential files. For direct access "R" files, EnhComp supports fielding and access exactly like Microsoft. EnhComp also supports a special "X" type of direct access file which uses list-directed fielding (i.e. XFIELD 1, VAR#, (32)VAR$, BIG!, LITTLE%). The "X" type allows for logical record lengths up to 32767!

For statements, EnhComp does not support CALL, COMMON, ERASE, OPTION BASE, CHAIN, WHILE WEND, WAIT, DEFUSR, NAME, or SOUND. On the other hand, EnhComp supports BKOFF and BKON to disable/enable the BREAK key; DRAW with ROT and SCALE for turtle-like graphics; COMPL(), INVERT, PAINT(), PLOT, RESET(), and SET() for pixel graphics; COMMAND and ENDCOM to define extensions to BASIC statements, INC and DEC to increment and decrement integer variables; UP, DOWN, LEFT, and RIGHT to scroll the video screen in four directions; IF ELSE ENDIF conditional construct; FUNCTION and ENDFUNC for multi-line user functions; labeled statements for readability; INPUT@pos for better control of input prompts; the ability to SORT arrays with SCLEAR, KEY, TAG, and SORT; ON BREAK GOTO address for programmer control over BREAK key handling; POP to remove one level of GOSUB; POSFIL to reposition sequential files; a SYSTEM command which allows you to invoke ANY command; PZONE and SZONE to establish actual tab stops for printer and video output used with PRINTing; REPEAT and UNTIL for looping (similar to WHILE WEND); and finally WPEEK and WPOKE for easier memory access of 16-bit "words". Note also that EnhComp's PRINT# statement supports redirection to video, printer, or disk file by the value of the buffer number (-3 for printer, 0 for video, 1-n for file).

EnhComp supports every Microsoft numeric function; however, we also support double precision arguments with DOUBLE PRECISION results. That includes ABS, ATN, COS, EXP, LOG, SIN, SQR, and TAN. The "EXISTS" function can be used to check if a particular file is available before you proceed to OPEN it. We don't support ERRS$, INPUT$, or SPACE$ (which can be derived from STRING$) but we do add BIN$ and WINKEY$ (which of course can be derived from INKEY$). Note that EnhComp's USING is actually a function which returns a string value. This means that it can be used by itself without a PRINT statement, which adds the flexibility of continued processing of the formatted string by your program before printing (perhaps like the

sprintf() function of C) EnhComp does not support SPC (again, this can be accomplished with STRING$ as EnhComp's STRING$ function uses no string buffer space!).

Now chaining and commons - both used to pass in-memory data from one program to another - can be easily accomplished by EnhComp. Since EnhComp allows you to establish the origin of the compiled /CMD program, simply set it to an address higher than the default (5200H I/III or 2600H 4). Then use the region between the base and your origin to poke the variable contents which you want to pass from one program to another. Since you can RUN a compiled program from another, the poked data can be easily peeked by the second program. There, that's chaining and common!

What more do you want? Okay, we have received a request for a SETEOF statement which allows you to reset the EOF pointer of a random access file. We have also been asked for support of the Tandy HIRES graphics board. The former is very easy compared to the latter. Here's an example program illustrating an easy user command designed to set the EOF of a random access file.

```
COMMAND SETEOF(BUFNUM%)
IF NOT EOF(BUFNUM%)
Z80-MODE
LD HL,(&(BUFNUM%)):CALL @CALADR
LD A,(IX+16+5):LD (IX+16+8),A
LD A,(IX+16+10):LD (IX+16+12),A
LD A,(IX+16+11):LD (IX+16+13),A
HIGH-MODE
ENDIF
RETURN
ENDCOM
ALLOCATE 1
OPEN "R",1,"testfile/dat",32
FIELD 1,32 AS ARG$
FOR I = 1 TO 20
LSET ARG$="This is test "+STR$(I)
PUT 1,I: NEXT
CLOSE 1
SYSTEM"List testfile/dat (hex)"
OPEN "r",1,"TESTFILE/DAT",32
FIELD 1,32 AS ARG$
GET 1,10: PRINT ARG$
%SETEOF(1)
CLOSE 1
END"list testfile/dat (hex)"
```

The BASIC supplied with LDOS documents the operation of SETEOF conforming to the manner in which I have used it in this test program. The random file is positioned to the last record desired by means of a GET on that record. Then the SETEOF [of course coded as %SETEOF(bufnum) since EnhComp requires the "%" character to indicate a user command] will set the EOF pointers to the current record pointers. The little Z80 routine does that. "@CALADR" is a routine from the EnhComp SUPPORT/DAT library which calculates the address of a bufnum's file buffer allocation (documented on page 5-2) and returns the value in register IX. To be fancy about it, the CF would be set on return from @CALADR if the bufnum referenced a file which was not already open. Of course, under that case, nothing damaging would result as the accessing of the FCB region would be of unused memory addresses. Make note that if the bufnum exceeds the value established by ALLOCATE (the maximum number of open files), @CALADR would report a runtime error 104.

Now I can't pull an easy rabbit out of the hat for HIRES graphics. However, the simplicity of including assembly language routines and the availability of public domain HIRES libraries should enable some dedicated graphics hackers to come up with usable functions.

Where is MISOSYS taking EnhComp from here? Well, to start, we are dedicated to eliminating any other bug which becomes documented. We are also making it known to Phil Oliver that we are interested in an MS-DOS version of EnhComp. Also, as additional requests for features are made, we will be considering each and every one of them. I really feel that every BASIC programmer has ample reason to purchase a copy of EnhComp. If you don't like it, you tell me why [Roy Soltoff].

Here's a couple of items from our Compuserve Special Interest Group (PCS49).

#: 66854 23-Jun-86 Fm: Bill Warner 75126,603 -> Roy: Two questions about EhnComp: (1) How close is the syntax to

LBASIC (e.g. would it be fairly easy to convert an existing program)? (2) Is the final program self-contained or must the support library be on-line at run-time? Thanks -Bill

(RE) EnhComp is pretty close. It has some different reserved words. For example, RDGOTO is used to reposition a DATA pointer. I used 'SYSTEM"string"' to implement the 'CMD"..."' of LBASIC. However, EnhComp includes a preponderance of extra statements. For example, it supports: INPUT@pos, "printstring"; var..... It supports BREAK key handling with 'ON BREAK GOTO label' which is similar to 'ON ERROR GOTO label'. The sort syntax is different. It doesn't use CMD"O" but a series of statements: SCLEAR to clear the sort work space; KEY to specify the key array, TAG to specify the tagalong arrays, if any, and SORT to invoke the actual sort. But it also has REPEAT-UNTIL, program-controlled print zones, WPOKE and WPEEK. USING is actually a string function; thus, you can set the result of a USING into a string for further manipulation. That's pretty powerful! It even provides INC and DEC of integer variables for speed (like ++i and --i in c). Enabling and disabling of the BREAK key is done via BKON and BKOFF rather than via CMD"B,"sw". It uses LOAD to load an object file instead of CMD"L","spec". Certainly, MERGE is irrelevant. RUN invokes another CMD program (no saving of variables). Also, the date and time are retrieved as DATE$ and TIME$ (8-characters each). There is no USR function as the built-in assembler lets you have the "assembly" routines right with the BASIC. Besides, with an assembly interface, you could easily create your own USR. By the way, the generated CMD file is standalone - whatever routines which are needed from the SUPPORT/DAT library are linked during the compilation process. Enough? -Roy

Here's an interesting "accuracy" benchmark from Jeff Brenton [CLMFORUM] 76703,1065. Once I fixed up the rounding kludge in Enhcomp, it's result was 2499.999999.

```
100 ' time and accuracy test in BASIC
110 DEFINT I: DEFDBL A
120 ILOOP=2499
130 A = 1
135 PRINT TIME$
140 FOR I=1 TO ILOOP
150 A=TAN(ATN(EXP(LOG(SQR(A*A)))))+1.0
160 NEXT I
170 PRINT USING "A = ####.######";A
175 PRINT TIME$
180 STOP
```

The answer SHOULD be 2500.000000, but MS-DOS will come up with 2700+, and TRSDOS/CPM will come up with 2300 or so. Even Microsoft FORTRAN for 8080 machines will come up with the same horrid results. [by the way, the run time is much more than a few minutes - don't start thinking that the program crashed -Roy]

The C Language

In case you may not have realized, MISOSYS has released its full-C compiler implementation for the Model I/III under LDOS (called MC) and the TRSDOS-6 compatible version called PRO-MC. This compiler was released on 07/10/86 and is version 1.5a. The compiler requires the use of a relocatable macro assembler which supports Microsoft's REL conventions. Either M80 or our own MRAS or PRO-MRAS assemblers are suitable.

The following notes covering this release may be useful for your consideration.

For Model I/III users, the MC compiler is a very big set of executable command files. As such, only about 5-6K of memory space is available for the compilation process. You must utilize the smallest possible amount of high memory for purposes other than the compiler. The size of the source code file which can be compiled is limited by this high memory space available.

The following are the only known language enhancements and limitations:

(1) Typedef is supported. All typedefs are global; that is, none are local to any particular block. The full typedef syntax and usage is supported, with the following exceptions: typedefs may NOT be modified with the keywords "short", "long", "unsigned"; likewise, the keywords

"short", "long", "float", "char", and "int" may NOT be modified by a typedef.

(2) Enumerated types are supported. Like structure and union types, enums are global; there are no local enums within a block. Enum variables are treated identically to signed short integers, and can be used interchangeably. The full enum constant list syntax is accepted; each enum constant is treated as if it were the equivalent integral constant. Strict type checking between enum variables, enum constants, signed short integers, and integral constants is NOT done.

(3) Bit fields in structures are fully supported. All bit fields are treated as unsigned short integers; there are no signed bit fields. Bit fields are ordered from high (most significant bit) to low (least significant bit) in a machine word (16 bits). The ordering is such that the first declared bit field will start in bit 15 of a double Z80 register (e.g. bit 15 of HL, or bit 7 of H), and so on. Note that when a word containing bit fields is fetched from or stored into memory, the high and low bytes are reversed! This reversal must be taken into account when creating a set of bit fields to describe an externally imposed format (e.g. a bit structure maintained by the operating system). The reversal is of no importance if the structure is wholly contained within the C program, and not used outside of it.

(4) When you initialize a pointer with a static object, you can use the address of the object plus or minus an offset (which gets scaled); you cannot use the address of a subscripted object. Similiarly, you cannot take the address of a structure/union member in an initializer.

Here is some dialog concerning C which recently appeared on our Compuserve Special Interest Group. I suspect that some QUARTERLY readers will find it educational, as well.

#66168 07-Jun-86 Les Mikesell 70010,266

Confusing = and == is a fairly common error in C, especially in statements like "if(x=y)". The compiler doesn't complain because this is a legitimate statement - it assigns the value of y to x and the "if" test will evaluate TRUE if y is non-zero. However, the programmer usually means "if(x==y)" which just tests if the values are equal. Since BASIC uses the same syntax for assignment and tests (A=1 vs IF A=1) the operator is context sensitive. Only the left-most "=" will do an assignment, the others yield boolean results of tests for equality. - Les

#66268 08-Jun-86 JOHN DEHELIAN 72667,1744

Simple question to anyone who would care to answer me; does PRO-MC support the 'void' keyword. I am reading a book by Jack Purdum that makes reference to it and from what I understand (I haven't actually programmed in C yet) this makes functions work as subroutines in pascal in that the function doesn't pass a value back to a variable assigned to that function. I guess an example might be best here. Functions normally operate as such: a = func(parms); Now from what I understand this keyword allows one to call a function in the same format as a pascal sub: ex: exchange(parm1,parm2) where you don't actually want to pass a single value. By the way, is it possible to pass back several parameters to the calling program through a parameter list in C? I'm thinking of a Fortran subroutine where any variable in the parameter list that is changed during the subroutine is passed back to the calling program.

[MC does support type void; however, its use is strictly to trap as an error, an attempt to assign the non-existant return value of a void function. -Roy]

#66278 09-Jun-86 H. Brothers 70007,1150

MC supports the use of "void" but it is not necessary in most implementations of C (including MC). Just because a function (equivalent to both functions and procedures in Pascal) returns a value, there is no necessity to use that value nor even assign it to another variable. In C, all parameters sent to a function are passed "by value." This would seem to be a handicap, because it implies that the receiving function cannot change the

"original" copy of any variable it receives. However, if you pass a pointer to a variable, the receiving function can then change the original value, since the only thing it can't change is the value of the pointer (i.e., the address of the storage location for the variable). And, if you wanted to change a pointer for some reason, you could always pass a pointer to the pointer, etc. ad infinitum (well, at least to the limit of most human brains to comprehend what's happening). -- Hardin

#66287 09-Jun-86 Les Mikesell 70010,266

The keyword "void" simply means that a function does not return ANY value. This is only useful when an advanced syntax checking program (like "lint" under unix) parses the program to determine if the function return value is the same data type as the variable being assigned. In other words, MC will simply ignore the word void (as opposed to giving an error message if it didn't support it). This has nothing to do with the second part of your question that related to "call by value" as opposed to "call by reference". In C, when a variable is passed to a function, the function gets a *copy* of the variable that can be treated as conveniently initialized local storage. However, if you want the function to be able to alter the callers copy of the variable, just pass the "address of" the variable using the & operator, and manipulate in the function using the "pointer to" operater *. Your exchange function would look like this:

```
int a = 1; int b = 2;
exchange (&a,&b);
..more program
exchange(x,y)
int *x,*y;
{ int tmp;
    tmp = *x;
    *x = *y;
    *y = tmp;
}
```

Note that this only applies to "normal" variables - array names are treated like the address of element 0 of the array, so that arrays are passed to functions by reference. However, the boundaries of the array are not automatically known by the function. - Les

#66345 11-Jun-86 jeff brenton 76703,1065

but the new compilers [on larger systems - ed] will generate a warning if you have: extern int puts(char*); in a header [as the prototype for puts()] but fail to cast it when you ignore it's return value, as in: (void)puts("this is a string\n"); the added overhead is to insure that the programmer KNOWS that puts() actually returns a value, and that s/he INTENDED to ignore it.

#66336 10-Jun-86 JOHN DEHELIAN 72667,1744

Thanks Les, What significance do the & operators have in the parameter list of your example exchange(&a,&b)? Is this operator necessary whenever variables are 'passed by reference'? Thanks again.

#66341 11-Jun-86 John J. Stein 74056,673

The '&' operator tells the function that it is getting an address of a variable rather than a copy of it's value. Using this address, you can change the value of the variable from within the function. The scanf() function uses this operator. When you want to input a decimal number from the keyboard called num, you use scanf() like this: scanf("%d",&num); The & will pass the address of num to scanf so it can put the number you input into the variable.

#66378 12-Jun-86 Les Mikesell 70010,266

I think you have the right idea but your comment that the & "tells the function that it is getting an address" is a bit misleading. Actually the & is processed BEFORE the function is called, so that an address is passed instead of the value of the variable. The function has to already know to expect the address rather than a value. - Les

#66340 11-Jun-86 John J. Stein 74056,673

Like Hardin has said, just because a function returns a value doesn't mean it has to be used. For instance, the function puts() returns an integer, representing a status code, depending on the outcome of the function. If the status is equal to

EOF, then the puts() was unsuccessful. You'll notice, though, that no one [few, not necessarily "no one" -ed] uses this value. So in programs you'll see just puts("This is my message"); rather than status = puts("This is my message"); Both examples are equally valid, and will execute the same way. From what I've seen so far, the void type does nothing but make sure that you don't try to assign a return value to a function that wasn't supposed to get one. You don't seem to be "saving" anything by using void.

#66334 10-Jun-86 JOHN DEHELIAN 72667,1744

Thanks Hardin, I would just like to say that I've admired your work in 80-Micro. Do you mean, when you say there's no need to assign a variable to a function, that I can define a function without the void statement and then call it by simply writing its name with all necessary parameters? For example, say I wrote a function to swap two variables. Could I call the function like this? --> swap(i,j) <-- or would I have to assign some variable, void=swap(i,j), where void is defined as int. This is the method the author suggests as an alternative to using the void statement if it is not supported. He explained that in this way the information passed back to void will not be used by mistake as useful data.

#66405 13-Jun-86 Roy Soltoff 70140,310

That's NOT how MC's void function works. The 'void' keyword is used to declare a function {similar to what you would do to declare a function which returns a value}. Thus, the declaration, void func(); declares "func" to be a function which has no returned value. If later in your program, you have coded, var = func(), then MC will flag that as an error! It is one more means of ensuring that your coding does not get you into trouble. -Roy

#66377 12-Jun-86 Les Mikesell 70010,266

You are perfectly free to use or ignore the return value from a function as you like. Unless otherwise declared, the compiler will assume that all functions return an int value. - Les

#66376 12-Jun-86 Les Mikesell 70010,266

OK, let's start from the beginning here. When you use a simple variable name in 'C' you refer to the current value of the variable. If you use "&variable" you refer to the "address of" the variable or the place in memory where the variable value is stored. This is also called a "pointer to" the variable. A reference like &variable actually generates a constant, but C allows you to declare variables with types of "pointer to" the other data types so that address values can be manipulated. To access the value of a variable given its address, you use the construction "*address" which means "get (or set) the value this pointer is pointing to". Now, to confuse things just a bit more, consider how a function call works. First, all the parameters are evaluated and their values are placed on the stack, the function is called and its return value may be used or ignored. The compiler does not do any particular checking to see if the parameters are in any way what the function expects - that is up to the programmer (the function may be compiled separately). So, when ordinary variables are passed (i.e by value), the function just gets a copy of that value pushed on the stack to appear like a local variable to the function. It *cannot* affect the callers copy of that variable. However, if you want the function to be able to manipulate the callers variables directly, you can pass the address of the variable (using the &variable syntax in the function call) and the function can then modify the contents of that address. This is the way you achieve a "call by reference". Note that the function must be aware of the type of data passed and use the *address operator to modify the contents of the passed address. Some general points: global variables are available to functions without being passed at all (and may be manipulated by the functions). Array names are generally equivalent to the address of element 0 of the array (except that sizeof(arrayname) returns the size of the array, not the size of a pointer).

Turning now to our correspondance file, someone asked us why our new MC C-compiler

required a relocatable assembler that was M-80 compatible (that's Microsoft's assembler). The question originated from someone already owning Tandy's ALDS relocatable macro assembler. Good question; here's the answer. All relocating macro assemblers may appear to work the same way; however, there is a world of difference between ALDS and MRAS. As far as MISOSYS is concerned, the "standard" protocol of a relocatable module is that defined by Microsoft since their MACRO80 assembler was first on the 8-bit market many years ago. The relocatable module file structure is used by Microsoft's M80 assembler, F80 FORTRAN compiler, BASCOM BASIC compiler, and their COBOL compiler. ALDS is not a Microsoft product. It was written by Tandy. ALDS is not compatible with Microsoft's relocatable module structure. When we wrote MRAS, it was our decision to support the Microsoft format; thus, all of the libraries provided with the MC compiler are in Microsoft relocatable format. That is the primary rationale for requiring either M80 or MRAS. We cannot support another format. It is unfortunate that Tandy chose to implement a non-standard format in their ALDS package, especially considering that they already SOLD Microsoft's product. M80 is included with their FORTRAN package.

Scott A. Loomer of Newburgh, NY writes of EDAS Version 4.3, "the EDAS enhancements sound very useful. I've already patched in the DATE and TIME feature. In my LC/ASM file, I've added the following right after the ORG [statement] and before the line labeled "@START":

```
        ORG nnnn
CDATE   DATE
        DB    0
CTIME   TIME
        DB    0
@START  ...
```

This allows me to place the following line in my title headers for 'LC' programs

```
printf("Revision: %s, %s\n",CDATE,CTIME);
```

so that I can tell the dozens of versions that seem to hang around my system." [Note to MC users: You can use the __DATE__ preprocessor macro for this purpose. -ed]

The following book reference comes from David B. Lamkins, of Canton, MA. "C: A REFERENCE MANUAL", by Harbison and Steele published by Addison & Wesley, 1984. David writes, "This is very thorough and well-organized, having been written from the viewpoint of compiler implementors. This is for the more experienced C programmers, as there is virtually no tutorial material. I would like to see this become a 'standard supplement' to K&R, as it brings C into the 1980's with enum types and separate name spaces for structure tags. Furthermore, ideosyncracies are brought to light throughout the book; some may surprise even veteran C programmers."

Ian Kluft writes, "I tried the C stack eater [from NOTES Issue IV. -ed] under CP/M+ Alcor C on my 128K system. It didn't stack up (no pun intended) well against the competition. It did 1256 recursions before giving a 'stack overflow' error. If I could have used the extra bank for the stack, it might have done better. Your LC is quite a product.

Michel R. Coutu writes, "I would like to provide you with a simple encryption program for your next edition of NOTES. I do not know if you have many of these encryption programs, but this one is simple and works very fast. I use it at work on an IBM PC in order to protect some text files. Incidentally, I have this program in compiled BASIC and it crypts/decrypts a 24K data file in 2:54 minutes while LC does it in 1:04 minutes. Excellent!

```
/* simple exclusive OR encryption/decryption filter */
```

This program is a simple encryption/decryption filter using rotating half sums with the key. If used with two successive keys (under UNIX, QNX, ZSHELL with piping) whose lengths are relatively prime, crypt is equivalent to a key of length to the product of given key lengths, this will deter all but the CIA or very determined hackers. The encrypted file can be made with the following command:

```
crypt key1 <fnamein | crypt key2 >fnameout
```

You can recover the plain text file by the same command but reversing the order of fnameout & fnamein.

One caution: This program works very well with ASCII text files. However, you must be careful with non-ASCII files as an exclusive OR with 00H will produce the character in the key.

```
/* crypt/ccc  Program.
   Public domain provided by Quantum Software, Inc. Ottawa, publishers
   of the QNX operating system for the IBM PC.
   This version for MISOSYS' MC running on a TRS-80 Model 4.
*/
#include stdio.h
main(argc, argv)
int argc;
char *argv[];
{
    int strlen(), getchar(), putchar(), int fputs();
    int i, c, keylen;
    if(argc==2)
        keylen = strlen(*++argv);
    else {
        fputs("Use: crypt key\n",stderr);
        fputs("To encrypt  crypt key <fnamein >fnameout\n",stderr)
        fputs("To decrypt  crypt key <fnameout >fnamein\n",stderr);
        exit(1);
        }
    for(i = 0; c = getchar()) != EOF; ++i)
    putchar(c ^ (*argv[i % keylen]);
}
```

A CC for MC by Mike Bedore

First, I would like to express my gratitude to both Roy Soltoff and Rich Deglin for the fine job done with the MRAS package and the MC compiler. I was especially impressed with the effort taken to maintain as much compatibility with UNIX as possible, perhaps because I do a bit of C programming on a Xenix system owned by a local fellow. It is really nice to be able to easily port programs from the Xenix environment back to my Model I.

Along this line, I wanted to be able to compile, assemble, and link several source and object files with a single command line on my Model I, much like you can do in the UNIX/Xenix environment. Well, I was mildly successful in achieving this objective, and I thought you might be interested in the result.

What follows is a program called "cc" (what else?), the C source code (in 2 parts) "ccmain" and "ccpart2", and the documentation for the program.

The syntax isn't perfect (at least not exactly the same as most C compilers under UNIX/Xenix), but the program will eliminate much of the work associated with

compiling, assembling, and linking separate files and modules using MC, MRAS, and MLINK (other than writing the code in the first place!).

CC was compiled on my Model I and thus (alas) can't be used on a Model 4 as is. The documentation for CC contains an additional note about compiling cc on a Model 4.

```
NAME
    CC - Compile MC Programs (requires MISOSYS' MRAS package)
SYNOPSIS
    CC builds and executes an LDOS JCL file to compile,
    optionally optimize, assemble and link one or more source
    and object files into an executable program file.
SYNTAX
    cc [-option] [-option] [...] main.c [file.c...] [prog.o...]

    This documentation is for Version 1.0, 26-Jul-86,
    By: Mike Bedore [75206,3164]

The options are as follows:

-c  Compile and assemble only (no link).
    (default: compile, assemble and link).

-C  Compile only (no assembly or link).
    (Default: see -c option)

-d n    Look for source files and write /CMD file to drive "n".
    (defaults to drive 1)

-e  Do NOT execute the JCL file.

-k  Delete temporary files as the work progresses. This may
    help those who are short on disk space (i.e. 2 drive
    system). (default: no delete)
    NOTE: /REL files are NOT deleted in any case.

-m  This option is only available when cc is compiled on a
    model 4 (with DOS6 defined). When running on a model 4,
    cc will normally use the single program MC/CMD rather
    than MC1 followed by MC2. If for some reason you want
    to use the MC1/MC2 combo, specify -m

-o name Write linker output to "name" instead of the default
    (defaults to same name as the first source file found).
    Do NOT include an extension or drivespec! The extension
    will be "/CMD"; the file will be written to the drive
    specified with the -d option.

-O  Optimize all source files prior to assembly.
    (default: no optimization.)

-t n    Write temporary (/TOK,/ASM,/REL) files to drive "n"
    (defaults to drive 1)

-v n    Use a virtual memory file for MLINK when linking the
    final output. The MLINK/VMF file will be on drive "n".
```

(default: will NOT use VM)

NOTES

Options may be in any order, may NOT be concatenated, and MUST precede all input filenames.

The -d, -t and -v options MUST have a space between the option flag and the drive number ("n" above), and the drive number MUST be present in the form of a single decimal digit.

Input filenames must NOT include extensions or drivespecs (other than .c or .o).

All input filenames MUST end in ".c" (source files) or ".o" (relocatable object modules).

Source (".c") files and object (".o") files may be in any order, but they must appear after all options, and the first "file.c" encountered must contain main().

A maximum of 8 source files and 8 object files in any one "run" is supported.

Any "file.c" will actually use "FILE/CCC" and any "file.o" will use "FILE/REL".

The /JCL file "CCJOB/JCL" is written to the first available drive.

For model 4 use, #define DOS6 in "CCMAIN/CCC" before compiling it.

```
/* cc.c - Vers 1.0, 26-Jul-86, by: M. J. Bedore [75206,3164]
 * An alternative to MC/JCL for compiling, assembling
 * and linking MC produced program modules.
 * NOTE: this program requires the MISOSYS MRAS package
*/
#include <stdio.h>
#define MAXF    8
#define JCL "ccjob/jcl"
#define NSIZE   15
/* #define DOS6 */
/*  The above #define is needed for compilation on a model 4 */

extern int split();
extern void doopt(),doasm(),dolink(),abend();

char *cfiles[MAXF],*ofiles[MAXF],ofname[NSIZE];

/* drive defaults */
char *drive = ":1",     /* input and output drive */
     *tdrive = ":1",    /* temporary files drive */
     *vdrive = "  ";    /* actually just storage for -v
                           option if selected */

char *ext = "/asm";     /* <-- should NOT be changed! */
```

```
int cctr,octr,rmof,offlag,vflag;
#ifdef DOS6
    char *rmcmd = "remove";
#else
    char *rmcmd = "kill";
#endif
/*-------------------------------------------------------------*/
main(argc,argv)
int argc;
char **argv;
{
    int asm,link,opt,exec;
#ifdef DOS6
    int mc = TRUE;
#endif

    char *strncat(),*strcpy();
    FILE *fp, *fopen();

/* set option defaults */
    asm = exec = link = TRUE;
    opt = offlag = rmof = vflag = FALSE;
    cctr = octr = 0;

/* ok, now parse  option flags from command line */
    while(--argc > 0 && (*++argv)[0] == '-')
        switch( *(argv[0]+1) ) {
            case 'c': link = FALSE; break;
            case 'C': asm = link = FALSE; break;
            case 'd': strcpy(drive,":");
                      strncat(drive,*++argv,1);
                      argc--; break;
            case 'e': exec = FALSE; break;
            case 'k': rmof = TRUE; break;
#ifdef DOS6
            case 'm': mc = FALSE; break;
#endif
            case 'O': opt = TRUE; break;
            case 'o': strcpy(ofname,*++argv);
                      argc--; offlag = TRUE; break;
            case 't': strcpy(tdrive,":");
                      strncat(tdrive,*++argv,1);
                      argc--; break;
            case 'v': strcpy(vdrive,":");
                      strncat(vdrive,*++argv,1);
                      argc--; vflag = TRUE; break;
            default:
                abend("cc: illegal option %c\n",*(argv[0]+1));
                break;
        }
    if(! link) offlag = FALSE; /* no point in output w/o link */
    if( ! argc ) /* no arguments left! */
        abend("cc: NO input file(s) found!\n",NULL);
    if((fp=fopen(JCL,"w")) == NULL)
        abend("cc: Can't open %s for output!\n",JCL);

/* now process the input filename(s) */
```

```
    do {
        if(split(*argv++)) { /* if split() file had ".c" */
            fprintf(fp,"mcp %s%s +o=%s\n",cfiles[cctr],drive,tdrive);
#ifdef DOS6
            if(mc)
                fprintf(fp,"mc %s%s +o=%s\n",cfiles[cctr],tdrive,tdrive);
            else {
#endif
            fprintf(fp,"mc1 %s%s\n",cfiles[cctr],tdrive);
            fprintf(fp,"mc2 %s%s +o=%s\n",cfiles[cctr],tdrive,tdrive);
#ifdef DOS6
            }
#endif
        if(rmof)
            fprintf(fp,"%s %s%s%s\n",rmcmd,cfiles[cctr],"/tok",tdrive);
        cctr++;
        }
        else octr++;
    } while( --argc > 0 && cctr < MAXF && octr < MAXF );

    if(argc)
        abend("cc: Too many input files!\n",NULL);
    if(opt) doopt(fp);
    if(asm) doasm(fp);
    if(link) {
        dolink(fp);
        fprintf(fp,".cc: Completed Compilation of \"%s\"\n",offlag == TRUE ? ofname :
cfiles[0]);

    }
    fprintf(fp,"//exit\n");
    fclose(fp);
    if(exec)
        execl("do","do","=",JCL,NULL);
    else
        exit(0);
}
```

```
/* ccpart2.c  Vers 1.0, 26-Jul-86, M.J.Bedore */
#include <stdio.h>
#define NSIZE   15

extern char *cfiles[],*ofiles[],ofname[],*drive,*tdrive,*vdrive,*ext,*rmcmd;
extern int cctr,octr,offlag,rmof,vflag;

void doasm(fp)
FILE *fp;
{
    int i;
    for(i = 0; i < cctr; i++)  {
        if(i)  /* this is NOT the 1st file */
             fprintf(fp,"mras %s%s%s +o=%s -nl\n",cfiles[i],ext,tdrive,tdrive);
        else  /* 1st one seen...must have main() */
             fprintf(fp,"mras mc +I=%s%s%s +o=%s%s -
nl\n",cfiles[i],ext,tdrive,cfiles[i],tdrive);
        if(rmof)
            fprintf(fp,"%s %s%s%s\n",rmcmd,cfiles[i],ext,tdrive);
    }
}
/*-----------------------------------------------------------*/
void abend(s1,s2)
char *s1,*s2;
{
    fprintf(stderr,s1,s2);
    exit(1);
}
/*-----------------------------------------------------------*/
void doopt(fp)
FILE *fp;
{
    int i;

    for(i = 0; i < cctr; i++)  {
        fprintf(fp,"mcopt %s%s\n",cfiles[i],tdrive);
        if(rmof)
          fprintf(fp,"%s %s/asm%s\n",rmcmd,cfiles[i],tdrive);
    }
    strcpy(ext,"/opt");
}
/*-----------------------------------------------------------*/
int split(arg)
char *arg;
{
    int cflag = FALSE;
    char *pp,*p;
    char *strchr(),*strcpy(),*malloc();

    if((pp = strchr(arg,'.')) == NULL)
        abend("cc: missing type specifier on %s\n",arg);
    *pp = '\0'; /* terminate the name */
    if((p = malloc((unsigned) NSIZE)) == NULL)
        abend("cc: out of memory in split()\n",NULL);
    strcpy(p,arg);
    switch(*++pp) {
        case 'c': cfiles[cctr] = p; cflag = TRUE; break;
```

```
        case 'o': ofiles[octr] = p; break;
        default:
          *--pp = '.' /* restore string argument */
          abend("cc: bad type specifier %c on %s\n",arg);
    }
    return(cflag);
}
/*-----------------------------------------------------------*/
void dolink(fp)
FILE *fp;
{
    int i;

    fprintf(fp,"mlink -a=y\n");
    if(vflag) fprintf(fp,"-v=%s\n",vdrive);
    for(i = 0; i < cctr; i++)
        fprintf(fp,"%s%s\n",cfiles[i],tdrive);
    for(i = 0; i< octr; i++)
        fprintf(fp,"%s%s\n",ofiles[i],tdrive);
    if(offlag)
        fprintf(fp,"-n=%s%s -e\n",ofname,drive);
    else
        fprintf(fp,"-n=%s -e\n",drive);
}
```

EDITORS: SAID

Here is some information wich may clear up some confusion concerning SAID. There is no such thing as "insert tab". TAB is treated like any other character entered from the keyboard. If you are in overstrike mode, the depression of the TAB key will overstrike the character under the cursor with a TAB character. The display will then be refreshed to expand the TAB which you have just entered. SAID does not provide any detectable keystroke to be able to reposition the cursor to tab stops on the screen - there just aren't enough keys on the keyboard.

SAIDINS does indeed get confused with what gets displayed when you depress a <CLEAR> plus a letter <@,A-Z> key and/or the <SHIFT> key. This is because the keyboard driver inverts the sense of the SHIFT key when returning key codes for <@,-Z> depressed simultaneously with the <CLEAR> key. For instance, <CLEAR><A> generates X'C2' which should be noted as a shifted-A and CLEAR. Unfortunately, the DOS keyboard driver and hence the SAID internal keyboard driver takes care of this inverting. I believe that Karl may have overlooked this esoteric fact of operation when he coded the display result. It will be corrected if SAIDINS ever gets regenerated. At this point, we feel that the "bug" is not severe enough to warrant immediate attention.

If you want to change any of the command keys during SAIDINS, just depress the keystroke. SAIDINS will detect it and ask you to double check your entry by repeating it. It is not true that SAIDINS carries through #33 of its 36 commands. SAID has only 33 commands. The list shown in the SAID documentation is 36 command functions. The documentation states that function 34 is not mapped. All functions including 34-36 can be mapped to keys when you request the change of extra characters in SAIDINS. Functions 35 and 36 are normally invoked via a Meta-Block-Swap sequence and are available only under PRO-SAID. "Change extra keys" refers to the ability to assign functions, shown in the SAIDINS documentation, to specific keyboard keys. It is useful to use this for setting frequently used two-key functions to a function key. The TRS-80 Model 4 has three function keys while the MAX-80 has four function keys; you can make use of them. Using the Model III LDOS keyboard driver, <CLEAR><SHIFT><Ø> will

not work; however, <CLEAR><SPACE> is equivalent and takes up one less keystroke. <UP-ARROW> should work after you apply the patch to SAID for the MAX-80 as noted in the SAIDDOC/TXT documentation file (maybe its SAID/DOC).

As far as SAID goes, if you are using SAID under LDOS, then you cannot use the SAID internal keyboard driver. You must specify this in response to the driver query which is the first query in SAIDINS. The documentation states, "The installation program can be used to override this built-in keyboard driver for LDOS users." Perhaps because I didn't say "MUST" instead of "can", it lead a few users astray. The exception is if you don't have the LDOS keyboard driver installed; however, don't expect SAID's driver to deal with the extra keys on the MAX-80!

Here's another SAID quirk when used with the Model III mode LDOS driver. Here is the scenario. The Model I/III version of SAID has a little routine to dynamically change the value returned by the DOS's keyboard driver for the up arrow key. Up-arrow normally generates a 5BH code which is a left bracket "[". Since SAID desires to have up-arrow return 0BH, it changes the driver translation byte for that key. It is done on the fly. Next, recognize that SAID has a patch to be applied for the MAX-80. What it does is change the offset from the start of the keyboard driver to where the byte is that must be changed. Next, note that this little routine is not in SAIDINS; thus, if you are not going to use the SAID keyboard driver but use the LDOS keyboard driver, then that change is not going to be made. If you try to install the up-arrow as a operating function other than a defaulted key command, it will be entered as a 5BH. Then, when you invoke SAID, that key will generate a 0BH. That's the problem. The solution is to add the little routine to SAIDINS which is in SAID. We may eventually get around to that someday. In the interim, the LDOS keyboard driver user can specify the X'0B' keycode via the combination of <CTL-K> during installation and <UP-ARROW> when in SAID.

SEARCH was specifically coded to be case insensitive at your option. This can be changed with SAIDINS. Also, if you enter the search string in a combination of upper case and lower case, the search will be case sensitive. That obviously requires the searched for string to be in upper and lower case. By experiment, you can FIND any character - just enter it in the FIND string. The BLOCK key is a character. The only exception is the TAB function entered via CLEAR+RIGHT but re-interpreted to X'09' - an ASCII TAB. You can FIND it (or change it) via CTL-I (a TAB). Speaking of blocks, yes, you can unmark a single block via DELETEs. We decided against unmarking designated blocks for simplicity.

SAID was designed so that a <BREAK> would not terminate insert mode.

SAID was designed for <ENTER> to always be inserted and never to overstrike the character underneath the cursor. It is nice that way too!

As far as the menu status line goes, Bank represents what banks of RAM are available for use and in use by PRO-SAID. "Dir:For" indicates the direction state of the search command. Remember, you can search forward or reverse-search. "Cnt: 1" indicates that the Macro repeat count is currently set to 1. The "%" display is measuring the position of the cursor through the file as a percentage of the total number of characters. This figure is accurate once the buffer contains 100 characters.

One enters SAID from EDAS via the "Q" command. One can also enter EDAS from SAID via the "DOS COMMAND". These functions are like the "SYSTEM" function in BASIC; except that they permit you to invoke ANY command that is capable of being invoked from DOS Ready with certain restraints. You are inhibited from invoking a command which would attempt to alter HIGH$. So when you invoke SAID from EDAS then exit, of course you get the "Press any key to continue" prompt. That will continue you in EDAS! What a powerful capability to find in a program. Why, you could even FORMAT a new disk and make a BACKUP from inside SAID or EDAS!!!

The reason SAID will prompt you when you exit even if you have saved the changes

and no further changes exist is to give you an opportunity to save it again under either a new name or even on a new disk. Many times when I am working at the computer with a large piece of text or source code, I save the same buffer to two or three different disks. It saves me from having to go and make backups at the end of the day. An ounce of prevention is worth a pound of cure. Only once in my entire experience of programming on Tandy computers have I had to reconstruct a file. That was way back in the early days when I overwrote the source file with the object code during an assembly with a DISK*MODified EDTASM. That may be too early for most folks to remember. Luckily, I was able to reconstruct the source by disassembling the object code file with DSMBLR. To this day, a portion of the DSMBLR source code remains uncommented because of that error. When one makes a mistake such as that, one soon learns that humans as well as computers are fallible. I heartily recommend making at least two copies - each on a different disk. That procedure has saved me probably a dozen times.

SAID and EDAS do not interface with each other. They are both stand alone programs. The "Q" command of PRO-CREATE and the DOS COMMAND of PRO-SAID do not use the @CMNDR SVC because it limits you to DOS commands which execute solely within the library overlay region. The COMMAND functions of PRO-CREATE and PRO-SAID do not limit you. No, we could not have invoked @CMNDR from "protected memory" and avoided altering @EXIT linkage as most programs out commercially do not exit via a RETurn and do NOT maintain the stack properly. These two conditions must be met in order for @CMNDR to function.

Frank A. Yacucci wanted to make some changes to the way our ALTDISK and PRO-SAID programs dealt with command line parameters. With PRO-SAID, he wanted to change the assembler file extension of SAID from its normal "ASM" to "SRC". I advised him that , if he didn't already have LS-FED II from LSI (now a MISOSYS product), that he get a copy. It is invaluable for finding out how to do the things he had requested, and more. I also told him how to discover the patches he needed to effect a resolution of his questions. The techniques are applicable for altering a wide range of poducts (see my response to his questions on PRO-ESP).

In order to use SRC as the assembler extension default in PRO-SAID in lieu of ASM, all you would need to do is to change any character string in the SAID/CMD file from ASM to SRC. From then on, SRC is your default. To change the command line 'ASM' parameter, investigate the parameter table. One of the ASMs found in the file is this table. Note that there is also an 'A' one-character abbreviation. If you want to change the OPTIONS meta command to accept 'S' in lieu of 'A', you will have to dig further; however, with FED, it is easy. You should note that the options display shows "Asm" for the ASM option. Look for that string then find out what references it. You will be amazed at how easy it is to search out the needed patch. Of course, all this precludes knowledge of assembly language; however, since you stated that you are using PRO-SAID with an assembler that uses the SRC extension, I know you are familiar with assembler.

I realize that it is easy to ask someone for a patch; however, it becomes more of a learning exercise for you to be able to find the answer yourself. After he tackled my solutions, he came back for more. After receiving his letter of April 30th concerning DEFAULTING the SAID program to "/SRC", I realized what he was getting at. He didn't just want to use "/SRC" in lieu of "/ASM", he wanted SAID to be invoked automatically assuming the assembler source options.

I took a look at SAID and noted that Karl ignored testing the parameter options if no parameters were entered on the command line. That's why when Frank changed the contents of X'4DC3' from 0000 to FFFF, it didn't force a default to the "A" parameter. You can easily fix this up in PRO-SAID by the following patch:

```
PATCH SAID (D28,31=17:F28,31=6C)
```

This patch forces PRO-SAID to still check the option routines regardless of the entry of parameters on the command line.

In this way, any other default could also be changed by patching the parameter table or the word pointed to by the vector entry in the parameter table.

EDITORS: LED

It has been reported and confirmed that the first TAB stop position in LS-LED is at column 8, not column 7 as should be the case (columns count starting from 0; thus the 8th column is column 7). This will probably be examined in detail sufficient to work up a patch by the next issue of THE MISOSYS QUARTERLY.

FORTH: HartFORTH

We received a query concerning licensing rights for software developed with the HartFORTH compiler. The following should clear up any confusion concerning this matter.

The following points stipulate the rights of a purchasor of HartFORTH to develop application programs with HartFORTH and to distribute said application programs. These points in no way grant any other rights of distribution of the HartFORTH product.

1. No royalties are obligated to MISOSYS for a purchasor's distribution of his or her applications developed with HartFORTH, providing the purchasor adheres to items 2, 3, 4, and 5.

2. The HartFORTH copyright appearing in sector 0 of the VM file must be left intact in the purchasor's application distribution disk.

3. Any facility of HartFORTH's used in the purchasor's application must be distributed in compiled form; no SOURCE screens of HartFORTH shall be distributed.

4. Documentation accompanying the purchasor's application shall note the statement,

> This application includes portions of HartFORTH Copyright 1982 by A. M. Graham, All rights reserved.

5. No portion of the HartFORTH user manual may be distributed with the documentation accompanying a purchasor's application program.

Michael Houston had a problem with the Model I/III version of HartFORTH. It was our blunder stemming from the release of a User Manual which was printed from the PRO-HartFORTH version documentation files. Apparently, there were a few minor differences between the HartFORTH editor and the PRO-HartFORTH editor.

The problem was with the editor's CONTROL functions in the Model I/III version of HartFORTH. The author of HartFORTH uses CONTROL-I for "put line" and CONTROL-K for "empty the screen" in the Model I/III version. The Model IV version is as is documented in the manual. We regret the inconvience.

Early last year, I received a letter from Matthew A. Sohlstrom concerning his recent acquisition of PRO-HartFORTH. Now we don't get much correspondance about FORTH [nor do we sell a lot of FORTH compilers]; however, the questions raised in Matthew's letter and the depth of the response lead me to believe that the dialog would be useful for this FORTH column. Here it is.

Matthew writes, "I recently received your PRO-HartFORTH. Let me say, first of all, how pleased I am with your product. Good value at a reasonable price, prompt service and no (or in this case FEW) bugs.

As a BRAND NEW forth user, I am delighted with HartFORTH in general and in specific. I am running a 200 page HartFORTH under TRSDOS 6.2 on the first TRS-80 Model 4 + 128K + graphics that Radio Shack has sold here in Evansville. A ten meg hard disk and two external DSDD 80 track floppy disks, all from Software Support Inc., help store and backup the beast. HartFORTH's speed on the hard disk is astounding. A friend of mine (also a Digital Dog Soldier, currently working his giblets into a state of spontaneous collapse at Central Methodist University) commented, upon seeing HartFORTH load and

page up and down, "What took it so long," as he wiped the carpet lint off his eyeballs. Finally, the idea of a 2.5 Meg HartFORTH system (coming soon to a computer system near ME) leaves me a little breathless.

There are, however, a few points I would like to address.

1) First, the doublequote operator (") does not seem to generate nill-strings correctly. " " produces a one character length string with whatever junk was left in the PAD. This is a bit difficult to deal with intuitively. I field-fixed it with the function (possibly incorrect):

```
: "NILL PAD DUP 0 C! ;
```

which seems to do the job, but " " would be much cleaner and MUCH more easily used in the formulation of new functions and constants.

2) Second, the SVC function is in need of upgrading. Some SVC calls (most notably the @FLAGS call) use IX and/or IY registers for values. Moreover, some users, while defining SVCs, might find using those two more than slightly useful resisters desireable. The cost would be more numbers on the stack and a bit more machine code, but nothing FORTH programmers are not already dealing with (and successfully, we hope).

3) Third, a bit more documentation on the ASSEMBLER vocabulary would be nice. I am a full time programmer/laboratory technician here at the University of Evansville, where I write all sorts of neat systems and support routines for a PDP 11/34A and several semi-smart peripherals in BASIC, C, FORTRAN, PASCAL, MACRO-11, and 8085 assemblers, so I am reasonably well grounded in language use and implementation but I haven't yet been able to figure out how to do the machine language routines I need to service my Radio Shack graphics board. It is possible that I am being too third generation languagesque, but I can't even figure out if I'm on the right track! Which of the mentioned books might tell me more about machine dependent code and generation thereof?

4) FORTH (and almost apocryphally, or at least wildly inaccurately last), it would be nice to be able to change or eliminate the "ENTER FILESPEC OF FORTH VIRTUAL MEMORY" prompt so that a FORTH application could be generated that needs less nursing through by the end user.

HartFORTH is well suited to the development and generation of the sub systems that are needed to form a given application (like a word processor) or a given system utility (like a multi-volume backup for a hard disk) by assigning each individual sub system to its own vocabulary or, as in "Starting Forth", by leaving the code in overlays. This is all well and fine, but in any given application, APPLICATION/CMD should wind up being the virtual memory file, and a filespec or filespecs specified in the routine invocation, along with the appropriate arguments (all of which should be fetched and parsed with SVC calls on the Model 4!) should directly control the application/utility, NOT (I feel) THE END USER!

If you intend that HartFORTH, while running under the auspices of TRSDOS 6.x, remain a separate entity and never be allowed to generate a utility useful to TRSDOS 6.x users as a whole, then HartFORTH is already an astounding and delightful success. So long, and thanks for all the fish."

Naturally, when one is thanked for the fish, one has to also answer the questions. After all, I brought my towel that day. I had tinkered a little with the HartFORTH CMD file in the Model I/III environment to be able to make it transparently portable across those two machines. So I felt confident about doing a little digging for Matt. Unfortunately, since I am not a FORTH programmer, I decided to pass a few of his queries across the pond to HartFORTH's author.

To wit my response: "This is in response to your letter of February 26th. I cannot address all of your queries at this time. I want to let you know that I have forwarded a copy of your letter to Molimerx (and on to the author) so that

the "heavier" aspects of HartFORTH may be addressed.

I believe that I may shed some light on a couple points. Taking the last one first, you can accomplish part of your need to have a FORTH application automatically access the VM file without having to be prompted. I don't believe there is a procedure to accomplish this without patching the application/cmd file, unless someone has already written an application to do this. Actually, it is easy to accomplish. On page 14 of of the manual, the parameters associated with the SYS word are shown. Note that SYS+32 is the file specification used for the VM file if SYS+31 is non-zero. Thus, if you change the contents of these two fields in the application file's disk (this occurs in sector 0 of the file), the VM file will be accessed without a prompt for it.

You could probably also have a facility for scanning the command line for user-entered parameters. This would be coded as a FORTH word. Note that SYS+29 is the Code Field Address of the FORTH word executed after the VM file is initialized. If you change this to be the CFA of your parameter scanning word, it can then address parameters. Of course, since this procedure seems so straight forward, perhaps it has already been done? You see, the Model I/III and Model 4 versions of HartFORTH are just two of many other versions that have been implemented. Since the model 4 version has been published for a few years (in Europe), the items you are looking for have had time to be implemented by someone else. I'll pass this question along to Molimerx.

The DO.SVC function is in screen 36. I looked at this function. True, it is written in machine code (actually, its in the primitive assembler which permits you to specify 16-bit hexadecimal values). If you examine the code, it does the following OPs: EXX; POP HL; LD A,L; POP HL; POP DE; POP BC; RST 28H; PUSH BC; PUSH DE; PUSH HL; EXX. This is of course prefixed by ;CODE and suffixed with END-CODE. Now Andrew Graham may prove me wrong; but I think you should be able to modify the routine so that IX and IY can be loaded prior to the RST 28H and restored afterwards. One proviso would be that since HartFORTH uses the IY register as a subroutine linkage instead of CALL and RET, you may not be able to utilize the value returned in IY. In fact, I would suspect that you would have to restore IY prior to END-CODE to its value that existed immediately following ;CODE since END-CODE is coded as JP (IY). I would suggest the following: : EXX; POP HL; LD A,L; POP HL; POP DE; POP BC; POP IX; EX (SP),IY; RST 28H; EX (SP),IY; PUSH IX; PUSH BC; PUSH DE; PUSH HL; PUSH AF; EXX. That means two additional items on the stack (note where IX and IY would be). You could name this one DOXY.SVC to keep it separate when you needed to address IX and/or IY. Since the assembler would require the additional instructions in hex, these would, of course, be: E1D9 E17D C1D1 E1DD E3FD FDEF DDE3 C5E5 E5D5 D9F5.

I will defer to Mr. Graham on points 1 and 3; however, Mountain View Press, PO Box 4656, Mountain View, CA 94040 [415-961-4103] has an ad in March 85 BYTE which lists over 25 books and papers concerning FORTH. Although I have never dealt with MVP, they do advertise a lot of FORTH specific material."

The response from Andrew is rather lengthy. But then, we rarely get exposed to FORTH dialog so let's get on with it. Andrew Graham writes, "I am very pleased that Matthew seems to be satisfied with his purchase of HartFORTH. I am impressed with the speed with which he seems to have come to grips with the language. To deal with his queries in order.

1) Yes, I regret that the doublequote operator cannot generate a null string. This is due to its use of the word WORD to accept the string from the input stream. As WORD will ignore leading delimiters (this is necessary to allow multiple space characters between words as the compiler uses 32 WORD to parse the input stream) then trying to invoke a null string will have the following effect.

If " " (note the single space) is used then the first quote and its following space are used to identify the doublequote command leaving the input pointer pointing to the second double quote. As the first "

uses 44 WORD to get the next string of characters terminated with a ", and because WORD will ignore leading delimiters then the second " will be ignored as a leading delimiter and the following characters will be read in the search for the final ".

This may be demonstrated by entering : "NILL " " ; from the keyboard which will cause a compile error as the ; is ignored for the reasons explained above.

The way round this is as Matthew states, but his coding should be

```
: "NILL PAD Ø OVER C! ;
```

I usually generate a null string as follows: CREATE "Z Ø, or equivalently: Ø CONSTANT "Z which leaves the address of a zero when "Z is invoked. If " " (two spaces) is used then a string comprising a single space results.

2) It is important to note that both IX and IY need to be preserved as well as BC. IX holds the return stack pointer, (the SP is the data stack pointer) and IY holds the address of the inner interpreter while BC is FORTH's program counter used to point to the CFA of the next word of a definition to be executed.

Roy's suggested coding saves IY but not IX and could cause the system to crash if IX is altered [sorry, I didn't know that IX was used -Roy]. I suggest that the easiest route might be to save IX and IY to explicit variables and restore them after the SVC call but before the END-CODE thus avoiding some messy stack manipulations.

Incidentally, I have assumed, and appear to be correct (at least so far) that DOS does not use the alternate register set and normally preserves IX and IY except where documented in an SVC [that's true for LDOS and TRSDOS 6 -ed]. The EXX therefore serves to save BC, and should occur in any CODE word that wishes to use BC, DOS appearing to take care of IX and IY except where documented.

The following FORTH code word should achieve the desired effect.

```
VARIABLE OLD.IX       VARIABLE OLD.IY

CODE DOXY.SVC
 ( exx   pop hl   ld a,1   pop hl   pop de   pop bc )
   D9 C, E1 C,    7D C,    E1 C,    D1 C,    C1 C,
 ( ld (old.ix),ix   ld (old.iy),iy   pop ix   pop iy   rst28h )
   22DD , OLD.IX ,  22FD , OLD.IY ,  E1DD ,   E1FD ,   EF C,
 ( push iy   push ix   ld ix,<old.ix>   ld iy,<old.iy> )
   E5FD ,    E5DD ,    2ADD , OLD.IX ,  2AFD , OLD.IY ,
 ( push bc   push de   push hl   push af   exx )
   C5 C,     D5 C,     E5 C,     F5 C,     D9 C,  END-CODE

This has the stack picture
DOXY.SVC ( iy ix bc de hl a -> iy' ix' bc' de' hl' af' )
```

The register sequence on the stack could of course be altered as required.

3) As far as more documentation on the ASSEMBLER vocabulary is concerned, I am not sure that I can help. You can see above how I code Assembler words when I am more worried about them working than I am about saving space on a screen. I merely write them in Z80 assembly language and then hand code them. Jumps are no problem with the relative jump instructions, and generally the routines should be short enough for this to be no great pain. I chose merely to provide a LABEL command for subroutines called for CODE words and a CODE command to specify low-level words invokable as FORTH words as I believe as little as necessary should be written at low level. Further than that if it is the Z80 architecture that you need information on I would have thought that all you need is a book on Z80 assembler language such as Bill Barden's, and see if you can persuade Roy to sell you one of the Micro Logic Z80 reference cards [we actually

still have a few of those plastic cards left -ed]. There is a design for a more complete Z80 assembler in Loeliger's book Threaded Interpretive Languages together with a description of how it works.

4) HartFORTH was intended from the outset to generate end user stand alone applications, in fact that was the reason for its being written in the first place.

I will try to explain the facilities that are available by describing how HartFORTH initializes itself.

Firstly it executes a very small section of machine code to initialize its stacks and registers and then it takes the word from SYS+19 and EXECUTEs it assuming that it is the Code Field Address of a FORTH word. Normally this would be the Virtual Memory Initialization word. However, if the final application does not need Virtual Memory then the CFA of the final application may be stored here. In this case do not leave the application by the word DOS which tries to close the VM file but use the @EXIT SVC instead to re-enter DOS directly.

If the VM Initialization word is executed then it examines the byte count at SYS+31. If that byte is zero it asks the user for the filename, otherwise it uses the filename stored at SYS+32. Then, as long as the file is opened without error, it takes the word at SYS+29 and executes it, assuming again that it is the CFA of a FORTH word. Normally this word is the CFA of the word QUIT, but you may put the CFA of an application here to make a stand-alone system.

As far as parsing the DOS command string is concerned I must admit that I have not catered for this possibility. However, it can probably be added as follows.

I believe that the problem is that of retaining BC or HL on entry to HartFORTH as these point to DOS's command line and are the only source of that address, at least to my knowledge. Here is a short patch to save one of these registers. Fortunately the first few SYS bytes are not used in the Model 4 version (or indeed the Model I/III version) so the actual code will go at the beginning between SYS and SYS+6, and this will store the command line address at SYS+27 which is also unused in these versions. You could then pick it out and use it as you wished.

```
HEX                                  (all this is easier in hex)
SYS SYS 4 ; 3 CMOVE    ( copy initial jump to unused bytes )
Ø SYS C! 22 SYS 1+ C!      ( this is opcode for ld <nn>,hl )
   ( the zero is nop in case you wish to store bc instead )
                ( in which case use ED SYS C! 43 SYS 1+ C! )
SYS 1B + SYS 2 + !      ( in proper numbers this is SYS+27 )
DECIMAL
SAVE-SYSTEM
```

The address in HL points to the first non-blank character after the command name, BC points to the start of the command line. Whichever you want you have in SYS+27 by putting the appropriate op-codes at SYS and SYS+1.

You could of course get more complicated by writing a LABEL word to parse the command line and patching a jump to it at SYS, jumping on termination of this LABEL word to the original jump address at SYS+1. This LABEL word could parse a filename and write it as a string (upper-case please!) to SYS+31 for use as a VM file. Note that if the VM file is specified at SYS+31 and cannot be opened then HartFORTH will immediately retry. This causes a loop that needs reboot if the error is persistant.

I hope this is useful. Please check carefully the above, the flavour is right but the machine coding could do with an indepenmdent check as I haven't had time to try it all."

Macros by Timothy Adye

For those of you who have been with us for awhile and recollect the sprinklings of quotations from THE HITCHHIKER'S GUIDE TO THE GALAXY, you will appreciate the following letter from Timothy Adye. Tim's home is in England - also the origins of one Arthur Dent.

"Let me first express my support for the Model I! I do hope that you continue to support this machine. I have been using it now since 1980 [Tim's letter was dated 3rd February 1985 -ed] with only one fault (the expansion interface cable) despite extremely heavy use of maybe 14 hours a day. Since I got LDOS and EDAS, it has compared favorably in all but graphics with many later machines. Having written an LDOS utility for sale for use with both the Model I and III, I found that it was not that difficult to support both machines. I hope that you find the same is true.

As a digression from matters computational ("Yes - I know Marvin - but shut up, this is organism talk"), have you heard of ZZ9-Plural-Z-Alpha? This is an international Hitch-Hiker's Guide fan club based in the UK. You're probably already a member, [not really -ed] but if not, you might like to know that they produce a quarterly magazine/newsletter, "Mostly Harmless", a fanzine, "Gargleblaster" (though you should read the advice on the front: "Warning: This zine has much the same effect"), and are running a Con this summer ("Lazlar Lyricon" - it took me a while to work out that reference: the maker of custom-built "bourge-mobiles", to be identified by the infra-pink lizard emblem), though I suppose that this would be rather out of your way [also too late for our readers considering when I got this into print -ed]! For more information write to:

Hohn 'Grandad' Philpott
78 Watling St.
Bexleyheath, Kent
DA6 7QQ, ENGLAND

Enough of this frivolity, onto more important questions (no - not "Why are men born, why do they dye, (why can't they spell die), why do they spend most of the intervening time wearing digital watches").

I am a proud owner of EDAS (sorry - it does sound rather like a shoe advertisement), and will certainly be ordering the new version from Molimerx as soon as they tell me they have it. At the moment I use it to write and maintain a book cataloging program for an antiquarian book business, which consists of more than 100K of source code (you couldn't do that using EDTASM). One interesting feature is that it has its own disk I/O routines. This is to speed up disk I/O since the sectors do not have to be interleaved. It can search an entire 3/4M disk in just two minutes for any piece of information. It is self booting, and requires no DOS or any ROM routines, so it is quite easily transportable to other Z80 machines (I realise that this is a debatable point, but it does also mean that all of the routines work the way I want them to). I have also written several LDOS utilities including ACCESS, which Molimerx is selling.

Concerning the bug in macro handling with *GET, I had the same problem, and put it down to not understanding how to use them (they worked for test files but not in my large program with lots of *GETs). I shall refrain at this point from making comments about the Sirius Cybernetics Corporation, since you did supply a patch as soon as you found out.

Here are a few tid-bits for EDAS programmers. The following macro and subroutines are very nice in assembler programs that display a lot of messages (for example interactive programs). It need only be typed in once and then put into a module or PDS.

```
MSG     MACRO   #M1,#M2,#M3
        IFDEF   ENDMSG
        IFNE    ENDMSG,$
        CALL    DSPMSG
        ENDIF
        ELSE
        CALL    DSPMSG
        ENDIF
        IFEQ    %%,1
        DB      #M1
```

```
        ENDIF
        IFEQ    %%,2
        DB      #M1,#M2
        ENDIF
        IFEQ    %%,3
        DB      #M1,#M2,#M3
        ENDIF
ENDMSG  DEFL    $
        ENDM

DSPMSG  EX      (SP),HL
        PUSH    AF
DSPM1   LD      A,(HL)
        INC     HL
        OR      A
        JR      Z,DSPM2
        CALL    @DSP
        JR      DSPM1
DSPM2   POP     AF
        EX      (SP),HL
        RET
```

To use this, simply put in your program:

```
MSG     'This is a message',13
MSG     'With two lines',13,0
LD      A,0
MSG     'New string here',13,0
```

The MSG macro will put a CALL DSPMSG before each string, unless the previous byte was the end of another message, in which case it can all be done at once. Put a null (zero byte) to terminate the message after the last string.

To further demonstrate the beauty of macros, you can add what looks like more op-codes to the Z80. For example, the three byte "opcode", EXCH (EB,E3,EB) will exchange DE and (SP),

```
EXCH    MACRO
        EX      DE,HL
        EX      (SP),HL
        EX      DE,HL
        ENDM
```

or the two byte opcodes MOV dd,ss (e.g. MOV BC,DE is 42,4B) will do 16-bit register to register LDs. My favourites are SHOVE aa,bb,... and TUG aa,bb,... (which require slightly more complex macro definitions), which are similar to PUSH and POP, except they do it for more than one register.

Well, I'm sure you've had quite enough of that, so I'll sign off there. May all your Gargleblasters be pure gold."

POKEPR by Roy Soltoff

One of my customers wanted a little program to be able to poke values to the printer from the DOS Ready command line. That was an easy one to whip up. The following POKEPR program is for use with TRSDOS 6. It will accept a series of decimal values separated by a <SPACE> and terminated with an <ENTER>. For example, if you wanted to send the string "27 18" which sets a DMP-500 to correspondance quality characters, type the command

```
POKEPR 27 18<ENTER>
```

Here's the program:

```
        ORG     2600H
POKEPR  LD      A,96    ;@DECHEX
        RST     40
        LD      A,6     ;@PRT
        RST     40
        LD      A,(HL)
        INC     HL
        CP      13
        JR      NZ,POKEPR
        SBC     HL,HL   ;For exit
        RET
        END     POKEPR
```

This program is so short, it can be entered using the DOS BUILD command with the HEX parameter:

```
BUILD POKEPR/CMD (HEX)
3E 60 EF 3E 06 EF 7E 23
FE 0D 20 F4 ED 62 C9
<BREAK>
```

UNLOCK by Peter Lengsfeld

Back in July of last year, Peter Lengsfeld wrote me a letter in which he said, "I would like to start by letting you know I very much enjoy using your utilities. Together with my Model 4P, I feel I have an unbeatable combination. I certainly hope market conditions will enable you to continue providing top quality software for some time to come. Thanks to PRO-NTO,

I have been able to make room on my desk for a full time computer by eliminating such items as a calendar, Rolodex file, note pads, etc..."

Peter went on to make some suggestions for changes and additions to certain pieces of our software. He then closed with, "The disk enclosed with this letter contains a program that I would like to submit to you for possible inclusion on your next issue of DISK-NOTES. In one way it is a thank you to all of the other individuals who have previously submitted many fine and useful programs. The program is entitled UNLOCK/CMD and is a machine language program to decode BASIC programs saved with the ",P" option available with BASIC 01.01.00. The program was written to run on a Model 4/4P under 6.x." Well here is the program which is also included on DISK-NOTES 5.

```
;*=*=*
;       UNLOCK PROGRAM
;       By Peter Lengsfeld
;       Mar. 20th, 1985
;*=*=*
@HIGH$  EQU     100
@OPEN   EQU     59
@CLOSE  EQU     60
@FSPEC  EQU     78
@FLAGS$ EQU     101
@READ   EQU     67
@WRITE  EQU     75
@KEYIN  EQU     9
@DSPLY  EQU     10
@INIT   EQU     58
BUFPTR  EQU     3
EOF     EQU     8
NRN     EQU     10
ERN     EQU     12
@ERROR  EQU     26
        ORG     3000H
;*=*=*
;       INITIALIZE
;*=*=*
START   LD      (STACK),SP      ;save return point
BGNMEM  EQU     $-4
ENDMEM  EQU     $-2
        LD      HL,MSG1         ;logon msg
        CALL    DISPLY
GETINP  CALL    NZ,INVAL
        LD      HL,MSG2         ;prompt for input file
        CALL    CKSPEC          ;get keyboard input line
        JR      NZ,GETINP
        LD      A,@FLAGS$       ;point to base of flags
        RST     40
        PUSH    IY              ;put in DE
        POP     DE
        LD      HL,'S'-'A'      ;offset for SFLAG$
        ADD     HL,DE           ;add offset to base
        SET     0,(HL)          ;dont check LRL, force READ
;*=*=*
;       OPEN FILE
;*=*=*
        CALL    POINT
        LD      A,@OPEN
```

```
        CALL    DODOS
;*=*=*
;       LOAD BUFFER WITH FILE USING
;       FASTFILE TECHNIQUE
;*=*=*
FASTIO  LD      HL,0            ;lets get HIGH
        LD      B,L             ;SVC requires B=0
        LD      A,@HIGH$
        RST     40
        CALL    SETSUB
        LD      HL,BUFFER       ;have room for
AGAIN   LD      (FCB+BUFPTR),HL ;put buf addr in
        LD      A,@READ         ;FCB
        RST     40              ;READ sector
        JP      NZ,CKERR        ;is err EOF or DOS?
        INC     H               ;inc buf addr
        DJNZ    AGAIN           ;keep reading sectors
        PUSH    HL              ;save
        LD      HL,(FCB+NRN)    ;check if ERN was last
        LD      DE,(FCB+ERN)    ;to fit in end of mem
        OR      A
        SBC     HL,DE
        POP     HL
        JP      NZ,NOMEM
ISEOF   LD      A,(FCB+EOF)     ;get EOF offset byte
        DEC     A
        LD      E,A
        LD      D,0             ;DE = offset of EOF
        DEC     H               ;backup 1 full sector
        ADD     HL,DE           ;add offset
        INC     HL
        LD      (ENDMEM),HL     ;store end of buffer
;*=*=*
        CALL    CLOFIL          ;close file
        LD      HL,BUFFER       ;check to see if file
        LD      A,(HL)          ;is a protected BASIC
        CP      0FEH            ;program
        JP      NZ,NOLOCK
        INC     (HL)            ;set to FFH
        INC     HL
        LD      (BGNMEM),HL
        CALL    UNLOCK
GETOUT  CALL    NZ,INVAL
        LD      HL,MSG3
        CALL    CKSPEC
        JR      NZ,GETOUT
        CALL    POINT
        LD      A,@INIT
        CALL    DODOS
        LD      HL,(ENDMEM)
        CALL    SETSUB
        LD      C,L
        INC     B
        LD      HL,BUFFER
WRMORE  LD      (FCB+BUFPTR),HL
        LD      A,@WRITE
        CALL    DODOS
```

```
            INC     H
            DJNZ    WRMORE
            LD      A,C
            LD      (FCB+EOF),A
            LD      A,@CLOSE
            RST     40
            JP      NZ,ABORT
            LD      HL,0
            JP      QUIT
;*=*=*
*GET        LOCKCODA
;           CLOSE FILE
;*=*=*
CLOFIL      LD      DE,FCB
            LD      A,@CLOSE
            RST     40
            RET
;*=*=*
;           GET KEYBOARD INPUT AND CHECK FOR VALID FSPEC
;*=*=*
CKSPEC      CALL    DISPLY
            LD      HL,TEMP         ;keyin buffer pntr
            LD      BC,1F00H        ;1FH = max line input
            LD      A,@KEYIN
            RST     40
            JR      C,ABORT         ;<BREAK> was pressed
            JR      NZ,DOSERR
            LD      DE,FCB
            LD      A,@FSPEC
            RST     40
            RET
;*=*=*
POINT       LD      HL,BUFFER       ;open file
            LD      DE,FCB
            LD      B,0             ;set LRL=256
            RET
;*=*=*
SETSUB      LD      BC,BUFFER       ;get start of BUFFER
            OR      A               ;clear CARRY FLAG
            SBC     HL,BC           ;subtract (HIGH-BUF)
            LD      B,H             ;num of sectors we
            RET
;*=*=*
INVAL       LD      HL,MSG4
DISPLY      LD      A,@DSPLY
DODOS       RST     40
            JR      NZ,DOSERR
            RET
;*=*=*
;           ERROR ROUTINES
;*=*=*
CKERR       CP      1CH             ;EOF?
            JP      Z,ISEOF
            CP      1DH             ;past EOF?
            JP      Z,ISEOF
DOSERR      OR      0C0H            ;set short msg
            LD      C,A
```

```
        LD      A,@ERROR
        RST     40
ABORT   LD      HL,-1
QUIT    LD      SP,$-$
STACK   EQU     $-2
        RET
;*=*=*
;       ERROR MESSAGES
;*=*=*
NOLOCK$ DB      'NOT a PROTECTED BASIC PROGRAM',0DH
NOMEM$  DB      'File to large',0DH
;*=*=*
NOLOCK  LD      HL,NOLOCK$
        DB      0DDH
NOMEM   LD      HL,NOMEM$
        LD      A,@DSPLY
        RST     40
        JR      ABORT
;*=*=*
;       MESSAGES
;*=*=*
MSG1    DB      'UNLOCK By P. Lengsfeld',0DH
MSG2    DB      0AH,'Enter LOAD filespec ..> ',03H
MSG3    DB      'Enter SAVE filespec ..> ',03H
MSG4    DB      0AH,'Invalid filespec',0DH
;*=*=*
;       END
;*=*=*
TEMP    EQU     $
FCB     EQU     TEMP+32
BUFFER  EQU     FCB+32<-8+1<8
        END     START


;*=*=*
;       UNLOCK ROUTINE
;       THIS ROUTINE IS USED TO UNLOCK BASIC
;       PROGRAMS SAVED WITH "P" OPTION
;
;       NOTES:  ROUTINE SHOULD BE CALLED FROM MAIN PROGRAM
;               BGNMEM SHOULD POINT TO FIRST CHAR AFTER 0FEH
;               ENDMEM SHOULD POINT TO LAST CHAR +1
;*=*=*
UNLOCK  LD      BC,0D0BH
        LD      DE,(BGNMEM)
UNLOCK1 LD      HL,(ENDMEM)
        OR      A
        SBC     HL,DE
        RET     Z
        LD      HL,TABLE1
        LD      A,L
        ADD     A,B
        LD      L,A
        LD      A,H
        LD      H,A
        LD      A,(DE)
        SUB     C
```

```
            XOR     (HL)
            PUSH    AF
            LD      HL,TABLE2
            LD      A,L
            ADD     A,C
            LD      L,A
            LD      A,H
            LD      H,A
            POP     AF
            XOR     (HL)
            ADD     A,B
            LD      (DE),A
            INC     DE
            DEC     C
            JR      NZ,UNLOCK2
            LD      C,0BH
UNLOCK2     DEC     B
            JR      NZ,UNLOCK1
            LD      B,0DH
            JR      UNLOCK1
TABLE1      EQU     $-1
            DB      0FBH,0D7H,1EH,86H,65H,26H,99H,87H,58H,34H,23H,87H,0E1H
TABLE2      EQU     $-1
            DB      4AH,0D7H,3BH,78H,02H,6EH,84H,7BH,0FEH,0C1H,2FH
;*=*=*
;           END OF UNLOCK CODE
;*=*=*
```

BASICSVC by Robert M. Connors

Speaking of BASIC, Robert M. Connors also wrote me last year with a couple of things for "NOTES". Now Robert has been a customer of mine for about as long as I can remember. He has kept me on my toes on more than one occassion - primarily over bugs in earlier versions of EDAS. In any event, Bob writes, "I have enclosed a few things with this letter. First is a procedure I use to create /LIB files for use by EDAS. The second thing concerns the CALLing of any DOS 6.x SVC by a BASIC program. I know that you usually do not deal with BASIC issues, but this is really a machine language routine for use by BASIC [actually, now that MISOSYS has released a BASIC compiler, we will be dealing more and more with "BASIC" issues -ed]. You may want to include these two items in your next issue of NOTES."

[For those readers interested in the library creation procedure, it is included in this issue over in the PaDS section. The BASIC SVC interface follows here. It may be interesting to point out at this time that the BASIC in the TRSDOS 6.3 version to be released by Logical Systems, Inc., sometime after the first of the year will support a means of directly invoking DOS service calls via USR10. -ed]

If you look in your Model 4/4P BASIC manual under 'CALL', you will note that upon entry to a machine language routine, the HL register pair points to the first parameter, DE the second, and BC the third unless there are more than 3, in which case BC will point to the LSB of the remaining parameters. The problem here is that the machine language subroutine must know how many parameters there are while SVC routines use any number of parameters. The solution, which this routine provides, is to put all the SVC parameters into one CALL parameter and let HL point to it on entry. That way, we need not concern ourselves with the number of parameters nor where they are located. On entry to this machine language routine, HL will be pointing to the contents of PARAM# and we are not concerned about where BC and DE point.

This routine, CALLed from BASIC 01.01.00 (under DOS 6.2.x), will perform any SVC that is available, except the @BANK SVC. The @BANK SVC cannot be used because BASIC itself overlaps address 8000H which is the start of the next bank, not to mention the fact that the BASIC program which calls the SVC is also in the bank of memory that will be swapped out. Therefore, the program would crash if the @BANK SVC is invoked.

To use this routine, include the following lines in your BASIC program:

```
DIM ROUTINE$,A%,BC%,DE%,HL%,I%,
    PARAM#,SVC%,V%
DATA 126,35,78,35,70,35,94,35
DATA 86,35,229,221,225,221,43,221
DATA 110,1,221,102,2,239,201,-1
'
WHILE I%=>0
  READ I%
  IF I%>0 THEN ROUTINE$=ROUTINE$+
     CHR$(I%)
WEND
```

The DIM statement sets aside memory and predefines the variables we are using. The DATA statements are the decimal equivalents of the machine language routine shown below. After reading, this entire machine language subroutine will be contained in ROUTINE$.

Somewhere in your program, include the following subroutine:

```
A%=0:BC%=0:DE%=0:HL%=0:RETURN
```

As can be seen, this sets the variables used to zero so that any previously assigned values are erased. This subroutine must be called prior to setting the values of these variables.

As you must have figured out by now, A%, BC%, DE%, and HL% are the variables which will hold the values for register A, and register pairs BC, DE, and HL respectively. To pass a value to any of the registers, use statements similar to these,

```
LET A% = a
LET BC% = CVI(CHR$(c)+CHR$(b))
```

where 'a', 'c', and 'b' are the values to pass to registers A, C, and B respectively. Note that A% is assigned the actual number, but BC% is assigned the values as if we were converting it from a value just read in from a random disk file. Also note the order in which the values for BC% were specified (LSB/MSB). If no value is to be assigned to register C, then 'c' should be 0. This also applies to all other registers when the value should be zero. Register A must always have a value since it will contain the number of the SVC you want to perform.

Once all the values are assigned to A%, BC%, DE%, and HL%, GOSUB to the following routine:

```
I%=VARPTR(PARAM#):POKE I%,
    PEEK(VARPTR(A%))
I%=I%+1:POKE I%,PEEK(VARPTR(BC%))
I%=I%+1:POKE I%,PEEK(VARPTR(BC%)+1)
I%=I%+1:POKE I%,PEEK(VARPTR(DE%))
I%=I%+1:POKE I%,PEEK(VARPTR(DE%)+1)
I%=I%+1:POKE I%,PEEK(VARPTR(HL%))
I%=I%+1:POKE I%,PEEK(VARPTR(HL%)+1)
V%=VARPTR(ROUTINE$)
SVC%=CVI(CHR$(PEEK(V%+1))+
    CHR$(PEEK(V%+2)))
CALL SVC% (PARAM#)
RETURN
```

The first thing that this subroutine does is find where in memory the double precision variable PARAM# resides. Once it is found, it pokes in the values for A%, BC%, DE%, and HL% in that sequence. A double precision variable was used because it takes up eight bytes in memory, and we need seven bytes to hold our register values. A string could be used, but that would require more code.

Next, the subroutine finds ROUTINE$ in memory, and assigns the memory address of the string to SVC%. SVC% is called using PARAM# as the only parameter. The machine language routine in ROUTINE$ then picks up each of the register values and assigns them to the appropriate register and then executes the desired SVC.

With some thought, both the BASIC program and machine language routine could be modified to perform functions other than a

SVC. Also, you could conceivably pass values back to BASIC through PARAM#.

The following assembly listing is the source code for the routine contained in the BASIC DATA statements above. On entry to this routine, HL points to the data contained in PARAM#. The data, of course, must be transfered to A, BC, DE, and HL before it can be used. This is the purpose of the routine, and the comments pretty much document what is going on. I point out, however, why it is necessary to decrement IX before transferring data to register pair HL. BASIC uses a 00H byte to indicate an end of a line and the end of a string. Since the command, "LD HL,(IX)" is actually a "LD HL,(IX+00H)" which will generate a 00H byte, we cannot include it as part of ROUTINE$ or else BASIC will interpret it as the string end. So we must decrement IX in order to prevent premature termination of the machine language routine in ROUTINE$.

```
USRSVC  LD      A,(HL)          ;get service number
        INC     HL              ;bump pointer
        LD      C,(HL)          ;get value
        INC     HL              ;  for
        LD      B,(HL)          ;     BC
        INC     HL              ;bump pointer
        LD      E,(HL)          ;DE value
        INC     HL              ;  is
        LD      D,(HL)          ;     next
        INC     HL              ;bump pointer
        PUSH    HL              ;transfer pointer
        POP     IX              ;  to IX and backup one byte to avoid
        DEC     IX              ;     a 00H byte in BASIC's ROUTINE$
        LD      L,(IX+01H)      ;get HL
        LD      H,(IX+02H)      ;  value
        RST     28H             ;perform requested SVC
        RET                     ;back to BASIC
        END
```

RELOCATABLE ASSEMBLERS by Roy Soltoff

It's time now for me to add my two cents concerning programming. With the release of my MRAS relocatable macro assembler last Fall, I thought it appropriate to speak a little about these types of assemblers. First of all, let me clear up an understanding of the term "relocatable". A lot of folks biting their assembler teeth on the TRS-80 have been taught that a relocatable program was one that was able to be run at any memory location in the TRS-80 usable address space. Such an understanding is not hard to grasp. In fact, I recollect many potential DSMBLR customers asking me about that very point - is the disassembler relocatable? That question was asked because they wanted to disassemble something in memory and did not want the disassembler itself to load over the program that was the intended target of the disassembly. Well, DSMBLR was not relocatable - but it allowed for the targeted program to be moved to some other non-interfering address. Neither is MRAS relocatable; it is designed to load and execute at a particular address. Nor are ALDS, M80, and MZAL relocatable. They all load and execute at specific addresses.

The term "relocatable" as applied to these types of assemblers is that they usually do not directly generate an executable program file [I say "usually" because MRAS has a switch to do just that]. What they do is generate an intermediate file which is not address specific as far as its execution address. This file can be processed by another utility called a linker with the actual process being called "linking". During the linking process, the intermediate file can be linked with other already assembled modules and can be specified to originate at a defined address. It is because these intermediate files can be relocated by the

linker that we coin the term, relocatable. In fact, the intermediate files usually have a file extension of "REL".

So what's the big deal? Why go through another step before you can get to a runnable program? The answer has to do with program development. Some companies use the term "program development system". Anyone who has written a program of substantial size regardless of the language used has realized that managing the task of developing the program is as important as the programming itself. For large scale programming projects, it is absolutely essential that program development be properly managed. Many times, the task of programming is divided up amongst many programmers. Even if one programmer is tackling the entire job, he or she will find it infinitely better to break up the programming into small, easily managed modules. Modules are designed to be standalone, usable as functions or subroutines for other modules. The relocatable assembler has the advantage of being able to link many modules together. Since they are already assembled, time is saved. If the programmer was using an assembler which required all modules to be assembled during the assembly process, it could take a lot longer to test one little change. Besides, programmers have found over the years that they could reuse many of the functions developed for one program in other programs. These wise programmers have built up libraries of functions and routines to aid them in their program development.

Now the EDAS/PRO-CREATE assembler certainly provides the ability of utilizing libraries of routines. The "*SEARCH" process was developed specifically for the needs of the LC compiler we released over three years ago. This assembler has served us well and continues to serve us for most of our needs. Of course, when LC was compiling a large program, the necessity to assemble the entire set of libraries for one little change precluded that assembler environment for our MC compiler - the MC compiler needed something more. That something was the ability to separately assemble modules and link them together with the already assembled library modules.

Linking together already assembled modules can also be done by an assembler such as EDAS. In fact, The first version of EDAS [version 3.4] was assembled by our disk-modified EDTASM. The source code consisted of two files. Now what is usually the case when you have more than one source file is that there are symbols in one file which are accessed by the other. What I had to do was to assemble EDAS1/ASM then alter the EQUates in EDAS2/ASM for any symbol used by EDAS2 which was in EDAS1. That wasn't too much work; but then I had to assemble EDAS2 and note any symbol in it which was used in EDAS1. Thus, I usually had to go back to EDAS1 and change some EQUates in it then reassemble it. This iteration had to be redone for every change which affected any symbol used by one file which was defined in the other. Now you can understand why the "generate EQU file" facility was programmed into XREF! How does a relocatable assembler make this easy?

MRAS provides two special pseudo-OPs called PUBLIC and EXTRN (which is short for external). Some assemblers use GLOBAL or ENTRY for PUBLIC - MRAS accepts all three. MRAS also accepts EXT for EXTRN. Here's how we use these pseudo-OPs. If we have defined a symbol in one module which we want to use in other modules, we declare the symbol PUBLIC in the module where it is defined. Some assemblers, such as MRAS, allow us to declare a PUBLIC symbol by appending two colons after its identifier. Now any other module which wants to use that symbol simply declares it EXTRN. This can be done by an explicit EXTRN statement, or in the case of MRAS and M80, by appending two numbersigns to the end of the symbol where it is referenced. This process tells the assembler, in the defining module, to write information about the public symbol in the REL file. Similarly, all modules which have the symbol declared external will have this information written to their REL files. The information is used by the linker during the linkage process to resolve the absolute address of the public symbols where needed by those modules declaring them external. This

external symbol resolution is the automatic process of the tasks I outlined above for generating EDAS 3.4. Let me tell you, it's far easier to resolve those externals in a relocatable assembler - especially when you can have hundreds of such "cross-used" symbols amongst dozens of REL modules.

Another good use of this PUBLIC and EXTRN business is actually NOT using it on symbols which you don't want to consider as public. Unless a particular symbol is declared PUBLIC, its symbol identifier is not known to any other module. Thus, that identifer may be reused in other modules without any conflict whatsoever. Symbols which are not declared public are known only to the module which defines them; they are considered LOCAL symbols.

The next topic concerning relocatable assemblers is the use of segment identifiers. But first, we have to define this term, "segment". For the typical TRS-80 program, there is absolutely no reason to differentiate between pieces of the program module which actually contain program code and pieces which contain data areas. The program code is the actual machine instructions which get executed by the CPU whereas the data area is referenced by the machine instructions for storing bytes or "words". If we want to assign a name to each area, we use the terms "code segment" and "data segment". A particular reason to "segment" a program into code and data regions is when the program is to be encoded in a read-only memory (ROM). For instance, the BASIC interpreter of the Model I or III computer is in a ROM. Since it accesses data areas which must be in read/write memory (RAM), it requires that the interpreter's source code be written in such a manner that the data segments are isolated from the code segments. In that way, the data segment can be declared to originate in the RAM address space of the machine. The linker usually has a facility for this address designation.

Now, don't get the wrong idea. It doesn't require a relocatable assembler to provide for more than one segment type. An absolute code generating assembler such as EDAS provides internally for only one segment - it doesn't care whether this is code, data, or anything else. However, with the use of the DEFL pseudo-OP and some clever programming, more than one segment could be utilized. This technique is discussed in the EDAS user manual. However, the clever technique is made a little easier with the segment pseudo-OPs such as CSEG, DSEG, and COMMON. COMMON? What's that all about?.

My first exposure to a COMMON segment was rooted in my early FORTRAN programming days. Let's take a look at what it can do for us. Let us say you have a data segment which is to be accessed by many other modules. You would need to declare each of the symbols in this segment as PUBLIC. Then, in every other module, you would need to declare those symbols as EXTRN. Another way of accomplishing this would be to define the data areas as being in a COMMON segment. The declarations would most likely be input into a separate file which would then by *INCLUDEd into those files needing access to the data. This may not sound like an easier method because the EXTRNs in the former DSEG method may be just as well included in a separate file! But there is also another reason. Just as the linker has a means of placing code segments and data segments at address spaces specified by the programmer, it can do the same for a COMMON segment, as well. In fact, most assemblers and linkers support NAMED COMMONs so you can specify a multitude of common segments - each distinct. The usual rationale for such a facility is in more complex operating environments using overlays of program and data spaces. You may still find large program environments operating on the TRS-80 built up from an approach of a ROOT segment and overlay modules. On the other hand, I don't find much use for COMMON in the TRS-80 environment.

Now that we have touched on the segment issue, when do you need to deal with it. In most cases, you don't. There is no restriction on code being only in a code segment and data only in a data segment. You can put code into a data segment as well as you can put data into a code segment. Unless you need to segregate your code and data for other reasons, you don't have to use a data segment. Unless

otherwise told by means of an explicit segment pseudo-OP, MRAS defaults to an assumed CSEG. It's perfectly suitable to keep your entire program in a code segment if your only intended use is for the TRS-80. Of course, if you are building up a library of modules, it won't hurt to keep data separate from code and it may well be helpful later on when those modules are to be used under a different operating environment.

I'll close with a caution. If you are going to use MRAS to generate a program from a single file which uses no "relocatable" pseudo-OPs or techniques, then use MRAS' "-GC" switch to directly generate an executable program file. Don't bother to generate the intermediate REL file followed up by its linking. I state this because a relocatable assembler usually has restrictions on the evaluation of expressions. These restrictions may be more restrictive than those found in an absolute code generating assembler like EDAS, or MRAS itself with the "-GC" switch. One of the traps you can fall into when you don't follow this advice is discussed in the section on EDAS/MRAS. Study it, and don't be afraid of jumping into a relocatable assembler - now that you know something about its environs.

MACHINE SENSING by Jeffrey R Brenton

It all started when someone put a request up on the LSI board on CompuServe, looking for an easy way to change the baud rate on an LX-80' serial ports. Normally, you would have to reset the device, then set it to RS232x again. But this decreases your available memory (RS232 drivers do not re-use their old locations) and you have to specify all the other parameters again. It is just too much when all you need to do is change from 300 to 1200 baud. I have the same need on my MAX-80, as it uses much the same circuitry as the LX-80 for its two ports, and had thought, "I should write a little program to do that." Well, now I had a REAL reason to do it, FAME! (Remember the first steps to becoming an expert assembly language programmer? Well, forget them! By the 15th time through a program with DEBUG to find out why the system reboots after the sign-on message, all vestiges of ego will have disappeared!) I soon had a working program that would handle the Lobo MAX-80's ports and, with a little patching, the LX-80's as well. It was flexible, and in keeping with my Philosophy of Perfect Inertia (read laziness), it made use of system calls to reduce my programming effort.

The next step was to make some money on it by writing an article for the Journal. After reading my first effort, though, the folks at LSI said, "It's too machine-specific. Could you make it so it runs on all machines?" "Uh, yeah, I *guess* I could....." was my reply.

And so was opened that can of worms known as hardware dependency. Telling a model I from a model III is easy, but that is not enough. With the model I, you have both the Radio Shack expansion interface (with one serial port) and the Lobo LX-80 (with two). The LX-80 can be tested for by looking for its special ROM located at x'3000'. If the first byte in this area isn't x'F3', the interface is not an LX-80.

As for model III machines, there are really three of them. There is the genuine model III, the model 4 operating in model III mode (both with a single port) and the MAX-80 (with two), which emulates a model III. A real model III can be told by looking at location x'3029'. If this location is non-zero, you have a real model III. Both the model 4 and the MAX have a x'00'. How, then, can you tell a MAX from a model 4?

Originally, you could find out by trying to change a location in the ROM area. If it changed, you had a MAX. If not, it was a model 4. Enter Duane Saylor and MDISK4, a program to allow you to use the extra 64K of RAM in the model 4 as a RAM-disk in model III mode. To do this, Duane had to copy the ROM into RAM and switch the machine to all-RAM mode. This means that a model 4 can now pass the MAX test. NOW how could you find out? There are a few ways that involve trying to switch banks, but these are potentially destructive. After a while, though, I did find a very reliable and completely non-destructive test for the MAX.

The MAX-80 (and the LX-80 interface) supports a variety of disk drive types. There is a DIP switch on the back of the keyboard unit to select the drive the system will use for booting. Under LDOS, this switch is memory-mapped to location x'37F8'. Lobo, however, did not bother to decode the two least-significant bits of this address. The switch's value can actually be read from any address in the range of x'37F8' through x'37FB'. The test, then, is to compare any two of these addresses and, if they contain the same value, we have a MAX, as the 'C' ROM in the model 4 contains some patch code at these addresses.

Once I had this information, adding the machine sensing code was easy. So easy, in fact, that I added code to sense the model 4 in TRSDOS/LDOS 6.x mode, by looking at the first memory address. All LDOS 5.1.x systems contain a x'F3' here, which is the 'DI' (Disable Interupt) instruction, while a 6.x system will contain a jump instruction. If the program is used under 6.x, it must use SVC calls instead of the more familiar 'fixed location' calls of 5.1.x and earlier.

The actual working part of this program is very short - most of it is the machine-sensing code and tables for using the @Param vector so thoughtfully provided by LDOS. Although the manual is not exactly self-explanatory on the use of @Param, Roy Soltoff gave an excellent lesson on it in the April 1, 1982 issue of the Quarterly (page 42) and I will give a short one here myself.

At the simplest, a program must provide a table consisting of eight-byte entries, with a x'00' byte at the end of the table. Each entry is six-ASCII characters followed by a two-byte address, pointing to where @Param is to store the result.

If an entry is less than six characters long, it must be left-justified with spaces. This format works with both versions 5.1.x and 6.x. For the special format allowed under 6.x, see Les Mikesell's column in the October, 1983 Journal. You then load the DE register pair with the address of the start of your table, point HL at the command line (if you have been REALLY lazy, it is still pointed there from when LDOS gave your program control), and call @Param (x'4476' in the Model I, x'4454' in the MAX or Model III, or SVC 17 under 6.x). @Param then looks at the rest of the command line searching for matches to the table. If one is found, it is evaluated and the result is placed in the location pointed to by the two-byte address in the table entry. Clear as mud so far, right? Wait, it gets better.

What do you get out of all this? Well, @Param will evaluate decimal, hexidecimal, string or flag type entries attached to your parameters, and hand your program a two-byte value depending on certain rules. If the value is a string, @Param puts the address of its beginning into the address pointed to by the table. If it is a flag (ON, YES and Y are true flags, OFF, NO and N are false flags), either x'FFFF' (true) or x'0000' (false) is used instead. As an aside, the parameter by itself, as in 'INV', is considered true, while the parameter with only an equal sign, as in 'INV=', is false. Hexidecimal and decimal are converted to two-byte integers and also put into the address the table points to.

You should be familiar with how it works already, as LDOS makes extensive use of @Param throughout the system. Remember BACKUP :1 :2 (MPW="PASSWORD",INV,NEW,Q=) ? That is an example of @Param in action. In the backup program, there is a table that looks something like this:

```
TABLE   DB      'MPW    '       ;Master password
        DW      MPWADD          ;pointer to where starting address for
 ;master password string is to be stored
        DB      'INV    '       ;Invisible files
        DW      INVFLAG         ;pointer where flag for INVisible
 ;files is kept
        DB      'I      '       ;abbr. for INV
        DW      INVFLAG         ;same as above
```

```
        DB      'NEW    '       ;New files only
        DW      NEWFLAG         ;point to new-only flag
        DB      'QUERY  '       ;Query on each transfer
        DW      QFLAG           ;Where qflag is kept
        DB      'Q      '       ;abbr. for QUERY
        DW      QFLAG           ;same as above
        .
more entries
        .
        DB      0               ;end of table
```

Ah ha, you say, some of those entries had the same pointer! Yes, because @Param will only take exact matches to your table. Any alternate names or abbreviations you want to accept must be included in the table (except for the special table arrangement allowed under version 6, as noted by Les), but they should point to the same address as the 'official' spelling. Backup will take both 'INV' and 'I' as meaning 'backup the INVisible files', so both must have entries in the table.

When our previous command line is encountered by @Param with DE pointing to this table, it will see that INV matches a table entry. Since it is by itself, it must be a flag, and x'FFFF' is put into INVFLAG. The same is true for NEW. With Q=, however, QFLAG will be filled with x'0000'. What of MPW? Because a double-quote was encountered, this must be a string entry, and MPWADD will have the address of the first character in the string.

Now, you might wonder what happens to all those other entries that backup allows. The answer is nothing. If a parameter is not entered, it is not changed. This allows you to load your program with default values for anything the user does not specify.

What does all of this have to do with changing baud rates? Plenty! With each of these computers, there are 16 different baud rates to chose from. The LX-80 and MAX-80 also have two different ports to worry about. I did not want to write 32 versions to cover all possibilities, so some way was required of telling the program what you wanted. That left 2 ways of doing it, either parse the command line myself (too much work!) or let @Param do it and just grab the results. I, of course, chose the easy way. Let's take the program in pieces.

```
00100          TITLE   <SETBAUD/CMD>
00110 ;set baud rate without changing other parameters of RS-232C port
00120 ;Syntax is: SETBAUD (300 {,A}) where 'A' can be either port A
00130 ;or port B (Default A) and '300' is the baud rate desired
00140 ;(aborts if not specified)
00150 @PARAM3 EQU     4454H
00160 @PARAM1 EQU     4476H
00170 @DSPLY  EQU     4467H
00180          ORG     5200H
```

So far, it is just standard assembly language. We explain who we are, what we expect, and what we intend to do (You comment your code like this, don't you? No? You should!). Since @Param is different for both model I and model III systems, we define an EQUate for each. We also will use @Dsply, so let's define it too. Finally, we must tell EDAS where to start loading code, hence the 'ORG 5200H'.

Next slide please.

```
00190 START   JP      BEGIN
      ;We'll put the parameters here
00200 TABLE   DB      '50     '
00210         DW      TRY50
00220         DB      '75     '
00230         DW      TRY75
00240         DB      '110    '
00250         DW      TRY110
```

```
00260          DB      '134.5 '
00270          DW      TRY135
00280          DB      '150   '
00290          DW      TRY150
00300          DB      '300   '
00310          DW      TRY300
00320          DB      '600   '
00330          DW      TRY600
00340          DB      '1200  '
00350          DW      TRY120
00360          DB      '1800  '
00370          DW      TRY180
00380          DB      '2000  '
00390          DW      TRY200
00400          DB      '2400  '
00410          DW      TRY240
00420          DB      '3600  '
00430          DW      TRY360
00440          DB      '4800  '
00450          DW      TRY480
00460          DB      '7200  '
00470          DW      TRY720
00480          DB      '9600  '
00490          DW      TRY960
00500          DB      '19200 '
00510          DW      TRY192
00520          DB      'A     '
00530          DW      PORTA
00540          DB      'B     '
00550          DW      PORTB
00560          DB      0
00570 ;End of parameter table
```

The first instruction is a jump to bypass our table of parameters. This normally won't be executed, because later we will tell EDAS that the transfer address is BEGIN. Following the jump is the table of all baud rates supported by the hardware. The order in this table is not important, but I left them in lowest to highest order. At the end of the list are the entries 'A ' and 'B ', corresponding to the two ports available on the MAX-80 and the LX-80 interface. All pointers refer to a table at the end of the program, where I do want them in order. This is to keep the code necessary to compute the baud-rate mask value to a minimum. Now, let's find which machine we are on and act accordingly.

```
00580 BEGIN   PUSH    HL              ;save command line pointer
00590         LD      A,(0000)        ;Test to see if this is
00600         CP      0F3H            ;TRSDOS/LDOS 6.x system
00610         JR      NZ,L6           ;it is, so jump
00620         LD      A,(125H)        ;test for model III
00630         CP      'I'             ;if this is not 'I', this is a model I
00640         JR      NZ,MOD1         ;so go to the code for model I
00650         LD      A,(3029H)       ;test for what KIND of model III,
00660         OR      A               ;i.e., III, 4 in III mode, or MAX-80
00670         JR      NZ,MOD3         ;a non-zero value here means mIII
00680 ;We must now determine whether we are dealing with a model 4 in mIII
00690 ;mode or a MAX-80 under LDOS. On the MAX, location 37F8H is used for
00700 ;the boot-drive selection DIP switch, but the two low-order bits of
00710 ;the address are not decoded, so the DIP switch's value can be read
00720 ;from any address between 37F8H and 37FBH. On the model 4 in III
00730 ;mode, these locations all have different values. This is how we tell
00740 ;them apart.
00750         LD      HL,37F8H        ;first location of MAX DIP switch
00760         LD      A,(HL)          ;read DIP switch value
00770         INC     HL              ;point to next location
00780         CP      (HL)            ;same value?
00790         JR      Z,MAX           ;yes, this is a MAX-80
00800 ;Now that we have resolved which machine we're on, we can use this
00810 ;information to determine which way to call @PARAM and @DSPLY. All
00820 ;versions call the parsing routine through their own special @PARAM
00830 ;call, and PARSE returns the value to be used to set the baud rate
00840 ;in the A register unless an error is encountered in the parameter
00850 ;list, in which case it drops immediately into the error routine
00860 ;which will display a nasty message and abort.
00870 MOD3    CALL    SIGNON5         ;tell everyone we're here (v 5.1.x LDOS)
```

```
00880           POP     HL              ;retrieve command-line pointer
00890           LD      DE,TABLE        ;tell @PARAM where the table is
00900           CALL    PARSE3          ;parse command line using m3 @PARAM
00910           JR      SETRS           ;jump to R/S port setting routine
00920 MOD1      CALL    SIGNON5         ;same
00930           POP     HL              ;    as
00940           LD      DE,TABLE        ;        MOD3
00950           CALL    PARSE1          ;parse cmd-ln using m1 @PARAM
00960           PUSH    AF              ;save the results for later
00970           LD      A,(3000H)       ;Test for LX-80 interface as opposed
00980           CP      0F3H            ;  to a standard R/S unit
00990           JR      Z,LX80          ;there is a ROM at 3000H, so LX-80
01000           POP     AF              ;no ROM, so we can set R/S port
01010 SETRS     OUT     (0E9H),A        ;load value into baud-rate generator
01020 L5EXIT    LD      HL,DONEMSG      ;Tell people we're done
01030           CALL    @DSPLY
01040           SBC     HL,HL           ;clear HL
01050           RET                     ;return to LDOS
01060 LX80      LD      A,(PORTB)       ;check if it is port A or B to be set
01070           OR      A               ;non-zero value means yes
01080           JR      Z,LX80A         ;zero means not port B
01090           POP     AF              ;restore baud-rate value
01100           OUT     (0EDH),A        ;set port B
01110           JR      L5EXIT          ;and exit successfully
01120 LX80A     LD      A,(PORTA)       ;just in case user input 'A='
01130           OR      A               ;we test for port A
01140           JR      Z,ERROR         ;display the error message
01150           POP     AF              ;restore baud-rate value
01160           OUT     (0ECH),A        ;set port A
01170           JR      L5EXIT          ;and exit
01180 MAX       CALL    SIGNON5         ;anounce our presence
01190           POP     HL              ;restore command-line pointer
01200           LD      DE,TABLE        ;point to parameter table
01210           CALL    PARSE3          ;parse command line with mIII @PARAM
01220           PUSH    AF              ;save baud-rate value for later
01230           LD      A,(PORTB)       ;as with LX-80, we check port selected
01240           OR      A
01250           JR      Z,MAXA
01260           POP     AF
01270           LD      (37D4H),A       ;MAX baud-rate ports are memory-mapped
01280           JR      L5EXIT          ;save code by using MOD1's exit routine
01290 MAXA      LD      A,(PORTA)
01300           OR      A
01310           JR      Z,ERROR
01320           POP     AF              ;restore baud-rate value
01330           LD      (37D0H),A
01340           JR      L5EXIT
01350 ;This is where we start dealing with TRSDOS/LDOS 6.x systems
01360 ;Since the TRS-80 Model 4 is the only 6.x system so far, we
01370 ;will only deal with its port arrangement. The main difference
01380 ;between this section and the other machines is the use of SVC's
01390 ;for all system calls, instead of direct calls to fixed locations.
01400 L6        LD      HL,SIGNON       ;announce ourselves under 6.x
01410           LD      A,10            ;using @DSPLY SVC
01420           RST     28H
01430           POP     HL              ;restore command-line pointer
01440           LD      DE,TABLE        ;point to parameter table
```

```
01450           CALL    PARSE6          ;parse command line
01460           OUT     (0E9H),A        ;set baud rate
01470           LD      HL,DONEMSG      ;tell everybody we're done
01480           LD      A,10            ;again, we use the @DSPLY SVC
01490           RST     28H
01500           SBC     HL,HL           ;clear HL
01510           RET                     ;return to TRSDOS/LDOS
```

Since I have managed the stack properly (no extra stuff left on it and pointing to the same location as when LDOS gave SETBAUD control), a simple 'RET' instruction will send us back to 'LDOS Ready'. This is because LDOS places the return address (x'402D' on version 5 systems) onto the stack, essentially 'CALLing' our program.

As you can see, I have added special code in both the LX-80 and MAX-80 sections to deal with picking up the port selection, since each port is set separately. First the program checks for port B and, if it was not the one selected, the program then verifies that the user did not say to NOT program port A. This bit is not strictly necessary, but is added to show how to deal with default parameters.

Now for part of the program that does the actual baud rate selection. We start with the calls to @Param, move on to determining which rate was actually chosen, then format the result for use by the machine-dependent code.

```
01520 PARSE1  CALL    @PARAM1         ;use mI @PARAM
01530         JR      WHICH           ;then search for rate chosen
01540 PARSE3  CALL    @PARAM3         ;use mIII @PARAM
01550         JR      WHICH           ;and continue
01560 PARSE6  LD      A,17            ;use TRSDOS/LDOS @PARAM SVC
01570         RST     28H
01580         JR      NZ,ERROR6       ;6.x requires special error routine
01590 WHICH   JR      NZ,ERROR        ;command line error
01600         LD      HL,TRY50        ;start of where @param put results
01610         LD      C,0             ;C will hold baudrate when we finish
01620 LOOP    LD      A,(HL)          ;check table entry for non-zero value
01630         OR      A               ;is it this one?
01640         JR      NZ,GOTBAUD      ;If A <> 0, this is desired rate
01650         INC     C               ;try next rate
01660         INC     HL              ;since each entry is two bytes long, we
01670         INC     HL              ;have to step twice each time
01680         LD      A,C             ;check to see if c is still valid
01690         CP      10H             ;must be 15 or less
01700         JP      C,LOOP          ;still valid, so continue
01710         JR      ERROR           ;not valid, no baudrate found. give error
01720 ;Since R/S RS-232R boards require both the send and receive baud rates
01730 ;to be set separately, we must duplicate the value in the low-order
01740 ;bits of C in the high-order bits. We do this by moving C into A,
01750 ;rotating C four times, then adding the result to A before returning
01760 GOTBAUD LD      A,C             ;start conversion
01770         RLC     C               ;once
01780         RLC     C               ;twice
01790         RLC     C               ;three times
01800         RLC     C               ;four times and done
01810         ADD     A,C             ;add high-nibble to low-nibble
01820         RET                     ;then return to caller
```

You will notice I started a counter in the C register with the value 0. This corresponds to the value that is placed in the baud-rate generator to obtain 50 baud, the first entry in the table at the end of our program. Each time through the loop, we check A for a non-zero value. If we find one, we will have the right bit-mask in C to program the generator for the rate that we are testing for. (This is why I want the second table to be in the correct order) When the selected baud rate is located, we must duplicate its mask in the high-order nibble of A before returning to the caller. This is because Radio Shack boards, unlike Lobo, allow a different baud rate for transmit and receive. We must, therefore, set both nibbles. The Lobo hardware ignores the extra bits.

```
01830 ERROR   LD     HL,ERRMSG       ;here is where error message is displayed
01840         LD     A,(0000)        ;make sure this is not a 6.x system
01850         CP     0F3H            ;by looking for DI instruction
01860         JR     NZ,ERROR6+1     ;use 6.x exit
01870         CALL   @DSPLY
01880         JP     4030H           ;and jump to ABORT
01890 ERROR6  POP    HL              ;clear stack
01900         LD     HL,ERRMSG       ;same as ERROR, except using SVC's
01910         LD     A,10
01920         RST    28H
01930         RET                    ;because HL <> 0, this is an ABORT
```

Here is where we tell people that we did not understand their command line. The 'JP 4030H' sends us back to LDOS in a way that will stop any JCL file from executing further. We do not want to compound one error by assuming the last command was executed correctly, when the baud rate may be radically different from what we expect (ever run 19,200 baud into a 300 baud modem?). Under 6.x, EXITing with HL <> 0 is the same thing.

```
01940 SIGNON5 LD     HL,SIGNON       ;point to sign-on message
01950         CALL   @DSPLY          ;display it
01960         RET                    ;and return to caller
01970 DONEMSG DB     'Baud rate changed',0DH
01980 ERRMSG  DB     'Parameter error! Please re-check your command line.'
01990         DB     0AH,'Correct syntax is: SETBAUD (p,r)',0AH
02000         DB     'where p is either port A or port B, and r is the desired'
02010         DB     0AH,'baud rate, i.e., 50, 75, 110, 134.5, 150, 300, etc.',0DH
02020 SIGNON  DB     'SETBAUD/CMD, baud rate changer for '
02030         DB     'TRS-80 and MAX-80 systems',0AH
02040         DB     '(c) 1983 by Jeffrey R Brenton',0DH
```

These are all of our messages, along with the code to display the SIGNON message on systems running LDOS 5.1.x. Notice that ERRMSG is defined on four different lines, and SIGNON on three. Some of you that are unfamiliar with assembly code might wonder how the system handles this. Quite well, actually. @Dsply will take a string, pointed to by HL, and display it on the screen, starting at the current cursor position. It will keep copying to the screen until it encounters either a x'03' (end-of-text) or a x'0D' (<cr>). You can get multiple lines displayed in just one call by using a x'0A' (linefeed) instead of a <cr> to end each line. EDAS will assemble these multi-line definitions into a contiguous string of characters, with the last one being the <cr> we add to the end. If only BASIC made it so easy to make strings longer than normal program-line length! Last slide, please.

```
02050 TRY50   DW     0               ;this is the table where @Param puts the
02060 TRY75   DW     0               ;results of parsing our command line
02070 TRY110  DW     0
02080 TRY135  DW     0
02090 TRY150  DW     0
```

```
02100 TRY300  DW      0
02110 TRY600  DW      0
02120 TRY120  DW      0
02130 TRY180  DW      0
02140 TRY200  DW      0
02150 TRY240  DW      0
02160 TRY360  DW      0
02170 TRY480  DW      0
02180 TRY720  DW      0
02190 TRY960  DW      0
02200 TRY192  DW      0
02210 PORTB   DW      0
02220 PORTA   DW      -1              ;this is 0FFFFH, to default to A
02230         END     BEGIN
```

This is our table that @Param is to use to store the results. Each parameter is given one word (two bytes) for storing the result of parsing our line. Almost all are initialized to 0, except PORTA. This is because port A is our 'default'. We could make B the default, but, whichever you use as a default, it should be the last one checked so as to make sure no other selection was made.

How about that? A lesson in using a system vector, a series of tests that will uncover the identity of the machine your program is running on and a useful program all in one!

Machine-dependent routines are normally to be avoided, but sometimes there IS a need to know what the hardware supports. The machine-sensing routines within this program can be of great value when you come up against such instances.

This program has also made very simple use of @Param. As you can see, there are many other possibilities. With a few simple changes, you can add command-line interpretation to many of the programs you write, which will save experienced users the "agony" of having answering, one by one, all the questions your program needs answered as it runs. That is what separates LDOS from most other 8-bit operating systems, in that the system provides you with the tools necessary to build "friendly" programs without having to re-invent the wheel everytime.

Here is the HEX file listing of SETBAUD:

```
05 06 53 45 54 42 41 55 01 02 00 52 C3 94 52 35 30 20 20 20 20 AA 54 37 35 20 20 20 20
AC 54 31 31 30 20 20 20 AE 54 31 33 34 2E 35 20 B0 54 31 35 30 20 20 20 B2 54 33 30 30
20 20 20 B4 54 36 30 30 20 20 20 B6 54 31 32 30 30 20 20 B8 54 31 38 30 30 20 20 BA 54
32 30 30 30 20 20 BC 54 32 34 30 30 20 20 BE 54 33 36 30 30 20 20 C0 54 34 38 30 30 20
20 C2 54 37 32 30 30 20 20 C4 54 39 36 30 30 20 20 C6 54 31 39 32 30 30 20 C8 54 41 20
20 20 20 20 CC 54 42 20 20 20 20 20 CA 54 00 E5 3A 00 00 FE F3 20 78 3A 25 01 FE 49 20
1A 3A 29 30 B7 20 08 21 F8 37 7E 23 BE 28 40 CD 74 53 E1 11 03 52 CD 31 53 18 13 CD 74
53 E1 11 03 52 CD 2C 53 F5 3A 00 30 FE F3 28 0C F1 D3 E9 21 7B 53 CD 67 44 ED 62 C9 3A
CA 54 B7 28 05 F1 D3 ED 18 EC 3A CC 54 B7 28 70 F1 D3 EC 18 E1 CD 74 53 E1 11 03 52 CD
31 53 F5 3A CA 54 B7 01 02 00 53 28 06 F1 32 D4 37 18 CA 3A CC 54 B7 28 4E F1 32 D0 37
18 BE 21 4F 54 3E 0A EF E1 11 03 52 CD 36 53 D3 E9 21 7B 53 3E 0A EF ED 62 C9 CD 76 44
18 0A CD 54 44 18 05 3E 11 EF 20 31 20 1F 21 AA 54 0E 00 7E B7 20 0B 0C 23 23 79 FE 10
DA 42 53 18 0B 79 CB 01 CB 01 CB 01 CB 01 81 C9 21 8D 53 3A 00 00 FE F3 20 07 CD 67 44
C3 30 40 E1 21 8D 53 3E 0A EF C9 21 4F 54 CD 67 44 C9 42 61 75 64 20 72 61 74 65 20 63
68 61 6E 67 65 64 0D 50 61 72 61 6D 65 74 65 72 20 65 72 72 6F 72 21 20 50 6C 65 61 73
65 20 72 65 2D 63 68 65 63 6B 20 79 6F 75 72 20 63 6F 6D 6D 61 6E 64 20 6C 69 6E 65 2E
0A 43 6F 72 72 65 63 74 20 73 79 6E 74 61 78 20 69 73 3A 20 53 45 54 42 41 55 44 20 28
70 2C 72 29 0A 77 68 65 72 65 20 70 20 69 73 20 65 69 74 68 65 72 20 70 6F 72 74 20 41
20 6F 72 20 70 6F 01 D0 00 54 72 74 20 42 2C 20 61 6E 64 20 72 20 69 73 20 74 68 65 20
64 65 73 69 72 65 64 0A 62 61 75 64 20 72 61 74 65 2C 20 69 2E 65 2E 2C 20 35 30 2C 20
37 35 2C 20 31 31 30 2C 20 31 33 34 2E 35 2C 20 31 35 30 2C 20 33 30 30 2C 20 65 74 63
2E 0D 53 45 54 42 41 55 44 2F 43 4D 44 2C 20 62 61 75 64 20 72 61 74 65 20 63 68 61 6E
67 65 72 20 66 6F 72 20 54 52 53 2D 38 30 20 61 6E 64 20 4D 41 58 2D 38 30 20 73 79 73
74 65 6D 73 0A 28 63 29 20 31 39 38 33 20 62 79 20 4A 65 66 66 72 65 79 20 52 20 42 72
65 6E 74 6F 6E 0D 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 FF FF 02 02 94 52
*83
```

Make an ASCII file of the above WITHOUT SPACES. They are there for readability only. The checksum (*83) should not be included if you are using the /BAS version of BINHEX. After you have finished typing it in, run BINHEX. At the prompt, type '2' for hex-to-binary conversion and give the file name you gave to the ASCII file. You can give the binary file whatever name you wish. BINHEX will create a runnable file (provided you made no typing errors). Later versions of BINHEX will give you a checksum to compare against the one shown, and the compiled version will compare the checksums for you. If they disagree, check for errors (do NOT miss any of those 0's!).

This program is available in the DownLoad database of the LDOS/TRSDOS6 Special Interest Group sponsored by MISOSYS, Inc., on CompuServe, as SETBD.HEX under ppn 73105,532. Any questions regarding it may be directed to me there or by mail to:

Tandem-Flow Systems, Ltd.
P O Box 146
Woodstock, Illinois 60098
Attention: Jeff

Product Highlights: ADE

SCM exposed a rare problem when SYSGENning an ADE floppy when the host drive was not a hard drive. Let me summarize the problem. Using TRSDOS 6.2, he created an ADE floppy using a floppy disk as the host disk. He then SYSGENed this configuration. Upon reBOOTing, when he accessed the ADE drive before accessing the host drive, the drive head moved to the stop and just stayed there making a terrible banging noise. He suggested that the problem related to the value in DCT+5 (normally the current cylinder location). That is indeed the root of the problem. I confirmed this sequence of events; however, on my system, the drive recovered and eventually properly accessed the file on the ADE floppy that I had attempted to load into MSCRIPT. An investigation into the cause revealed the following scenario.

Under TRSDOS 6.1 and 6.2, the BOOT process concludes with a routine which scans all logical drives' DCTs other than zero starting from logical drive 7 and working its way to logical drive one. If the drive is not a hard drive (noted via DCT+3, bit-3), it loads an X'FF' into DCT+5 which sets the current cylinder out of range. It then proceeds to issue an @RESTORE to the disk driver for that drive unless the RESTORE=OFF option is set using the SYSTEM command. The @RESTORE function in the DOS disk driver loads a zero into DCT+5 to note that the drive is positioned at cylinder 0. The purpose of the X'FF' stuff into DCT+5 is to "cure" the old bang-the-head-against-the-pin problem that occurred under LDOS 5. Why then does the problem occur here?

It goes a little deeper. The ADE driver is smart to recognize that it only needs to operate on type-2 disk requests (those functions 8-15). Thus, for functions 0-7, ADE passes the request through to the host drive. However, when the BOOT loader stuffs the X'FF', it stuffs it into the DCT belonging to the ADE floppy. When ADE receives the @RESTORE request, it locates the DCT of the host disk and passes the function request to the host. The DOS disk driver then performs the @RESTORE operation on the host drive; thus zeroing DCT+5 on the host DCT - the ADE floppies DCT+5 remains at X'FF'.

When you attempt to open a file on the ADE floppy, the first thing done by the SYS2/SYS module which services the @OPEN request is to issue an @CKDRV on the requested drive. This is still the ADE floppy. The @CKDRV routine was "smartened" up in TRSDOS 6.x to first check if the current cylinder is beyond the maximum cylinder. It does this by comparing (DCT+5) to (DCT+6). If it finds that the current cylinder (X'FF' in this case) is greater than the maximum cylinder (X'27' for a 40-cylinder drive), guess what @CKDRV does? It issues an @RESTORE! This request goes to the ADE driver which sends it along to the host disk driver and the restore operation is performed on the host drive! Thus, the X'FF' contained in DCT+5 of the ADE drive remains an X'FF'. The next thing that @CKDRV does is make the assumption that (DCT+5) is in range, picks up the value, then issues an @SEEK to that cylinder. Bang! The head is stepped against the pin because the SEEK request is to cylinder 255!

The ADE driver does not use DCT+5 for any purpose - it starts out being zero, which is an OK value. Unfortunately, the BOOT initialization is the one thing in the system which will change this value in an ADE's DCT. The temporary solution is to NOT SYSRES the ADE driver if you are going to have a floppy disk as the host drive. Using DEVICE or LOG will not help since they both use @CKDRV. The ADE driver could be modified in a number of ways. First, it could trap the @RESTORE function and just reset DCT+5 to zero. It could also force DCT+5 to an arbitrary but "good" value. For the time being, the problem is rarely experienced. The suggested cure is to not SYSGEN ADE when the host drive is a floppy.

Product Highlights: BSORT

From our Compuserve Special Interest Group (PCS49), the following BSORT question was raised by KZ: " A while back I bought some programs from your firm and BSORT was one of them. I am working on a program I found in 80 MICRO for sorting Visicalc

spreadsheets and the line for sorting reads as follows:

```
330 CMD "O",RC,D$(RS)
```

and all I am getting is a syntax error - RC stands for the Row and Column number. Since it appears that I did not have a sort program for my Model 4P I went to the BSORT program instead. I typed in the following line:

```
330 system "BSORT RC,D$(RS)"
```

but all I get is "Command aborted".

Joe, our trusty support SIG SYSOP came up with the easy solution. Firstly, the line you indicate will not work, as the starting element and number of elements must be explicitly defined/typed integer variables. Secondly, it will not work because it is missing the TRSDOS "RUN" verb. For example, instead of [330 system"BSORT RC,D$(RS)"] you must use:

```
330 system"RUN BSORT RC%,D$(RS%)"
```

and RC%, RS% must be defined as integer first thing in the program, and occur as integer throughout. Alternately, if you don't have a full function editor to do a global search and replace, you might try something like this:

```
330 t1% = rc:t2% = rs
system"RUN BSORT T1%, D$(T2%)"
```

These points are explicitly covered in the BSORT documentation. You have also copied the file BSORT/CMD over to the working TRSDOS 6.2 diskette that you are trying to run this program on, and this disk is present in drive zero when trying to execute the sort, right? --jjkd--

Product Highlights: CONVCPM

Ian Kluft of Mt. Shasta CA writes, "For two issues of NOTES now, you have claimed that the Model 4 CP/M+ format is the same as IBM PC CP/M-86. I have my doubts that they are exactly alike. I know that they're close. I used to work at a dealer of an IBM compatible computer (the Fujitsu Micro 16s). It's CP/M-86 was claimed to be the same format as the PC's. I could read those disks with my Model 4 under CP/M+, but it couldn't read my CP/M+ disks. Also, I had problems writing to the CP/M-86 disks with my Model 4. It kept overwriting other data like it didn't know it was there. Indeed, the Fujitsu may be the one that's not exactly like IBM, but this is something worth looking at in a bit more detail, before Model 4 owners start losing data."

Product Highlights: DD&T

Claude E. Hunter was having trouble when both LS-diskDISK and PRO-DD&T's disassembler module were both installed. The problem with DD&T's DD/CMD when also using LS-diskDISK's DD/CMD stems from both /CMD programs using the same 2-letter module name, "DD", for their "terminate and stay resident" memory module. Use the DDTD61/FIX patch to be applied to the PRO-DD&T's DD/CMD file to fix this conflict. This patch will change it's module name to "D1"; thus, there will be no further conflict - other than both /CMD files having the same name.

Product Highlights: DESCRIBE

Here's another use for the extended file descriptors provided with our DESCRIBE package. It comes from Jeff Brenton 76703,1065 and recently appeared on our Compuserve Special Interest Group (PCS49). "Contact MISOSYS for a copy of Pro-DESCRIBE. it will let you add a 63-character description to each file on a disk, although you have to do it outside of SuperScripsit, at the DOS level. I use it to keep track of purchase orders on a disk at work.

Product Highlights: DSM

Joel Weisbrod had a problem running the Model I/III version of DSM from a Job Control Language file. When I investigated why DSM51 doesn't permit operation from JCL, I found that it was documented incorrectly. According to how the program is written, it is not going to even look for a command character (what the "S" was

purported to stand for). Thus, the "S" is being interpreted as the output file specification. This error propogated itself to the third line which was then expecting the "A". The easy solution is to omit the "S" line from the JCL file. Thus, the correct file syntax for operating DSM51 from JCL is as follows:

```
DSM51 SORTED/MAP (JCL)
SORTED/IND
A
4
3
```

LD had a large mailing list of 2000-2500 names which was segmented into about five files so he could sort each file with an in-memory BASIC sort. LD was interested in adapting DSM's disk sorting capability into his application. Here are some tips that may be useful for others in the same boat.

Since the datafile has 2500 or so names, LD had to break it up into smaller pieces so he could deal with it in BASIC. All well and good. However, that really hinders the operation. LD had to break it up strictly for the sorting which was RAM based. Let me outline another technique.

The concept of using an integer array which contains the sorted index of the total data file needs to be understood. What this means is that the program needs to be revised to use a single data file along with an index file. The index file is random accessed with an LRL of 2. Each index record is a packed integer. That integer represents the actual record position in the data file.

Assuming a single data file, DSM4 will sort it according to the sort criteria and generate the desired index file. Once the index file is created, if, for instance, you want to access the 50th sorted record, you read the 50th record of the index file then use that value as the GET pointer to read the data file. Now 2500 index entries of 2-bytes each is only 5K of memory. You probably could then keep the index file in memory for speedier access. The only memory space allocated for data would be for one record at a time. That's the entire purpose of DSM4 - to allow you to sort a big file and produce an index of the sorted order.

Product Highlights: FED II

Here's a tip that relates not just to FED, but to a lot of other programs which use a default file extension and you want to access a file which has no extension. It's just one more example of the type of support provided on our Compuserve Special Interest Group.

Alan H. Pesetsky 75675,1535 said, "I've been taking a look at the new SCRIPSIT-PRO. They have a sample file in their tutorial called TWAIN. When I tried to look at its structure using FED I got a "file not in directory message". Otherwise I can read the file using TSK and it lists using the DOS list command (though it pops into reverse after the header). Any clues as to why FED thinks it isn't there?"

(RE) If you are using FED-II, and the file is named "TWAIN" with no file extension, then tell FED the file is "TWAIN/". The "/" is important. It says don't add the default extension of "/CMD". I suspect if my assumptions hold true, that FED is telling you that "TWAIN/CMD" is not found!

On the other hand, the LIST command of DOS will try to access "TWAIN/TXT" first, then "TWAIN/" second if the "/TXT" file cannot be opened. That's why LIST was able to list the file.

Product Highlights: Little Brother

The following notice was added to the TRSDOS 6.x version of Little Brother on April 17, 1986. "For proper printing operation using Little Brother under TRSDOS 6.x, you need to install the FORMS filter if your printer does not CARRIAGE RETURN when it receives a LINEFEED. This can be accomplished by the following two TRSDOS 6 library commands:

```
SET *FF FORMS
FILTER *PR *FF
```

A Little Brother user needed help in getting to first base with LB and in

developing a screen definition. What was essentially needed was an outline of the steps neccessary to create a usable data base. What follows is such an outline."

There are approximately three processes that need to be accomplished in order to create a data management system using LB. The first is to define the data file environment. This task establishes the different items (data fields) which will appear in the data file. This part corresponds to your illustration of name, address, etc.

Once you have established the structure for a data file, two more things need to be accomplished. You have to provide a means for printing the information and you have to provide a means for entering information into the data file. You apparently understand the process of defining the output printing formats discussed on pages 19-58 of the LB manual.

The DEFINE SCREEN FORMAT section discusses that task associated with telling LB how you want to see the input screen for entering data. You see, other database programs which allow you no control over the input screen leave you no facility for customizing the input. With LB, you can easily create the input screen. All you do is follow the instructions on pages 7-18. You move the cursor around on the screen until it is positioned at the screen location where you want to see the request to enter a field then type in the appropriate prompt and field specifier. Using your example, if you just want to enter each item on a separate line, then assuming your example fields are numbered in the order presented in your letter, the following input screen would accomplish this:

```
Name: ^1^
Address: ^2^
City: ^3^
State: ^4^
ZIP code: ^5^
Phone: ^6^
```

Of course, you could dress up the screen with a heading such as, "Customer Address File". You certainly could position the input fields according to your taste - perhaps putting City, State, and ZIP on one line.

A useful reason for giving you the capability of defining your input screen would be to isolate data entry to a subset of the fields in your data file. You may have a large number of fields which do not get input but are calculated. Or maybe some fields are entered rarely. You can have different input screens to use at different times - each input screen tailored to the data which you want to enter. Having less fields on the input screen gives it a less cluttered look.

I am confident that you will find LB's methods easy to use. Don't forget that you can always change an input screen's presentation - even after you have begun using it to enter data.

Another user wanted to duplicate a Little Brother data base structure. Turns out the job is extremely easy IF you do a little preplanning. If you have any reason to suspect that you may want to duplicate your data base, save a copy of the DEF, LB, VDn, and PRn files BEFORE YOU ADD ANY RECORDS. Then all you need do to create subsequent data bases using the same data definition is to use backups of the "saved" set. Using a different set of "filenames" helps. If you have not preserved a copy of these original files, you will have to recreate the data file definition; however, the VDn and PRn files may be reused. The definition can be easily recreated by using a printout of the field definitions obtainable from the "define data field format" function using the existing data base.

Now speaking of existing data bases, I recently went through a revision of the customer file database here at MISOSYS. A little background is in order. Prior to the acquisition of LSI's retail operation, we used PowerMAIL to hold our base of registered customers (we only kept those interested enough in returning a registration card). Logical Systems used PROFILE in the past and then Little Brother after they completed that package. We, of course, acquired their data base. Now LSI had about 50,000 records in that LB data base. About 12,000 were identified

as recent purchasors or registered customers. The remaining 36,000 represented inquiries, subscription lists of defunct magazines, and other entries we didn't feel warranted our direct mailings. Thus we wanted to trim their list down to 12,000. We also wanted to drop some fields and add others (specifically fields relating to THE MISOSYS QUARTERLY subscriptions). That meant a significant use of the Little Brother Maintenance Utility. We also wanted to merge our data base into the revised LSI base. That meant a bulk loading using the AUTO option of Little Brother.

The LSI base runs on an old Leading Edge PC e/w a 10 Meg hard drive. Since the 50,000 record base used up 8.5 megs of the hard drive, we needed a larger hard disk to provide room to store both the old base and the new one. To do this, we acquired a 20Meg drive package with controller and cables for $399 - good deal. Turns out that it would not work in the Leading Edge machine but worked perfectly in the IBM PC we had. So much for compatibility of the older LE. So, using the PC for the job, we backed up the 50,000 record base onto about 25 floppies and restored the base to the IBM's 20Meg drive. We then defined a new data base structure for our new base using the LB Maintenance Utility (LBMU). We extracted the 12,000 "active" records and populated the new base with them - all this is part of the LBMU. We then backed up the new data base onto floppies and restored them to the LE machine after purging all of the old data base files.

The next job was to get our PowerMAIL data over to the new data base. Using the PMAIL's function to extract records into an adder file, we generated a 4,000 record adder file. Now the AUTO facility of LB works something like LDOS's redirection capability - it allows you to obtain the input, which normally would come from the keyboard, from a disk file. All I had to do now was create a disk file that would look like the keystrokes I would type to input 4,000 records. Easy stuff!

As part of their technical support literature, LSI had available an example of a conversion utility written in BASIC to take a fixed record length file and convert it to a LB AUTO-input file. Since it was not published in the last LSI JOURNAL, I think it appropriate to publish it here.

```
10 OPEN"R",1,"OLDDATA",LRL: 'LRL must match existing file
20 FIELD 1, ... ...: 'FIELD the old file to match its data structure
30 OPEN"O",2,"AUTO/JOB": ' OPEN the LB auto file
40 PRINT #2,"2";CHR$(13);"A";
50 FOR L=1 TO LOF(1): 'DO all records in old data file
55 GET 1,L
60 PRINT#2, field variable 1;CHR$(13);
70 PRINT#2, field variable 2;CHR$(13);
80 ETC... (DO ALL FIELDS IN OLD FILE)
90 PRINT #2,CHR$(27);
199 NEXT L
110 PRINT#2,CHR$(26);
120 CLOSE:END
```

The semicolons after the "PRINT #2" statements are VERY important - do not forget them! If there are blank records at the end of the old data file, line 50 should be changed to read: FOR L = 1 TO real number of records.

Now one more important thing to remember is that if your new Little Brother data base has additional fields or it no longer includes some of the fields which were in the old one, then take that into consideration. A carriage return (CHR$(13);) should be PRINT#'d in the proper position for each new field. And don't PRINT# any field from the old data file which is not used in the new file.

As a more concrete example, here's a copy of the BASIC program I used to convert my PowerMAIL file to the auto input file of Little Brother. This program was written

for use with our EnhComp BASIC compiler on the Model 4.

```
'MDATA/BAS - 06/09/86
ALLOCATE 2: OPEN "r",1,"pmail/add",128
FIELD 1, 15 AS F0$, 10 AS F1$, 20 AS F2$, 20 AS F3$, 10 AS F4$, 15 AS F5$, 8 AS F6$,
10 AS F7$, 5 AS F8$, 12 AS F9$, 1 AS FLG1$, 1 AS FLG2$, 1 AS FLG3$
OPEN "o",2,"mdata/dat:1"
PRINT#2,2:PRINT#2,"A";
FOR I = 3 TO LOF(1): GET 1,I: 'Skip the header sector in PMAIL/ADD
IF ASC(LEFT$(F0$,1)) = 255:' Do not bother with deleted records
    NEXT I
ENDIF
PRINT#2,!STRIP$(F2$):'Company name field
PRINT#2,!STRIP$(F0$):'Last name field
PRINT#2,!STRIP$(F1$):'First name field
PRINT#2,!STRIP$(F3$):'Address 1 field
PRINT#2,!STRIP$(F4$):'ADDRESS 2
PRINT#2,!STRIP$(F5$):'CITY - FIELD 6
IF ASC(FLG3$) AND 32:' if foreign customer
    PRINT#2,"":       ' do not print contents of STATE field
ELSE
    PRINT#2,!STRIP$(F6$)
ENDIF
PRINT#2,!STRIP$(F7$):'ZIP code field
IF ASC(FLG3$) AND 32: if foreign customer
    PRINT#2,!STRIP$(F6$):'print STATE field now (Country)
ELSE
    PRINT#2,"":'else country field blank for US
ENDIF
PRINT#2,"":'FIELD 10 - MAIL CODE
IF F8$ = "     ":'datal field stored last purchase date
    PRINT#2,""
ELSE
    PRINT#2,LEFT$(F8$,2)+"/"+MID$(F8$,3,2)
ENDIF
PRINT#2,"86/06":' ORIGIN DATE
PRINT#2,CHR$(13);CHR$(13);CHR$(13);:' PRINT 3 <ENTER>S
IF ASC(FLG1$) AND 128:'Keep record of MC purchase
    O$="C"
ELSE
    O$=""
ENDIF
PRINT#2,O$
PRINT#2,CHR$(27);:' SAVE THE RECORD
NEXT I
PRINT#2,CHR$(26);:CLOSE:END
FUNCTION STRIP$(S$):'   my function to strip trailing spaces
IF LEFT$(S$,1) = " ":'  just used to make the output file smaller
    RETURN ""
ENDIF
LOOP%=LEN(S$):INC LOOP%
REPEAT
    DEC LOOP%
UNTIL MID$(S$,LOOP%,1) <> " "
RETURN LEFT$(S$,LOOP%)
```

There you go. It's as simple as that. Now there is no excuse for not bringing your

old data files up to the flexibility of Little Brother data management. Oh, by the way. In order to get this processed file over to the PC from the MAX-80 where the old data was stored, I direct connected the MAX to the PC with a null modem and XMODEM'd the MDATA/DAT file. Now the file was about 400K so it took a while at 1200 baud; thet's why I added that STRIP function.

Product Highlights: PaDS

Here's some feedback which may help you in your use of PaDS. The PDS(APPEND) module of PRO-PaDS does not restrict programs which load at X'2600'. Even though the PaDS front end loader loads and executes there, it's not the FEL which loads the member, the system does. The FEL only extracts the name of the member requested from the command line and reads the member directory for a match. If found, it obtains the positioning information for the member and interfaces back to the system. Thus, the member which is loaded by the system can load into X'2600'.

If you have discovered certain "programs" which are forced to be data members by PDS(APPEND), this is caused not for loading at X'2600' but for other reasons. The only way a file can be considered to be a program member is if it's file adheres to the following three points: (1) the first file byte must be either 05H or 01H; (2) the file extension must be "/CMD"; and (3) the fourth byte from the end of the file must be 02H. Invariably, when a "program" is forced to data, its the third condition which is violated. It will mean that the "program" has an end-of-file pointer in the directory which is incorrect. If such is the case, the directory should be corrected then PDS(APPEND) will consider it a program. It can be corrected either manually with a file editor (such as FED/CMD or FED/APP), or by loading the offending file into PROCESS (CMDFILE for Model I/III users) and rewriting it.

Donald Gloistein had a problem with the PPADSD/FIX as listed in NOTES, Issue IV. I had previously verified that the patch worked as advertised. Since he was still having a problem with it, I wanted to double check everything.

It turns out that when I tested the results of applying the patch, I used the fix file that was issued on the DISK NOTES 4 diskette. I again used this fix with a copy of the release PRO-PaDS file and it again worked. After comparing the DISK NOTES 4 fix with that printed in NOTES, I recognized the problem. I don't for the life of me know how the NOTES listing could have been wrong.

Take another look at the listing on page 4-63. The second patch line starts at X'2D75' and has 16 bytes of patch. However, the next patch line starts at X'2D86'!!! This is certainly incorrect. It should be X'2D75'. That should clear up the problem he was having. To confirm the patch, I am listing it again in this issue's PATCH section.

Here's a caution for PaDS users. The caution deals with recovery from a disk full condition during a PDS(APPEND) operation. If PaDS can be absolutely sure that the error was a valid disk full, then an attempt could have been made to correct the EOF of the PaDS file during a PDS(APPEND) operation. When I was designing that module, my feeling was that the disk full error out of all the possible errors that could arise, should not be treated as a special case. To do so could jeopardize the integrity of the file without absolute certainty of the extent of the FCB corruption.

True, it is a simple matter for one to check a PaDS ISAM directory and infer the location of the EOF. A fixup could then be done to the system directory. I would rather have that a separate facility for the skilled user. For my personal taste, since the disk full probably didn't corrupt any other portion of the PaDS file (if it did, you see, then changing the EOF would not accomplish what would have been intended), it is also an easy matter to PDS(COPY) the valid members over to another file. That is the recommended solution at this time.

A. J. Hagers of THE NETHERLANDS supplied me with a cute little program to invoke a DIRectory command for selected date ranges. The program is designed to be invoked from a PaDS. The variation of the range selection is selected by a different entry point. Therefore, the utility is appended to a PaDS via the MAP option and has a number of entry points identified on the map data line.

As supplied by AJ, the program was Model I specific. I added code to support both the Model I and Model III. I also added conditional code for use with a conditional assembler such as EDAS/PRO-CREATE or MRAS to assemble a Model 4 version. I took advantage of the "P1" parameter in EDAS/MRAS so as to assemble a Model 4 version directly by a command such as:

```
MAS DAYS +O=DAY
MRAS DAYS +O=DAYS -GC
```

A model I/III version can be assembled with a command line such as:

```
MAS DAYS +O=DAYS (P1)
MRAS DAYS +O=DAYS -GC (P1)
```

In order to be able to append the program to a PaDS file, you would need a MAP file entry such as this:

```
DAYS,TODAY,5200,DAY3,520A,WEEK,520D,FORT,52
10,MONTH,5213,QUARTER,5216
```

Now since the Model I/III MAP file line is limited to 63 characters in length, you Model I/III folks will have to trim back some of the member names in the above MAP line to avoid exceeding that 63-character limit. Assuming you created a separate PaDS file of 6 members named DDIR, you could request today's directory with:

```
DDIR(TODAY) :2
```

A printed directory of all /CMD files of the last three days would be invoked with a command such as:

```
DDIR(DAY3) /CMD (PRINT)
```

Any parameters you enter on the "DDIR" command line will be passed to the normal DIR command generated by the DAYS program. Here's the program in its final form assembled for the Model 4.

```
                00001 ; DAYS routine for usage in a PDS library
                00002 ; A.J. Hagers programmed, Rotterdam 4th January 1986
                00003 ; usage could be    DDIR (week) :0 (i)
                00004 ;
                00005 ;       (today)         directory for today
                00006 ;       (day3)          directory of last 3 days
                00007 ;       (week)          idem week
                00008 ;       (fort)          idem fortnight
                00009 ;       (month)         idem month
                00010 ;       (quarter)       idem quarter
                00011 ;
4044            00012 DATE$1  EQU     4044H
421A            00013 DATE$3  EQU     421AH
0018            00014 @CMNDI  EQU     24
4405            00015 @CMNDI1 EQU     4405H
4299            00016 @CMNDI3 EQU     4299H
0012            00017 @DATE   EQU     18
4470            00018 @DATE1  EQU     4470H
3033            00019 @DATE3  EQU     3033H
                00020 ;
5200            00021         ORG     5200H
5200  0600      00022 TODAY   LD      B,0
5202  11220D    00023         LD      DE,0D22H        ;Cover up the "-"
5205  ED530053  00024         LD      (DPOS+8),DE     .
5209  11        00025         DB      11H             ;Ignore next LD B,n
520A  0603      00026 DAY3    LD      B,3             ; set counter
```

```
520C  11        00027           DB      11H             ;Ignore next LD B,n
                00028 ;
520D  0607      00029 WEEK      LD      B,7             ; idem
520F  11        00030           DB      11H             ;Ignore next LD B,n
                00031 ;
5210  060E      00032 FORT      LD      B,2*7           ; idem
5212  11        00033           DB      11H             ;Ignore next LD B,n
                00034 ;
5213  061C      00035 MONTH     LD      B,4*7
5215  11        00036           DB      11H             ;Ignore next LD B,n
                00037 ;
5216  065B      00038 QUART     LD      B,13*7
                00039 ;
5218  E5        00040 DAYS      PUSH    HL              ; save hl
5219  C5        00041           PUSH    BC              ;B needs to be saved
                00042           IF      @@1             ;Check on LDOS 5.x
                00043 CMDLEN    EQU     63
                00044           LD      A,(125H)        ;Model check
                00045           CP      'I'             ;'I' = Model III
                00046           LD      HL,DATE$3
                00047           JR      Z,IS3
                00048           LD      HL,@CMNDI1      ;Correct @CMNDI
                00049           LD      (CMNDI+1),HL
                00050           LD      HL,@DATE1       ;Correct @DATE
                00051           LD      (DATE),HL
                00052           LD      HL,DATE$1       ;Correct DATE$
                00053           LD      (DATEA+2),HL
                00054           LD      (DATEB+1),HL
                00055           ELSE
004F            00056 CMDLEN    EQU     79
521A  21F852    00057           LD      HL,DPOS         ;Find memory location
521D  3E12      00058           LD      A,@DATE         ;  of date storage
521F  EF        00059           RST     40
5220  EB        00060           EX      DE,HL           ;DATE$ => HL
5221  223252    00061           LD      (DATEA+2),HL
5224  226C52    00062           LD      (DATEB+1),HL
                00063           ENDIF
5227  110353    00064 IS3       LD      DE,TEMP_H       ;Save current system date
522A  010300    00065           LD      BC,3
522D  EDB0      00066           LDIR
522F  C1        00067           POP     BC
5230  DD211A42  00068 DATEA     LD      IX,DATE$3       ; set on system date$
5234  DD7E00    00069           LD      A,(IX+0)        ;If year is leap, then
5237  E603      00070           AND     3               ;  set Feb in table to 29
5239  2005      00071           JR      NZ,NOLEAP
523B  3E1D      00072           LD      A,29
523D  320753    00073           LD      (MNTAB+1),A
5240  AF        00074 NOLEAP    XOR     A               ; zero A
5241  57        00075           LD      D,A             ; and D
5242  B8        00076           CP      B               ; B=0 for a today
5243  281D      00077           JR      Z,NO_CNT        ; which has no countdown
5245  DD3501    00078 LOOP      DEC     (IX+1)          ; day:=day-1
5248  2016      00079           JR      NZ,NEXT         ; ?=0
524A  210553    00080           LD      HL,MNTAB-1      ; set HL on base of month
524D  DD5E02    00081           LD      E,(IX+2)        ; fetch current month
5250  1D        00082           DEC     E               ; mn:=mn-1
5251  2005      00083           JR      NZ,NOYEAR       ; not a new-year
```

```
5253  DD3500    00084          DEC     (IX+0)          ; year:=year-1
5256  1E0C      00085          LD      E,12            ; new month
5258  19        00086 NOYEAR   ADD     HL,DE           ; HL:= position in table
5259  7E        00087          LD      A,(HL)          ; fetch number of days
525A  DD7701    00088          LD      (IX+1),A        ; and place in date$
525D  DD7302    00089          LD      (IX+2),E        ;  idem
5260  10E3      00090 NEXT     DJNZ    LOOP            ; loop for number days
5262  21F852    00091 NO_CNT   LD      HL,DPOS         ; HL:= string space
                00092          IF      @@1
                00093 DATE     CALL    @DATE3          ; use system routine
                00094          ELSE
5265  3E12      00095          LD      A,@DATE
5267  EF        00096          RST     40
                00097          ENDIF
5268  210353    00098          LD      HL,TEMP_H       ; restore date$
526B  111A42    00099 DATEB    LD      DE,DATE$3
526E  010300    00100          LD      BC,3
5271  EDB0      00101          LDIR
5273  E1        00102          POP     HL              ; and HL
5274  11A552    00103          LD      DE,MOVE_IN      ; start pos in cmd$
5277  7E        00104 SCAN_MV  LD      A,(HL)          ; byte from cmd buffer
5278  FE0D      00105          CP      0DH             ; ?= CR
527A  2815      00106          JR      Z,ENDFND        ; end of cmd found
527C  FE29      00107          CP      ')'             ; ?= )
527E  2811      00108          JR      Z,ENDFND        ; considered an end o l
5280  FE28      00109          CP      '('             ; ?= (
5282  2804      00110          JR      Z,OPEN          ; para's in cmd
5284  EDA0      00111          LDI                     ; copy buffer into cmd$
5286  18EF      00112          JR      SCAN_MV         ; loop around
5288  EDA0      00113 OPEN     LDI                     ; copy buf to cmd$
528A  3E2C      00114          LD      A,','           ; place komma in cmd$
528C  32F452    00115          LD      (DTEXT),A       ; instead of (
528F  18E6      00116          JR      SCAN_MV         ; get into loop again
5291  21F452    00117 ENDFND   LD      HL,DTEXT        ; date para to be moved
5294  010F00    00118          LD      BC,DTLEN        ; place, BC:= length
5297  EDB0      00119          LDIR                    ; action word
5299  21A152    00120          LD      HL,CMDTXT       ; HL on completed command
                00121          IF      @@1
                00122 CMNDI    JP      @CMNDI3         ; and execute it
                00123          ELSE
529C  3E18      00124          LD      A,@CMNDI
529E  C32800    00125          JP      40
                00126          ENDIF
52A1  44        00127 CMDTXT   DB      'DIR '          ; first part of cmd$
      49 52 20
52A5            00128 MOVE_IN  DS      CMDLEN          ; worst case needed
52F4  28        00129 DTEXT    DB      '(d="'          ; part of date parameter
      64 3D 22
52F8  6D        00130 DPOS     DB      'mm/dd/yy-"',13 ; filled in by dos @date
      6D 2F 64 64 2F 79 79 2D
      22 0D
000F            00131 DTLEN    EQU     $-DTEXT
5303            00132 TEMP_H   DS      3               ; temp hold
5306  1F        00133 MNTAB    DB      31,28,31,30,31,30,31,31,30,31,30,31
      1C 1F 1E 1F 1E 1F 1F 1E
      1F 1E 1F
5210            00134          END     FORT
```

LIBRARY files for EDAS by Robert M Connors

The following procedure creates a series of PaDS library files which can be *SEARCHed from EDAS under LDOS 5.1.x using a multi-drive system. Instead of maintaining two separate files, EQUATE1/EQU and EQUATE3/EQU, this procedure will produce one /LIB file which contains addresses that are identical on the Model I and III, and two other /LIB files which contain EQUates for unique Model I/III addresses.

First, using EDAS, create a list of equates (e.g., @DSPLY EQU 4467H) for all addresses that are the same on the Model I and III (use the alphabetic memory map in the LDOS manual). The EQUATE1/EQU or EQUATE3/EQU file on your master LDOS diskette can be used to do this. Save this list as a file entitled 'LDOSEQU/ASM', or something similar. Next, create a separate file for the Model I and Model III which will contain unique addresses for each. They could be entitled something like LDOS1/ASM and LDOS3/ASM. These files may contain special symbols ('@', '$') as part of the label names, just as they appear in the Technical Information section of the LDOS manual. Then format a new data diskette and put it in drive 1. Load LBASIC and run the following program (Note: the spaces that are included must remain for the Model 4/4P but may be removed for Model I/III LBASIC):

```
100 CLS:CLEAR 5000:ON ERROR GOTO 360:'Delete '5000' on Model 4/4P
120 OPEN "I",1,"LDOS1/ASM:0"
140 IF LOC(1)=LOF(1) THEN CLOSE:END ELSE LINE INPUT#1,REC$
160 FT=INSTR(REC$,CHR$(9))-1:ST=FT+2
180 TT=INSTR(REC$,CHR$(9)):TT=LEN(REC$)-TT
200 F$=LEFT$(REC$,INSTR(REC$,CHR$(9))-1)+"ASM:1"
220 IF LEFT$(F$,1)="@" THEN MID$(F$,1,1)="A"
240 D=INSTR(F$,"$"):IF D<>0 THEN MID$(F$,D,1)="Z"
260 OPEN "O",2,F$
280 PRINT @0,CHR$(30);"Writing file ";F$;
300 PRINT @64,"Record = ";CHR$(30);LEFT$(REC$,FT);
    " ";MID$(REC$,ST,3);" ";RIGHT$(REC$,TT);
320 PRINT 2,REC$
340 CLOSE 2:GOTO 140
360 IF ERR<>122 AND ERR<>134 THEN CLOSE:END
380 CLOSE:KILL F$
400 CLS:PRINT @0,"Disk full!!  ";F$; " is the next file to create"
420 PRINT "Re-run the program when ready."
440 END
```

The file name in line 120 should match the name given the EQUate file created as described above. The program will open the file, and extract each line, creating a file for it using the EQUate label as the filespec. If the label contains an '@' or '$', they are changed to 'A' and 'Z' respectively in the filespec (not the label). Both the filespec, and EQUate line will be displayed on the video. The new file is closed, the next EQUate line read from the input file, and the process repeats itself until the end of file is reached. If your diskette in drive #1 becomes full before the program has completed processing (only people with 35 track drives will have that problem), error trapping is provided to allow insertion of a newly formatted data diskette in drive #1 so that processing can be completed. Of course, you'll have to delete all the previously processed lines from the /MAP file before continuing -- that's why the program tells you which one to start with again.

Once all the files are created, you may KILL the /ASM file which you just processed. Then, using PARMDIR or FM, create a PaDS /MAP file of all the files created by the BASIC program. You'll have to create two /MAP files if you used two diskettes. If PARMDIR or FM are not available, EDAS, a word processor, or even the LDOS 'BUILD' comand could be used to create the /MAP file. Regardless, the /MAP

file should not contain line numbers. Create a PaDS /LIB file using the PaDS(BUILD) function being sure to allocate enough members to cover the entries in your /MAP file(s). Next, using the PaDS(APPEND) function with the '(MAP)' parameter, append the files to your /LIB file.

Finally using FED, look at the /LIB file's PaDS directory entries and change all the entries starting with 'A' to '@' (note: LDOS has no labels starting with 'A'), and all entries ending with 'Z' back to '$' (note: LDOS has no labels ending with 'Z'). Use the arrow keys to move from record to record. Be sure you save each page (FED's 'S' command) before moving to the next page or the changes will not take effect. If you don't have FED, get a hex dump of the file's PaDS directory on your printer, and use the LDOS 'PATCH' utility to make the changes directly to the file. Note: PaDS directory entries containing special symbols such as '@' and '$' cannot be copied or listed using standard PaDS functions. For a way around that, see 'NOTES FROM MISOSYS' Number IV, page 61-62.

Repeat the above procedure for each /LIB file you have created. When you are done, you no longer need to *GET the equate files and clutter your assembly and cross reference listings with a lot of unused symbolic labels. Nor do you need to include your own equate labels for LDOS entry points in your program. Instead, just *SEARCH LDOSEQU/LIB for each program (LDOS 5.1.x DOS's only), and conditionally *SEARCH LDOS1/LIB or LDOS3/LIB depending on which machine you are writing code for. You will then get an EQUate listing for only those labels referenced in the program, thereby saving disk space, paper, and assembly time. Also, the cross reference listing and/or symbol tabels will be correspondingly small.

The above BASIC program could also be used to create an LDOS4/LIB file which would contain the equates for all the DOS6 SVC functions (e.g., @DSPLY EQU 0AH), which I have done. I also have a file for the MAX-80 for those addresses shown in the MAX-80 Programmer's Manual not used by LDOS. This means that I can, by setting the 'Pn' values when invoking EDAS, or by setting the '@@1' thru the '@@4' labels in my source listing, assemble a file for any of these computers with a minimal amount of re-write of routines. It is a great time saver.

Product Highlights: PARMDIR

Before NOTES Issue III was printed, RMC wanted to know why PARMDIR didn't suppress the "notes" generation when invoked using the MAP parameter since the PaDS MAP file doesn't suport a "dot" comment line. He was the first person commenting about the 'comments' of PARMDIR when using the MAP parm. Because that was a most-reasonable request, I worked up a patch that was to have appeared in the next issue of NOTES. I never gave it an official patch name and I don't know why the patch never made it into NOTES, Issue III. The patch is designated PARMDIRx/FIX and pertains to the Model I/III version. It's printed in the PATCH section of this issue.

Product Highlights: PRO-ESP

Frank A. Yacucci wanted to make some changes to the way our ALTDISK and PRO-SAID programs dealt with command line parameters. For ALTDISK, he wanted the DRIVE to default to something other than 7 and the HALF option to default ON instead of OFF. First, the changes to ALTDISK relate strictly to parameter behavior. So lets take a look at that. You should have a copy of either THE PROGRAMMER'S GUIDE TO TRSDOS 6 or Tandy's Technical Reference manual. Knowing the concept of the parameter scanner SVC, you would use FED to look at the ALTDISK file. The parameter table is found at address X'3525'; the table is an LDOS 5 type table. You see the DRIVE parameter entry. The parameter VECTOR has a value of X'3000'; thus, use the FIND command of FED to locate the address. At X'3000', you will find the two bytes, 07 00. You should know that parameter values are two bytes stored low-high. The '07' is the default of DRIVE=7. Thus a patch of the order, (D00,54=04:F00,54=07), changes the 7 to your desired default (I illustrated '04'

for DRIVE=4). The "D00,54" is determined by FED to be the position for the patch.

Back to the parameter table and you will find that the HALF parameter VECTOR points to X'3002'. At that address you will find the value, 00 00. You should learn from THE GUIDE that OFF is represented by X'0000' while ON is represented by X'FFFF'. Thus, the patch for HALF to default to ON is:

(D00,56=FFFF:F00/56=0000).

Product Highlights: ZCAT

Through the medium of our Compuserve Special Interest Group (PCS49), Ron Ungashick 70360,100 reported both a problem and solution. He advised, " Roy, I have found what I believe is a minor problem with PRO-ZCAT. When you have more than 68 disks cataloged and you "<L>ist disks on file" the first screen of 68 disks are displayed correctly. When you press <ENTER> to display the second screen, the first line of disks is not cleared. The new disks are displayed in the lines below it. The code to clear the disk display is as follows.

```
3B70    LD  HL,0500H
3B73    LD  B,03H
3B75    LD  A,0FH
3B77    RST 28H
3B78    LD  C,1FH
3B7A    LD  A,02H
3B7C    RST 28H
```

I have a patch to correct the problem as follows.

```
. Patch PRO-ZCAT Disk Display
X'3B72'=04
```

If you have time, perhaps you could verify this and include it in THE MISOSYS QUARTERLY. Thanks, -Ron". Well Ron was right. For those of you who want a direct patch, it's:

PATCH ZCAT (D0B,46=04:F0B,46=05)

Product Highlights: ZSHELL

Here's some tips on using our ZSHELL facility. First, if you want to add redirection to the JCL file resulting from an application of the WC utility, all you need to do is prefix the WC command line with a double quote mark and add the redirection specification where you want it to appear in the WC parameter as in:

"WC LIST */TXT:1 >>TEMP/TXT:5

It pays to read the documentation thoroughly. That happens to be a good use for the double quote which inhibits ZSHELL from scanning the current command line for redirection specifications.

Second, there is a little problem in redirecting SCRIPSIT's printer output. Here is the problem. Model 4 SCRIPSIT does indeed output a zero when first performing the PRINT operation; however, it is not output via the @PRT supervisor call but rather through @CTL referencing the *PR device control block. If you ROUTE *PR filespec at the system level, this redirects the device handling to the DOS's character I/O routines. The disk "output" handler traps @CTL calls and inhibits them from being put to the disk file. That's why you won't see a 00 byte in the ROUTEd disk file. Unfortunately, ZSHELL makes no distinction between a @CTL_Ø and a @PUT_Ø. Thus, when you invoke SCRIPSIT with ZSHELL's redirection, the hex Ø put out by SCRIPSIT at the start of printing winds up in the disk file.

This sounds like a useful modification to ZSHELL - that of inhibiting @CTL calls - but only if the output device is a disk file! Some day I'll have to look at that modification. By the way, the reason that SCRIPSIT outputs a @CTL_Ø is for the purpose of testing the availability of the printer. Until we decide on a permanent solution to this problem, be on guard.

PRO-WAM: A new name for PRO-NTO

In early 1986, MISOSYS received a letter from James J. Foster, attorney for Chemical Bank. It advised us that we were infringing on Chemical Bank's "Pronto" trademark. The normal "cease and desist" language was conveyed. Our attorney investigated the purported claims and advised us as to the position we should take. We then addressed the following to Chemical Bank's attorey.

"After reviewing the documents supplied by you and the documents associated with the announcement of our PRO-NTO product in December of 1984, our corporate attorney has advised us that we "should change the designation for our software." Counsel further added that, 'Some of the ads [MISOSYS] provided ... clearly would be confusing when compared to the use of those words by the Chemical Bank.' Thus, I have been advised by counsel to write you indicating that we will take the necessary steps to cease using the PRO-NTO (or PRONTO) designation for our software.

In light of these findings, I have instructed our attorney to perform a trademark search for a new name that has been selected for our product. As soon as we can determine that the selected name does not infringe on any other company, we will proceed to alter our advertising and other product information concerning our windowing and application manager product.

Since advertising leadtimes are in excess of two months, and a trademark search may take some time to accomplish, I have instructed our advertising department to incorporate the following notation in our PRO-NTO ad as soon as possible. "Coming soon: a new name for PRO-NTO - to avoid confusion with Chemical Bank's Pronto electronic banking system".

We apologize for any inconvenience our use of Chemical's trademark may have caused. However, at the time we selected the PRO-NTO name for our product in 1984, Chemical's electronic banking system was certainly not an item known by us to be a computer software program."

We then directed our trademark attorney to investigate other names which we could use for our "PRO-NTO" product. We selected "PRO-TON", "WAM!", "PRO-DIGY", and "PRO-MISE". It turns out that all of those were no good. There is a Proton corporation using that term, "WAM" and slight variations are trademarked as well as a number of pending applications, "PROMIS" is trademarked for computer programs as well as a pending application for "PROMISE", "PRODIGY" is already trademarked for microcomputer systems. We wound up spending about $500 to exhaust all of our reasonable choices. Thus, we settled in on "PRO-WAM". The "WAM" part comes from "Window and Application Manager". The PRO indicates a TRSDOS 6 product. You will soon see this term used in our advertising. Don't worry, it's still the same dependable PRO-NTO product (not to be confused with electronic banking).

Gregory Cleland writes of PRO-NTO, "I've had my new copy of PRO-NTO for a couple of days now and I've got to say that this is the best price/performance package that I've ever bought. As usual for MISOSYS products, the documentation was excellent and the software stunning. John Harrell's review in 80 Micro is right on target. PRO-NTO is a five star product.

I have only two questions, both regarding DIALER4P/FIX. Should the '@' macro really be "C**M", or do you mean "CC*FM"? Secondly, is there a cleaner way to force the 4P modem to hangup after connecting -- for voice communications -- other than LIBEXEC SETCOM (DTR=N)? This method requires one to reissue the SETCOM (DTR=Y) command prior to dialing out again."

Well the answer to both of Greg's questions are easy. We recommended "C**M" as the "@" macro string to clear the 4P modem and bring it to known conditions. Since the DIALER's output doesn't examine the response back from the modem, the second asterisk is used to ensure that at least one of them is seen by the modem. The "M" puts the modem into automatic mode. Of course, if you want to FAST DIAL, add the "F". The answer to question two can be found further down in a letter from Jack Savitz.

William E Connors writes concerning a bug he experienced with MED/APP - one of the applications in the Mister ED application pac, "I received your letter with the fix for my MED/APP problem. Thank you for your very quick response. Glitches do plague us all from time to time, and are often caused by our efforts at enhancement. I think my Grandmother recognized that since I can remember her saying, 'The enemy of best is better.'

I performed the indicated operations contained in your letter and I'm happy to say that MED's "A" & "H" commands now perform flawlessly. I am delighted! The function I had in mind for MR ED's MED/APP is the modification of LESCRIPT's KSM file, while resident in RAM. It works as expected, and makes LESCRIPT even better suited to my needs."

RJE asked whether PRO-NTO will run under MAXDOS 6. Be advised that Karl Hessinger, author of PRO-NTO, used a MAX-80 under MAXDOS 6 for development. Thus, PRO-NTO will certainly run under MAXDOS.

Here's an easy way to help isolate random problems while using PRO-NTO. It was directed to a user experiencing difficulties. Investigate the possibility of a problem in your external memory. Don't forget, PRO-NTO makes extensive use of alternate RAM. Perhaps you should run your machine for a few days without PRO-NTO installed but with a full memDISK installed. Write a large test file of known data to the memDISK to fill it up. Then examine the file's contents periodically to detect if there were any changes. For an intermittant problem, I would first suspect a RAM problem.

John Foote wanted to change the printing format of the ADDRESS application's Rolodex-type card layout to use continuous form 3x5 cards (like in CARD). His query concerned the "pitch" of the ADDRESS card printing and related to the "Rolodex" card layout of 13 lines versus the 3x5 card layout of 18 lines at 6 lines per inch. In response to his letter, I advised him of the following.

There are two choices. First, when you invoke the "card" command of ADDRESS, you have the option of keying in your own print format. This is documented in the manual on pages 2-3/2-4. What you need is a print format that uses 18 semicolons; the default format uses 13 semicolons to be able to print 13 lines. You must construct a format with that in mind. That gives you total flexibility. Depending on how frequently you print out your list, you may consider keeping the format in a record of the CARD application. Thus, when you want to print out an address list to 3x5 cards, when the prompt appears for "format", invoke CARD, search for the format record, then EXPORT the format string to ADDRESS.

You could still patch ADDRESS. Since the PATCH command does not like semicolons within a patch string, and the patch listed in hexadecimal would appear confusing, I prefer to tell you where in the file to alter the DEFAULT string. This would require that you have some sort of file editor (like our LS-FED II, or our Mister ED Editor Application pack). Alternatively, you could design your new format string and convert both the default string and your new string to their hexadecimal representation. The default string is:

```
";;0 1  9;2   8;3  4;5 6 7;$;;;;;;;"
```

This string appears at relative record 8 with byte offset 48H. This means a patch origin of "D08,48=" if you choose to work up a patch. To easily come up with a new format string, you could remove 4 spaces and the "$" from the default string to give you room to add the 5 additional semicolons. The string is terminated with a carriage return not shown above since it is already in the file following the last semicolon.

The BRINGUP application of PRO-NTO supports a date range of that currently supported by the DOS. When TRSDOS 6.x is upgraded to extend beyond December 31st, 1987, MISOSYS will shortly thereafter have

available a new release of PRO-NTO which deals with the extended date. Registered users will be notified at that time as to the procedures and costs of updating.

On 05/12/86, We added, to the PRO-NTO release disk, Lynn Sherman's modification to SuperScripsit for use with PRO-NTO. See the file SSFIX/TXT for the details. The patches are contained in the file, PRONTOSS/JCL.

On 05/12/86, we also upgraded PSORT to version 2.1. This permits you to sort a PowerMAIL adder file using PSORT. If you need to upgrade to this version, the charge is $5 ($6 if outside the United States). Please return your PRO-NTO master disk in a protective disk mailer with the required fee.

In order to utilize PSORT 2.1 for the ZIP-code ordering of a POWERMAIL adder file, you need to apply the following patch to your PMAIL/ADD disk file: PATCH PMAIL/ADD (D00,10=80 00 02 00 62 00 0A:F00,10=00 00 0 0 00 00 00 00). If you understand the nomenclature listed at the bottom of page 2-9 of your PRO-NTO manual, you will be able to redo the patch to sort on any field of your choosing. Note also that you can turn your ADDRESS/DAT data file into a PowerMAIL adder file by applying the following patch to ADDRESS/DAT:

```
D00,30=" PowerMAIL PLUS "
F00,30=00 00 00 00 00 00 00 00
       00 00 00 00 00 00 00 00
```

One PowerMAIL/PRO-NTO user wanted to sort his PMAIL/ADD file by ZIP code. Here are some detailed instructions for this scenario. It requires altering the adder file so that the second line contains the environment bytes needed by PSORT.

If you list the PMAIL/ADD file in hex using a command such as "LIST PMAIL/ADD (HEX)" and pause it after the first four or so lines have been listed, the second line needs to be altered so that it will look like the second line shown here identified by "<10>":

```
  0123456789ABCDEF          00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
================================================================================
| ..P.............. | <00> | 00 00 50 00 00 00 00 00 00 00 00 00 00 00 00 00 |
| ....b............ | <10> | 80 00 02 00 62 00 0A 00 00 00 00 00 00 00 00 00 |
| disknamediskname | <20> | 64 69 73 6B 6E 61 6D 65 64 69 73 6B 6E 61 6D 65 |
|  PowerMAIL PLUS  | <30> | 20 50 6F 77 65 72 4D 41 49 4C 20 50 4C 55 53 20 |
|                  | <40> | 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 |
```

Note that the first seven values are "80 00 02 00 62 00 0A" which are what is needed. This is what your patch needs to accomplish. The easiest way to zap in those values would be via a file editor (such as LS-FED-II or the Mister ED package available from us); edit the file so as to make the second line look like that shown above. It will enable you to edit any of your adder files as well as all other files, easily. Note that the normal ADDRESS/DAT file can also be so modified to enable it to be sorted into ZIP code order by PSORT 2.1.

We had a report of a problem with CHARSET not printing. The "problem" wasn't really a bug in CHARSET but was caused by the use of a line-buffered printer. Since most printers these days are line-buffered, it pays to put everyone on guard over this condition. The user had an MX-100 connected to the computer. The MX-100 is a line buffered printer and will not print out a line until either (1) a print action code is received, or (2) the line buffer is full. Thus, if you output characters to that printer, you won't see anything print. If you repeatedly depress the "Print" command until printing starts, you are actually outputting the CHARSET "string" once for each depression. Since the MX-100 print buffer probably has 132 or so characters, it did not start printing until that gets full. To see this behaviour, add one of the print action codes to the print string in CHARSET. For instance, move the cursor to the X'0A' or X'0D' values. Although the string will be loaded with what appears to be a printable character (at least displayable on the

screen), it will be interpreted as a LF or CR by your printer.

Jack Savitz wrote us how he solved the "disconnect" problem of the Modem-4P when used with DIALER/APP. He sets up his "@" macro to be "C**MF" do do fast dialing. He sets up the "A" macro with 16 "Ps". He then sets the "O" macro to a "T" to be able to output tone dialing. Now a dialing string such as "05165551212A" now lets him hit the "*" and disconnect at leisure which is a lot better than trying to hit it immediately after the last dial tone.

I sometimes wonder whether there are many different kinds of 4P modems. John Yanosky writes, "I tried and was successful in applying the patches included in DIALER4P/FIX. However, I was unable to get what I think is proper operation using the "@" macro, "C**M". In its place, I used the Radio Shack Model 4 DESKMATE computer dialing definition: "**C**MG@^DDT" and had success."

Here's another solution to the 4P disconnect problem that I worked up for John Fadok. He asked for a PATCH to have the PRO-NTO DIALER application automatically send a series of P's followed by an asterisk. The following patch should do this:

```
. Patch to DIALER for Model 4P to
. automatically send a string of P's
. at the end of the dialing line to
. automatically disconnect the modem.
. Note that this is usable only for
. telephone communication.
D04,50=06 08 3E 50 CD 83 2B 10
       F9 3E 2A C3 83 2B
F04,50=CD 81 2B 21 00 06 11 6C
       2F CD 0E 2E 01 00
. Eop
```

In the first line of the patch, the "08" is the repetition counter. It will cause DIALER to send 8 P's then the asterisk. You can change that to some other hexadecimal value (i.e. 06 for 6, 0A for 10, etc.) to have whatever count you want.

The following are brief descriptions of public domain applications which run under the PRO-NTO/PRO-WAM window controller. They are available in the DL3 section of our Compuserve User Group. We heartily thank the contributors of these software packages.

MAPMEM.APP [70130,441] 08-Jul-86 3935(1696) 28 -> A PRO-NTO application to display contents of selected low/high routines. Direct comments to Don Brandt 70130,441. Version 3.5 computes machine size and expands stack area.

MODEM.APP [70130,441] 07-Jul-86 5860(2528) 28 -> PRO-NTO application that supports xmodem file transfers. Requires that *CL be set to COM/DVR. Direct comments to Don Brandt 70130,441.

SVCAPP.NEW [70265,113] 01-Jul-86 2015(864) 34 -> Here is a new version of SVC/APP. By popular demand, it includes a Search by SVC Number ability as well as the ability to use the original data file without any special LRL. This version is from the author of SVC.APP, Doug Tittle, who would be glad to assist and/or improve as needs dictate. Drop me a line. Enjoy! Doug.

MXDUMP.APP [72436,3604] 28-May-86 2610(1120) 26 -> A PRO-NTO application to screen dump the Model 4 screen with lo-res graphic characters. Rename to MXDUMP/APP.

SSPRN2.FIX [72057,55] 07-Mar-86 3360 27 -> This is an update of SSPRON.FIX. These patches will allow somewhat smoother access to PRO-NTO from SS if you change your PRO-NTO activation code to one of the invalid control codes in SS that normally produce an error.

LIST.APP [76317,2576] 28-Dec-85 5545(2368) 63 -> File lister module for PRO-NTO.

DM.BIN [71426,625] 19-Oct-85 3905(1696) 51 -> PRO-NTO application. Download and rename DM/APP. Condensed version of DMAP.BIN. Shows where programs reside on your disk. Works on 40 track single sided double density disks only. Requires PRO-NTO to operate. -Dick Newman.

OLYMPI.APP [70406,1107] 22-Jun-85 835 66 -> This is an updated version of a PRO-NTO

window application for your Olympia NP (or Epson) printer. If anyone would like a version customed to their printer just leave a message and I'll see what I can do! This version fixed a small initialization problem that affected the Pronto Application Window.

OKIDAT.BIN [72145,640] 04-May-85
2930(1280) 21 -> Okidata printer setter. Is an application program designed to run with PRO-NTO. Rename to Okidata.app.

The following PRO-NTO application has been submitted by Bryan Headley. "Here is a diskette, with a submission for your upcoming MISOSYS QUARTERLY. Hopefully, it will arrive in time! Moreso, in that in preparing this diskette for you, I decided to create a new version of the software. Everything as of 10 P.M. this evening is going great [July 23rd -ed]; the new version was re-assembled and tested, documentation revised, and this letter is being written. A close shave!

I decided not to place my submission into the public domain. Instead, I own the copyright, and have granted users the right to reproduce for personal, non-profit use. I find this a preferable resolution, in that all parties come out winners. Both the documentation and source code state this.

This might be used in your introduction to the submission:

"WinCalc is a programmer's calculator. It provides all the mathematical and logical operations a Z-80 programmer may require during the development and maintenence of assembly-language programs."

Actually, I find WinCalc a blessing. I've been stuck too many times, trying to remember what 65 OR 20H is, or what 14312 is in Hex. After the millionth time, I decided to do something about it. Originally, the program was to merely support addition, subtraction, AND, OR, NOT, and the rotate commands. But, then came the shift commands, Octal and ASCII modes, the flag commands, etc... All in all, the code is dense! I looked, and there's ONLY ONE BYTE stuck in the end of the code as the filler (to make the code end at x'3000')!!

BinCalc/WinCalc


```
+-------------------------------------------------+
| +==========={ Bryan's Binary Calculator }=======+ |
| ||    0            N Sign N Zero N Junk N BCD  || |
| ||Dec  8 Bit (- .) N Junk N P/O  N Subt N Carry|| |
| +=={ MODES }========{ BITWISE }==={ ROTATE/SHIFT }=+ |
| [^B]in [^D]ec  [&] AND [|] OR   [L] RLCA [S]  SLA |
| He[^X] [^O]ct  [^] XOR [!] NEG  [l] RLA  [s]  SRA |
| [^A]SCII       [~] CPL [`] DAA  [R] RRCA [^R] SRL |
| --{ MATH }------{ COMMANDS }---- [r] RRA          |
| [+] Addition   [^C] All Clear   -{ WORD LENGTH }- |
| [-] Subtract   [^N] Clr. Entry  [^E]  8-Bit Entry |
| [=] Equals     [^L] Set Flags   [^S] 16-Bit Entry |
| [\] Set Carry  [/]  Cpl Carry                     |
+-------------------------------------------------+
```

Introduction

BinCalc is a programmer's calculator, oriented towards those users who develop or maintain Z-80 assembly language code. The calculator exists in two forms: the stand-alone BinCalc (BINCALC/CMD) and the PRO-NTO application WinCalc (WINCALC/APP). The PRO-NTO application implies that the calculator exists as a pop-up, similar in concept to the capacities associated with SideKick. **Capacities**

As BinCalc is a programmer's calculator, absolutely no effort has been expended in providing support for floating point arithmetic nor numbers beyond x'FFFF'. These functions are of little use to assembly language programmers.

BinCalc offers two operational modes: 8-bit and 16-bit. When in 8-bit, all logical and math operations are done on 8-bit quantities. This corrolates to working with a member of a register pair in Z-80. When in 16-bit mode, math is done on 16-bit quantities, akin to working with register pairs. Note that logic instructions are not valid in this mode; this corresponds to the capacity of the Z-80 processor.

Features

BinCalc allows the user to determine the resultant value from a mathematical or logical formula. In so doing, BinCalc maintains and reports the state of the Z-80 registers after each instruction. This is very useful when developing code, and trying to determine value range filtration/determination routines.

Entry Modes

BinCalc supports a number of numeric entry bases. These include Binary, Octal, Decimal, Hexidecimal, and Ascii. When in any except ASCII, data entry and system output are accepted/displayed in the selected base. When in ASCII mode, all keystrokes (1 for 8-bit, 2 for 16-bit modes) are accepted as input; the ASCII value of these keystrokes are taken as entry. Any and all keystrokes, including backspace, Enter, Break, etc., are accepted as entry. Upon completion of entry, the numeric base of the system automatically reverts to the previous setting.

Example: In decimal mode, ASCII mode is selected. The user presses "A". The system returns to decimal, and indicates that "65" is the value entered by the user.

```
+-------------------------------------------------+
| +==========( Bryan's Binary Calculator )========+ |
| ||    0            N Sign N Zero N Junk N BCD  || |
| ||Dec  8 Bit (- .) N Junk N P/O  N Subt N Carry|| |
| +=( MODES )=======( BITWISE )==( ROTATE/SHIFT )=+ |
+-------------------------------------------------+
```

Status Line

BinCalc uses the status line to display information to the user. The status line is used to display numeric entries as typed, the response from a calculation, the current numeric base, operand width, flag status, and the current number's representation in ASCII.

Entry of Formulae

BinCalc is an algebriac calculator; formulas to be calculated are entered in "normalized" fashion. For example, to add 2 to 5, the user enters the following:

2+5=

Note that the equation terminator is required. In the above example, the equal sign is used (=). The user may also opt to use the <Enter> key in this capacity.

Mixed-Mode Calculations

Circumstances arise where the user may wish to add a decimal number to a hexidecimal, seeing the result in binary. BinCalc has provisions for handling these situations. The four primary modes may be selected prior to the entry of the mathematical operator. For example, let us assume that the user is to add 2 decimal to 00000011 binary, and wishes to see the result in octal. Further, [^D] selects decimal mode, [^B] selects binary, and [^O] selects octal. The sequence (with the principals separated by spaces) then would be:

2 [^B] + 00000011 = [^O]

Setting Flags

BinCalc has the ability for the user to set up the initial state of the Z-80 flag register prior to a calculation. To initiate this, press [^L]. The current number in display is removed, to be replaced with the following:

```
+-------------------------------------------------+
| +==========( Bryan's Binary Calculator )========+ |
| ||SZ?H?PSC          N Sign N Zero N Junk N BCD  || |
| ||Flg  8 Bit (- .) N Junk N P/O  N Subt N Carry|| |
| +=( MODES )=======( BITWISE )==( ROTATE/SHIFT )=+ |
```

The 'S' signifies the Sign Flag, the 'Z' the Zero Flag, etc. The cursor is position at the current flag to be modified. The user may press the plus key (+) to signify that a flag indicated by the position of the cursor is to be set, or the minus key (-) to signify that the flag is to be reset. Pressing any other keystroke indicates that the current setting of the given flag is to be retained. All keystrokes are legitimate entries, including <Enter>, <Break>, etc.; there's no escaping Flag mode until the values of all 8 flags have been indicated.

In addition to explicitly setting flags, you may also use the Complement and Set Carry commands. These behave as per CCF and SCF respectively.

```
| +=( MODES )=======( BITWISE )==( ROTATE/SHIFT )=+ |
| [^B]in [^D]ec  [&] AND [|] OR   [L] RLCA [S]  SLA |
| He[^X] [^O]ct  [^] XOR [!] NEG  [l] RLA  [s]  SRA |
| [^A]SCII       [~] CPL [`] DAA  [R] RRCA [^R] SRL |
| --( MATH )------( COMMANDS )---  [r] RRA          |
| [+] Addition   [^C] All Clear   -( WORD LENGTH )- |
| [-] Subtract   [^N] Clr. Entry  [^E]  8-Bit Entry |
| [=] Equals     [^L] Set Flags   [^S] 16-Bit Entry |
| [\] Set Carry  [/]  Cpl Carry                     |
+---------------------------------------------------+
```

Unary Operators

Unary Operators take immediate effect upon the numeric entry currently being displayed. Unary Operators include RLCA, RLA, RRCA, RRA, SLA, SRA, SRL, NEG, CPL, and DAA. Of these, Unary operations may only be performed upon 8-bit values, with the exception of the DAA command. For example, the user wishes to rotate the binary value 01010101 2 times to the right (RRCA). The sequence used is:

[^B] 01010101 [R] [R]

Mathematic and Logical Operators

Mathematic and Logical Operators require two numeric entries. Mathematic operations allows for addition and subtraction on both 8 and 16-bit operands. Logical operations may only be performed on 8-bit operands, and include AND, OR, and XOR. Examples below show sample math and logical operations:

2 + 3 = [^B] 01010101 & 11110000 =

Erroneous Entries

Erroneous numeric entries often may be corrected by pressing the backspace key, so long as the user has not already entered the symbol associated with a mathematical, logical, or unary operator. In cases where an entire number has been entered (but for some reason is incorrect), the user may opt to clear the entry with the [^N] key.

If the user has entirely botched everything up, he may press the 'All Clear' button, [^C]. This key resets all formulae entered to date in the system. The user may then proceed to re-enter this.

Should an entry be erroneous (e.g. a key

sequence not understood by BinCalc), a short tone will sound.

Data Importation and Exportation

The Windowing BinCalc (WINCALC/APP), which operates under PRO-NTO, has the ability to import and export information. Numbers and formulae may be imported (read) from the screen of the "foreground" application and handled by BinCalc as though keystrokes from the keyboard. Results from calculations performed by BinCalc may also be returned to the foreground application.

To initiate importation, press <CLEAR> <LEFT-ARROW>. The BinCalc window disappears, and the "foreground" application's screen is returned to it's initial state. The cursor is found blinking in the upper left corner. Use the arrow keys to position the cursor to the area where the user wishes importation to begin. Press <^B>. An "[" character will mark this location. Now position the cursor to the position of the last character to be imported, and press <SHIFT> <ENTER>. All text, from the beginning marker to the location of where the cursor was when the <SHIFT> <ENTER> keys are pressed are read from the screen, and sent to BinCalc as though typed at the keyboard.

To initiate Exportation, press <CLEAR> <RIGHT-ARROW>. The cursor is positioned to the upper left corner of the BinCalc window. Using the arrow keys, position the cursor to the beginning of where the user wishes to begin exporting (writing) text to the "foreground" application. Usually this is the portion of the status line where an answer is stored. Press <^B>. An "[" marker will mark this location. Now position the cursor to the position of the last character to be exported, and press <SHIFT> <ENTER>. BinCalc terminates, exiting to the "foreground" application. The text to be exported is buffered as keyboard entry. The nature of the "foreground" application determines how the exported text is handled.

Exiting BinCalc

To exit BinCalc, press the <BREAK> key.

Capacities

1) Numeric Bases Supported:

Binary Octal Decimal Hexidecimal ASCII

2) Range of numeric entry:

0...0 0...0 ... 1...1 1...1	Binary
0 ... 177777	Octal
0 ... 65535	Decimal
0000 ... ffff	Hexidecimal
\0\0 .. \ff \ff	ASCII

3) Operators (16-bit numbers)

Addition, Subtraction, DAA

4) Operators (8-bit numbers)

Addition, Subtraction, AND, OR, XOR, CPL, NEG,DAA, NOT, RLCA, RRCA, RRA, RLA, SLA, SRA, SRL

Of Interest

The Z-80 supports two subtraction commands: SBC and SUB. SUB is only provided for 8-bit operations, while SBC is supported across the board. So, all subtraction commands are implemented via SBC. The decision to do this came from the author's determination that no intuitative command symbol could be thought of to differentiate a subtract from subtract with carry. Additionally, the user may opt to complement the carry flag prior to subtraction, thereby simulating SUB (one would assume that the user would complement the carry flag back!)

On the other hand, 8-bit and 16-bit additions are implemented through the ADD instruction. Again, no intuitative symbols for ADC were thought of. ADC is easily emulated by adding 1 to the result if the carry flag is set.

THE following is the WINCALC/APP application in BINHEX format. The /APP file and a documentlation file are available on the DISK NOTES 4 disk (see THE BLURB). If you feel like typing all of this in to a file, us the BINHEX command to convert it into a binary file named

WINCALC/APP. Thereafter, it should be available as a PRO-NTO application pop-up.

```
50524F4E544F42696E617279204361632003000000000000000000000000
000003000A000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000003037
2F32332F383632313A35323A3438436F707972696768742031393836204 2
7279616E20572E20486561646C657920210A0311330C06073E7CEF280606
053E68EFC921222606103E20772310FC3E037721000006033E7CEF21122D
060A3E7CEFC3452ACDC12B0100003E7CEF4FFE30380EFE3A382EE6DFFE41
3804FE4738247921712F012100EDA12814E26228232318F506053E68EF3A
FC2FB720CA18C55E2356EBE9F53AFC2FB7CCA82BF14F3ADB2FFE10792804
FE0030D83AFC2FFE0028D13C32FC2F792AFD2F772322FD2F06093E7CEF3A
FB2FF60132FB2FC337283AFC2FB7C44029AF32FC2F21002622FD2F3ADA2F
FE55473AFB2F2010E60128062AD42F22D62F3E0232FB2FC9FE01200A2AD4
2F22D62FF6021802F60132FB2F78FE4FC8B7CA34282AD82FE52AD62FED5B
D42FFE2B2004F119181FFE2D2005F1ED52181606A3FE26280806B3FE7C28
0206AB78322529F17DA36F22D62FF5E122D82F3ADC2FB73E00200332D72F
32DA2FC334282100263AFC2F471100003ADB2FFE10280AFE0A2848FE0828
1E1832C57ED630FE0A3802D607EB06042910FDEBB35F23C110E9EB22D42F
C9C57ED6300603EB2910FDEBB35F23C110EFEB22D42FC97ED630EB29EBB3
5F2310F5EB22D42FC92AFD2F360D2100263E60EFC5E122D42FC911ED4418
2B11002F18261100271828110000F181C11001F181711000718121100171 8
0D11CB27180811CB2F180311CB3F3ADC2FB7C26228D53E5532DA2FCDAE28
E122FB292AD82FE5F12AD62F7DCB3F6F22D62FF5E122D82FAF32DA2FC334
284F3ADC2FB7C2622879F53E4F32DA2FCDAE28F132DA2FC334283E10180A
3E0A18063E0818023E02F53E4F32DA2FCDAE28AF32DA2FF132DB2F3ADB2F
FE1020070E47210204181BFE0A20070E3A2103051810FE0820070E382103
0618050E322108103ADC2FB77C20017D328E2879328728C3342821030206
033E7CEF21F12F060A3E7CEFCDA82B21582F060A3E7CEF21030106033E7C
EF06082AD82FC50100003E7CEFC10E19FE2D20077D0E2DE67F180CFE2B20
077D0E2BF68018017D076FC506093E7CEFC110D422D82FC334282AD82FE5
F13FF5E122D82FC334282AD82FE5F137F5E122D82FC3342821030206033E
7CEF21DD2F060A3E7CEFCDA82B060221D52F3ADC2FB72002052BC5010000
3E7CEFC1772B4FC506093E7CEFC110ECAF32FC2F3AFB2FF60132FB2FC3AE
283AFC2FB7CA62282AFD2F2B22FD2F3D32FC2F0E0806093E7CEFC337283E
0018023E0132DC2FAF32D52F32D72FC3452A21000022D62F22D82FAF32DA
2F32FB2F21000022D42F21002622FD2FAF32FC2F3AFB2FE60232FB2FC362
280E00FE8920010C06083E7CEFC9060B3E7CEFC3342821030106033E7CEF
212226060A3E7CEF21030106033E7CEFC9CDA82B21030206033E7CEF21E1
2F3ADB2FFE10281121E92FFE0A280A21ED2FFE08280321E52F060A3E7CEF
21070206033E7CEF21F52F3ADC2FB7F5200321F82F060A3E7CEF210F0206
033E7CEF2AD62F11892DEB7BCDE52C772B2B7ACDE52C57F12002162D7A77
06037E4FC506093E7CEFC12310F421030106033E7CEF2AD62F1111263ADB
2FFE10286FFE0A2872FE08281A3ADC2F0610B720050608652E007CE68007
F63012291310F518573ADC2FB73E0101058020070102C0652E003C32AB2C
32B42C7832A92C79329E2CCDA62C3E0332AB2C32B42C3EE032A92C0600CD
A62C10FB181EC57CE68006000710FDC6301213060029 10FDC1C9EB3E63EF
EB18033E61EF3E0312211126060A3E7CEF2AD82F11612F0608CDF02C10FB
21030106033E7CEFC9FE203804FEC038023E2EC9C5D5E5EB5E2356EB0603
3E7CEF0E59E17DE68020020E4E06093E7CEFD1C1291313C9202B3D3D3D3D
3D3D3D3D3D3D7B20427279616E27732042696E6172792043616C63756C61
746F72207D3D3D3D3D3D3D3D3D3D2B0A207C7C2020202020202020202020
20202020204E205369676E204E205A65726F204E204A756E6B204E204243
```

```
4420207C7C0A207C7C2020202031362042697420282020202920 4E204A75
6E6B204E20502F4F20204E2053756274204E2043617272797C7C0A202B3D
7B204D4F444553207D3D3D3D3D3D3D3D7B204249545749534520 7D3D3D7B
20524F544154452F5348494654207D3D2B0A205B5E425D696E205B5E445D
656320205B265D20414E44205B7C5D204F522020205B4C5D20524C434120
5B535D2020534C410A2048655B5E585D205B5E4F5D637420205B5E5D2058
4F52205B215D204E454720205B6C5D20524C4120205B735D202053524 10A
205B5E415D534349492020202020202205B7E5D2043504C205B605D204441
4120205B525D2052524341205B5E525D2053524C0A202D2D7B204D415448
207D2D2D2D2D2D2D7B20434F4D4D414E4453207D2D2D2D205B725D205252
410A205B2B5D204164646974696F6E2020205B5E435D20416C6C20436C65
61722020202D7B20574F5244204C454E475448207D2D0A205B2D5D205375
6274726163742020205B5E4E5D20436C722E20456E74727920205B5E455D
2020382D42697420456E7472790A205B3D5D20457175616C732020202020
5B5E4C5D2053657420466C6167732020205B5E535D2031362D4269742045
6E7472790A205B5C5D2053657420436172727920205B2F5D202043706C20
436172727903535A3F483F5053430314011B012201290114021B02220229
0208392B02322A042A2A18262A0F2E2A01F62A2B162A2D162A3DAE280DAE
28260D2A7C0D2A5E0D2A21AE297EB32960B8294CC7296CCC2952BD2972C2
2953D12973D62912DB2905552B13592B03682B0E782B0C802A80932B8993
2B88A02B5CE82A2FDA2A0000000000000000A0041736303486578034269 6E
03446563034F637403466C67033136032038030000002600
*7A
```

Bigger bubbles for your 5-megger

Here's a tip from Ray Pelzer 70475,1263 which concerns a method of adapting other Tandon hard drive bubbles to the Radio Shack 5Meg hard drive system. It was posted on our Compuserve SIG. " To make a Tandon drive work in a Radio Shack controller, 3 jumpers are added to the TM600 main microprocessor board. The first is a line from the feed-thru hole close to and between pins 8 & 10 of connector J1, over to the closer end of resistor R68 (the only resistor close to it). Then, you'll find 2 stubs of wire-wrap wire. Both go to U9 near J2. One stub from pin 1 to the near end of R64, and the other from pin 2 to the near end of R14 beside it. Those three wires are not on the original Tandon drives, and are added to make the Active light and Write Protect switch work in a Tandy case/controller set. I added those jumpers to a 15-meg bubble, reconnected the 3 wires from the controller & case, and bingo! All set.

Notes on the AlphaTech memory board

The following information concerns the Alpha Technology SuperMEM memory board for the Model 4. This board was reviewed by Hardin Brothers in the January 86 issue [RAM Tough]. MISOSYS has patches available for TRSDOS 6.2 so that the DOS can properly address the memory board by Alpha Technology. As the primary author of TRSDOS 6.x, I steadfastly believe in standardization; thus, I spent my time working with the folks at Alpha Technology as well as with Bentley Mitchell, the now deceased author of the RAMdrive package, in order to ensure that programmers had a uniform protocol of accessing the extra memory.

Since the DOS already supported a scheme of bank switching via the @BANK supervisor call, I worked to extend that scheme so as to support the switching of 31 banks rather than the 3 available in a 128K machine. The extended @BANK support was implemented via three patches to the DOS - two of them quite large. As Hardin reported, MISOSYS makes these patches available to anyone wanting them. There is no charge for the patches; MISOSYS is supplying them free to encourage the maintenance of standards. We only ask everyone's cooperation to minimize the time it takes us to deal with these "freebie" requests. Thus, those wanting a copy of the patches, must adhere to the following. (1) Send a diskette in a

diskette mailer to: MISOSYS, Inc. (attn: ATP), PO Box 239, Sterling, VA 22170-0239. (2) Enclose a return address label that we can apply to your mailer. (3) Enclose postage in US stamps adequate to return the mailer and disk or enclose US funds rounded up to the next dollar for the postage. The patches are also available on Compuserve from the DL3 section of our SIG, PCS49.

In addition, anyone having a copy of the patches may freely give them to anyone else or place them on any bulletin board of their choosing or print them in any TRS-80 computer club newsletter. MISOSYS should be credited as the originator of the patches. Note that the patches have been placed into the public domain with commercial rights reserved by the author.

Note: There is a public domain ramdrive software package described below. It is available from our Compuserve SIG DL3 section.

A few users cannot BOOT the TRSDOS 6.2 disk after application of my patches for the Alpha Technology memory board. This points to a defective memory board. I am aware of a few instances where Alpha Tech has shipped a board which would not work with our patches.

The Alpha Tech board has a RAMPORT which is used to enable the memory banks of the board. This port is supposed to be two-way; i.e. you are supposed to be able to OUTPUT to the port as well as INPUT from the port. There have been cases where the board was defective and INPUT from the port resulted in a constant X'FF' value. My software patches require the port to be fully functional in both directions. You can easily test for two-direction capabilities as follows: (1) From DOS Ready, type "DEBUG"; then depress the <BREAK> key to cause the DOS debugger to become invoked. (2) From the debugger, type the command "Q43" followed by the <ENTER> key. If the display to the right of the "43" shows "FF", the board is defective. If it shows "00", the board is NOT defective. (3) After observing the result, type the letter <O> followed by <ENTER> This causes an exit from DEBUG and returns you to DOS Ready. At this point, type "DEBUG (N)" and the debugger will be deactivated. Your tests should prove a defective board; contact your board supplier for resolution.

The Alpha board is sold by AlA Computer Division in Indian Harbor Beach, FL. I spoke with the folks down there to apprise them of the few instances of defective boards and I am sure they will assist you in clearing up any troubles you may have.

We are also aware that other hardware companies have developed add-on memory boards for the Model 4. In the interest of standardization, I would hope that those companies get in touch with us so that appropriate patches to TRSDOS 6.2 can be developed to communicate with their boards via @BANK.

I am sometimes surprised that the MSDOS world boldly announces new breakthroughs in memory addressability of the IBM PC with the Lotus/Intel/Microsoft standard for banked memory! The Model 4 had that over two years ago. What is needed are a continuation of standards for the memory add-ons to the Model 4. The MISOSYS patches provide the extension to the @BANK standard Model 4 users have enjoyed for over 2 years now.

RAMDRV.LQR [74076,762] 17-Mar-86 63190(26976) 25 -> RAMDRIVE is a series of drivers that utilize the ALPHA TECH memory board and Roy Soltoff's TRSDOS 6.2.x @BANK patches. The drivers allow for multiple memory drives and 40/80 track 1/2 sided drives so that backup and lsqfb can be used. FDR6 has been included with the appropriate @BANK changes. Comments and suggestions appreciated. Michael Jacobs.

The PATCH Corner: General Information

The following information should be read before you type into a file, any of the patches noted in THE MISOSYS QUARTERLY.

It is unfortunate that our printer prints the letter "O" and the number "0" almost identically. Unless we utilize a filter to "slash" the number zero, the two are difficult to distinguish. However, when it comes to patches, all is not lost. In an LDOS 5 or TRSDOS 6 direct patch, the letter "oh" is not used in the patch code (it may appear in comments which are lines beginning with a dot). The direct patch format of TRSDOS 6 which we use in our patches is:

```
Drr,bb=xx xx xx xx xx xx ...
Frr,bb=xx xx xx xx xx xx ...
```

The patch is usually a pair of lines. The first line begins with the capital letter, "dee". This is immediately followed by the "rr" field (which stands for record). The "rr" field is always two hexadecimal digits. Actually, it can be a 4-hexadecimal digit number if the file to be patched has more than 256 sectors. Hex digits use nothing but the numbers zero through nine and the first six letters of the alphabet: A,B,C,D,E,F, or a,b,c,d,e,f. The record number is immediately followed by the "bb" field (which stands for byte). The byte field is also two hexadecimal digits - just like the record field. This is immediately followed by an equal sign, "=". The equal sign is immediately followed by the first patch byte (the "xx" shown above). The patch byte is again two hexadecimal digits. Where more than one patch byte is included on a line, it is separated from its predecessor by a single SPACE. The line is terminated with an ENTER.

TRSDOS 6 patch format uses a "find" line record. This is used to verify that the the file being patched is actually the file you want patched. All of the bytes noted in the "F" line or lines must be matched in the file before any of the "D" patches will be utilized. The second line of the pair begins with the letter "F" which stands for FIND. The next six positions are identical to the preceding "D" line. Following the equal sign on the FIND line are pairs of hexadecimal digits which should align themselves with the preceding line.

So far, the letter "oh" is not used. The only place outside of a comment line where you could find the letter "oh" used is if instead of showing the patch bytes as a series of hexadecimal pairs, it was depicted as a string. A string could be used if one was patching a string of displayable ASCII characters. For instance, the patch:

```
D03,14="This is a new string"
F03,14="extra space for what"
```

would replace the string, "extra space for what", with the string, "This is a new string". Strings are shown within double quotes. That's the only place where a letter "G" through "Z" could be used.

Also, even though TRSDOS supports the colon notation to put more than one patch line on the command line (e.g. "PATCH TEST (D01,27=56:F01,27=65)"), it does not support the colon separator when used in a FIX file. The "D" and corresponding "F" records must be on physically sepapate lines. In order to conserve space in THE MISOSYS QUARTERLY, we may logically print more than one FIX line on a printed line; HOWEVER, ALWAYS USE A HARD <ENTER> FOR THE COLON WHEN TYPING IN A FIX FILE.

```
. M4F80JCL/FIX - Harry G Clayton Jr - 10-Oct-1985
. Apply to Model 4 (F80/CMD) FORTRAN-80 Ver. 3.44
. fixes JCL abort when F80 exits
X'5CBA'=21 00 00
```

```
.M4M80TTL/FIX - 28-Feb-1986 - Harry G Clayton Jr
. This patch causes M80 to read current date for listings
. Also displays logon message
. This patch is for M80 Ver. 3.44 (comes with Model 4 FORTRAN) only
X'3006'=C3 0A 76
X'4908'=09 44 4F 53 36 20 4D 38 30 20 33 2E 34 34 20 09
X'4918'=58 58 20 58 58 58 20 58 58 09 50 67 2E
X'760A'=E5 21 62 76 3E 12 EF 2A 65 76 22 18 49 2A 68 76
X'761A'=22 1F 49 2A 62 76 7D D6 30 06 00 28 02 06 0A 7C
X'762A'=D6 30 80 3D 47 87 80 21 6A 76 16 00 5F 19 11 1B
X'763A'=49 01 03 00 ED B0 3E 0D 32 21 49 3E 20 32 17 49
X'764A'=21 09 49 3E 0A EF 3E 09 32 21 49 32 17 49 21 2F
X'765A'=71 22 07 30 E1 C3 2F 71
X'766A'="JanFebMarAprMayJunJulAugSepOctNovDec"
X'768E'=30

. ADEDCT53/FIX - 02/07/85 - Patch to Model I/III ADE/DCT
. file dated October 26, 1984. Patch corrects operation on Model I
. Install via: PATCH ADE/DCT ADEDCT53
D0C,D7=C1; WAS 4E
D0D,25=22 37 63 21 4E 44 22 9B 6E C9; WAS ZEROES
D0F,19=CD E1 6F; WAS 22 37 63

. ALTDISK1/FIX - 02/07/85 - Patch to PRO-ESP module ALTDISK/CMD
. This patch corrects operation of ALTDISK's use of a low-memory
. area for stack space during bank swap operations.
. Apply via: PATCH ALTDISK ALTDISK1
D01,F0=23:F01,F0=17:D01,F7=15:F01,F7=17
D02,09=23:F02,09=17:D02,12=15:F02,12=17
D05,71=2E 36 47 36 83 36 94 36 00 00 B6 36
F05,71=16 36 2E 36 47 36 83 36 94 36 00 00
D06,7E=00 00 00 00 00 00 00
F06,7E=ED 73 4C 36 31 00 04
D06,B5=21:F06,B5=31
D07,21=F3 ED 73 C9 36 31 20 03 3E 66 EF C5 01 00 01 ED
F07,21=3E 66 EF C5 01 00 01 ED B0 C1 3E 66 EF C9 00 00
D07,31=B0 C1 3E 66 EF 31 00 00 FB C9
F07,31=00 00 00 00 00 00 00 00 00 00

. Patch to DIALER for Model 4P to automatically send a string of P's
. at the end of the dialing line to automatically disconnect the modem.
. Note that this is usable only for telephone communication.
D04,50=06 08 3E 50 CD 83 2B 10 F9 3E 2A C3 83 2B
F04,50=CD 81 2B 21 00 06 11 6C 2F CD 0E 2E 01 00

. DDII/FIX - 02/02/85 -
. PATCH PRO-DD&T VERSION OF DD FOR USE WITH LS-DOS 6.2 MODEL II
D00,F8=E1:F00,F8=DC:D05,01=21:F05,01=26:D05,04=22:F05,04=27
D05,0C=E0:F05,0C=DB:D05,0F=E1:F05,0F=DC:D05,41=E9:F05,41=E4
D06,7D=1F:F06,7D=1A:D08,43=1A:F08,43=15:D08,49=1F:F08,49=1A
D08,A6=1F:F08,A6=1A

. DDTD51/FIX - 06/03/86 - Patch to DD&T's DD/CMD module
. Patch to change module name to "D1" from "DD" to
. avoid conflict with the module name of diskDISK.
D03,6F=31:D05,40=31
```

```
. DDTD61/FIX - 06/03/86 - Patch to PRO-DD&T's DD/CMD module
. Patch to change module name to "D1" from "DD" to
. avoid conflict with the module name of diskDISK.
D03,25=31:F03,25=44:D04,ED=31:F04,ED=44

. DED51/FIX - Patch to DED 3.01a of MSP-02
. Patch corrects operation on multi-head hard drives
D17,FE=11 07 00 19 7E E6 1F 5F 7E AB 07 07 07 3C 1C 57
. WAS =E5 21 07 00 D1 19 6E 26 00 E5 21 1F 00 D1 CD 46
D18,0E=AF 67 83 15 20 FC 6F:. WAS =8F E5 21 01 00 D1 19

. DED61/FIX - Patch to DED 3.01a of PRO-ESP
. Patch corrects operation on multi-head hard drives
D18,0D=11 07 00 19 7E E6 1F 5F 7E AB 07 07 07 3C 1C 57
F18,0D=E5 21 07 00 D1 19 6E 26 00 E5 21 1F 00 D1 CD 04
D18,1D=AF 67 83 15 20 FC 6F:F18,1D=7E E5 21 01 00 D1 19

. DESC51/FIX - 01/18/85 - Applied 00004
. This fix must be applied to be able to <C>reate descriptor data under
. Model III LDOS. Note: The DESCRIBE documentation stated that the password
. used for DESCRIBE/CMD was DESCRIBE. The actual password used on serial #s
. 00001 through 00003 was PDS. Apply via PATCH DESCRIBE.PDS DESC51
D0F,60=CD 8D 6F:D1C,01=03 22 74 5E 21 90 42 22 57 69 C9 00 00

. DESC52/FIX - Patch to DESCRIBE
. This fix corrects the titling for printing directories
. Install via: PATCH DESCRIBE.DESCRIBE DESC52
. Inhibit ETX from being output in title nouns
D0C,0C=FE 20 D8 CD 2C 5C 18 F6:. WAS =CD 2C 5C FE 20 30 F7 C9
. Output the title display via <P> on every page
D0B,37=CD A7 5E:. WAS =CD 3E 5C
D0D,92=11 03 CD 3E 5C 3E 3C 32 4A 5C C9
. WAS =69 66 69 63 61 74 69 6F 6E 11 03

. DESC53/FIX - Patch to DESCRIBE - April 22, 1985
. This fix corrects the <C> command when extending the directory of certain
. hard drive partitions; inhibits extended directory records from <F> command.
. Apply via: PATCH DESCRIBE.DESCRIBE DESC53
D0A,53=CD 78 5F 20:. WAS =1A E6 10 28
D0E,69=1A E6 90 FE 10 C9:. WAS =00 00 00 00 00 00
D1E,59=CD 91 76:. WAS =CD 39 44
D22,F3=CD 39 44 C8 FE 06 C9:. WAS =00 00 00 00 00 00 00

. DESC54/FIX:3 - Patch to DESCRIBE - 05/01/85
. Patch to add to DESC53/FIX to correct operation with large capacity hard
. drives when VERIFY=ON. Apply via: PATCH DESCRIBE.DESCRIBE DESC54
D22,f9=C0 D5 DD E1 DD 34 0A 20 03 DD 34 0B AF C9
. was =C9 00 00 00 00 00 00 00 00 00 00 00 00 00

. DESC61/FIX - 02/14/85 - Patch for PRO-DESCRIBE
. Patch corrects the operation of the REMOVE DESCRIPTORS command so that the
. DOS de-allocates the freed up space. Apply via: PATCH DESCRIBE.PDS DESC61
. Note: DESCRIBE/CMD is documented to have a password of "DESCRIBE"; however,
. it was set to "PDS" when DESCRIBE was released. Please change the password
. via the command: ATTRIB DESCRIBE/CMD.PDS (O=DESCRIBE)
D15,D9=CD AE 53:F15,D9=3E 3C EF
D21,8F=EB CB F6 EB 3E 3C EF C9:F21,8F=00 00 00 00 00 00 00 00
```

```
. DESC62/FIX - Patch to PRO-DESCRIBE
. This fix corrects the titling for printing directories
. Install via: PATCH DESCRIBE.DESCRIBE DESC62
. Inhibit ETX from being output in title nouns
D0B,DD=FE 20 D8 CD F4 39 18 F6:F0B,DD=CD F4 39 FE 20 30 F7 C9
. Output the title display via <P> on every page
D0B,0C=CD 6E 3C:F0B,0C=CD 05 3A
D0D,66=11 03 CD 05 3A 3E 3C 32 12 3A C9
F0D,66=69 66 69 63 61 74 69 6F 6E 11 03

. DESC63/FIX - Patch to PRO-DESCRIBE - April 22, 1985
. This fix corrects the <C> command when extending the directory of certain
. hard drive partitions; inhibits extended directory records from <F> command.
. Apply via: PATCH DESCRIBE.DESCRIBE DESC63
D0A,28=CD EE 3C 20:F0A,28=1A E6 10 28
D0D,E8=1A E6 90 FE 10 C9:F0D,E8=00 00 00 00 00 00
D1D,39=CD B6 53:F1D,39=3E 4B EF
D21,97=3E 4B EF C8 FE 06 C9:F21,97=00 00 00 00 00 00 00

. DESC64/FIX:3 - Patch to PRO-DESCRIBE - 05/01/85
. Patch to add to DESC63/FIX to correct operation with large capacity hard
. drives when VERIFY=ON. Apply via: PATCH DESCRIBE.DESCRIBE DESC64
D21,9D=C0 D5 DD E1 DD 34 0A 20 03 DD 34 0B AF C9
F21,9D=C9 00 00 00 00 00 00 00 00 00 00 00 00 00

. DIALNOCL/FIX - 07/18/85 - Updated 11/23/85 - RS
. This patch can be applied to DIALER/APP to eliminate the "Call"
. and "Input" commands so as to not require the *CL driver's
. availability. Note, that after this patch is applied, DIALER will
. NOT be able to operate a modem! Apply to a working copy of DIALER/APP only!
. Apply via: PATCH DIALER/APP DIALNOCL
.
D01,4B=00:F01,4B=EF:D01,7C=C3 AD 28:F01,7C=11 43 4C:D02,44=C3 56 29
F02,44=3E 41 CD:D02,AF=11:F02,AF=CA:D02,C0=11:F02,C0=CA
D07,75="    ":F07,75="Call":D07,94="     ":F07,94="Input"

. DOEDIT61/FIX - 02/01/85 - Apply to PRO-ESP release of DOEDIT/FLT
. Patch permits the entry of one or more parameters. Previously, only all or
. none could be entered. Apply via: PATCH DOEDIT/FLT DOEDIT61
D01,30=B7 28 09 FA E8 30 CB 6F 28 04:F01,30=CB 7F 20 08 CB 6F 28 06 E6 1F

. EDAS51/FIX - 01/25/85 - Patch EDAS Version 4.3
. This fix corrects macro comments and Model I KFLAG$
D04,F7=23 44; WAS 9F 42:D2B,F3=C3 FE 87 00; WAS FE 3B 28 48
D2E,26=79 00; WAS 78 81
D2F,85=FE 3B CA C8 84 23 77 23 CD 4C 62 CA C8 84 18 F6

. EDAS52/FIX - Patch to correct behavior of OPTION using a '+'
D31,2A=03; was 04 at X'899F'

. EDAS53/FIX - Patch to correct behavior after a "Q" command
D0E,94=00:D0F,39=00

. EDAS54/FIX - Patch to correct behavior of -MF assembly option
D1E,AD=38 2D 22 89 77 06 00 ED 42 CB 7E 28 22 D1; at X'776A'
D1E,E1=AF; at X'779E'
```

```
. EDAS55/FIX - Patch to EDAS 1/3  EDAS module - 05/03/85
. This fix corrects the operation of LORG $
. Apply via: PATCH EDAS EDAS55
D27,DB=0E 88:. WAS =95 8B
D2F,95=3A 42 57 FE 03 C0 C3 95 8B:. WAS =00 00 00 00 00 00 00 00 00

. EDAS56/FIX 05/10/85 - Patch to EDAS 4.3
. This fix corrects EDAS when you assemble with the -WS and -LP switches
. while generating object file. Also, it corrects EDAS * by removing a
. version date time stamp which loaded into the text buffer.
. Apply via: PATCH EDAS EDAS56
D1C,48=C3 F8 81:. WAS =C4 E6 30:D3A,54=10:. WAS =01

. EDAS61/FIX - 01/25/85 - Patch TRSDOS 6.x EDAS Version 4.3
. This fix corrects macro comments
D2B,45=C3 C2 5A 00:F2B,45=FE 3B 28 48:D2D,78=79 00:F2D,78=78 81
D2E,D3=FE 3B CA 8C 57 23 77 23 CD 37 35 CA 8C 57 18 F6
F2E,D3=00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

. EDAS62/FIX - Patch to correct behavior of OPTION using a '+'
D30,7C=03:F30,7C=04

. EDAS63/FIX - Patch to correct behavior after a "Q" command
D0E,19=00:F0E,19=A0:D0E,C3=00:F0E,C3=A0

. EDAS64/FIX - Patch to correct behavior of -MF assembly option
D1E,17=38 2D 22 65 4A 06 00 ED 42 CB 7E 28 22 D1
F1E,17=D1 38 2C 22 65 4A 06 00 ED 42 CB 7E 28 21
D1E,4B=AF:F1E,4B=AE

. EDAS65/FIX - Patch to PRO-CREATE EDAS module - 05/03/85
. This fix corrects the operation of LORG $: Apply via: PATCH EDAS EDAS65
D27,45=D2 5A:F27,45=59 5E
D2E,E3=3A F2 2A FE 03 C0 C3 59 5E:F2E,E3=00 00 00 00 00 00 00 00 00

. EDAS66/FIX 05/10/85 - Patch to PRO-CREATE 4.3
. This fix corrects EDAS when you assemble with the -WS and -LP switches
. while generating object file. Also, it corrects EDAS * by removing a
. version date time stamp which loaded into the text buffer.
. Apply via: PATCH EDAS EDAS66
D1B,AE=C3 BC 54:F1B,AE=C4 E6 30:D39,A6=10:F39,A6=01

. EDFED1/FIX - 03/27/86 - Patch to FED/APP of Mister ED
. Patch corrects closing of the file being edited.
. Apply via: PATCH FED/APP EDFED1
D01,58=04:F01,58=0E

. EDMED1/FIX - 03/18/86 - Patch to Mister ED's MED application
. Patch corrects the operation of the "A" and "H" commands
. Apply via: PATCH MED/APP EDMED1
D02,9D=95 2F:F02,9D=1C 2C:D02,F3=95 2F:F02,F3=1C 2C
D08,19=31:F08,19=30:D08,95=F5 C3 1C 2C:F08,95=00 00 00 00
```

```
. EDMED2/FIX - 06/06/86 - Patch to Mister ED's MED application
. Patch corrects for a DOS error in @BANK, function 4
. as well as cure's obscure bug on exit to permit EXPORT
. Apply via: PATCH MED/APP EDMED2/FIX
D01,0E=31 00 27 C5 C3 99 2F:F01,0E=C5 01 00 00 3E 66 EF
D01,38=CD A3 2F 32 9A 2F:F01,38=3E 66 EF 32 10 28
D08,19=32:F08,19=31
D08,99=01 00 00 3E 66 EF C1 C3 15 28 0E 00 3E 66 EF C9
F08,99=00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

. EDTED1/FIX - 03/31/86 - Patch to Mister ED's TED/APP & OOPS/APP
. Patch corrects abort exit when no 32K bank is available.
. Note that no beep tone will now be heard under this condition.
. Apply via: PATCH TED/APP EDTED1/FIX and PATCH OOPS/APP EDTED1
D01,6F=65:F01,6F=D9

. EDTED2/FIX - 06/06/86 - Patch to Mister ED's TED/APP application
. Patch corrects for a DOS bug in the @bank function 4 SVC
. Apply via: PATCH TED/APP EDTED2/FIX. Also: PATCH OOPS/APP EDTED2/FIX.
D01,62=01 00 04 3E 66 EF DD 77 00 01 20 02 0D 28 64 C5
F01,62=06 04 3E 66 EF DD 77 00 01 20 02 0D 28 65 C5 3E
D01,72=3E 66 EF C1 20 F6 DD 71 01 04 3E 66 EF ED 62
F01,72=66 EF C1 20 F6 DD 71 01 04 3E 66 EF 21 00 00
D08,9B=32:F08,9B=30

. FMTD1/FIX - Patch to Model 1 LDOS 5.1.4 FORMAT/CMD
. Patch resets DCT for 2-sided format capability
. Apply via: PATCH FORMAT.RRW3 FMTD1
X'6EAD'=42 75:X'7542'=FD CB 03 A6 C3 E7 70

. FMTD3/FIX - Patch to Model 3 LDOS 5.1.4 FORMAT/CMD
. Patch resets DCT for 2-sided format capability
. Apply via: PATCH FORMAT.RRW3 FMTD3
X'6EA5'=37 75:X'7537'=FD CB 03 A6 C3 DF 70

. FMTD6/FIX - Patch to TRSDOS 6.2.X FORMAT/CMD
. Patch resets DCT for 2-sided format capability
. Apply via: PATCH FORMAT.UTILITY FMTD6
X'32A5'=16 3A:X'3A16'=FD CB 03 A6 C3 59 35

. HELPADD1/FIX - Applied 04/12/85
. This fix corrects the MAP option of the HELPADD utility provided with the
. PRO-HELP package. Apply via: PATCH HELPADD HELPADD1
D03,6E=1D:F03,6E=2C:D03,94=00 00 00 00 00 00 00:F03,94=CD 0C 2A FE 0D 20 F9

. HTH611/FIX - Patch to LS-Host/Term - HOST61/CMD module - 04/15/86
. Patch corrects logon password length for 7-10 characters.
. Apply via: PATCH HOST61 HTH611
D07,EE=0A:F07,EE=06

. HTH6A1/FIX - Patch to LS-Host/Term - HOST6A/CMD module - 04/15/86
. Patch corrects logon password length for 7-10 characters.
. Apply via: PATCH HOST6A HTH6A1
D08,26=0A:F08,26=06
```

```
. HTH6V1/FIX - Patch to LS-Host/Term - HOST6V/CMD module - 04/15/86
. Patch corrects logon password length for 7-10 characters.
. Apply via: PATCH HOST6V HTH6V1
D07,07=0A:F07,07=06

. Patch to IFC5 Version 3.4b
.Fix to move the tag display char to the start of the line
D09,56=CD:D09,5B=85 6A 00: D19,1C=3E 06 CD 15 6A DD CB 13 7E C9
D08,E4=09:D08,F2=19:D09,31=20:D09,3B=2E:D0B,66=06
D0B,82=06:D1A,56=03:D1A,5C=03:D1B,0F=03
.Fix to allow copy of files which end at the end of the copy buffer
D0F,69=CD 8F 6A:D19,26=78 B7 3A 21 73 C0 C1 C1 C9
.Change version # to revision c
D1A,89="c"

. IFC53/FIX - 07/09/86 - Patch to IFC
. Patch corrects calculation of the number of directory sectors for certain
. hard drive configurations. Apply via: PATCH IFC IFC53
D02,4C=CD 98 6A:F02,4C=FD 7E 04
D19,2F=FD 7E 07 E6 E0 FD 7E 04 C8 CB 20 C9
F19,2F=00 00 00 00 00 00 00 00 00 00 00 00

. Patch to IFC6 Version 3.4b
.Fix to move the tag display char to the start of the line
D09,56=CD:F09,56=DD:D09,5B=93 3C 00:F09,5B=CB 13 7E
D17,22=3E 04 CD 58 3C DD CB 13 7E C9:F17,22=00 00 00 00 00 00 00 00 00 00
D08,E8=06:F08,E8=05:D08,F6=1E:F08,F6=1D:D09,31=2A:F09,31=29:D09,3B=3D
F09,3B=3C:D0B,65=04:F0B,65=48:D0B,81=04:F0B,81=48:D19,18=03:F19,18=20
D18,5F=03:F18,5F=20:D18,65=03:F18,65=20
.Fix to allow copy of files which end at the end of the copy buffer
D0F,61=CD 9D 3C:F0F,61=3A 86 45
D17,2C=78 B7 3A 86 45 C0 C1 C1 C9:F17,2C=00 00 00 00 00 00 00 00 00 00
.Change version # to revision c
D18,92="c":F18,92="b"

. IFC63/FIX - 07/09/86 - Patch to PRO-IFC
. Patch corrects calculation of the number of directory sectors for certain
. hard drive configurations. Apply via: PATCH IFC IFC63
D02,51=CD A6 3C:F02,51=FD 7E 04:D17,35=FD 7E 07 E6 E0 FD 7E 04 C8 CB 20 C9
F17,35=00 00 00 00 00 00 00 00 00 00 00 00

. M80344/FIX - Patch to Model 4 M80 V3.44 06-May-84
. Apply via: PATCH M80 M80344
. Correct M80 to handle 7-character externs, entries, .REQ names.
D1A,4A=00 00 FE 02 28:F1A,4A=28 04 FE 03 20
. Correct M80 to inhibit JCL from aborting on exit
D45,64=21 00 00:F45,64=CD AD 74

. MAS51/FIX - 01/25/85 - Patch MODEL I/III MAS
. This fix corrects macro comments and Model I KFLAG$
D03,CB=23 44; WAS 9F 42:D1F,60=C3 C2 7C 00; WAS FE 3B 28 48
D21,97=79 00; WAS 78 81
D22,F2=FE 3B CA 8C 79 23 77 23 CD 86 5F CA 8C 79 18 F6

. MAS52/FIX - Patch to correct behavior of OPTION using a '+'
D24,9B=03; was 04 at X'7E63'
```

```
. MAS53/FIX - Patch to correct behavior of -MF assembly option
D12,2B=38 2D 22 5E 6C 06 00 ED 42 CB 7E 28 22 D1; at X'6C3F'
D12,63=AF; at X'6C73'

. MAS54/FIX - Patch to EDAS 1/3  MAS module - 05/03/85
. This fix corrects the operation of LORG $. Apply via: PATCH MAS MAS54
D1B,59=D2 7C:. WAS =59 80
D23,02=3A 42 58 FE 03 C0 C3 59 80:. WAS =00 00 00 00 00 00 00 00 00

. MAS55/FIX 05/10/85 - Patch to EDAS 4.3
. This fix corrects MAS when you assemble with the -WS and -LP switches while
. generating object file. Apply via: PATCH MAS MAS55
D0F,76=C3 C1 76:. WAS =C4 C3 5D

. MAS61/FIX - 01/25/85 - Patch TRSDOS 6.x MAS
. This fix corrects macro comments
D1F,0B=C3 A6 4F 00:F1F,0B=FE 3B 28 48:D21,3E=79 00:F21,3E=78 81
D22,9D=FE 3B CA 70 4C 23 77 23 CD 9D 32 CA 70 4C 18 F6
F22,9D=00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

. MAS62/FIX - Patch to correct behavior of OPTION using a '+'
D24,42=03:F24,42=04

. MAS63/FIX - Patch to correct behavior of -MF assembly option
D11,EE=38 2D 22 5A 3F 06 00 ED 42 CB 7E 28 22 D1
F11,EE=D1 38 2C 22 5A 3F 06 00 ED 42 CB 7E 28 21
D12,22=AF:F12,22=AE

. MAS64/FIX - Patch to PRO-CREATE MAS module - 05/03/85
. This fix corrects the operation of LORG $. Apply via: PATCH MAS MAS64
D1B,1C=B6 4F:F1B,1C=3D 53
D22,AD=3A F2 2B FE 03 C0 C3 3D 53:F22,AD=00 00 00 00 00 00 00 00 00

. MAS65/FIX 05/10/85 - Patch to PRO-CREATE 4.3
. This fix corrects MAS when you assemble with the -WS and -LP switches while
. generating object file. Apply via: PATCH MAS MAS65
D0F,39=C3 A5 49:F0F,39=C4 F5 30

. MED51/FIX - 01/25/85 - Patch MODEL I/III MED
. This fix corrects the Model I KFLAG$
D04,42=23 44; WAS 9F 42

. MED52/FIX - Patch to correct behavior after a "Q" command
D0D,49=00:D0D,EE=00

. MED61/FIX - Patch to correct behavior after a "Q" command
D0C,DB=00:F0C,DB=A0:D0D,85=00:F0D,85=A0

. MLB51/FIX - Corrects "Replace module" - Apply via: PATCH MLIB MLB51
D0A,4F=EB:. was =E5:D0A,53=EB B7 ED 52:. was =D1 CD 80 7B
D0A,5A=00 00:. was =E5 21
D0A,5E=00 00 00 00 00 00 CA 19 5C DA:. was =D1 CD E6 7B 7C B5 CA F4 5B C3

. MLB61/FIX - Corrects "Replace module". Apply via: PATCH MLIB MLB61
D09,35=EB:F09,35=E5:D09,39=EB B7 ED 52:F09,39=D1 CD A9 55
D09,40=00 00:F09,40=E5 21
D09,44=00 00 00 00 00 00 CA 03 39 DA:F09,44=D1 CD 0F 56 7C B5 CA DE 38 C3
```

```
. MLIB61/FIX - 02/19/85 - Patch to PRO-MLIB
. This fix corrects buffer operation when HIGH$ is X'FFFF'
. Apply via: PATCH MLIB MLIB61
D04,A7=00:F04,A7=23

. MLK51/FIX - Corrects *REQ handling - Apply via: PATCH MLINK MLK51
D06,D0=21 5B 60 E3:. WAS =E1 06 00 78
D07,EC=C3 AD 60 00 EB:. WAS =E1 73 23 72 23

. MLK52/FIX - Corrects extern+offset if it's the first reference
. Apply via: PATCH MLINK MLK52
D09,1C=EB:. was =C9:D0A,BF=CD 83 61 D1 19 EB:. was =EB CD 84 61 D1 19

. MLK53/FIX - Adds "$", "_", "@", and "*" to acceptable
. filespecs and symbols for -E=symbol - Apply via: PATCH MLINK MLK53
D01,FF=40:. WAS =41 @ 5A82H:D02,2C=F4 69:. WAS =81 5A @ 5AAFH:D02,41=F4 69
. WAS =81 5A @ 5AC4:D11,AD=FE 24 C8 FE 7F C8 FE 2A C8 C3 81 5A
. WAS =00 00 00 00 00 00 00 00 00 00 00 00 @ 69F4H

. MLK54/FIX - Various corrections
. (1) corrects forced change of transfer address, (2) corrects link of
. symbols defined as absolute, (3) inhibits REQ library search if all symbols
. are defined. Apply via: PATCH MLINK MLK54
D04,4A=14:F04,4A=12:D07,FA=0D 6A:F07,FA=13 69:D0A,04=CD 00 6A:F0A,04=5E 23 56
D11,BD=E5 CD 6A 66 E1 5E 23 56 D8 E1 C3 92 6A
F11,BD=00 00 00 00 00 00 00 00 00 00 00 00 00
D11,CA=B7 C2 13 69 67 6F C9:F11,CA=00 00 00 00 00 00 00

. MLK55/FIX - Corrects one byte in MLK53/FIX
. Apply via: PATCH MLINK (D11,B1=5F)

. MLK56/FIX - 11/15/85 - Patch to MLINK in MRAS
. This fix corrects a bug when .REQ is used in the root of an overlay
. generation and the command line containing the -O switch also contains
. additional linker commands. Apply via: PATCH MLINK MLK56
D0A,0C=30 55:. Was =43 56 at X'626F'

. MLK57/FIX - 03/25/86 - Patch to Model I/III MLINK/CMD - corrects operation
. of the -Z switch. Apply via:(LDOS's PATCH command): PATCH MLINK MLK57
D04,0C=C9:. was  C8 at X'5C87'

. MLK58/FIX - Patch to Model I/III MLINK - (1) corrects MLK54/FIX
D11,C8=7E 62:. WAS =92 6A

. MLK61/FIX - Corrects *REQ handling - Apply via: PATCH MLINK MLK61
D06,DC=21 7F 33 E3:F06,DC=E1 06 00 78
D07,F8=C3 D1 33 00 EB:F07,F8=E1 73 23 72 23

. MLK62/FIX - Corrects extern+offset if it's the first reference
. Apply via: PATCH MLINK MLK62
D09,28=EB:F09,28=C9:D0A,CB=CD A7 34 D1 19 EB:F0A,CB=EB CD A8 34 D1 19

. MLK63/FIX - Adds "$", "_", "@", and "*" to acceptable filespecs and symbols
. for -E=symbol. Apply via: PATCH MLINK MLK63
D01,EB=40:F01,EB=41:D02,18=30 3D:F02,18=85 2D:D02,2D=30 3D:F02,2D=85 2D
D11,D5=FE 24 C8 FE 7F C8 FE 2A C8 C3 85 2D
F11,D5=00 00 00 00 00 00 00 00 00 00 00 00
```

```
. MLK64/FIX - Various corrections - (1) corrects forced change of transfer
. address, (2) corrects link of symbols defined as absolute, (3) inhibits REQ
. library search if all symbols are defined. Apply via: PATCH MLINK MLK64
D04,4D=14:F04,4D=12:D08,06=49 3D:F08,06=37 3C:D0A,10=CD 3C 3D:F0A,10=5E 23 56
D11,E1=E5 CD 8E 39 E1 5E 23 56 D8 E1 C3 B6 35
F11,E1=00 00 00 00 00 00 00 00 00 00 00 00 00
D11,EE=B7 C2 37 3C 67 6F C9:F11,EE=00 00 00 00 00 00 00

. MLK65/FIX - Corrects one byte in MLK63/FIX
. Apply via: PATCH MLINK (D11,D9=5F:F11,D9=7F)

. MLK66/FIX - 11/15/85 - Patch to MLINK in PRO-MRAS
. This fix corrects a bug when .REQ is used in the root of an overlay
. generation and the command line containing the -O switch also contains
. additional linker commands. Apply via: PATCH MLINK MLK66
D0A,18=20:F0A,18=F3

. MLK67/FIX - 03/25/86 - Patch to Model 4 MLINK/CMD
. Patch corrects operation of the -Z switch: Apply via: PATCH MLINK MLK67
D04,0F=ED:F04,0F=EC

. MRS51/FIX - Corrects EXT at end and file option
. Apply via: PATCH MRAS MRS51
D14,94=7A 82:. WAS =A3 6E
D28,C2=23 79 E6 C0 B6 77 E1 D1 C9:. WAS =00 00 00 00 00 00 00 00 00
D0E,42=C3 83 82:. WAS =0D 20 F4
D28,CB=0D F2 59 68 C3 72 68:. WAS =00 00 00 00 00 00 00

. MRS52/FIX - Corrects DW in rel segment with absolute value operand
. Apply via: PATCH MRAS MRS52
D28,B3=9B 65 B7 28:F28,B3=B3 67 B7 20

. MRS54/FIX - Corrects transfer address generation for certain cases
. Apply via: PATCH MRAS MRS54
D1F,C3=CD 8A 82 28 08 CD B7 62 EB 3A 9B 65 47 22 30 58 78
. was =21 00 00 45 28 05 CD B7 62 EB 04 22 30 58 3A 9B 65 @ 799FH
D28,D2=21 00 00 45 C9:. was=00 00 00 00 00 @ 828AH

. MRS55/FIX - Patch to Model I/III MRAS - 11/27/85
. Patch corrects chain external across segments
. Apply via: PATCH MRAS MRS55
D14,A4=C3:; WAS =F2 @ 6EACH

. MRS56/FIX - 02/10/86 - Patch to Model 1/3 (MRAS) MRAS/CMD
. Patch corrects crash when PUBLIC, EXTRN, or REF Pseudo-OPs have an argument
. name greater than 15 characters in length. Also corrects logic tests (IFREF,
. IFDEF, etc) under same. Apply via: PATCH MRAS MRS56
D1C,A3=CD 99 82:. WAS =CD 92 76:D1E,35=CD 8F 82:. WAS =3A 33 58
D28,D7=3E 0F B9 30 01 4F 3A 33 58 C9 3E 0F B9 30 01 4F C3 92 76
. WAS =00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

. MRS61/FIX - Corrects EXT at end and file option
. Apply via: PATCH MRAS MRS61
D14,57=5E 55: F14,57=9F 41
D28,6D=23 79 E6 C0 B6 77 E1 D1 C9:F28,6D=00 00 00 00 00 00 00 00 00
D0E,00=C3 67 55:F0E,00=0D 20 F4
D28,76=0D F2 50 3B C3 69 3B:F28,76=00 00 00 00 00 00 00
```

```
. MRS62/FIX - Corrects DW in rel segment with absolute value operand
. Apply via: PATCH MRAS MRS62
D28,5E=92 38 B7 28:F28,5E=AA 3A B7 20

. MRS63/FIX - Corrects "ENTRY exp" if "exp" is not program relative
. Apply via: PATCH MRAS MRS63
D1F,89=28 08 CD AE 35 EB 3A 92 38 47 22 E0 2B 78
F1F,89=45 28 05 CD AE 35 EB 04 22 E0 2B 3A 92 38

. MRS64/FIX - Corrects transfer address generation for certain cases
. Apply via: PATCH MRAS MRS64
D1F,86=CD 6E 55 28 08 CD AE 35 EB 3A 92 38 47 22 E0 2B 78
F1F,86=21 00 00 45 28 05 CD AE 35 EB 04 22 E0 2B 3A 92 38
D28,7D=21 00 00 45 C9:F28,7D=00 00 00 00 00

. MRS65/FIX - Patch to PRO-MRAS - 11/27/85
. Patch corrects chain external across segments - Apply via: PATCH MRAS MRS65
D14,67=C3:F14,67=F2

. MRS66/FIX - 02/10/86 - Patch to Model 4 (PRO-MRAS) MRAS/CMD
. Patch corrects crash when PUBLIC, EXTRN, or REF Pseudo-OPs have an argument
. name greater than 15 characters in length. Also corrects logic tests (IFREF,
. IFDEF, etc) under same.Apply via: PATCH MRAS MRS66
D1C,66=CD 7D 55: F1C,66=CD 8E 49: D1D,F8=CD 73 55: F1D,F8=3A E3 2B
D28,82=3E 0F B9 30 01 4F 3A E3 2B C9 3E 0F B9 30 01 4F C3 8E 49
F28,82=00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

. MSD51/FIX - Correct abort of save block - Apply via: PATCH SAID MSD51
D0A,FE=C3 8E 78 D5:D26,B2=CD 19 6E D1 D8 C3 CD 5C

. MSD52/FIX - Corrects Query S&R - Apply via: PATCH SAID MSD52
D09,1C=C3 96 78:. was =CD 2B 71 at 5AF0H
D0C,AA=00 00 00:. was =DA DF 6F at 5E6EH
D26,BA=CD 2B 71 DA DF 6F C3 F3 5A:. was =00 00 00 00 00 00 00 00 00 at 7896H

. MSD53/FIX - Patch to Model I/III Said version 1.1
. correct line number count when carriage returns are entered
D01,DF=CD 9F 78:. WAS =22 EF 7F
D26,C3=22 EF 7F 2A F1 7F 2B 7E FE 0D C0 C3 A3 6A
. WAS =00 00 00 00 00 00 00 00 00 00 00 00 00 00
. allow delete all to restore default extension
D0D,0B=C3 BB 78:. WAS =C3 FC 6B
D26,D1=21 9E 7F 11 E8 79 01 03 00 ED B0 60 69 C9
. WAS =00 00 00 00 00 00 00 00 00 00 00 00 00 00
D26,DF=21 E8 79 11 9E 7F 01 03 00 ED B0 C3 FC 6B
. WAS =00 00 00 00 00 00 00 00 00 00 00 00 00 00
D29,B5=CD AD 78:. WAS =21 00 00
.eop

. MSD54/FIX - Patch to Model I/III SAID version 1.1
. Corrects DOS command when the invoked program sets a high-memory stack
. which would be in the region of the originally located text. Also ensures
. that system cursor stays off on return to SAID
D0E,62=F3:. WAS =FB:D0E,6E=C3 C9 78:. WAS =C3 B0 5F:D24,43=0F:. WAS =2E
D26,ED=ED 7B EC 79 FB C3 B0 5F:. WAS =00 00 00 00 00 00 00 00
```

```
. MSD61/FIX - Correct abort of save block - Apply via: PATCH SAID MSD61
D0B,42=C3 20 4C D5:F0B,42=CD B4 44 D8
D26,C4=CD B4 44 D1 D8 C3 0D 31:F26,C4=00 00 00 00 00 00 00 00

. MSD62/FIX - Corrects Query S&R - Apply via: PATCH SAID MSD62
D09,1E=C3 28 4C:F09,1E=CD A8 47:D0C,EA=00 00 00:F0C,EA=DA 5A 46
D26,CC=CD A8 47 DA 5A 46 C3 F5 2E:F26,CC=00 00 00 00 00 00 00 00 00

. MSD63/fix - 02/10/86 - Patch to Model 4 SAID version 1.1
. Apply via: PATCH SAID MSD63
. correct line number count when carriage returns are entered
D01,E2=CD 31 4C:F01,E2=22 97 57
D26,D5=22 97 57 2A 99 57 2B 7E FE 0D C0 C3 00 41
F26,D5=00 00 00 00 00 00 00 00 00 00 00 00 00 00
.  allow delete all to restore default extension
D0D,4F=C3 4D 4C:F0D,4F=C3 65 42
D26,E3=21 46 57 11 88 4D 01 03 00 ED B0 60 69 C9
F26,E3=00 00 00 00 00 00 00 00 00 00 00 00 00 00
D26,F1=21 88 4D 11 46 57 01 03 00 ED B0 C3 65 42
F26,F1=00 00 00 00 00 00 00 00 00 00 00 00 00 00
D29,C4=CD 3F 4C:F29,C4=21 00 00
.eop

. MSD64/FIX - Patch to Model 4 PRO-SAID version 1.1
. Corrects DOS command when the invoked program sets a high-memory stack
. which would be in the region of the originally located text. Also ensures
. that system cursor stays off on return to SAID
D0E,6E=F3:F0E,6E=FB:D0E,7A=C3 5B 4C:F0E,7A=C3 DF 33:D24,36=05:F24,36=2E
D26,FF=ED 7B 8C 4D FB C3 DF 33:F26,FF=00 00 00 00 00 00 00 00

. PARMDIRx/FIX - undated - This fix suppresses the NOTES on a MAP parm.
X'54CE'=CD BA 61:X'61BA'=21 00 00 22 4D 55 21 49 5F C9

. PDSC/FIX by David F. Roberts - Patch to write file modification date to
. PaDS directory  instead of date of addition.
X'52E6'=18 16:X'5966'=C2 F2 55 ED 4B 70 58 CD 10 4B C2 F2 55 23 D5 11
X'5976'=F3 58 7E E6 0F 12 23 13 7E 12 D1 C9:X'53E9'=CD 66 59

. PPADSD/FIX Suggested by David F. Roberts - Patch to write file modification
. date to PaDS directory instead of date of addition.
X'26CE'=18 19:X'2D75'=C2 EE 29 ED 4B 7F 2C 3E 57 EF C2 EE 29 23 D5 11
X'2D85'=02 2D 7E E6 0F 12 23 13 7E 12 D1 C9:X'27DC'=CD 75 2D

. PPADSE/FIX - 10/24/85 - Corrects PDS(LIST) numbering in PRO-PaDS
. Apply via: PATCH PDS.PDS PPADSE
D08,B3=CD 7C 29:F08,B3=3E 62 EF
D0A,B1=6E 6F 74 20 61 20 50 44 53 21 0D C5 4F 3E 62 EF C1 C9
F0A,B1=66 69 6C 65 20 69 73 20 6E 6F 74 20 61 20 50 44 53 21

. PROESP1/FIX - Patch to MINIDOS/FLT of PRO-ESP - Patch corrects system
. lockup on error conditions. Apply via: PATCH MINIDOS/FLT PROESP1
D02,D1=06 3E 22 EF C8 E1 3E 08 18 E6:F02,D1=EF 3E 21 EF 3E 08 20 E8 3E 22

. PROESP2/FIX - Patch to MINIDOS/FLT of PRO-ESP - Patch corrects error
. recovery of the "K" command. Apply via: PATCH MINIDOS/FLT PROESP2
D02,94=C3 5B 32:F02,94=CA A8 31
```

```
. PRONTO13/FIX - Patch to PSORT V2.1a of PRO-NTO - 07/17/86 - Patch corrects
. PSORT's PACK option when nrec > 256. Apply via: PATCH PSORT PRONTO13
D0D,24=7C:F0D,24=7D

. PTRACE61/FIX - 06/21/85 - Patch to PRO-DD&T package - This fix corrects
. PTRACE running under TRSDOS 6.2 so that it does not hang the system. Note
. that there will no longer be a blinking asterisk in the upper right hand
. corner of the display  while PTRACE is active as stated in the documentation.
D00,9E=FE:F00,9E=FF:D01,35=CD C2 37:F01,35=01 6E 03:D01,39=E6:F01,39=DD
D04,E4=00:F04,E4=EF:D07,B1=01 6E 03 7C FE F4 D8 21 FF F3 C9
F07,B1=00 00 00 00 00 00 00 00 00 00 00

. PZSHELL1/FIX - 02/01/85 - For PRO-ZSHELL release only! - This fix corrects
. redirection of input and piping. Apply via: PATCH ZSHELL PZSHELL1
D09,64=43:F09,64=41

. SAIDIN51/FIX - Patch to SAID's SAIDINS/CMD - Patch corrects display of
. SHIFT key sense for keys with CLEAR key and A-Z shifted/unshifted
. Apply via: PATCH SAIDINS SAIDIN51/FIX
D20,82=7E 3D 07 E6 01 32 7D 67 00 00 00 00 00
. WAS =6E 26 00 E5 21 80 00 D1 CD 2F 71 7C B5
D22,AE=7E FE 60 3E 01 30 01 3D EE 00 00 00 00
. WAS =6E 26 00 E5 21 60 00 D1 CD 2F 71 7C B5

. SAIDIN61/FIX - Patch to PRO-SAID's SAIDINS/CMD - Patch corrects display of
. SHIFT key sense for keys with CLEAR key and A-Z shifted/unshifted
. Apply via: PATCH SAIDINS SAIDINS61/FIX
D14,2B=7E 3D 07 E6 01 32 51 3F 00 00 00 00 00
F14,2B=6E 26 00 E5 21 80 00 D1 CD B6 47 7C B5
D15,E6=7E FE 60 3E 01 30 01 3D EE 00 00 00 00
F15,E6=6E 26 00 E5 21 60 00 D1 CD B6 47 7C B5

. UD1UNK1/FIX - Patch to Utility Disk I UNKILL/CMD - Patch corrects operation
. if the target file uses an extended directory record
. Apply via: PATCH UNKILL (D02,92=00)

. XREF51/FIX - Patch to EDAS 4.3's XREF V4.3a - Patch relocates out-of-memory
. test routine so that really BIG /REF files will be properly trapped for OM
. errors. Apply via: PATCH XREF XREF51
D01,AD=54:. WAS =47:D01,F3=2A FA 5B ED 5B FE 5B AF ED 52 DA 8D 59 18 A7
. WAS =18 B4 2A FA 5B ED 5B FE 5B AF ED 52 DA 8D 59

. XREF61/FIX - Patch to PRO-CREATE's XREF V4.3a - Patch relocates out-of-memory
. test routine so that really BIG /REF files will be properly trapped for OM
. errors. Apply via: PATCH XREF XREF61
D01,6F=54:F01,6F=47:D01,B5=2A D2 39 ED 5B D6 39 AF ED 52 DA 92 37 18 A7
F01,B5=18 B4 2A D2 39 ED 5B D6 39 AF ED 52 DA 92 37

. ZCAT61/FIX - Patch to PRO-ZCAT - 02/19/86 - This patch corrects for
. printing problems when "Printer NOT available" messages result from print
. buffer full conditions. Apply via: PATCH ZCAT ZCAT61
X'3638'=E5 C5 21 00 16 C3 9F 3B 3E 02 EF C1 E1 C9
X'3B9C'=C3 38 36:X'3BBC'=C3 40 36
```