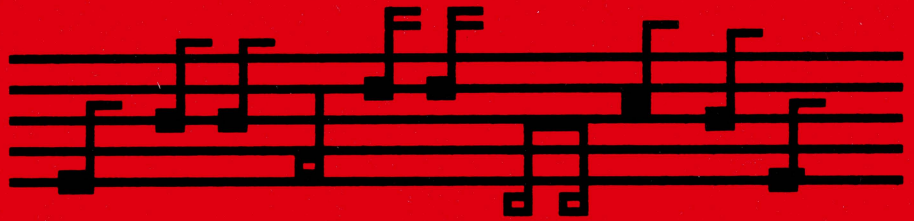


NOTES FROM MISOSYS

ISSUE II - December 1, 1983

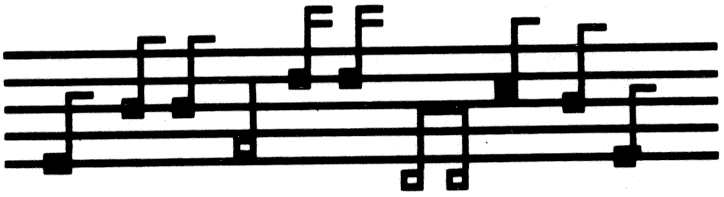


TABLE OF CONTENTS

NOTES FROM MISOSYS is a publication of MISOSYS, PO Box 4848, Alexandria VA 22303.

NOTES FROM MISOSYS

THE BLURB

This is the second issue of NOTES FROM MISOSYS. We have received a great deal of encouragement to continue this publication. I wish to thank all of you who have expressed such feelings. We certainly intend to continue to provide this vehicle of information to our customers. MISOSYS firmly believes in product support. We are making every attempt to produce products of extremely high quality and excellent value for your dollars - quality in the software, the documentation, and the support. That's the kind of environment we strive for with the product itself. The NOTES publication is just one example of after-sale support to the customer.

With the release of Issue II of NOTES, issue I will no longer be automatically sent to newly registered customers. Therefore, if you do not have issue I and wish to obtain it, please send us $2 [$3 if non-US] and request the issue.

Our original goal has been to publish NOTES on a three to four times a year schedule. Since the last issue was dated May 1 and this issue is dated December 1, obviously this goal has not been met. The primary reason has been the tremendous amount of activity taking place at MISOSYS - activity that was prioritized higher than the production of this issue. For example, since the last issue was released, MISOSYS has authored and published THE PROGRAMMER'S GUIDE TO LDOS/TRSDOS VERSION 6, developed PRO-HELP, PRO-GENY, and the VRHARD hard disk driver. We completely rewrote CMDFILE to introduce version 2 CMD-FILE and PRO-CESS - a major undertaking. We assisted Karl Hessinger in preparing the version 5 ZGRAPH documentation, assisted Rich Deglin in preparing the MLIB documentation, upgraded the ZSHELL package to add WC, and aided in planning the MACH2 package. We converted the LC library package to LDOS 6 for use on the Model 4 (the PRO-LC package). We also helped Jim Frimmel groom the compiler itself. You can't imagine the degree of logistics necessary in producing a finished LC package. We also published a new catalog. While all of these activities were going on, Brenda gave birth to our first daughter, Stacey Elizabeth, on June 8th. I want to spend lots of time with Stacey in the first few years; therefore, no more working nights unless absolutely necessary - our days have usually been non-stop. Thus, this issue of NOTES comes seven months after the previous one.

This was typed by Stacey: -----> vbco.1 O/H <-----

The next thing that we usually discuss in NOTES is a summation of our new products. I don't want to go into excessive detail here as our complete catalog was recently mailed to our entire registered customer base. However, I would like to point out some of the significant items that you should be aware of. First, if you are looking at Model 4 products (or products for release 6 of LDOS), you should pay attention to our PROfessional software series. All products that are named beginning with "PRO" are for you. We currently have quite a selection of top-notch software for use with this DOS. There is PRO-LC, a C-language compiler and Macro-assembler; PRO-CREATE, a Macro-assembler; PRO-CESS, a load module maintenance utility; PRO-CON80Z, a utility to translate 8080 source files to Z-80; PRO-CURE, a utility to transfer files from CP/M media to LDOS; PRO-HELP, a help facility of reasonable size; PRO-MACH2, a file allocation package that let's you put-it-where-you-want; PRO-GENY, a collection of four unique support

utilities; PRO-PaDS, a partitioned data set utility; PRO-ZCAT, a disk cataloger; PRO-ZGRAPH, a pixel graphic editor and printing package; and the PROGRAMMER'S GUIDE. We also have the VRHARD hard disk driver package for both Model I/III LDOS 5.1 and LDOS 6.x.

The new products for the Model I/III user (including the Max-80) include CMD-FILE version 2 which is similar to PRO-CESS; CONVCPM version 2, which is similar to PRO-CURE; ZCAT, the Model I/III disk cataloger for use with LDOS; and the enhanced version 5 of ZGRAPH. Rich Deglin joins our group of free-lance programmers with the publication of MLIB by MISOSYS. Let me say that if you are currently using Microsoft's M-80 assembler which generates relocatable code files (/REL) or FORTRAN-80, COBOL-80, or BASCOM, you need MLIB. MLIB is a librarian for creating, building, managing, and maintaining relocatable libraries. MLIB, as supplied by MISOSYS, picks up where Microsoft left off. Since Microsoft has never supplied their librarian as is done with the price-inflated CP/M packages, TRS-80 users have never had the librarian capability. You can get it now with MLIB - as well as get readable documentation which describes the REL format.

When you read through this issue of NOTES, you will find offers to update CONVCPM, ZSHELL, ZGRAPH, LC 1.0, and PRO-CREATE 4.1a. Other products may have patches identified to either correct known bugs or add improvements. Unless otherwise specified, anytime that you want to return a diskette to get the most recent copy of your version release (i.e. 1.1, 1.2, 1.3, etc. and not 1.1 to 2.1), instead of applying the patches yourself, you may return the diskette in a protective mailer with $5 per diskette.

Now let me say a word about sending diskettes in the mail. Radio Shack makes a disk mailer (Cat No 26-1317) which a lot of you use for mailing 5-1/4" diskettes. This is a good mailer. Somewhere I have seen the statement that the mailer is designed for ONE, repeat ONE diskette. Why is this so? There appears to be sufficient depth for a single diskette in the interior of the mailer. When you try to squeeze more than one disk into the mailer, the constant over-pressure tends to compress the edges of the diskette. A diskette compressed in this manner sometimes has difficulty in freely turning in the diskette jacket because of the jacket's pressure on the media. Other types of mailers - both homemade and commercial - may also exhibit this tendency. I have received reports from users getting an update that the diskette is unreadable. Invariably, the difficulty is caused by the inability of the diskette to turn stemming from the compression. Therefore, whatever you use to mail diskettes, make sure that the wrapping does indeed protect your diskettes. There is no reason for you to have to incur a delay in getting a needed and desired update - as well as no need for us to have to duplicate a second time.

Another problem that has come up concerns only our non-US retail customers and the payment for merchandise ordered with foreign checks [a foreign check is a check that is not drawn against an American bank]. Since our commercial bank has started to deduct an excessive fee to clear foreign checks, we will no longer accept checks drawn on a foreign bank as payment for retail orders. Our foreign customers are urged to use either MasterCard or VISA credit card accounts, International Postal Money Orders, foreign bank checks/drafts drawn against an American bank, or cash sent certified and registered (we would prefer any of the first three methods over cash).

Now for the specials. Since this issue of NOTES should reach you during the end-of-the-year buying season, MISOSYS is going to do something it hasn't attempted before. Although our philosophy has been to offer a continuing stability of our pricing structure, the time has come to reward our loyal customers with the opportunity to stock up on quality products at year end reduced prices. First, our catalog announces a discount of 15% on everything in our 83-2 catalog. The discount expires 12/31/83. This gives you the opportunity of purchasing LC/PRO-LC for $127.50 [plus shipping] - a savings of $22.50. Perhaps you want to take advantage of this offer to get our new MACH2 package for $34.

Alternatively, we are going to offer some item specials. These specials will be strictly for our NOTES FROM MISOSYS readers. These specials will last from now until January 31st, 1984. The 15% off discount does not apply to these specials.

```
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
$                                      $
$            SPECIAL PRICES            $
$            UNTIL 01/31/84            $
$                                      $
$   CMD-FILE or PRO-CESS 2.0....$25    $
$                                      $
$   ZSHELL......................$25    $
$                                      $
$   MLIB........................$39    $
$                                      $
$   MSP-01 or PRO-GENY..........$25    $
$                                      $
$  You must mention NOTES to qualify   $
$                                      $
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
```

Turning to another subject, I think that it is time to dispense with all of the rumors concerning my relationship with Logical Systems, Inc. and my activities concerning LDOS. Anyone familiar with LDOS knows that I was the individual primarily responsible for software development of LDOS version 5 and the re-design of version 5 into version 6. I have nurtured LDOS for a long time now and have no intentions of deserting it. A few years of my life have been spent taking care of that system and it has become a part of me.

With the completion of LDOS 6.0, LSI decided that all software development of LSI products will be performed at the LSI headquarters. Since LSI is located at Milwaukee, WI and I am located at Alexandria, VA, my services were no longer required. Although I am still a minority stockholder of LSI, I have no control over how that company is run. I regret being ejected from any further active involvement with "my system", however, it was LSI's decision. I have no intentions of relocating to Wisconsin in order to continue such activities; I still intend to respond to questions concerning the DOS that I may be able to answer. Now let's get back to work.

The last issue of NOTES was sent SURFACE to our foreign (foreign is the term we shall apply for mailing purposes to all countries other than the United States and Canada). I have had a few requests from foreign customers to send NOTES by air mail. This issue is being sent BULK MAIL to our United

States customers, First Class to Canada, and AO Air to our foreign customers. We will attempt to continue these mailing methods in 1984 with a projection for three issues of NOTES. As an example of the difference in rates for surface versus AO Air, a 4-ounce issue can be mailed surface to any country for $0.71 while the cost would be $1.68 for Europe and $2.08 for Australia and Japan. Domestic BULK RATE is $0.11. Therefore, we would appreciate it if our foreign customers would remit $5 per year for the continuation of NOTES via Air.

Don't forget that we are also open to receive small articles concerning the use of any MISOSYS product. We will make every attempt to publish your item in the next issue. Also, contributed programs are acceptable. I now call your attention to the following items of commercial interest.

Riclin Computer Products - Software for the Lobo MAX-80

LCOPY is all you will ever need to transfer files from LDOS disks to MAX-80 CP/M disks. LCOPY runs under CP/M. It is simple to use, as its command syntax is similar to that of PIP. LCOPY supports the following LDOS formats: 8-inch SSDD or DSDD; 5-1/4-inch, 35, 40, 77, or 80 cylinders, SSSD, SSDD, DSSD, and DSDD. LCOPY supports all MAX-80 CP/M formats. Commands may be entered on the CP/M command line, or one at a time following the LCOPY prompt. The following command formats are supported: X:=Y:AFN (wildcarded filename); X:=Y:UFN (one file only); and X:UFN1=Y:UFN2 (one file only with a new name). If the file already exists on the destination CP/M disk, it will be saved as a backup (.BAK) file. Parameter specifications may be added to any command line: V - copy visible files; I - copy invisible files; S - copy system files; M - copy modified files; N - copy new files; O - copy old files; Q - query by file; E - erase previous copy of destination file (no .BAK); D - copy files by date (all LDOS date forms are supported). CVTEXT is supplied which converts TRS-80 ASCII text files to CP/M format. LCOPY is priced at $40; available in either 8" or 5-1/4" CP/M formats.

MAXFONT allows you to create alternate MAX-80 video fonts. It includes an interface to GRASP; this enables direct printing of these fonts on an Epson M series printer with Graphtrax. MAXFONT runs under LDOS and is menu-driven. Commands include: Load/Write font from/to disk; Get/Store font from/to character generator; View/Print font; Clear from memory; Restore original font; Disk directory; Kill disk file; and Edit font. Edit is also menu-driven, and is similar to the GRASP editor. MAXFONT is supplied with sample fonts and INSTALL which loads a font from disk to the character generator from DOS Ready. MAXFONT is available in 8" and 5-1/4" LDOS formats, and is priced at $40.

Please send check or money order to: Riclin Software Products, 4901 Seminary Road, Suite 1003, Alexandria, VA 22311.

E.S.P. - Enhanced Systems Packages for the Model IV and MAX-80

Make use of those "extra" features in your advanced machine under LDOS 5.1.x! Use E.S.P...

- o MAXDISK and M4DISK - alternate memory bank, 64K, high speed RAM disk. Can be system or any other drive. Drive contents are recoverable after reboot.
- o DO80 - full function 80x24 video driver, including scroll protect, special character mode, BLINK support, PRINT@, more.
- o STATLINE - 25th status line support for MAX-80 80x24 video.
- o DOEDIT80 - full screen 80x24 LDOS command line editor.
- o SWAPMODE - change from/to 64x16/80x24 mode with one keystroke
- o PRTOGGLE - engage/disengage a *DO|*PR LINK with one keystroke
- o NOROM - run your Model IV in Model III mode without the ROM.
- o ALTLD - dump/load the 64K alternate memory bank to/from disk.
- o ALTRES - reside system overlays in the alternate memory bank.
- o FKEY - program the MAX-80's function keys to your desire

Programs fully compatible; all filters and drivers SYSGENable.

Order from Riclin Computer Products [see address above] or MicrConsultants - East, 7509 Wellesley, College Park MD 20740. Model IV or MAX-80 packages, $40 each, $70 for the set of two.

CMD-FILE/PRO-CESS Version 2

When I set out to port my CMDFILE program over to LDOS 6.x and the Model 4, I knew that I wanted to introduce many new functions that I have never seen existing in any other load module maintenance utility. The primary function desired was the reorganization (packing) of load records that existed in non-sequential load order as well as in less than maximum length (the maximum size of a load record is a block of 256 load bytes plus the 4-byte per record overhead). Cassette operations were going to be dropped. Also, since I have become interested in the user-acceptance of friendly menu-driven applications, the new version was to be menu controlled.

After the product was complete, I wanted to bring the powerful features back to the Models I and III since MISOSYS has no intention of deserting our Model I/III audience. Wherever possible, we will develop a product for Model I/III users as well as the LDOS/TRSDOS 6.x audience [and other systems in the near future]. The big problem came in naming the Model I/III product. Since the product was derived from and represented a major enhancement of CMDFILE, I decided to continue to name the Model I/III product CMD-FILE 2. If you do not use cassette functions, then the new version practically replaces the old version of CMDFILE [the only missing function is the addition of the "appendage"].

Let's take a look at what I have implemented in the load module "packing" function. First, since a module may have had LDOS's X-type patch records applied, CMD-FILE/PRO-CESS removes all X-patches and converts them to "direct" D-type patches where possible. Any remaining X-type patches are placed into new load records. The file is then sorted sequentially by load address. Finally, the file is written with maximum-sized load records for all load bytes that are contiguous.

As an example of the utility of this operation, take a look at the output of the LC compiler. LC generates separate regions for code and data. Everytime you specify a static data element [i.e. "character string" or static variables], the compiler has to switch the program counter to the data region then switch back to the code region. When this sequence is assembled, it results in generating new load records EVERY TIME THE PC IS SWITCHED. Thus, the resulting load module is not optimized for maximum-sized load records. If you are using LC to generate commercial software, you owe your customers the best load module. You should use CMD-FILE/PRO-CESS to pack that file.

As another example, I analyzed some very inefficient programs as far as load record size. Take RSCOBOL and RUNCOBOL, for instance. Both of these programs which are included with the Radio Shack COBOL package are composed of 16-byte load records. Since each load record has four bytes of overhead, 256 bytes of program take up 320 bytes of file space in both programs. This compares to 260 bytes for programs written with 256-byte load records. After processing RUNCOBOL with CMD-FILE 2, I reduced RUNCOBOL from 1537 load records down to 97 load records while the disk space taken up by the file shrank from 121 sectors down to 98 sectors - A SAVINGS OF 6K OF DISK SPACE! Not only disk space was saved but also loading time. It took approximately 11 seconds to load RUNCOBOL under TRSDOS 1.3 prior to packing it. After packing, it took about 10 seconds. That 9% improvement over all the invocations of

RUNCOBOL by all the COBOL users represents a great deal of time!

In case you wonder how I got the password protected RUNCOBOL and RSCOBOL accessed, I'll tell you. I just read the TRSDOS diskette's directory using the LDOS extended debugger and changed the password and attributes to enable the access.

Another use of the packing function is to generate a load module with sequential load records. Non-sequential load record files cannot be properly disassembled by DSMBLR/PRO-DUCE with data screening. Therefore, by processing such a file with CMD-FILE/PRO-CESS, it's ready for disassembly.

Now that I have given you a little lead-in to some uses of this load module maintenance package, I can shift over to supplying two patches. These patches are only for PRO-CESS, not the Model I/III CMD-FILE version.

```
. PROCESS1/FIX - 10/13/83 - Applied 520021
. Corrects cursor UP/DOWN after BREAK
. and MAP printing after printer error.
D00,DD=98
F00,DD=74
D15,E9=F5 CD 74 45 F1 C9
F15,E9=00 00 00 00 00 00
D07,27=D9
F07,27=D1
. End of patch

. PROCESS2/FIX - 10/27/83 - Applied 520021
. This patch corrects the END address
. after an IMAGE LOAD invocation
D02,3C=9E 45
F02,3C=E4 34
D15,EF=06 00 C3 E4 34
F15,EF=00 00 00 00 00
. This patch corrects PACK of X-patches
. where the last byte of a load record
. is being patched.
D09,9C=2F 6F 23
F09,9C=ED 44 6F
. End of patch
```

CON80Z/PRO-CON80Z

Bernd Jung of Dusseldorf, West Germany reports a few problems with the CON80Z program that apparently slipped by our testing. Specifically, Bernd discovered that: 1) "CNC XYZ" was not decoded to "CALL NC,XYZ"; 2) "CPO XYZ" was not decoded to "CALL PO,XYZ"; 3) "ACI n" was decoded to "CP A,n" instead of "ADC A,n"; 4) "RST n" was passed unchanged instead of converting "n" to "p"; and 5) "HLT" was not decoded to "HALT" but was left as "HLT".

Two of the five bugs can be fixed by patches; however, since three of the five cannot, I decided to correct my source and reassemble. Therefore, MISOSYS is now shipping CON80Z version 1.1 and PRO-CON80Z version 1.1. If you

are currently in possession of a 1.0 version vopy of CON80Z and wish to obtain the corrected version 1.1, please return your master diskette in a protective mailer. There will not be a charge for the update to your disk.

CONVCPM/PRO-CURE

A patch is needed to CONVCPM version 1.3. It corrects a problem when using CONVCPM with high-memory drivers. The patch is as follows:

```
PATCH CONVCPM (D01,13=03)
```

The address in question is X'5303' and the old value was an X'01'.

Our new catalog presents information on CONVCPM/PRO-CURE version 2. This release expands the supported CP/M media types. If you currently have the CONVCPM version 1.x and want to upgrade to the CONVCPM version 2.x, please return your CONVCPM master diskette with $20. A completely new package will be returned to you. Note that there is no offer to upgrade from CONVCPM 1.x to PRO-CURE.

DSMBLR/PRO-DUCE

We have had so many positive reports on version III of our disassembler that I am glad to have taken the time to enhance the DSMBLR II package. You may be aware that the DSMBLR is one of the first products ever released by MISOSYS. The original version I was for Model I cassette machines. A version was done for the Exatron stringy floppy. We expanded DSMBLR to version 2.0 and added disk source file output. I believe it was version 2.2 which supported Model III. The third version supports disassembly direct from disk - a function that we received so many requests for. We went beyond just disk input and added a screening mode to provide user definition of data areas. Thus, Version III provides for the specification of literals (messages and other strings), words (DEFWs/DWs for the hackers), and bytes (DBs and DEFBs again for the hackers).

Version III is supported under TRSDOS/LDOS 6.x under the product name PRO-DUCE - since it produces assembly code. The Model I/III compatible product called DSMBLR III was designed to be fully compatible with LDOS 5.0, LDOS 5.1, TRSDOS 2.3 (Model I) and TRSDOS 1.3 (Model III). DSMBLR probably runs under other Model I/III operating systems; however, we only guarantee it under LDOS and TRSDOS (sorry, we don't support anything under TRSDOS 2.7DD).

The disassembler is a powerful product. Although there are still some features that you may want (such as user-defined labels and online entry of screening ranges), I feel it is a bargain at the $40 price. As good as the disassembler is, a bug or two still found its way past our testing. Therefore, a couple of patches follow. The patch syntax shown is for Model I/III LDOS or TRSDOS 6.x (in the case of PRO-DUCE). For TRSDOS 1.3 PATCH, convert the "Dxx,Byy=zz zz zz" syntax to "ADD=aaaa,FIND=bbbb,CHG=cccccc" using the "zz zz zz" for "cccccc", the values shown as WAS for "bbbb", and the address shown as @X'hhhh' for "aaaa". DSMBLR3x refers to Model I/III

[note that DSMBLR31 and DSMBLR32 were applied prior to release] whereas PRODUCEx refers to the PRO-DUCE version:

```
. DSMBLR33/FIX - Applied starting with 530099
. Patch corrects command entry of parameters w/o filespec
D00,D5=00 00; WAS D6 0D @X'5491'
D09,95=C3 3E 67; WAS C2 7B 5E @X'5D2D'
D13,CE=FE 41 DA 30 5D C3 7B 5E; Were zeroes @X'673E'
. End of patch

. PRODUCE1/FIX - Applied starting with 530013
. Patch corrects command entry of parameters w/o filespec
D00,97=00 00
F00,97=D6 0D
D09,3C=C3 97 43
F09,3C=C2 34 3B
D12,1F=FE 41 DA DB 3A C3 34 3B
F12,1F=00 00 00 00 00 00 00 00
. End of patch

. DSMBLR34/FIX - Applied starting with 530310
. Patch corrects swapping disks when output disk is full
D11,60=C3 46 67; WAS F6 01 C9 @X'64DC'
D13,D6=11 BE 73 F6 01 C9; Were zeroes @X'6746'
. END OF PATCH

. PRODUCE2/FIX - Applied starting with 530080
. Patch corrects swapping disks when output disk is full
D0F,CD=C3 9F 43
F0F,CD=F6 01 C9
D12,27=11 34 4F F6 01 C9
F12,27=00 00 00 00 00 00
. END OF PATCH
```

The following patch is derived from an X-patch that was placed on the LDOS Compuserve bulletin board. The name of the contributor had scrolled off the display before I was able to SCREEN PRINT the text so I cannot give credit where credit is due. I can suggest that if you have developed any patch to a MISOSYS product, why not send it to us so that we may evaluate it. Perhaps there is a better way of accomplishing the desired result. Perhaps we can at least give the other users a chance at implementing your change. In any event, I rewrote the patch to shorten it a little and converted it to a DIRECT patch. It now is official and uses a small portion of the patch area I reserved in the disassembler product.

```
. DSMBLR35/FIX - 10/06/83 - APPLIED 530409
. CORRECTS BUG WHEN A LITERAL SCREENING RANGE
. IS SPECIFIED THAT INCLUDES A LABEL EQUATE
. OF NEGATIVE DISPLACEMENT GREATER THAN 9
.
D09,33=CD 4C 67
. WAS  CD 47 5C @X'5CCF'
D13,DC=FE 3A 38 08 F5 3E 31 12 13 F1 D6 0A C3 47 5C
. WAS  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 @X'674C'
. END OF PATCH
```

```
. PRODUCE3/FIX - 10/06/83 - APPLIED 530080
. CORRECTS BUG WHEN A LITERAL SCREENING RANGE
. IS SPECIFIED THAT INCLUDES A LABEL EQUATE
. OF NEGATIVE DISPLACEMENT GREATER THAN 9
.
D08,E3=CD A5 43
F08,E3=CD F7 39
D12,2D=FE 3A 38 08 F5 3E 31 12 13 F1 D6 0A C3 F7 39
F12,2D=00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
. END OF PATCH

. DSMBLR36/FIX - 11/21/83 - Applied 530486
. Corrects possible crash on excess interstitial labels
D09,38=CD; Was BF @X'5CD4'
. End of patch

. PRODUCE4/FIX - 11/21/83 - Applied 530100
. Corrects possible crash on excess interstitial labels
D08,E8=CD
F08,E8=BF
. End of patch
```

Now that the buggy patches have been identified, let me turn to some improvements. Bernd Jung is one of my devoted users in West Germany. I sometimes feel that I know the guy personally having communicated so frequently via letters. In any event, Bernd does a lot of work with the 8085 chip. This chip is an improved 8080. The disassembler can handle 8085 code to produce Z-80 mnemonics [not that useless of a result as you may imagine]. The only problem is that the 8085 uses the op codes 20H and 30H for the RIM and SIM instructions [Read Interrupt Mask and Set Interrupt Mask]. These op codes are used in the Z-80 as two-byte instructions. Thus, DSMBLR would cough a little on RIM and SIM instructions.

To help Bernd out of this limitation, I worked up a little patch to DSMBLR III to support the 8085 [actually I had to work it up twice since the first patch created a bug in the disassembly of two other instructions: "LD (nnnn),HL" and "LD (nnnn),A". If I get sufficient requests, I'll try to work up such a patch for PRO-DUCE [or maybe yet someone who has both DSMBLR and PRO-DUCE could work up the patch]. The change is shown in assembly source code form. It is left to the reader as an exercise to convert it to a patch. Don't forget, the Model 100 uses an 8085 compatible chip!

```
;D8085
MOVOP   EQU     5A3BH
SHFT3   EQU     572DH
        ORG     6BB8H
        DB      '1'         ;Change 20H & 30H
        ORG     56E9H
        CALL    PATCH+1 ;Patch single-byte OPs
        ORG     5A9DH
PATCH   RET                 ;Mask B2LBL
        CP      20H         ;Is it RIM?
        LD      HL,SIM$ ;Init to SIM string
        JR      Z,DORIM ;Go if RIM
```

```
        CP      30H     ;Test for SIM and do it
        JR      NZ,MOVOP ;Else do the original code
        DB      0C2H    ;Ignore next inst
DORIM   LD      L,RIM$.AND.0FFH
        POP     AF      ;Clean the stack
        JP      SHFT3
RIM$    DB      'RIM'
SIM$    DB      'SIM'
        END
```

Some people are never happy with the designer's choice of expression. That's why we always want to customize canned software. Bernd was unhappy with our choice of "M" as the label prefix [we actually got complaints about the letter "Z" used in DSMBLR II so I changed to an "M" for MISOSYS]. To satisfy those that want their own character, you can patch DSMBLR III at X'5AE5' with the character of your choice - make it a character acceptable to your assembler. The corresponding PRO-DUCE location is X'3895'. The value is currently an X'4D' [note: giving addresses helps promote the sale of LSI's FED utility which runs circles around all other utilities for zapping load modules].

James A. Sladek, of Norfolk, VA, points out a minor typo in the documentation on page 19. Step 4 of the DEBUG procedure used with Model I TRSDOS should specify the new value as "3C" not "7C" as noted. In fact, the "7C" is what the old value is! While I'm at it, an easier way from BASIC is to just "POKE &H46B0,60". One of these days I might write an article on the entire procedure to convert TRSDOS 2.3 to the Model III data address mark convention.

James must be one of my early DSMBLR users as he mentioned the "infamous TRS-80 printer DCB bug of initialization to 67 lines per page". This pertains to Model I only. To remind the Model I users, the *PR device control block lines-per-page field was initialized by TRSDOS to 67 although the printer driver's counter started from zero. Thus, paper tended to migrate up one line per page (11" paper has 66 lines per page). My old DSMBLR documentation mentioned this and suggested that you poke a 66 into X'4028'. As an aside, Tandy "corrected" the problem on the Model III by keeping the 67 but reindexing the starting value to one. Enough of that! James supplied a patch to Model I TRSDOS SYS0 to have SYS0 correct the value to 66. The values shown are for DISK TRACK, SECTOR and BYTE:

```
T01,S05,BF2=3E 42 32 28 40; WAS 2C 20 00 3E 03
T01,S05,BFD=00; WAS F5
T01,S06,B12=00 00 00; WAS F1 3D 20
T01,S06,B16=00 00 00; WAS 3A EC 37
```

As a final note from James, he wanted more space for the title on disassembler listings. He there worked up a patch to "sacrifice" my title and lengthen the user title to 51 characters. James suggested that I implement his change in a future release; unfortunately, vanity precludes me from doing so. I will share his patch with you. This patch is for DSMBLR III only. If someone wants to work up the equivalent patch for PRO-DUCE, I will publish it in a future NOTES.

5FC1 from OC to 33
5FC3 from 9E to 75
5FCC from OC to 33
625B from 9C to 75
6874 from 4D to 03
6875-68AA from TEXT to 20

Let me share a few responses to letters received concerning the disassembler version III

Ervan Darnell qustioned me about the "MFFFF" label which always seemed to be generated even though the program being disassembled neither referenced X'FFFF' nor had a -1 (X'FFFF') reference as an operand to a 16-bit instruction. The "MFFFF" label is always generated as that value is used to prime the table which collects address references. Its presence in no way affects the integrity of the output nor is it detrimental in any way, short of using one line of a printout.

Ervan also was one of the first to report difficulty in screening a program direct from disk when the program contained load records that were not in sequential order [actually, Robert Newton first identified such a quirk]. The problem, that of not "screening load blocks which load into lower memory than the previous block", is a limitation due to the design of the disassembler. I would venture to guess that 95% of all programs are in sequential load order. The DSMBLR properly handles those 95%. It does this by first sorting your array of screening data then searching the "array" at each logical PC address. If DSMBLR wanted to handle the 5% of programs that are out of sequence, it would have to search the table at each logical PC. This was felt to be inefficient and would occupy considerable time. DSMBLR was programmed to handle the 95% of the cases rapidly! I suspect that the DSMBLR documentation will be tightened a little to reflect on this limitation. As part of this "problem", you are restricted to not have a screening range that is lower than the first logical PC address.

Another problem also stems from a limitation that can be easily rectified by paying careful attention to your screening data ranges. If a file has a sequential but non-contiguous load record, the disassembler by itself properly handles the disassembly and generates another ORG statement. However, if you have specified a screening data range that traverses the boundary between such a load record and its previous record, the disassembler cannot handle it. We may revise the decoding algorithm to handle such a case, but for now, tighten up your screening data. For instance, a program has a message extending from X'7777' through X'7787', a 16-byte gap (perhaps a DEFS 16), then another message from X'7797' through X'779F'. Do not enter one range as "$7777-779F" but specify the two ranges, "$7777-7787,$7797-779F".

As long as I am talking about screening data, let me talk a little on why I came up with the scheme that I did. This discussion is for the programming users. Every one of you knows that you love to tinker with programs that you have purchased. Its difficult, however, to understand every program you want to tinker with. Wouldn't it be great if those that tinkered and wanted to offer their information to others, had a convenient means of doing so. With assemblers as powerful as EDAS/PRO-CREATE, all you need is a good set of source code to make whatever customization you wanted to make. If you have spent the time to disassemble a machine language program and want to

share the results of your efforts, all that is needed is for you to publish your screening data file - which can be placed into the public domain. If you wanted to make it commercially available, you could publish it as a product. Imagine what you could do by creating a set of screening files for FORTRAN, or SCRIPSIT, or PROFILE, or ... With your screening files, the MISOSYS disassembler, and a purchased copy of the product referenced by the files, most individuals could easily alter the canned software product to suit their own desires. The screening data file is a legal means to develop and promote the generation of source code files. Enough said?

EDAS VERSION 4.1/PRO-CREATE

The EDAS/PRO-CREATE column this issue will first start with a correction to a patch published in the last issue. The line in EDAS410/FIX that read D1B,7E=28 08... should have read D1B,7E=28 04... The corrected patch is shown below.

```
. EDAS410/FIX - 01/04/83 - Roy Soltoff
. This FIX corrects the symbol table addition on the
. statement: LABEL <TAB> ;COMMENT
. Patched starting with 820428
.
D12,55=FE 3B C2 63 72 4D C3 28 72
.; WAS 00 00 00 00 00 00 00 00 00
D1B,7E=28 04 C3 54 69 00 FE 3B 28 DE
.; WAS 20 04 FE 3B 28 DA FE 3B 28 C5
. end of patch
```

The next three patches are for Model I/III EDAS. They should be applied to your working copy of EDAS based on the serial number of your master.

```
. EDAS415/FIX - 05/18/83 - Applied starting 820663
. This fix corrects the bug where the -IM assembly option
. may cause an erroneous "File not open" error.
.
. PATCH EDAS/CMD USING EDAS415
.
D12,7A=B6 21 01 5A B6 C9
. was  00 00 00 00 00 00
D1A,5F=CD 79 69
. was  21 26 38
. End of fix

. EDAS416/FIX - 05/18/83 - Applied starting 820663
. This fix corrects the bug introduced by EDAS414/FIX
. when MACROs are used and -WS option is specified.
.
. PATCH EDAS/CMD USING EDAS416
.
D12,80=2A B4 56 22 CE 57 C9
. was  00 00 00 00 00 00 00
D19,DA=CD 7F 69
. was  2A B4 56
```

```
. End of fix

. EDAS417/FIX - 10/18/83 - Applied starting 820996
. This fix corrects the bug that sometimes occurred
. when you entered EDAS with a wrong parameter.
.
. PATCH EDAS/CMD USING EDAS417
D02,FD=67 44; WAS 2C 5B
D06,87=0D; WAS A1
. End of patch
```

There are three patches for PRO-DUCE Version 4.1. They are as follows:

```
. PCREAT01/FIX - 05/18/83 - Applied starting 820013
. This fix corrects the bug where the -IM assembly option
. causes low memory to be altered.
.
. PATCH EDAS/CMD USING PCREAT01
.
D12,6B=B6 21 26 38 B6 C9
F12,6B=00 00 00 00 00 00
D1A,55=CD FC 47
F1A,55=21 26 38
. End of fix

. PCREAT02/FIX - 05/18/83 - Applied starting 820013
. This fix corrects the bug introduced
. when MACROs are used and -WS option is specified.
.
. PATCH EDAS/CMD USING PCREAT02
.
D12,71=2A B4 34 22 CE 35 C9
. was  00 00 00 00 00 00 00
D19,D0=CD 02 48
. was  2A B4 34
. End of fix

. PCREAT03/FIX - 10/18/83 - Applied starting 820996
. This fix corrects the bug that sometimes occurred
. when you entered EDAS with a wrong parameter.
.
. PATCH EDAS/CMD USING EDAS417
D02,9C=3E 0A EF
F02,96=CD 5F 39
D06,8A=0D
F06,8A=A1
. End of patch
```

Now that I am discussing PRO-CREATE, a minor drawback in its operation was discovered when testing the PRO-LC package. If an error occurred during the assembly process, the assembler aborts to DOS ready. This left any output files open. In order to correct this problem, it was necessary to alter the source code and reassemble the assembler instead of patching. Thus, starting with serial number 820072, MISOSYS is shipping PRO-CREATE version 4.2a. If you have PRO-CREATE version 4.1 and want the 4.2a version, you can return

your master disk with $5 to cover the update handling costs. Be aware that the PRO-LC package contains only one 40-track double density diskette. That diskette includes the entire compiler and assembler files. Therefore, if you have tendered an order for PRO-LC and have already received version 4.1 of the assembler, you will NOT have to return your PRO-CREATE diskette as you will automatically get version 4.2 with the PRO-LC. However, if you are not ordering PRO-LC, you may want to get the 4.2 release to keep current. Let me stress that this does NOT pertain to the Model I/III EDAS.

Let me now get on to some discussions concerning the use of EDAS. Robert Newton asks, "A subject that I would like to see explained in your lucid style in the next NOTES is the use of LORG. I recently had occasion to use it, but just never got the hang of it. For example, how do you revert back to where you really are?" Bob raises a good point.

The EDAS manual states that "A load-origin assembler directive, 'LORG', is provided to cause the load addresses of the object file to be based on the LORG operand while the execution code address references will still be based on the 'ORG' operand." Microsoft's M-80 assembler uses .PHASE and .DEPHASE to switch the "load origin" on and off, whereas EDAS provides only the one pseudo-OP.

Before I state exactly what needs to be entered to answer Bob's question, let me address the concept of LORG. Being able to generate a load module that loads at an address different from where it executes is most important when you are writing what is called ROMable code. Many TRS-80's are used as Z-80 (or even 8080) development systems. Such a system is used to generate software which will run on some other machine. The typical use is the generation of software to be placed in a Programmable Read Only Memory (PROM). There are a few companies that sell "PROM burners". Take a look at any TRS-80 publication and you will see ads for such equipment.

The PROM burner gets its name from the method in which a PROM is programmed. The typical PROM direct from the factory essentially contains memory cells that are all 1's. That is, each bit is turned on by a diode connection. In order to transfer a memory image to the PROM, wherever the image has a zero bit, the corresponding PROM bit must be changed from a one to a zero. This is done by electrically destroying the diode - the process sometimes called "burning" [Erasable PROMs, or EPROMS, are programmed differently]. Now the function of the PROM burner is to read the memory image and burn the appropriate bits in the PROM. The memory image usually must be scanned hundreds of times in the process. Where the PROM burner connects to the development machine's bus, it addresses the RAM. On a Model I, about the lowest address you could load a program is X'5200'. What do you do when the program that is to be placed into the PROM needs to execute at X'1000'. You have the requirement to get the program into memory at an address different from where it executes.

There are various ways of accomplishing the address offset function. You could use TRSDOS 1.3's RELO command. You could use CMDFILE/PRO-CESS. You could also directly generate the file to load at X'5200' but execute at X'1000' by assembling it with EDAS using "ORG 1000H" followed by "LORG 5200H".

Let's look at what EDAS does with this set of statements. EDAS maintains an offset counter whose contents are always added to the load address when a load record is generated. If EDAS encounters an LORG, it evaluates the operand and subtracts the current program counter value from the LORG operand. This result becomes the LORG offset counter value. Thus, with the above ORG/LORG statements, the offset counter would contain a value of X'4200'.

Say that we have a situation where the "offset" is to be in effect for only a portion of the assembled program. How can we SWITCH OFF the offset? In order to switch the offset OFF, we have to return it to a zero value. Since we know its value is actually the difference between the LORG operand and the current program counter, it becomes easy to switch the LORG off by specifying a new ORG and LORG with the operand values identical. I originally told Bob that he can do this by specifying "LORG $". That statement will reset the LORG offset to zero since the "$" specifies the current program counter. However, since there may have been code that was generated but not yet output as a load block, we want to force it to be output as a load record before resetting the LORG offset. This can only occur if either the load record is at maximum size (256 bytes) or an ORG with a non-sequential address is encontered. Therefore, the proper way to turn off the LORG is to specify both an ORG and LORG with identical addresses but non-sequential to the last address assembled.

There are other reasons for using LORG. As an example, let's discuss the LDOS BACKUP program. This program contains three modules of code. The first is the module which handles all of your parameters, checks the source and destination disks, etc. The second module performs the mirror-image backup function. The third module performs the reconstruct or by-class backup function. In order to maximize buffer space for the actual backup performed, it would be ideal to relocate the mirror-image or reconstruct module to the lowest available memory. Also, a portion of the first module is not needed after the options are parsed and evaluated. Thus, we have a classic case of a large program divided into three modules, two of which must be loaded at an address different from where they will execute.

Prior to EDAS 4.1, Each module was assembled separately. The second and third modules referenced labels in the first so an EQUATE file was generated with XREF so the second and third modules could be properly assembled. After the second and third modules were assembled, they were offset using CMDFILE, and rewritten. The entire process was clumsy although easily accomplished by Job Control Language. LORG was added to EDAS specifically to handle BACKUP (and FORMAT) as a single assembly. Thus, the two modules that needed offset loading from their execution were generated as part of the single assembly process.

So much for LORG. The next discussion stems from a request forwarded by Ray D. Greet, of Lockleys Australia. Ray wanted to be able to use the *SEARCH directive of EDAS to load macros from a macro library just like *SEARCH is used to obtain assembler source subroutines need to resolve your program's references. Let me share my response to Ray.

The *SEARCH directive was implemented to permit the inclusion of library modules during the assembly process. This function corresponds to the library search process during a linkage of separately assembled relocatable modules.

The Microsoft LINK80 (L-80) performs a sequential search of a REL library file which may contain numerous modules. LINK80 performs a SINGLE sequential search. Thus, if the tenth module needed a routine located in the third module, the third module would not be linked unless referenced by some other module linked prior to reaching the tenth module. That deficiency is due to the single search of the library.

When I implemented the *SEARCH directive, I felt that it was necessary to resolve all references (i.e. externals) regardless of order within the library. To accomplish this, EDAS performs multiple searches of the library's directory until it makes a complete pass through the directory without bringing in a member. EDAS brings in a module when it finds the member name matches a symbol in the symbol table which is currently referenced but undefined. So that EDAS does not lock up in a condition whereby a module is brought in which does not define the label that caused it to be brought in, EDAS will provide a member definition error if the module does not define the label which caused its inclusion.

In the case of MACROs, you will not have a fixed label within the MACRO. By its very nature, they must have different labels on each expansion or else multiple symbol definition errors will occur. Therefore, the "self-defining symbol" problem would prevail. Another restriction with MACROs is that they need to be defined prior to their reference (how else would the assembler know how many addresses to advance the program counter?). All is not lost. EDAS can be fooled into loading a MACRO from a library. Consider that the name of the MACRO can be referenced BUT NOT VIA A MACRO CALL REQUEST. For instance, if you have a MACRO named "ADDHLA" stored in a PaDS library with member name ADDHLA, use the following statements at the beginning of your assembler program:

```
DUMMY   DEFL  ADDHLA!MACRO2!MACRO3
*SEARCH MACLIB
```

Note that you can have more than one MACRO name referenced by ORing the names. You could OR, AND, ADD, etc. The above DEFL statement can be correlated with an EXTERNAL statement on other assemblers. Yes, you will define the symbol, DUMMY, but so what? If you do not want to "waste" another label, use "@@1", "@@2", "@@3", or "@@4" which are always defined by the assembler (note: don't use the @@n labels if you are using command-line arguments P1, P2, P3, and P4).

The above will work since EDAS will not flag as an error, a reference to a symbol of the same name as a MACRO. It will flag as an error a definition of a symbol the same name as a MACRO. Good luck in your "search".

Another issue involves the *GET directive and actually also pertains to *SEARCH. One of my users had experienced difficulty when nesting *GETs. EDAS can handle up to five levels of nesting (this is version 4.1). However, when EDAS tried to read the second level, it gave this user a "Bad parameters" error. The error pertains to a bad file format. Read on to find out why this error appeared and what you can do about it.

The standard header of assembler source files is a X'D3' followed by a 6-character name. Since this convention was established by Microsoft in the original version of EDTASM and carried forward by Apparat in their disk

version of Radio Shacks EDTASM, I can only guess at its significance. For cassette files, it's useful to be able to identify the type of file. The X'D3' serves that purpose. Is it a coincidence that X'D3' is the letter "S" with the high order bit set ("S" for source). Anyway, that's the header record. Microsoft also established the assembler statement format as a 5-character line number with each high order bit set followed by a space (X'20') followed by the source statement, and terminated by a carriage return. M-80 and FORTRAN actually use a TAB in lieu of the space as used in EDTASM.

Our position is that headers and line numbers take up space and waste file loading time. EDAS 3.5 permitted you to load a file either with/without a header and with/without line numbers. This was done by specifying auxilliary characters with the "L" command. The EDAS 3.5 *GET directive required a headered and numbered file. EDAS 4.1 automatically detects the presence of a header and line numbers. The header is ascertained by examining the first byte of the file. If it's a X'D3', the file is assumed to have a header. Once the header determination is made, the first character of the first statement is examined. If it has the high order bit set, the file is assumed numbered.

When *GET is used, the determination is performed for the first file. If a *GET is nested, the determination has already been made. Thus the secondary file must have the same arrangement as the higher level file in terms of header and line numbers. This means that if the higher level file has no header nor line numbers, the nested file cannot have line numbers nor can it have a header. It is possible to redesign the assembler so that the nesting routine would save the current header/number flags and restore them at the nest exit; however, it does not appear to be too necessary. Just remember to keep all files used with *GET with the same header/number configuration. Also, if a *GET file has a *SEARCH directive, all of the library routines should be the same construction as the *GET file.

Michael D. Caubarreaux, of Houston, TX, has a problem with the conflict of pagination when using the LDOS printer filter (FORMS filter for TRSDOS 6.x). Since EDAS and XREF both do their own line counting, If you have a filter in your printer device that also does line counting and forms control, the two programs are fighting each other. Let me share my response to Michael.

Let me attempt to resolve the queries concerning paging conflicts. To begin with, both EDAS and XREF incorporate the capabilities of pagination and titling the listings that each produces (assembler listing for EDAS and cross-reference listing for XREF). In order to accomplish this listing pagination, it is necessary to keep an internal line counter. Any external filtering that does its own pagination is self defeating for the purposes of titling. Printer filters are nice to be able to supply customization to meet certain formats (such as long/short paper, wrap-around, indent, paper eject, etc.). However, when you have an application that generates a title on each page of output, either the application must take care of its own line counting, or there must exist STANDARDIZED feedback from the OS to the application.

The feedback from the system to the application is insufficiently standard, in my opinion, to deal with its use by an application (such as EDAS

or XREF). Therefore, it is necessary to reach some compromise. If you demand to be able to control your paging external to our applications, then you will have to give up the titling generated by EDAS/XREF. Patches to do this will follow this text. Alternatively, you can recognize what parameters you have set for page-length and lines printed-per-page within the printer filtering process and tell EDAS/XREF what those are. XREF provides command-line options to adjust its line-width (LEN), page-length (PAGE), and lines-per-page (LINES). EDAS provides a command (1) to set PAGE and LINES for its internal use. Why not use them?

Herewith are patches to disable paging checks:

```
Patch to EDAS/CMD Version 4.1
D04,87=C9; was C0

Patch to XREF/CMD Version 4.1a
D06,62=C9; was C0
```

I hope that these patches are sufficient to resolve what you consider to be conflicts with EDAS/XREF and printer filtering.

Note: PRO-CREATE users may utilize the following patches:

```
Patch to PRO-CREATE 4.1
D04,44=C9
F04,44=C0

Patch to PRO-CREATE 4.2
D04,8E=C9
F04,8E=C0

Patch to XREF supplied with PRO-CREATE
D06,38=C9
F06,38=C0
```

Finally, D. F. Roberts of Cirencester, England reports two problems. I won't bore you with the first one since it was corrected with EDAS48/FIX. His second problem was as follows: "The second is one which strange to say is similar to one which also existed on the Microsoft Editor Assembler Plus. Namely if one symbol is equated to a label using the EQU directive, a 'Multiple Definition Error' is incorrectly given. This only happens if the value of the label is non-zero!!. If the DEFL directive is used the error is not generated so this can be used to avoid the problem, however if there is a simple cure it would be nice to have it corrected."

D.F went on to illustrate sample code which produced the error:

```
ABC   EQU   DEF
DEF   EQU   1
      END
```

I believe that D.F has totally misunderstood the differences between EQU and DEFL because what he/she is purporting is ABSOLUTELY NO ERROR in EDAS!!! It is not a bug in EDTASM+, either. Since it is possible that someone else may also misunderstand the difference, let me emphasize. The EDAS manual states

that, "The 'EQU' pseudo-OP is the generally accepted way to define constant values for use in your program". I should emphasize, CONSTANT. The manual says of DEFL that, "This declaration is similar to the 'EQU' declaration except that the label value is permitted to change during the course of the assembly without producing phase errors".

Let's also remember that the assembler also operates in at least two passes in order to generate the object code. The first pass through the source, the assembler is building a table of all labels referenced or defined in the code. Any time that a label is referenced before it is defined, the assumed value is zero. Thus in D.F.'s sample, the assembler's first pass assigns a value of zero to ABC since "DEF" is not yet defined with a value. The second pass finds that "DEF" has a value of one and thus "ABC" is equated to one. However, since ABC was defined via an EQU pseudo-OP which must assign a CONSTANT value, the symbol's value has been redefined in error. That is what a "multiply defined symbol" error means! The fact that the "bug" didn't materialize if DEF was equated to zero is only due to the assembler's assumption of a zero value for any undefined symbol. If the sample code is revised to read:

```
DEF   EQU   1
ABC   EQU   DEF
```

the desired result will be achieved. The use of DEFL permits the redifinition of symbols with new values because that is the function of DEFL. The EQU is an "equate" of a symbol to a constant value. It is wrong to interchange the two since that will most likely result in a program bug. DEFL is usually used when you want to change the value of a symbol. I recommend that those who are unclear on the EQU/DEFL distinction re-read the manual covering the two pseudo-OPs.

GRASP

In the last issue of NOTES, I delivered a pitch that asked for you GRASP users to submit any character fonts that you have developed. MISOSYS would then build up a font disk and make it available to all GRASP users. I am happy to report that the first disk of character fonts, GRASPF1, is available. Via the courtesy of both R. W. Odlin of Sedro-Woolley, WA and Karl Hessinger of College Park, MD, GRASPF1 contains ten different character fonts. Karl has supplied CLASSIC, BIGCHAR, COMPUTER, and COMPRESS. R. W. has developed some nifty character sets complete with KSM files and JCL installation procedures. R.W.'s include Anglo-Saxon, Anglo-Saxon BOLD, Irish, Irish BOLD, Coptic, and Egyptological.

To whet your appetite, I will cite some of the text of R.W.'s letters. "The Anglo-Saxon set is designed to provide vowels with macron with CLEAR-{vowel}; the two "th" letters with CLEAR-d and CLEAR-t, on grounds of visual resemblance; the ligature "ae" with "j". In all cases, Capitalized forms may be ascertained by pressing CLEAR-{letter}, backspacing, and typing SHIFT plus the key indicated.

The great abundance of special forms of 'H' needed by Egyptologists has made it impossible for the KSM to cover all needed letters in that character

set, but it should be easy enough to find those needed (for instance, the Greek colon appearing in verbal constructions is printed with either CLEAR-I or CLEAR-N).

Had the Egyptological set been restricted to the transliteration conventions customary for Middle Egyptian, the set could well have been much shorter: but there are many off-the-wall methods of transliterating Demotic, some with an 'e', some with an equals sign, some with neither but something equally unexpected, and I have tried to satisfy as many as I could learn of.

The Irish set now provides all the accented vowels in upper-case; aspirated lower-case consonants with the mark of aspiration pleasingly placed; and aspirated Capital "S" by typing SHIFT-Z.

The bold-face sets turned out so nicely. The COPTIC/ALT filter is not an upper- and lower-case alphabet but two distinct character sets, since Coptic does not as a rule employ upper- and lower- case distinctions (nor anything much in the way of punctuation. I have added some marks for the convenience of scholars).

I am emboldened to go on, to this effect: two new Greek sets, to supplement Mr Knight's in 80 MICRO. These will be, tentatively, UNCIAL/ALT (all u.c., few 'scholastical tittle-tattles') and ALDINE/ALT (a XVIth Century font with all the squiggles and ligatures of which the period was enamored).

I need not say these are offered for public domain use, in the hopes of easing the lot of scholars in the indicated disciplines." [end of citation]

If you are a registered GRASP user and want to obtain this disk of fonts, please send $10 and request GRASPF1. The $10 charge covers the cost of the disk, mailer, processing, and shipping. MISOSYS will continue to seek character fonts to collect for building GRASPF2. I am indebted to R. W. Odlin and Karl A. Hessinger for their generosity in making these fonts available.

The next issue was raised by Mike Kaizen. Mike discovered that the use of COMPRESS in the SETMX80G/CMD program did not enable the compressed (or condensed) mode. He tracked the problem down to the values being used in SETMX80G for compressed ON/OFF at address X'5785/6'. His "MX PRINTER MANUAL with GRAFTRAX" suggests SI (SHIFT-IN or 0FH) to enable and DC2 (Device Control 2 or 12H) to disable. Somehow, values of 50H and 51H are in SETMX80G [the compressed enable/disable values are correct for SETMX100]. Mike said that his manual also indicated that 8FH and 92H are equally usable. Karl Hessinger advised me that his manual stated that in the "text" mode, bit 7 is ignored: thus 0FH and 8FH are equivalent. Appendix 4 of my MX-80 manual confirm that action.

Now the result of this discussion is to inform you of a patch needed to SETMX80G/CMD. The patch is as follows:

```
. SETG1/FIX - 11/10/83 - Applied 400140
. PATCH SETMX80G/CMD using ...
D05,A5=0F 12; WAS 50 51
. End of patch
. Note: this patch can be applied as: PATCH SETMX80G (D05,A5=0F 12)
```

While I am on the subject of modifications, let me present a slight change to the ALTCHAR/BAS program. Currently, it defaults file specifications to drive one if a drive spec is not part of the file spec. You can change this so that it doesn't default to any particular drive by changing the code:

```
IFT=OTHENFS$=IN$+"/ALT:1":... to
IFT=OTHENFS$=IN$+"/ALT"
```

By dropping the ":1" from the default string, file specifications that do not include a drive spec will not be restricted to drive one.

HELP/PRO-HELP

As you may be aware, Scott Loomer authored the Model I/III HELP package and was the inspiration for the rewrite to bring up the PRO-HELP package. Scott has worked up a filter which can be used with the Model I/III HELP package. Rather than try to use my own words to discuss Scott's latest invention, let me cite a portion of his letter.

"I thought that I'd offer the HELP/FLT as a contribution to your next 'Notes from MISOSYS'. The filter and necessary explanation are relatively short and it might prove useful to those who have bought HELP. As you'll notice in the text accompanying the hex code (in HELP/SCR), I'll be glad to provide the filter on a disk along with HELP files for LScript, VisiCalc and EDAS IV for a nominal charge.

Note that I've also sent along the latest (final) versions of the HELP/FLT and the above mentioned HELP files for you to do with as you wish. The work in developing the filter has been rewarded in that there is a fellow here in Madison that is putting together a typesetting software package that will include the filter as a means of getting on-line aid in using the programs.

Incidently, with your PRO series, what are you going to do when you run out of suitable names? Most of the ones you're using are pretty clever, but I thought that PROgeny was stretching it quite a bit!"

H E L P / F L T

The HELP filter (HELP/FLT) allows access to HELP modules from within applications that use the LDOS(tm) keyboard driver. These include LScript and Visicalc(tm) when patched to run on LDOS(tm). The syntax to install the filter is:

```
FILTER *KI with HELP/FLT (Char=ddd/x'hh'/"c")

Char - optional activation character parameter
       default is <Clear><Shift><H>

Length when installed: 350 bytes
```

To get HELP from within an application after the filter has been installed press <CLEAR><SHIFT><H>. The prompt '{h} help' will be displayed at the bottom of the screen. Type the appropriate letter(s) to indicate the help file to be accessed (e.g. to use HELPP type <P>). Next, type a left parenthesis and the name of the command, option or program to be explained. The prompt will now look like this example: '{h} helpp(build)'. The closing paren is optional. Press <ENTER> and the text will be displayed on the screen. To return control to the program, press <BREAK>. In VisiCalc, the screen display is restored by then typing </>,<->,<ENTER>.

To create the HELP/FLT, convert the hex code listing below to a binary file named 'HELP/FLT'. The BINHEX hex to binary conversion program has been printed in an LDOS Quarterly and is also included with DSMBLR III from MISOSYS.

Additional HELP modules describing the features and functions of LScript (including LScriptX), EDAS IV and Visicalc are available for $10 from MicroConsultants - Central, P.O. Box 55023, Madison, WI 53705. A copy of HELP/FLT will also be included.

LDOS is a registered trademark of Logical Systems, Inc.
VisiCalc is a registered trademark of VisiCorp

```
HELP/FLT
050648454C50202001020052 1AE601CA2053D5E5212953CD67441104543A
2501FE49202D215444224752211144227352228452218A42222453219942
227A5421144422295421 2B4422C15221BE4222C852E1CD7644C21B5321E8
007CBDCA1B53FE0020037D18057E2100006F322354E521B853CD6744E1CD
EA52DDE1DDE52A4940018C017D3217547C321854AFED4222494023DD7E01
321F54DD7E02322054E5C5011554B7ED424D44DD210153DD6E00DD66017C
B72811E55E2356EB09EBE1732372DD23DD2318E5C1D1DDE13A0F43CB6F28
0721BE432BE5DDE1DD7E01321F54DD7E02322054F3DD7301DD7202211554
EDB0FBC32D40CD9A0ACDBD0F3E3D3230413E0D323641213041CD67440102
0053C93C5442544F5458545E546E5477547E5484548C5499542E54000021
DE531803219B53CD7B44C330400A48454C502F464C54202D2048656C7020
496E766F68657220202D2056657273696F6E20352E31622C203520417567
757374203139383 30A436F70797269676874202863292031393833206279
2053636F747420412E204C6F6F6D6572202D204D6963726F436F6E73756C
74616E74730A0D0A5468697320666696C74657220666F7220696E70757420
6F6E6C79210D41637469766174696F6E2063686172616374657220076616C
7565202864656369 6D616C292003506172616D6574657206572726F7221
207B436861723D2263222F6464642F78276401C2005464277D0D43484152
20204D5243202020202042001807A1550448454C50CD0000F5FEE82802
F1C93A0E43E68F32A0543E0132244021C03F22204021A354CD674421AC54
060BCD4000384B21005211C05401E100EDB0ED73B854212D4011BA540106
00EDB03EC3322D40323040217C54222E4022314021A854C30544ED7BB854
112D4021BA54010600EDB021C0541100520 1E100EDB03E0332AC54CD4900
F13E00EFC91E7B687D2068656C70032020202020202020202000000000
0000000002020052
*D6
```

Now for the PRO-HELP users, all I have for this issue is a patch. I did a booboo when I made up the HELP(COMM) screen. Here is a patch to correct it.

```
. PHELP1/FIX - 11/04/83 - APPLIED 420025
. FIX TO CORRECT HELP(COMM)
D15,CA=04 03 26 00 13 04 02 00 28
F15,CA=02 C3 2D 01 04 03 26 00 13
. END OF PATCH
```

The bug stemmed from a trailing block of spaces in the original text file. I had used SCRIPSIT to prepare the HELP screens. It is absloutely essential that you delete-to-end-of-file after the last ENTER. Make sure that you don't make this mistake. I may look into a modification of HELPTXT/BAS to pick up this problem and correct for it.

LC/PRO-LC

Let me start out this issue's column on LC with a reminder. If you are still using LC version 1.0, then you had better send in your disk for an update to LC version 1.1. This update offer is still at no charge to you UNTIL DECEMBER 31, 1983. Beginning January 1, 1984, it will cost you $5 to update your LC disk from 1.0 to 1.1. Patches to bring your EDAS 4.1 diskette up to date for version 4.1 were included in the last issue of NOTES and are continued in this issue of NOTES. If you want us to update your EDAS, there is a $5 charge.

MISOSYS began shipments of the PRO-LC (TRSDOS/LDOS 6.x compatible release of LC 1.1) during October 1983. I am proud that this release is nearly 100% C-source code compatible to the Model I/III version in its support. To my recollection, the only difference in source code function interface is the "ploc(address)" function which had to be different in order to support the non-resident video of LDOS 6.x type systems. The PRO-LC release provides for the user-specification of the stack placement: either high or low memory. This gives the capability of running LC-generated programs with the DOS spooler active in external memory.

Now with the compiler available on the TRS-80 Models I and III, and computers running TRSDOS 6.x or LDOS 6.x, it becomes more important to stress the concepts of portability. All of you C programmers should already be aware that C is a portable language. As long as you keep machine dependent code out of your source, you can easily port a program from one computer to another. All you need is a compatible compiler.

In testing out the PRO-LC implementation, I tried to deal with compiling the programs in the LC interest group library. Unfortunately, most programs were machine dependent. Some used floating point which wasn't yet available with PRO-LC. Some programs were adapted with minor changes. However, let me point out that you really ought to pay more attention to keeping portability over slight variations of machine dependence. As a for-instance, I noticed many C-source programs with the statement:

```
#define CLS (0x1c9)();
```

Now it is a correct statement that programmers have used to clear the video screen since it defines a function which essentially "calls" the Model I or III ROM clear-screen routine. But what happens if you don't have a ROM? Such a program needs to be changed to compile under PRO-LC. It is good that the machine dependent "piece" was implemented as a "#define" which could be easily edited; however, wouldn't it be better to have used something like:

```
#define CLS puts("\x1c\x1f");
```

or include a function, cls(), which does the 'puts("\1c\1f")'? I am sure that many other examples can be illustrated; however, I hope that my point is clear without burdening you with excessive text. As LC is populated onto more machines, the more portable you keep your code, the better off you will be.

Now let me insert an errata to the PRO-LC manual [and the current printing of the LC manual] concerning the call() function. The second

paragraph of call() on page 4-36 should be: "On SuperVisor Call accessible systems, an "address" value of less than 256 will be interpreted as an SVC reference in lieu of a CALL address. The call() function will then use the "address" as the SVC number and ignore regs[0]."

It is important for LC/PRO-LC users to realize that there is an active LC Interest Group. This group was formed in 1982 and is chaired by Earl C. Terwilliger, of Akron Ohio. Earl's dedication and support of the "C"ers has resulted in a very useful focal point for the novice, intermediate, and advanced C-programmer. Earl sums up in his own words, the following synopsis of the group he started.

"An LC interest group (LCIG) has been formed as a hobby by Earl Terwilliger. The group can be used to help new users learn LC, to provide useful programs/functions for the more experienced 'C' programmer and to provide any member with some useful/functional 'C' programs. It provides a central place to collect and distribute LC source programs and related files contributed by its members. The membership fee is $5. The group serves as a non-profit vehicle to share ideas and programs. Programs submitted by members are kept on diskettes. There are now 4 diskettes (CSAMPLER, LCIG8301, LCIG8302, and LCIG8303) [I wouldn't be surprised if by now, Earl has collected material sufficient for a fifth diskette]. A copy of any of the standard LCIG diskettes will be sent to any member for $5 per diskette or $2 per diskette if the member sends in their own media. Members are also encouraged to contribute programs by sending them on a diskette to the group. The address for "The LC Interest Group" is as follows:

The LC Interest Group
c/o Earl C. Terwilliger Jr.
647 North Hawkins Ave.
Akron, Ohio 44313
Phone (216) 836-7389"

Let me go through some of the customer queries received concerning LC that may help you in avoiding problems. The first was submitted by James Campbell, of Aloha, OR. I appreciated the depth with which James explored the problem that he was having. Because of this, I would like to share his letter. James wrote: "I have to tell you how much I enjoy using Elsie, although I am new to the product. I think I have found a bug. You will have to bear with my explanation, since C is not my native language and Z-80 assembler is something I do not know well.

My problem started when I keyed in your sample from Appendix D-9, to show the use of call() [note: this sample program first appeared in the 2nd printing of the LC manual. The example appeared in the LC errata included with the first printing of the manual]. The program worked to a point, telling me the current status of my drives--through drive 8 and climbing. I cut the power about drive 156 or so. Obviously the test was not working [specifically, the test is 'for (d=0;d<8;++d)'].

After checking my text several times, I got an assembler listing and went into DEBUG. I soon determined that the value being used for the drive check was not the same one being used for the test against the value 8. I think the problem is the translation of the inequality test 'd<8'. To get back the current value of D$, the instruction at 01410 should be $GETW D$,

not LD HL,D$. I can say that with some confidence, because I used DEBUG to change the value at 5231 from 210200 to 2A6A5F and the program executed successfully.

Then again, I may be wrong. In any event, I hope the enclosed documents will help you determine where my problem lies."

James included printouts of his source and the assembler listing. He was right about the need for "$GETW LABEL", which is used to fetch a static integer. Assembler code of the form, "LD HL,LABEL" is used to obtain the pointer to a function. So did LC go wrong? My first gut feeling is that it must be a source code error - don't assume a bug in LC. Armed with this hypothesis, I easily discovered the bug. In the "for" statement mentioned above, the inequality was entered as "D<8" instead of "d<8". Since LC is case sensitive, "D" and "d" are two different variables. However, since the assembler requires that all variables be in upper case, LC converts its symbols on output to upper case. Therefore, if you look at an assembler listing, you cannot tell the difference.

The significance of James's mistype goes beyond just the simple error. It sheds light on the importance of keeping all variables in lower case. Recognizing that the compiler is case sensitive, you could have two different variables spelled the same but different cases! At least as far as the compiler is concerned - but not the assembler. It has always been recommended that "defines" use upper case.

I felt it interesting to explore the reason why the compiler generated the "LD HL,D$" instead of the "$GETW D$" macro. Kernighan & Ritchie state on page 209, "There are only two things that can be done with a function: call it, or take its address. If the name of a function appears in an expression not in the function name position of a call, a pointer to the function is generated. ... [the function] must be declared explicitly in the calling routine since its appearance ... was not followed by (." LC doesn't require the explicit declaration; therefore, an identifier which has not been defined, is interpreted as a function reference whether it is followed by a left parenthesis or not. It follows from K&R on page 68 that, "If a name which has not been previously declared occurs in an expression [omitting 'and is followed by a left parenthesis'], it is declared by context to be a function name. Furthermore, by default, the function is assumed to return an int." The LC manual states that, "A function declared within another function body is assumed to have a storage class of external. The compiler regards the declaration as if an 'extern' statement preceded it."

What we then have is an assumed declaration of the form: "extern D();". When "d" was declared, LC converts this to "d$" to avoid assembler restrictions on labels. Since "d" was declared as an "int", LC referenced the variable via the $GETW and $PUTW macros. The assembler requires upper case labels so LC passes "D$" to the /ASM file. The macro argument converts the "D$" to an offset from zero and adds the data origin. When "D" was referenced, it was converted to "D$" to avoid assembler restrictions. Since "D$" was interpreted as an external function name, LC referenced the variable strictly by its name. The assembler did not reject this as an undefined symbol error since "D$" was actually defined based on the original "d". However, "D$" by itself was only a relative offset from zero. If you examine the original C declaration of "int rc,d;", you will understand that the value

of "RC$" should be zero while the value of "D$" should be two. That's exactly what occurred in James's case. The data origin was 5F68 resulting in 5F6A when "d$" was referenced (D$+$$STORG) but only 2 when referencing "D$".

The next problem to discuss was submmitted by P. Bailey, of Derby, England. You should be able to gather the queries raised by Mr. Bailey from my response to him. It went as follows: The "problems" you addressed in your letter postmarked May 23rd were analyzed by our "C" experts. You may be surprised at their conclusions.

The first problem, that of being unable to effect a tab using the "\t" or "\x09" escape sequences, is one of misunderstanding. You actually did insert a tab character in the string. What you may fail to realize is that there is nothing in the output stream that will EXPAND the tab. You can rest assured that the tab was inserted by redirecting the output to a disk file and then listing the file in hex (or in ASCII with the TAB=ON parameter).

In order to assure ourselves of the above, the following program was compiled and run:

```
main()
{ puts("This is a tab ->\t<- character\n");}
```

The compiled and assembled tabtest/ccc program performed as expected. [as an aside, under the LDOS/TRSDOS-type systems, the video driver does not expand the TAB character. This can be accomplished by the LIST command, however, when listing a file]

Your second problem represented another degree of confusion. The illustrated program fragment you supplied can be reduced to the following:

```
main()
{ loop:
        c=getchar();
        goto loop;
}
```

Notice that you have no test for end-of-file in the getchar() function invocation. It is true that the program will pause awaiting a character upon reaching the getchar() function. Even though getchar() will be receiving its input from the keyboard (unless redirected), it nevertheless is still possible to transmit an EOF. This is performed by depressing the BREAK key. Once the BREAK key is depressed, getchar() will continue to return EOF regardless of any further keyboard operation. One possible recoding of this loop is as follows:

```
main()
{ loop:
        if ((c=getchar()) != EOF )
        goto loop;
        else exit;
}
```

The second problem arose from a need to detect an ABORT condition to return to a primary menu prompt. This was clarified in a subsequent letter

which follows:

Without going into a great explanation, your use of BREAK to escape into a menu in a "word processor" type of C application can be accomplished easily. Under C, one needs to be able to maintain device independence; thus, the BREAK key was selected to provide an end-of-file. Considering in your application that the primary means of input would be by keyboard, your need to regain keyboard control after a getchar() and BREAK-EOF is quite important. An elegantly simple solution would be to re-open the standard input device (stdin) with fopen() when the EOF indication is encountered. Then have the C program go to the menu mode.

The reopening of stdin also serves to support the redirection of standard input at the command line to start the input from a disk file and have it automatically switch to the keyboard upon reaching the EOF from the disk file. In any event, all you then need is a command to exit your program.

If our NOTES readers have any other solutions, perhaps they may be submitted.

R. D. Greet, of Lockleys, Australia reports that, "A book that has turned up locally which I would recommend for first reading on C is THE C PRIMER by Hancock & Krieger (Byte Books). This gem made clear in my mind aspects of C that was causing difficulty."

Concerning the issue of books, I have been advised that Paul Chirlian's book, "Introduction to C", will soon be published by Matrix Publishing (Matrix is closely associated with Dilithium Press). Paul, as you may know, is the author of many books concerning microcomputers including the book on Microsoft FORTRAN.

I thought I would relate the findings that Truman Krumholz discovered concerning the "Sieve" program listed in the last issue of NOTES. He writes:

"In Issue 1 of NOTES FROM MISOSYS is the sieve program in C which was taken from the January BYTE. As the program appears, my TRS-80 Model I executes the program in 117 seconds. This agrees with your time taking into consideration different clock speeds. However, if the line containing the main statement and the integer declaration line are reversed in order, the program runs on my Model I in 68 seconds. This is a significant improvement in speed. Being a beginner with C, I tried this because this is the order in which the statements appear in the article in BYTE. [actually, Truman is in error here. The statements are ordered as he states but only in the PASCAL version of the sieve program. The C version orders the statements as shown in NOTES].

For your interest, I also have a home-brewed LNW-80 Model I. The only part of this machine that is LNW are the boards. My machine runs at three speeds, 1.77 mHz, 4mHz, and 5.33 mHz. It also has the capability of switching out the ROM and the ROM wait states and substituting RAM. Running this program on the LNW at 5.33 mHz, it executes in 22 seconds. Both the improved program and the version using multiplication as listed in January BYTE run on my LNW in 18 seconds at 5.33 mHz.

I am enjoying LC very much, but I wish I had more examples to look at. This is the best way for me to learn a language. I copy programs then modify them to see what happens. By the way, my best versions of FORTH, writtern by a local fellow here, ran the program listed in BYTE in 54 seconds (clock speed at 5.33 mHz). I clocked MMSForth at 188 seconds on my TRS-80 Model I and 62 seconds on the LNW at 5.33 mHz."

The reason why Truman detected a significant improvement in execution speed was due to the pronounced effect of re-ordering the declaration of variables. The variables i, prime, k, count, and iter are local to main() when defined within main(). This means that they are referenced from the stack pointer. If their declaration precedes main(), they are global static variables having a fixed address. In order to access an integer that is local, the following code must be performed:

```
LD   HL,OFFSET      ;OFFSET is variable based on current SP
ADD  HL,SP          ;Calculate actual address of storage
CALL @GINT          ;Obtain value
```

The @GINT subroutine performs: \LD A,(HL)\INC HL\LD H,(HL)\LD L,A\RET\. The access of a static global requires only, "LD HL,($$STORG+#NAME)", with the argument referenced at assembly time. If we evaluate t states, I calculate that the local takes 72 t-states versus 16 for the global. At 2 mHz, a t-state is 0.5 microseconds. Thus the additional 56 t-states to access a local take 28 microseconds. That's PER ACCESS. If your program has hundreds of thousands of accesses of its variables, then the time difference is significant.

The utility of local variables arises when dealing with recursive functions. K&R state that "when a function calls itself recursively, each invocation gets a fresh set of all the automatic variables, quite independent of the previous set." This is not true if only statics are used. There are some very definite advantages for using automatic variables - but the advantage is not speed. FORTRAN, for example, has no such thing as an automatic variable. All variable addresses are fixed at compile (and link) time. The same thing is true of compiled BASICs. C gives you a choice. Thus if you are interested in speed, it may be to your advantage to define your variables outside of main().

Darach Foskett of Beecher Falls, VT forwarded an interesting problem which had me going around for awhile. It originates from the text in K&R on page 207 and again on page 86 which concerns token replacement associated with #define and macro substitutions. I must admit, I had to reread K&R a number of times on this point to understand the discussion. As Darach stated, his was not so much an LC question since LC does not support macro substitutions but a general C question. Herewith my response:

The problem stems from a possible misinterpretation of the K&R statement that, "text inside a string or character constant is not subject to replacement". This does not mean that 'identifier arguments' within the #define macro are not subject to replacement but rather if an 'identifier' happens to appear in the program, then it will not be substituted. K&R provide a very brief example of this on page 86 where they illustrate that a '#define YES whatever' will not be substituted for the 'YES' within the C statement, 'printf("YES")'.

On page 9 of THE C PUZZLE BOOK, it uses the following #define statement:

```
#define PRINT(int) printf("int = %d\n",int)
```

The character sequence "int" appears in the #define three times. The first is as an argument identifier. The second is within the printf() string argument. The third happens to be the second argument of the printf(). According to my interpretation of K&R, all three appearances of 'int' will be replaced by the token string. Thus, a statement such as 'PRINT (x < y ? y : x);' will find the 'PRINT' substituted according to the #define to read as follows:

```
printf(" x < y ? y : x  = %d\n", x < y ? y : x );
```

This statement computes to the result printed in PUZZLE. If I had coded a statement, 'printf("This is a PRINT line!\n");', the 'PRINT' within my statement would not be subject to any replacement by the PRINT defined by the #define pre-processor statement.

The answer to your question concerning the use of "int" in the #define is that the formal macro arguments are totally independent of all other parts of the source file. You could have 15 #defines - all using the same dummy parameter name. It is sometimes confusing to look at a program that repeats the use of the same name because one sometimes forgets which edition of the name one is referring to. Such is the case with the reuse of identifiers when you open a new block (i.e. { code }). Although you can repeat the name for a new local variable, it can be confusing to the programmer.

The next topic is one that I am sure the Model I/III LC users will love to read. Let me give you a little background. All of the LC development work done at the MISOSYS office has been on hard drives. In the past, a Lobo 5-meg drive as well as a Radio Shack 5-meg drive have been used. These drives are very quiet in seeking. In fact, you really cannot hear the movement of the head if you are listening in a normal level of room noise. Even when LC was invoked from floppies, the Tandon drives in use were quiet seekers. In fact, the only noisy drives I recollect were the MPI drives.

In any event, when floppies were used, I always was aware that there was a perceptively long time when a program loaded before it seemed to do anything. This effect was just in the back of my mind and the quietness of the various drives in use masked the "excessive" seek activity actually taking place.

I presently use a VR Data package which uses Syquest drives with buffered seek. Although the fan in this cabinet is quite noisy, the drive itself exhibits a kind of "bubbling" sound when the drive is rapidly stepped over distances. Since the PRO-LC package was developed using the Model 4 and VR Data hard drive combination, I became noticably aware of the great number of seeks occuring not only when LC first started, but programs compiled with LC.

Now aware that something bothered me, I decided to look into the matter. It did not take too long to discover that the culprit was the effect of opening the three standard files. Not that opening in itself was the root cause, but rather that @FSPEC followed by @OPEN (or @INIT) was successively

invoked for each of the three standard files. Now @FSPEC is in SYS1/SYS whereas @OPEN and @INIT are in SYS2/SYS. Thus, executing a loop that opened the three files successively resulted in the access and loading of SYS1\SYS2\SYS1\SYS2\SYS1\SYS2. This means directory access as well. When a program first executes, SYS2 is the resident overlay since the DOS just had to OPEN the program file for loading.

After discussing the "problem" with Jim Frimmel (I also talked it over with Karl Hessinger), I decided to add an internal FSPEC-type routine to LC. This was introduced first into PRO-LC at release time. Because I feel that the improvement in apparent (and actual) execution speed is significant, I wanted to apply the same technique to LC. Thus, I have the following procedure to recommend. If we do an LC 1.2 release, this will become a part of it. For now, you can easily achieve the same improvement by adopting the following procedure:

```
. LC SPEED IMPROVEMENT
. This patch modifies LC/CMD so as to improve the
. speed at which the compiler initializes the
. standard files.
. PATCH LC/CMD.LC USING ...
D6A,08=2F BE; WAS 1C 44
D6E,0B=E5 D5 7E FE 21 38 0F FE 61 38 06 FE 7B 30 02 EE
D6E,1B=20 12 23 13 18 EC 3E 03 12 D1 E1 C9; WERE ZEROES
. End of patch
.
. This patch modifies LC/LIB so as to improve the
. speed at which compiled programs initialize the
. standard files.
. PATCH LC/LIB.LC USING ...
D2C,BA="@SPEC"; WAS "441CH"
. End of patch
.
. The following routine must be added to LC/ASM
. and should be placed to follow the statement:
.     *SEARCH LC/LIB
. This can be accomplished by loading LC/ASM,
. inserting this routine, then the updated file
. is saved with: W LC.LC
.
        IFDEF   FOPEN
*M
@SPEC   PUSH    HL              ;Save registers
        PUSH    DE
$?1     LD      A,(HL)          ;Exit on space or less
        CP      33
        JR      C,$?3
        CP      97              ;Convert l/c to U/C
        JR      C,$?2
        CP      123
        JR      NC,$?2
        XOR     32
$?2     LD      (DE),A
        INC     HL              ;Bump pointers
        INC     DE
```

```
            JR      $?1             ;Loop until exit char
    $?3     LD      A,3             ;Terminate with ETX
            LD      (DE),A
            POP     DE              ;Restore regs
            POP     HL
            RET
            ENDIF
```

The next thing to offer is a program submitted by Karl Hessinger, of College Park, MD. Karl wrote a conversion program which aids in the conversion of Model I/III BASIC programs to Model 4 BASIC. Not only is this program useful in what it does, it is one more source program to help you learn C. Although it is long, it may prove beneficial. Karl's program works from compressed BASIC program files rather than ASCII program files.

```
#include stdio/csh
#option redirect OFF
FILE *fp1, *fp2;
char *keyword[128];
int remflag,strflag;
main(argc,argv)
int argc, *argv;
{  char c;
   if ( argc != 3 )
      abort("\n** Parameter error **\n");
   puts("\x1c\x1f");
   fp1 = getfile(*++argv,'r');
   fp2 = getfile(*++argv,'w');
   if ((c=getc(fp1)) != '\xff')
      abort("Not a compressed BASIC file!");
   setup();
   while ( !end() )
      {  linenumber();
         remflag = strflag = FALSE;
         while ((c=getc(fp1)) != '\0' )
            if ( remflag || strflag || c != ' ' )
               {  if ( c > 127 && !strflag )
                     {  if (token(c))
                           goto lineend;
                     }
                  else
                     out(c);
                  if ( c == '"' )
                     strflag = strflag ? FALSE : TRUE;
               }
         lineend : if ( strflag )
            out('"');
         out('\n');
      }
   fclose(fp1);
   fclose(fp2);
   exit(0);
}
linenumber()
{  int num;
```

```
   num = getc(fp1);
   num += getc(fp1)*256;
   fprintf(fp2,"%d ",num);
   printf("%d ",num);
   return(0);
}
end()
{  int flag; char c;
   if ((c = getc(fp1)) == EOF)
      return TRUE;
   flag = c;
   if ((c = getc(fp1)) == EOF)
      return TRUE;
   flag += c * 256;
   if ( flag == 0 )
      return TRUE;
   else
      return FALSE;
}
token(val)
char val;
{  char c; int pos,flag;
   fputs(keyword[val-128],fp2);
   puts(keyword[val-128]);
   if ( val == 184 )      /* Strip value after CLEAR */
      {  while ((c=getc(fp1)) != ':' && c != '\0' )
            {}
         if (c=='\0')
            return(TRUE);
         else
            {  out(c);
               return(FALSE);
            }
      }
   if ( val == 178 )      /* PRINT change PRINT@ values */
      {  while ((c=getc(fp1)) == ' ')
            {}
         if ( c == '@' )
            {  out(c);
               pos = 0;
               flag = FALSE;
               while (isdigit(c=getc(fp1)))
                  {  flag = TRUE;
                     pos *= 10;
                     pos += c - '0';
                  }
               if (flag)
                  {  fprintf(fp2,"(%d,%d)",pos/64,pos%64);
                     printf("(%d,%d)",pos/64,pos%64);
                  }
            }
         if ( c == '\0' )
            return(TRUE);
         if ( c > 127 )
            return(token(c));
```

```
         else
            {  if ( c=='"' )
                  strflag = strflag ? FALSE : TRUE;
               out(c);
            }
      }
   if ( val == 147 || val == 251 )
      remflag = TRUE;
   return(FALSE);
}
getfile(fname,mode)
char *fname, mode;
{   char *fp;
    if (mode == 'r')
    {   if ((fp=fopen(fname,"r")) == NULL)
        {   printf("Open error - %-20s\n",fname);
            exit(1);
        }
        else return fp;
    }
    else if (mode == 'w')
    {   if ((fp=fopen(fname,"w")) == NULL)
        {   printf("Open error - %-20s\n",fname);
            exit(1);
        }
        else return fp;
    }
}
out(c)
char c;
{  putchar(c);
   if (c != putc(c,fp2))
      {  puts("File I/O error!\n");
         fclose(fp1);
         fclose(fp2);
         exit(1);
      }
   return(0);
}
setup()
{  keyword[0] = "END ";          keyword[1] = "FOR ";
   keyword[2] = "RESET";         keyword[3] = "SET";
   keyword[4] = "CLS ";          keyword[5] = "CMD ";
   keyword[6] = "RANDOM ";       keyword[7] = "NEXT ";
   keyword[8] = "DATA ";         keyword[9] = "INPUT ";
   keyword[10] = "DIM ";         keyword[11] = "READ ";
   keyword[12] = "LET ";         keyword[13] = " GOTO ";
   keyword[14] = "RUN ";         keyword[15] = "IF ";
   keyword[16] = "RESTORE ";     keyword[17] = " GOSUB ";
   keyword[18] = "RETURN ";      keyword[19] = "REM ";
   keyword[20] = "STOP ";        keyword[21] = " ELSE ";
   keyword[22] = "TRON ";        keyword[23] = "TROFF ";
   keyword[24] = "DEFSTR ";      keyword[25] = "DEFINT ";
   keyword[26] = "DEFSNG ";      keyword[27] = "DEFDBL ";
   keyword[28] = "LINE ";        keyword[29] = "EDIT ";
```

```
    keyword[30] = "ERROR ";      keyword[31] = "RESUME ";
    keyword[32] = "OUT ";        keyword[33] = "ON ";
    keyword[34] = "OPEN ";       keyword[35] = "FIELD ";
    keyword[36] = "GET ";        keyword[37] = "PUT ";
    keyword[38] = "CLOSE ";      keyword[39] = "LOAD ";
    keyword[40] = "MERGE ";      keyword[41] = "NAME ";
    keyword[42] = "KILL ";       keyword[43] = "LSET ";
    keyword[44] = "RSET ";       keyword[45] = "SAVE ";
    keyword[46] = "SYSTEM ";     keyword[47] = "LPRINT ";
    keyword[48] = "DEF ";        keyword[49] = "POKE ";
    keyword[50] = "PRINT ";      keyword[51] = "CONT ";
    keyword[52] = "LIST ";       keyword[53] = "LLIST ";
    keyword[54] = "DELETE ";     keyword[55] = "AUTO ";
    keyword[56] = "CLEAR ";      keyword[57] = "CLOAD ";
    keyword[58] = "CSAVE ";      keyword[59] = "NEW ";
    keyword[60] = "TAB(";        keyword[61] = " TO ";
    keyword[62] = "FN ";         keyword[63] = "USING ";
    keyword[64] = "VARPTR ";     keyword[65] = "USR ";
    keyword[66] = "ERL ";        keyword[67] = "ERR ";
    keyword[68] = " STRING$";    keyword[69] = " INSTR";
    keyword[70] = " POINT";      keyword[71] = " TIME$ ";
    keyword[72] = " MEM ";       keyword[73] = " INKEY$ ";
    keyword[74] = " THEN ";      keyword[75] = " NOT ";
    keyword[76] = " STEP ";      keyword[77] = "+";
    keyword[78] = "-";           keyword[79] = "*";
    keyword[80] = "/";           keyword[81] = "^";
    keyword[82] = " AND ";       keyword[83] = " OR ";
    keyword[84] = ">";           keyword[85] = "=";
    keyword[86] = "<";           keyword[87] = " SGN";
    keyword[88] = " INT";        keyword[89] = " ABS";
    keyword[90] = " FRE";        keyword[91] = " INP";
    keyword[92] = " POS";        keyword[93] = " SQR";
    keyword[94] = " RND";        keyword[95] = " LOG";
    keyword[96] = " EXP";        keyword[97] = " COS";
    keyword[98] = " SIN";        keyword[99] = " TAN";
    keyword[100] = " ATN";       keyword[101] = " PEEK";
    keyword[102] = " CVI";       keyword[103] = " CVS";
    keyword[104] = " CVD";       keyword[105] = " EOF";
    keyword[106] = " LOC";       keyword[107] = " LOF";
    keyword[108] = " MKI$";      keyword[109] = " MKS$";
    keyword[110] = " MKD$";      keyword[111] = " CINT";
    keyword[112] = " CSNG";      keyword[113] = " CDBL";
    keyword[114] = " FIX";       keyword[115] = " LEN";
    keyword[116] = " STR$";      keyword[117] = " VAL";
    keyword[118] = " ASC";       keyword[119] = " CHR$";
    keyword[120] = " LEFT$";     keyword[121] = " RIGHT$";
    keyword[122] = " MID$";      keyword[123] = " ' ";
    return;
}
abort(msg)
char *msg;
{  fputs(msg,stderr);
   putc(eol,stderr);
   exit(1);
}
```

Finally, I discussed this last idea with Jim Frimmel and he concurred in its appearance in NOTES. Although C is considered to be a portable language, situations arise whereby it would be useful to be able to write C-source in conditional blocks - much like the assembler permits conditional pseudo-OPs.

There do exist conditional processes in a full C preprocessor. Although not implemented in Elsie, they are #if, #ifdef, #ifndef, #else, and #endif. Now all of these "conditional" operations are supported in EDAS. Since Elsie supports an "#option variable" specification which can define "@_variable" and set it to TRUE or FALSE (actually ON or OFF as used in the assembler), it becomes a simple matter to construct conditional blocks of code that are not conditional as far as the compiler is concerned but rather passed through to the assembler for evaluation of the conditional operations.

To achieve this in the C source, assume that you define an assembler symbol of "@_TEST" an set it to -1 via the C statement, "#option TEST ON". Next, a block of C-source code that is to be included in the resultant program would be surrounded by the statement pair,

```
#asm                          #asm
<TAB>IF @_TEST        and     <TAB>ENDIF
#endasm                       #endasm
```

If you don't want to include the block, specify "#option TEST OFF". Of course, there are variations such as using, "IFDEF @_TEST" and either specifying the #option or not.

This procedure may not be standard K&R; however, under the present implementation of Elsie, it does enable conditional C source.

Speaking of Jim Frimmel, it turns out that the previous item was not the final part of this issue's LC column. Jim has supplied us with the first of his "C'erious Subjects" columns. Although it may be a little advanced for the beginning LC user, if you all spend a little time to explore Jim's column, I am sure you will gain a better understanding of LC.

C'erious Subjects - by Jim Frimmel

Hiya!!! I'd like to discuss with you a few subjects that have been brought to my attention by LC users, and show you some techniques that may be useful to have in your bag of LC tricks.

The first thing is to show you a way to shorten the time required to assemble LC programs. The method shown is illustrated for Model I/III LC; however, it can be easily adapted to PRO-LC. The idea is to pre-assemble the library(s), so that you can use an equate file during each assembly. Then the assembled program is appended to a copy of the pre-assembled code, resulting in a runnable program. This technique saves me 30 seconds on every compile/assembly during development, on a MAX-80 with 8" disks. It will probably save one minute or so on model I/III's. For your final version of a program, use the regular LC/JCL file to get the most compact code possible.

Listing 1 shows the assembly file, LCLIB/ASM, that I used to pre-assemble the LC standard library. The REF macro references the symbol

following it, so that EDAS IV will retrieve it from the library. You can adapt this to pre-assemble any library module or combination of library modules. You don't have to pre-assemble all of a library, and you can pre-assemble ANY function, whether compiled LC functions or assembly language functions that you've written yourself. The LC standard library is trickier than other libraries, so I recommend that you pre-assemble the entire library, as shown here.

Note also that the options that are set in the LCLIB/ASM file, plus the defaults in LCMACS/ASM, will determine the options in effect in the pre-assembled library. Once assembled, the options cannot be changed, so if special options are needed, either pre-assemble a separate module with those options, or don't use this technique.

The transfer address of the pre-assembled /CMD file must be removed so that LC program /CMD files may be appended to it. I actually had to write a program to do this, since I couldn't figure out a way to do it using LDOS's extensive utilities. Why, you may ask, don't I just use Command File Utility to combine the two command files together? I found that it added 15 seconds (MAX-80 time) to the process, just because of all the friendly prompts in CMDFILE. The answers require quite a few JCL lines. Over in listing 2 is the program, XTRA/CCC, to strip the transfer address. Compile and assemble XTRA/CCC using the normal LC/JCL file.

At this point you are ready to actually generate the LCLIB/CMD and LCLIB/EQU files. The JCL file in listing 3 will do all but one step. If you have already typed in and compiled XTRA/CCC, and have typed in LCLIB/ASM, you're ready to DO LCLIB/JCL. The end products will be LCLIB/CMD, the pre-assembled and stripped version of the LC library, and LCLIB/EQU, an equate file that will define addresses of library functions when assembling LC programs for use with LCLIB/CMD. One comment about the LCLIB/JCL file: the I/O redirection in the XTRA command line is a little strange, in that standard input and standard output are the SAME FILE! The reason that I can get away with this is that XTRA writes out exactly what it reads in, minus four bytes. As a general rule, this can be done as long as the program writes out one byte for each one read, and shortens the file, or leaves it the same length. In any other case, results are usually unexpected.

Not all the symbols in the equate file just generated will be useful. No temporary labels, local labels, macro names, etc. need be included. Try and get the equate file looking like the one in listing 4, by going in under EDAS and deleting the unnecessary lines, and adding the few new ones.

We now have to create a modified version of LC/ASM to GET the equate file. The equate file must be included at the beginning of LC/ASM. The actual program being assembled must then be ORG'ed past the end of the pre-assembled code. Since the program may vary in size, the pre-assembled module should reside below the rest of the program. Listing 5 shows my QLC/ASM file ("Quick Levelance Cod").

We can now assemble a program using QLC/ASM. A new JCL file (listing 6) is needed if we're not going to issue the commands manually every time. In addition to the normal stuff LC/JCL does, it must also combine the two /CMD files into the final, runnable form. Note that the assembly's /CMD output file is sent to a temporary file. Then the copy and append are used to

generate the final /CMD file.

One thing that you should notice about the new QLC/ASM, and the pre-assembled standard library: The variables used by the library are placed in upper memory, at OC000H. This is so that @PROGEND (which determines where alloc() and sbrk() reserve memory) will be defined above all code and variables. This imposes some limitations on the programmer. If your program is greater than about 24K bytes long, the symbol, $$STORG (in LCLIB/ASM), must be equated to a higher value to make room for the larger program. On the other hand, if your program uses large amounts of space reserved by alloc() and sbrk(), $$STORG may be decreased to make more room available to alloc() and free(). In any case, be sure that your program, once assembled, does not overlap the variables used by LCLIB, which begin at $$STORG (in LCLIB/ASM).

Well, it seems that some of us actually write programs with LC that are pushing the limits of the memory size in the TRS-80! I've been asked to explain how to share variables between programs, so I guess someone's program has gotten big enough to need splitting up. Well, this one is easy, since all you have to do is make sure that all programs have the same variables, at the same place. Also, if initialization must be done on variables, it must take place in the first program to be run, and the ZVAR option should be OFF for all subsequent programs. Also, any space gotten by calls to alloc() will no longer be available when the next program runs. This includes open files as well. All files must be re-opened when the next program runs.

To implement this, first collect all variables that are to be shared into a separate file that can be included into all programs. You don't have to put all variables in this file, just the ones to be shared. This file must be #include'd in each program BEFORE ANY OTHER VARIABLES ARE DECLARED! If you define other variables first, then the shared variables will be at the wrong place. Finally, compile and assemble all the programs and check the address given for $$STORG in each, which is the base of variables in the program. Choose the highest value, then add some to allow for growth. Now to make it really happen, make a copy of LC/ASM, and modify the line that equates $$STORG to the program counter, to set $$STORG to the value you have chosen. Reassemble all the programs using the new /ASM file, and you should be all set.

That's all for this issue. Look for details on using true overlays in the next issue of NOTES - if I can get to it. 'Till then, as Olivia says, "Let's get C'erious!".

LISTING 1

```
;   LCLIB/ASM  -  pre-assembly of LC standard library
;
*GET LCMACS
;
;***  If you wish any different options, set them here *
@_ZVAR DEFL    0                ; don't initialize var's
;***    Macro to reference a name  ***
REF    MACRO   #NAME
$TEMP  DEFL    #NAME
       ENDM
;
```

```
;*****   NOTE: This next statement determines how much
;*****         space there is for the program, and
;*****         divides the program (with var's) from
;*****         the areas for alloc() and free().
;*****         Change the value as needed.
@PROGEND        EQU     0C000H          ;24K for program
;***  REFerences to all LC standard library functions ***
;
        ORG     5200H
        REF     @AND
        REF     @ASL
        REF     @ASR
        REF     @COM
        REF     @DIV
        REF     @EQ
        REF     @GE
        REF     @GINT
        REF     @GO
        REF     @GT
        REF     @LE
        REF     @LT
        REF     @MULT
        REF     @NE
        REF     @NOT
        REF     @OR
        REF     @PINT
        REF     @SUB
        REF     @UDIV
        REF     @UGE
        REF     @UGT
        REF     @ULE
        REF     @ULT
        REF     @XOR
        REF     ATOI
        REF     EXIT
        REF     FGETS
        REF     FOPEN
        REF     FPRINTF
        REF     FPUTS
        REF     GETC
        REF     GETCHAR
        REF     GETS
        REF     ISALPHA
        REF     ISDIGIT
        REF     ISLOWER
        REF     ISUPPER
        REF     ITOA
        REF     ITOX
        REF     MOVE
        REF     PRINTF
        REF     PUTC
        REF     PUTCHAR
        REF     PUTS
        REF     STRCAT
        REF     STRCMP
```

```
       REF     STRCPY
       REF     STRLEN
       REF     TOLOWER
       REF     TOUPPER
       REF     XTOI
       REF     ZERO
;*** now get all the modules ***
$$STEMP        DEFL    0               ;init rel. storage
*SEARCH LC/LIB
$$STORG        EQU     $               ;storage areas here
@LCLIBEND      DEFL    $$STORG+$$STEMP ;label to ORG to
       END     402DH           ;back to LDOS if exec'ed
```

LISTING 2

```
#include stdio/csh
char buf[4], c;
main()
{   /* short filter to strip last 4 bytes from stdin */
    /* written to strip transfer address from load   */
    /* modules.      Jim Frimmel   11/2/83           */
    buf[0] = getchar();
    buf[1] = getchar();
    buf[2] = getchar();
    buf[3] = getchar();
    while ((c = getchar()) != EOF)
    {   putchar(buf[0]);
        move(buf+1,buf,3);
        buf[3] = c;
    }
}
```

LISTING 3

```
. LCLIB/JCL - Pre-assemble the LC standard library
.  Format: DO LCLIB (DRIVE=D)   DEFAULT: DRIVE=1
//IF -DRIVE
//ASSIGN DRIVE=1
//END
EDAS (JCL,ABORT)
L LCLIB
A LCLIB:#DRIVE#,LCLIB:#DRIVE# -SL -XR -NM -NC
B
XREF LCLIB:#DRIVE# (EQU)
XTRA <LCLIB/CMD:#DRIVE# >LCLIB/CMD:#DRIVE#
. Pre-assembly completed
.  Now edit the file LCLIB/EQU to prepare it for use
```

LISTING 4

```
@AND    EQU     60D4H           @ULE    EQU     6162H
@AP     EQU     5C0BH           @ULT    EQU     6155H
@APMOD  EQU     522CH           @WRMOD  EQU     522BH
@ASL    EQU     60DBH           @XOR    EQU     6174H
@ASR    EQU     60E3H           ALLOC   EQU     5ACBH
```

```
@CMP    EQU     613FH           ATOI    EQU     5C37H
@COM    EQU     60F8H           EXIT    EQU     5300H
@DIV    EQU     618FH           FCLOSE  EQU     5A67H
@EQ     EQU     6119H           FGETS   EQU     585AH
@ERRET  EQU     60B6H           FOPEN   EQU     598EH
@FCLS9  EQU     5A84H           FPRINTF EQU     5378H
@FCNT   EQU     0010H           FPUTS   EQU     5807H
@FTEND  EQU     5222H           FREE    EQU     5B4EH
@FVTBL  EQU     5202H           GETC    EQU     590AH
@GE     EQU     6133H           GETCHAR EQU     536FH
@GETFV  EQU     60BCH           GETS    EQU     5346H
@GINT   EQU     6107H           ISALPHA EQU     5FE3H
@GO     EQU     522DH           ISDIGIT EQU     5FFBH
@GT     EQU     6125H           ISLOWER EQU     600BH
@GTFV1  EQU     60C4H           ISUPPER EQU     601BH
@LCLIBEND DEFL  6211H           ITOA    EQU     5CF1H
@LE     EQU     612CH           ITOX    EQU     5E2DH
@LOMEM  EQU     5200H           MOVE    EQU     6084H
@LT     EQU     6139H           PRINTF  EQU     5325H
@MOD13  EQU     60B0H           PUTC    EQU     5957H
@MULT   EQU     617BH           PUTCHAR EQU     535DH
@NE     EQU     611FH           PUTS    EQU     5334H
@NEG    EQU     60F3H           SBRK    EQU     5C11H
@NOT    EQU     60FFH           STDERR  EQU     5206H
@OR     EQU     6112H           STDIN   EQU     5202H
@PINT   EQU     610CH           STDOUT  EQU     5204H
@PUTHL  EQU     60CDH           STRCAT  EQU     602BH
@RDMOD  EQU     522AH           STRCMP  EQU     603FH
@SENAM  EQU     5226H           STRCPY  EQU     6059H
@SINAM  EQU     5222H           STRLEN  EQU     6067H
@SONAM  EQU     5226H           TOLOWER EQU     5FCDH
@SUB    EQU     60EEH           TOUPPER EQU     5FB7H
@UCMP   EQU     6169H           XTOI    EQU     5EE9H
@UDIV   EQU     61AAH           ZERO    EQU     6075H
@UGE    EQU     614FH           @UGT    EQU     615BH
```

LISTING 5

```
;LC/ASM - Modifications
;*=*=*
@START EQU      5200H           ;Set to the code origin
*GET LCLIB/EQU                  ;Pre-assembled LC/LIB
       ORG      @LCLIBEND       ;Reference next available code address
;*=*=*
;      Continue with initial LC/ASM code
;*=*=*
       LD       HL,(4049H)      ;Remove "@START" label *****
;
;
       END      @LCLIBEND       ;Change entry point *****
```

LISTING 6

```
. QLC/JCL - Quick compile and assembly of LC programs
. format is:   do lc (file=<progname>,{drive=d},{show})
//if -drive
//assign drive=1
//end
//if show
lc #file#:#drive# +l
//else
lc #file#:#drive#
//end
edas (jcl,abort)
l qlc
c/cprogram/#file#
//if show
atempxxxx:#drive# -we
//else
atempxxxx:#drive# -nl
//end
b
copy lclib/cmd #file#:#drive#
append tempxxxx/cmd:#drive# #file#:#drive#
. completed compilation
```

PaDS/PRO-PaDS VERSION 1.0

In this issue of NOTES, the PaDS section will correct a few typos that appeared in the last issue. Next, there is one additional patch that must be applied to the utility. Following the patches, I will discuss some more uses of this interesting utility. First notice that there is a slight change in the name of the Partitioned Data Set utility. After saying the name of the utility hundreds of times, I felt that I, like others, were pronouncing it as P - D - S. That's three syllables - entirely too long for most folks. Perhaps some of you were already trying to pronounce PDS as a single-syllable "word". My guess would be that "pids" may have been the result. Not liking "pids" as the name of my product, I inserted a lower-case "a". Now the utility can be pronounced "pads". This change was installed at the time that the LDOS/TRSDOS 6.x compatible version, PRO-PaDS, was released. Enough of this smalltalk.

This next patch corrects a problem in the PaDS utility's PDS(APPEND) member when the file you are trying to append is a null file. The patch was listed in the last NOTES. The PATCH as printed was correct; however there was a typo in the procedure. In addition, somehow I did not get this patch into the MASTER DISK. I found that out when Robert J. Newton, of Houston, brought another PaDS problem to my attention. When I gave him a "D"irect patch, it turned out to be in the wrong location. My working copy of PaDS had the PDSA/FIX installed but the MASTER copy did not. Therefore, this patch was not installed until PaDS serial number 220473. Therefore, if your PaDS serial number is between 220001 and 220472, and you have not already installed the PDSA/FIX, please do so. Since this is an X-patch, you will have to patch PaDS by first copying APPEND to workspace, patching the separate file, killing and purging the APPEND in the PaDS, then appending the patched copy back to the PaDS. Do this with a BACKUP copy of PaDS. This is detailed as follows:

1. BUILD the PDSA/FIX file with:

```
. PDSA/FIX - PATCH TO THE APPEND MEMBER OF PDS (PaDS only!)
X'5499'=CD 3D 59
X'593D'=11 8A 58 2A A5 54 AF ED 42 CO 21 4D 59 C3 2E 56
        OA 41 70 70 65 6E 64 69 6E 67 20 66 69 6C 65 20
        69 73 20 6E 75 6C 6C 21 OD
. End of patch
```

2. Execute the command, "PDS(C) PDS(APPEND) APPEND"

3. Execute the command, "PATCH APPEND PDSA"

4. Execute the command, "PDS(K) PDS.PDS(APPEND)"

5. Execute the command, "PDS(P) PDS.PDS"

6. Execute the command, "!APPEND APPEND PDS.PDS"

Note specifically the exclamation point in step 6 - it is required! You now have corrected the PDS(APPEND) member.

The next patch MUST BE APPLIED. It corrects a problem in the PDS(PURGE) utility that trashes the file being purged WHEN A KILLED MEMBER ENDS ON A

SECTOR BOUNDARY". Since the patch is a DIRECT patch, its location depends on the result of applying the PDSA/FIX. Therefore, this patch must be applied AFTER you implement the PDSA/FIX. The PDSB/FIX was applied to the MASTER disk starting with serial number 220473. Again, this patch is for PaDS.

1. BUILD the PDSB/FIX file with:

```
. PDSB/FIX - PATCH TO THE PURGE MEMBER OF PDS (PaDS only!)
. APPLIED 10/06/83 - 220473
D1A,55=00
. End of patch
```

2. Execute the command, "PATCH PDS.PDS PDSB"

The PRO-PaDS utility MASTER disk had the A and B fixes applied starting with serial number 220044. Therefore, PRO-PaDS users with serial numbers between 220001 and 220043 need to apply the following fixes. The procedure to follow for PROPADSA/FIX is the same as that shown for PDSA/FIX and the procedure to install the PROPADSB/FIX is the same as that for PDSB/FIX; however, use the following patches:

```
. PROPADSA/FIX - PATCH TO THE APPEND MEMBER OF PDS (PRO-PaDS only!)
X'288C'=CD 4C 2D
X'2D4C'=11 99 2C 2A 9B 28 AF ED 42 C0 21 5C 2D C3 2D 2A
        0A 41 70 70 65 6E 64 69 6E 67 20 66 69 6C 65 20
        69 73 20 6E 75 6C 6C 21 0D
. End of patch
.
.
. PROPADSB/FIX - PATCH TO THE PURGE MEMBER OF PDS (PRO-PaDS only!)
. APPLIED 10/13/83 - 220044
D15,E0=00
F15,E0=28
. End of patch
```

Now that the patches are out of the way, let me correct one more typo that appeared in the last issue of NOTES. Page 1-10 was discussing a MAP entry to use in appending a module that had two entry points, 5200 and 5203. Unfortunately, where the specific map line was listed, one of the transfer addresses was mistyped. The correct line follows:

```
COPY,APPEND,5200,COPY,5203
```

Since I am talking about MAP files, let me mention a difficulty experienced by one of the PRO-PaDS users who was attempting to use the MAP method to establish a subroutine library for use by the PRO-CREATE assembler. It seems that he was using the EDAS editor to prepare the MAP file. However, when the PDS(APPEND) utility was invoked, an "End of file encountered" error was displayed after the messages that indicated all of his modules were appended. Of course, APPEND could not update the directory. The reason for this was that EDAS normally expects that an assembler source file is being generated and automatically adds an X'1A' terminator to the file. The APPEND utility thought that this was the beginning of another MAP record and continued to read the file for the "member name" and transfer address. Since the file created by EDAS ended at the X'1A', APPEND issued the appropriate

error and aborted. Therefore, if you are going to use EDAS to prepare MAP files, use the "W!! filespec" command when writing your MAP file. The double exclamation marks indicate that no terminator is to be written; thus the X'1A' is suppressed. The APPEND module does trap a NULL to indicate the file end in case you use SCRIPSIT to prepare the MAP file.

I rarely encounter a user with a problem in appending a CMD-type program. When a problem does develop, the solution is simple. The problem encountered is getting the error message, "Load file format error" when attempting to execute a CMD program that was appended to a PaDS. Where does this error come from and why? To answer these questions, a little background is in order.

When I developed the PaDS utility, the most important thing I wanted to accomplish was to provide a "library" function that used absolutely no high memory and was fast. If you were around the LDOS bulletin board on Compuserve during the latter part of 1981, you will realize that the LDOS users wanted a facility to collect executable programs similar to the SYS6 and SYS7 library files. This is why I designed PaDS. To that end, I strongly feel that PaDS met its goal.

In order to accomplish the friendly member-name access procedure and provide the direct execution of members, I needed a way to interface the DOS with each PaDS. I used a technique which capitalized on the method in which the DOS loads and executes programs - as well as its own LIB members. Thus, each PaDS that is built by the PaDS utility, incorporates a program that executes when the PaDS is invoked. Thus, when you enter a command such as:

```
MYLIB(TEST)
```

the DOS executes a program called "MYLIB/CMD" and leaves register pair HL pointing to the left parenthesis when control is passed to MYLIB/CMD. In addition, register pair DE is still pointing to the system's File Control Block which was established to access that program. Recognizing that, the program which executes in MYLIB/CMD can parse the command line and determine if a member specification was entered. Using the open FCB, it can also access any part of the MYLIB/CMD file. This technique can actually be used to distinct advantage in specific applications where a CMD file can be constructed which has a data region following the program that the program accesses.

I have termed the executing program a "Front End Loader" since it is in the front part of the PaDS and is used to load the requested member. The FEL is written to each PaDS that is built with the PaDS utility. Under LDOS 5.1 (Model I/III), the FEL loads starting from X'5200'. The PRO-PaDS FEL loads into the library overlay region at X'2600'. I wanted to make the FEL short and sweet. That I did. Surprisingly, the FEL is less than 256 bytes. In order to keep it short, the error message feedback had to be kept to a minimum. Since messages can take up a lot of space, the FEL uses standard errors that reside in the DOS error dictionary. Therefore, if you try to execute a PaDS member that is not a CMD program [a data file member, for example], instead of displaying a message such as "Attempted to execute a data member", PaDS passes an error code 34 to @ERROR. Now error-34 is "Load file format error" which is actually correct for this case. A data file generally is not a load module! In using error-34, I saved about 40 bytes.

Now you know what that "Load file format error" is all about. OK you say, you told me what it means; but I tried to execute a member that was a CMD program! What gives? Most likely, if you would do a PDS(D) of your PaDS, you would find that the member in question was listed as a DATA member. Since you know that you appended a CMD program, how then did it get classified as data? That's an easy one. In order to classify a file as either DATA or PROGRAM, the PDS(APPEND) module imposes a few tests. If the file extension is "/CMD" AND the first byte of the file is X'05' or X'01' [which indicate a load module HEADER record or LOAD record respectively] AND the fourth from last byte is an X'02' [which indicates a TRANSFER record], then the file is interpreted as a PROGRAM; otherwise it is interpreted as data. The reason for the classification is that CMD programs must have the TRANSFER record changed from an X'02' to an X'04' [which indicates END of partitioned data set member] so that they may be properly loaded and executed by the DOS loader. I certaintly don't want every member to have a byte changed.

Now you say that your "troublesome" CMD program executed properly from DOS but won't as a PaDS member. When DOS loads a plain-vanilla CMD program, it reads through the file and loads X'01' records into memory. The DOS reads through until it reaches the TRANSFER record. It doesn't care if anything is beyond the TRANSFER record [if it did, I would not have been able to implement the PaDS utility as I did]. Thus, you will find that the CMD program did not have a proper end-of-file pointer in the directory - proper meaning that it indicates the last byte of the TRANSFER record as the end of the file. Thus, when PaDS read the file, it could not detect the X'02'.

There is a simple solution to the dilemma. First, confirm that what I say is true by LISTing the CMD file in hex and noting where the listing ends. If you don't see as the last four bytes, "02 02 LL HH", where LL and HH refer to the low-order and high-order bytes of the transfer address, then the file's EOF is wrong. All that you need do is run CMDFILE, load the CMD file, then write it back out. CMDFILE reads a file until the TRANSFER record is reached. When CMDFILE writes out its buffer, it will create a file with an EOF that is correct for the CMD file. PRO-PaDS users can use the PRO-CESS utility for this purpose.

Since the inception of PaDS, many users have used PaDS to store more than just CMD files. PaDS is just as useful to store BASIC programs, DOS filters, and other files just for archival purposes. However, when you want to access those non-CMD members, you have to use PaDS utilities. There is no way around the necessity for high-memory interfacing routines when the goal is to provide DOS-type access to members. Witness the 3K of memory required by a "library option" utility available for the TRS-80. To get flexibility and powerful procedures, you have to give up something. Our criteria for no high memory utilization to implement PaDS has worked well. Until I find the time to develop PaDS version 2 [which is to be designed to provide READ access to members through standard DOS calls], there are simple procedures that can be used to access members.

For instance, if you want to install a filter that is resident in a PaDS, all you need to do is to PDS(COPY) the filter member to a work file then filter the device using the work file. If you want to delete the work file after the filter is applied, you can KILL/REMOVE it. This procedure is perfect for Job Control Language. Consider the JCL file:

```
. JCL to invoke a filter stored in a PaDS
//if -dev
//assign dev=pr
//end
pds(copy) filters/cmd(#file#) #work#/flt
//if p1&p2
filter *#dev# using #work#/flt (#p1#,#p2#)
//end
//if p1&-p2
filter *#dev# using #work#/flt (#p1#)
//end
//if -p1&-p2
filter *#dev# using #work#/flt
//end
//exit
```

With this JCL, you could invoke the LDOS 5.1 minidos filter via a command:

```
do flt (dev=ki,file=minidos,p1=type)
```

This, of course, assumes that you have a PaDS called FILTERS/CMD that has the MINIDOS/FLT filter as a member. The //ASSIGN macro is shown to provide a default for "dev" of the *PR device. Thus, if *PR is the device to filter, you don't have to enter "dev=pr". You could choose the default to be whatever device you most frequently filter. You could also add a JCL statement to delete the "#work#/flt" file after the filtering. I also demonstrated the method to add parameters to the filter command. This procedure works for PaDS and PRO-PaDS users.

Model I/III users of LBASIC could also RUN members that are BASIC programs by specifying the following sequence of BASIC statements:

```
CMD"PDS(COPY) BLIB/CMD(DEMO) WORK/BAS
RUN"WORK
```

This can't be done from TRSDOS 6.x BASIC because Microsoft's BASIC does not perform the program shift to high-memoryy and protection as is done in LBASIC.

I have had a few requests from the more experienced hackers as to the structure of our PaDS file. These queries are looking for information sufficient to write specialized access routines. The PaDS is a load module and contains various types of load module records. Since I have discussed the structure of load module records in my book, THE PROGRAMMER'S GUIDE TO LDOS/TRSDOS VERSION 6, I will not repeat myself here. What I will cover is a block sketch of the PaDS.

The PaDS starts with the Front End Loader. This is a standard program as discussed above. The only difference is that the header record begins with an X'06' instead of an X'05'. This byte is used in the PaDS commands to identify the file as a partitioned data set usable with PaDS. The MEMBER directory follows the FEL. This directory X'0C' records as discussed in the GUIDE. The member directory is terminated with a X'0E' record used to determine if the member specification passed is contained in the directory [if the X'0E'

record is reached prior to a match in the directory, the searching immediately stops]. Following the member directory is the ISAM directory composed of X'08' records. Each member record corresponds to a unique ISAM record matched up via the ISAM number. The ISAM directory is terminated by a X'0A' record which is used by the DOS loader to detect that the ISAM member was not found [if it did not match up the ISAM number requested with any in the ISAM directory]. What follows is a X'04' record, again used by the DOS loader in a pure library file to inhibit execution of the first member if the ISAM linkage was never established. The member modules follow. The ISAM directory contains a three-byte pointer to the relative record and byte which starts the member. The member directory record contains a three-byte length value which specifies the exact length of its member.

There you have it. The hacker should find it easy to write specialized routines to access the PaDS. The non-programming user may want to wait until we release version 2 of PaDS. Don't forget Scott Loomer's PDS TUTOR package. Scott can be reached at 315 Palomino Lane, Madison, WI 53705.

THE PROGRAMMER'S GUIDE

MISOSYS began publishing THE PROGRAMMER'S GUIDE TO LDOS/TRSDOS VERSION 6 back in August of this year (1983). It, of course, was written by Roy Soltoff - the primary designer of TRSDOS 6.0 (a la LDOS 6.0). It was authored and published due to the reluctance on the part of the Tandy Corporation to make available their Technical Reference Manual in a timely manner. The GUIDE [not to be confused with THE HITCHHIKER'S GUIDE] is designed to be used by the LDOS Version 6 assembly language programmer. The GUIDE, in its 214 pages, covers the procedures to be followed when writing programs for use under version 6 LDOS.

The GUIDE also contains a great deal of background information on device handling, including the most in-depth discussion of device filtering and routing that you have ever read. A description of the GUIDE is contained in catalog 83-2. As an example of the GUIDE's content, Figure 1-2:System Map is an example of one figure excerpted from chapter 1.

The GUIDE also makes good reading for the Model I or Model III LDOS user since LDOS has a great deal of similarity across the Model I/III version 5 and the RAM-based LDOS version 6.

For those that have already acquired a copy of the GUIDE, there are a few corrections to make. First, pages 3-47 and 3-48 have been revised to change all references of "IX" to "IY". If your book does not have the revised page, contact MISOSYS and request the new page.

Other corrections are as follows:

1) page 2-32, change the statement beginning with "+3 - Flag to indicate" to read "+3 - Contains the receive character code interpreted as BREAK." Note: I am suggesting to LSI that they change the COM/DVR module to interpret a value of NULL to indicate that no checking for BREAK/PAUSE/ENTER conditions should be done if the BREAK character code value is set to NULL.

2) Page 6-139, change VIDEO POKE register B value from "B => 1", to "B => 2".

3) Page 6-139, SET CURSOR POSITION; change the statement, "A <= Will ..." to read, "A <= Will contain the error code if an error was encountered."

4) Page A-150, paragraph 4; change the phrase "256 load bytes respectfully" to read "256 load bytes respectively".

LOWCORE: X'0000' - @$SYS RST vectors, NMI vector, System flags, Date, Time, System FCB, DEBUG register save area, JCL FCB, Command FCB, SVC Table, DCB Table, System stack, Miscellaneous data, Command input buffer, Drive Control Table, Device I/O handler, Clock task, Memory management routines.
IOR: @$SYS - X'12FF' Keyboard, Video, Printer, and Disk drivers.
SYSRES: X'1300' - X'1DFF' File access routines, SVC processor, System overlay handler, System program loader, Interrupt Task Schedular, System buffer.
SOR: X'1E00' - X'23FF' Execution region for system overlays 2-5, 9-13, overlay disk file buffer.
LOR: X'2400' - X'2FFF' Execution region for system library comands contained in libraries A, B, & C.
UPR: X'3000' - (HIGH$) Execution region for user transient programs (note: programs not accessing the system libraries can start at X'2600'.
HIMEM: (HIGH$)+1 - X'FFFF' Region for relocation of extended system and user static modules.

Figure 1-2: System Map

SOLE

I have a few odds and ends to bring up in this issue of NOTES concerning the use of SOLE. The first item to be discussed is a big problem because another publication, the LSI JOURNAL, published totally erroneous information

concerning the subject. Since LDOS users have been led to believe that whatever LSI publishes must be fact, when they present material that is factually in error - especially concerning a MISOSYS product - it can be harmful to our customers. Let me emphatically state that when you have questions concerning MISOSYS products, please give us the opportunity of respoding to your query. Do not go off half-cocked and develop what you think is a solution to a suspected problem.

I did get quite perturbed with LSI for publishing a particular article that appeared in their April 1, 1983 LDOS QUARTERLY. I first saw the article when I received a copy of that QUARTERLY [as an aside, I am no longer on LSI's registration list for QUARTERLIES or JOURNALS]. I subsequently investigated the problem and solution posed in the article since the author HAD NEVER MADE A BUG REPORT TO MISOSYS. I then drafted a letter to LSI protesting their publishing such erroneous information. The text of my letter to LSI is repeated here so that any SOLE user who read the LSI article would be advised as to the erroneous information contained in that article. The text of the letter is followed by THE OFFICIAL PATCH which contains the method of fixing the bug.

Excerpts from letter to LSI dated May 18, 1983:

"Rather than wait until the next LDOS QUARTERLY before speaking my piece concerning one article in the April 1, 1983 issue of THE LDOS QUARTERLY, I thought it best to bring this to your attention now. This letter addresses the article entitled "SOLEFIX -- Fix that GAT error!".

Let me first state that I find it highly ludicrous that the content of this article was not brought to my attention prior to its publication. Considering the relationship our respective companies entertain, I would certainly expect that your organization would at the barest minimum, attempt to query MISOSYS as to the technical accuracy of an article relating to a MISOSYS product.

Let me get to another point concerning "SOLEFIX". It was technically INACCURATE. The article purports to describe a granule allocation problem where a system disk has been altered with SOLE. The article states that, 'SOLE also alters the diskette's directory by setting the allocation bits for all 3 granules of cylinder 0... Unfortunately, SOLE fails to allocate one of these granules to a file; this creates a granule allocation table error.'

Far be it for me to be "the expert" on LDOS; however, let me point out that when SOLE reformats the BOOT track of the double density system diskette, the track now has TWO GRANULES OF 5 SECTORS EACH. The author would have you believe that the track still contains three granules. The author goes on to say that "Super Utility Plus fixes this error for you, and thus allows the system to write on cylinder 0, granule 2." WHAT ERROR? WHAT GRANULE 2? Because SU+ alters directory information, why does the author consider the problem to be a "BAD" GAT?

The next error the author performs is to come up with a procedure to allocate "THE THIRD" boot granule to the BOOT/SYS file!!! This technique is so illustrated in his SOLEFIX/ASM program. Now what the author's program does is change the allocation information for BOOT/SYS to three contiguous granules from two contiguous granules. He sets the CREATE bit to inhibit

deallocating this fictituous "space" SINCE THE FILE STILL SHOWS ONLY 7 SECTORS.

I am glad that Super Utility Plus is happy with this altered arrangement. Unfortunately, in my opinion, the alteration is wrong because BOOT/SYS is only two granules in length and track zero contains only TWO granules. If the author intended for the article to point out a deficiency in SU+ and how one can alter a SOLEd disk to fool SU+, it failed to convey that thought. If it intended to accurately document the allocation schema of a mixed density diskette, then the author also failed that category. If the author wanted to use the LSI newsletter as a medium for "documenting" erroneous data, he succeeded. The one commendable point is that the author's program serves to illustrate code interfaced to standard system entry points [I do not know why he thinks he has to "AND 63" the error return code. I would also recommend his END statement be changed to "END START"].

End of excerpt of letter to LSI.

The LSI Journal is certainly not the vehicle to be used for calling program problems to the attention of a program's non-LSI publisher. I readily admit that SOLE did in fact have a bug. The bug was NOT in allocating only two granules for BOOT/SYS. The problem was due to the fact that I neglected to SET bit 2 of the lockout byte for cylinder 0. Thus, even though the allocation byte for cylinder 0 was correct, the lockout byte showed that the cylinder had a third granule. Thus, if a directory checking utility was used on such disk, it would have shown cylinder 0, granule 2 as allocated but not used for any file. Unfortunately, SU+ automatically "corrected" the "problem" by deallocating the "fictituous" third granule. The proper and CORRECT solution is to correct SOLE1/CMD to lockout granule 2. The following patch is the OFFICIAL PATCH to correct this problem.

```
. SOLE1A/FIX - Applied 06/01/83 starting from serial number 210694
. Corrects problem of NOT locking out the eliminated boot track
granule
X'522B'=47 53
X'5347'=2E 60 36 FC C3 39 52
. End of patch
```

While we are in this discussion, let me relate that Rick McDonald and Bob White both reported that SOLEd disks that have been backed up by the LDOS BACKUP utility per the procedure specified on page 7 of the SOLE documentation results in a disk that has 1.5K more space available than the source disk. After investigating this bug report, I realized that the "extra 1.5K" derived again from the BOOT track having the third granule free. BACKUP did just what SU+ did. As it turns out, it wasn't so much a fault of BACKUP as it was that BACKUP is very confused by a mixed density disk. Suffice it to say for now that you must correct the procedure listed in the SOLE manual (page 7, paragraph 1, sentence 3) to read as follows:

> ONCE YOU HAVE THE DISKETTE FORMATTED, RUN SOLE1 FOLLOWED BY SOLE2, EACH TIME REFERENCING THE DRIVE CONTAINING THE NEWLY FORMATTED DISK.

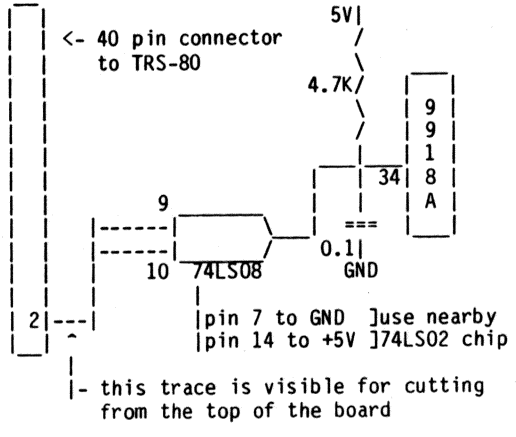
I have printed the sentence in all capitals here only for emphasis. The important point is that SOLE1 must be run first so as to alter the GAT's

allocation and lockout tables for cylinder 0 to reflect 2 granules only and that both granules are in use.

One SOLE user has passed on this patch to use SOLE with the LNW at 4 Megs. In SOLE2, change the byte at X'56CC' to an X'10'. The current value at that location is X'08'. This serves to lengthen the delay after passing a command to the disk controller.

Mike Kaizen, of Rockville, MD, has been a devoted follower of MISOSYS products. Mike has been a customer of mine starting with DUTIL and DISK*MOD (Ashton-Tate, eat your heart out - DUTIL was a name of a MISOSYS product back in 1979). Mike recently experienced a problem in trying to re-BOOT a Model I using a SOLEd system disk. I believe that the disk booted fine if the RESET button was pressed but the BOOT command would cause the system to hang. Mike traced to problem to use of the CHROMAtrs color board - which seems to be a hot item for inexpensively adding color to your TRS-80 [actually, it seems that the cheapest way to add color is to buy a 16K Atari 400 which are going for $49 these days with the Atari rebate]. Mike discovered that the CHROMAtrs board did not buffer the RESET line. There was already too much load on the CPU reset bus and the additional load from the color unit was preventing the reset from actually taking place. The result was that the disk controller was not reset to single density from double density and couldn't read the single density BOOT sector.

Mike's solution was not to unplug the CHROMAtrs board but to modify it so as to buffer the reset line. I'll try to draw the schematic change using the DW-II character set. What you need to do is add a 74LS08 on the CHROMAtrs board in line with the reset bus line. Note the schematic below:



I have received a few queries from Model I SOLE users of how to backup a double sided SOLEd diskette. An attempt to do so by using BACKUP results in a DATA SECTOR NOT FOUND DURING READ error. I cannot spend the time to work up a solution as the DOS doesn't generally support booting double-sided system disks and very few individuals are attempting to use such disks. Also, the

only 2-sided disk drive at MISOSYS is an 80-track drive which is never used on our old Model I test station. However, what I would like to do is to explain why such a diskette cannot be backed up. I would also like to recommend that someone using a 2-sided system disk on a Model I with SOLE investigate my proposal more specifically and arrive at a patch to BACKUP. Before I go into the discussion, let me state that the application of SOLE to a two-sided diskette results in only the first side being reformatted to single density - the second side remains in double density [but should be left unused].

To begin, when BACKUP is invoked for mirror-image copying, one value it calculates is the number of sectors per cylinder. This value is then plugged into the BACKUP code to be used where it is needed. If we are talking about a 5" double-density diskette, the number of sectors per cylinder is 18 for a single sided disk and 36 for a two sided disk [a two-sided disk has sides numbered 0 and 1]. Let's look at the one-sided situation first.

The SOLEd disk has 18 sectors per cylinder on all cylinders except the first, where it has only 10 (numbered 0-9). When BACKUP tries to read sector 10 of cylinder 0, the disk driver has already switched to single-density and has automatically updated the DCT to show that 9 is the highest numbered sector. Thus it says, "whoa, this disk only has sectors 0-9." Since the driver is a little intelligent, it attempts to see if the sector in question is on side 1. After subtracting off a track's quantity of sectors, the driver next would try side 1 with sector 0 instead of side 0 with sector 10. Before actually going to the controller with this information, it checks the DCT to see if the drive is two-sided. Finding only a single sided drive indicated, it readdresses side 0 and does not pass a "sector number out of range" error back to the caller. Thus, when BACKUP passes logical sector numbers 10-17, the driver rereads sectors 0-8 - thereby backing up those sectors twice. Early users of SOLE may recall a problem of drive 3 selection when backing up a SOLEd disk. This was due to the LDOS disk driver not checking if the disk was two sided thus promping a change to the LDOS driver.

Now, let's look at the two-sided case. First, we now have BACKUP computing 36 sectors per cylinder. Again, when sector 10 is passed to the driver, sector 10 is readdressed to sector 0 of side 1 - BUT IN SINGLE DENSITY! When the driver tries to read this sector, it gets a record not found error because side 1 is in double density. The driver (we are talking about PDUBL or RDUBL) then switches density to double (part of the automatic density recognition) which now updates the DCT to show 17 as the highest numbered sector. Now before the driver tries to reread the sector, it must invoke the SEEK routine which now calculates that sector number 10 is on side 0! Therefore, the second attempt still results in a sector not found error. The driver switches to single density and repeats the process until it times out after the ten retries. Sectors numbered 10-17 of cylinder zero cannot be backed up because they do not exist!

The root cause of the problem is that BACKUP was never programmed to deal with a mixed density diskette. The solution, of course, is to correct BACKUP so that it essentially recalculates the number of sectors per cylinder while it is copying cylinder 0. We would not want BACKUP to perform the calculation on every cylinder as then there would be no way of ensuring a mirror image condition. Suffice it to say that the only practical case is where cylinder zero is of single density and all other cylinders are double

density.

BACKUP uses the sectors-per-cylinder value in three routines critical to the backup of a SOLEd diskette. These are the routines for LOADING, DUMPING, and VERIFYING cylinders. If we want to correct BACKUP, we have to come up with a technique that will determine when sectors 10-17 of cylinder zero OF A SOLED diskette are being referenced and inhibit the reading, writing, or verifying sector request from being performed.

I will address this issue using LDOS 5.1.3 BACKUP as a base [I will not address the issue for 5.1.4 or any other release of LDOS]. A few words are in order for the BACKUP module itself. The mirror-image piece is assembled to execute at X'5900' but initially loads at X'6100' [see my discussion of LORG under EDAS/PRO-CREATE]. Therefore, if you need to disassemble BACKUP and add patch code, your patches must execute at an address different from where they load. The "loading" routine invokes RDSEC with "CALL NZ,RDSEC" at address X'5A23' which loads at X'6223'. The "dumping" routine invokes WRSEC with "CALL WRSEC" at X'5A98' while the "verifying" routine invokes VERSEC with "CALL NZ,VERSEC" at X'5AF1'. The code from X'5992' through X'59B9' calculates the sectors per cylinder and stuffs the result into the three critical routines plus two others you need not be concerned with. Your fix may need to recover the initially calculated value.

The big problem is in making sure your changes do not effect the backup of two-sided single density or double density disks. Since a SOLEd system disk has the entire cylinder zero locked out, you need not worry about trying to use the second side - the loss of disk space is minor. What is needed, therefore, is a technique to determine whether a particular source disk is a SOLEd disk. If it is, then if we are copying cylinder zero and the sector number is greater than nine, we should ignore it. That sounds simple enough. However, how do we ascertain the disk as one that has been SOLEd?

I propose that you investigate patching BACKUP at the three locations identified above. When your patch code is entered, register D contains the cylinder and register E contains the sector. Also, register IY points to the DCT for the drive. For the sake of speed, I suggest code such as:

```
INC   D
DEC   D
JR    NZ,DO_THE_REQUEST
ELSE TEST_FOR_INHIBIT
```

This tests for cylinder zero or otherwise. If BACKUP is not on cylinder zero, no further checks need be made and the request is performed. If it is on cylinder 0, you could then check for a sector number less than 10 or greater than 17 and also perform the requested function if the number is not in the range 10-17. Once you find that the cylinder is zero, and the sector is in the critical range, you must determine if the disk is a SOLEd disk. After thinking about this problem, I suggest that you test if the current number of sectors per cylinder is equal to the original number of sectors per cylinder, then the request is performed, otherwise it is not performed. In order to calculate the current number of sectors-per-cylinder, it should be sufficient to execute code such as:

```
LD    A,(IY+7)
AND   1FH
INC   A
BIT   5,(IY+4)
JR    Z,$+3
ADD   A,A
```

which doesn't take the number of heads-per-partition into account [that is unneeded for single disk floppies]. If the disk is a SOLEd disk, reading sectors 0-9 would have automatically conditioned the DCT for single density and any current computation of sectors per cylinder would result in 10 (single sided) or 20 (two sided). By comparing this result with the original value calculated in the routine starting at X'5992', assume that a different value indicates a SOLEd disk.

Do not affect registers DE, HL, nor IY. You should only need to use the accumulator. It appears that the area from X'5CE5'-X'5CFF' is available. Patch code for the mirror image part cannot extend past X'5CFF'. I doubt that this amount is sufficient to implement the proposed modification - thus, you will have to get cute and use some of the messages for patch space. If anyone successfully implements such a patch or has some other usable technique for accomplishing the desired result, please write it up and submit it for the next issue of NOTES.

ZGRAPH/PRO-ZGRAPH
==========

In this issue of NOTES, I am proud to report that Karl Hessinger has developed an enhanced version of ZGRAPH - version 5. First, this version is written completely in assembly language - it is fast. Second, it includes an improved circle generating routine [improved for blinding speed]. It now supports a "fill" function that fills in all pixels of a boundary. This operation is similar to the "paint" operation under graphic BASICs.

Version 5 ZGRAPH now uses all available RAM for buffer space. Instead of limiting you to four buffers, it creates as many as can fit in the RAM space which follows the program. With the ability to store 15-25 screens and save all the screens into a single file, Karl decided to develop another utility which "plays" the screen images in a timed succession. This is great for making up a graphics slide show.

Karl added magnification and reduction functions to ZGRAPH so that you can take a rectangular block of the screen image and either expand it or shrink it. You get nine levels of expansion and two modes of reduction.

The ZGRAPH package now supports the Radio Shack DMP-series of printers - including the DMP-2100. Printer support has been shifted over to a separate group of utilities - the xxBINCAT/CMD programs. The RSBINCAT supports the Radio Shack DMP printers while the EPBINCAT supports the Epson printers. Each BINCAT program provides the features to concatenate separate ZGRAPH screens horizontally as well as vertically to produce large graphic printouts from separate screen images. The xxBINCAT programs also permit magnification of the concatenated image.

NOTES FROM MISOSYS

If you presently own Model I/III ZGRAPH version 4 and wish to upgrade to Model I/III ZGRAPH version 5, you can do so by returning your ZGRAPH master diskette to MISOSYS. There will be a $20 charge. In return, you will get an updated diskette and a brand new version 5 ZGRAPH manual.

ZSHELL

Some of the ZSHELL users have asked for additional features for the ZSHELL package. Therefore, starting with registration number 430063, the program, WC, has been included with the ZSHELL package. WC is a "shell" processor that allows you to invoke compatible commands on a number of file specifications that match a wildcardspec entered on the command line. You enter the command line once while the WC shell processor searches the designated disk drive(s) for files that match your wildcard specification.

WC builds a Job Control Language file of your command line (minus the "WC") substituting each matching file specification for the wildcard specification on a separate command line. WC then automatically executes the JCL file. The "wildcardspec" uses the file name and file extension as two distinct fields for matching purposes. If the drive specification is entered, WC will search that specific drive for all files matching the name-extension wildcard fields. If the drive specification is omitted, then all drives will be searched. Within each field, WC accepts two wild characters, "?" and "*". The question mark will match any character in that character position. The asterisk is used to match all trailing characters in the field. For example, "?SHELL/TXT:1" will match with ASHELL/TXT, BSHELL/TXT, etc. but ASHELL1/TXT will not match. A global match of all filespecs would be an entry of the form, "*/*"; whereas a match of all /CMD files would be an entry of the form, "*/CMD". If a minus sign, "-", precedes the filename field, WC will select files that do not match the wildcardspec. Any entered password will be used in the full file specifications generated by the selection process. Note that this wildcard syntax is different from the DOS partspec - and more powerful!

WC is quite useful to perform repetitive tasks on files whose file specifications are similarly constructed. For example, to list out all /TXT files on drive 1, you could use a WILDCARD entry of:

WC LIST */TXT:1

What DOS commands are compatible? All of the following DOS commands are: APPEND, ATTRIB, LIST, LOAD, REMOVE/KILL RENAME, RESET(V6), and RUN. Other programs that expect a filespec on the command line are also "compatible".

Aside from just adding WC, ZSHELL itself has been slightly altered to improve its performance when interacting with Job Control Language. Also, for our LC users, Karl added an "escape" character that can be entered to inhibit ZSHELL from scanning a command line. If a double-quote character (") is the first character entered on a command line, it is trapped by ZSHELL and will inhibit ZSHELL from performing any redirection functions on that command. Thus, the invocation of an LC-created program could utilize the I/O redirection capabilities of LC even though ZSHELL is active.

NOTES FROM MISOSYS

If you own a ZSHELL package with a registration number 430001 through 430062, you may return it for an update. The fee is $5 which includes the disk upgrade, additional documentation, and shipping.

Let me point out a very significant use of ZSHELL that is easily overlooked. Many LDOS users have realized the potential of Job Control Language for automating many repetative operations. The knowledgable JCL user understands that JCL functions only with LINE INPUT: the @KEYIN routine or LINEINPUT from LBASIC. Programs which use the single character keyboard calls (@KEY, @KBD, or LBASIC's INKEY$) cannot be operated via JCL. As there exists many fine menu-driven programs which rely on single-character inputs, they are unusable with JCL. Rather than attempt to patch this, and patch that, to utilize some "typein" process, ZSHELL permits extremely powerful user-controlled file input within its redirection capabilities.

ZSHELL's standard input redirection (the keyboard device, *KI) provides for three end-of-file options selectable at command invocation. Each option selects a particular operation that will be undertaken upon reaching the end of the redirected input device prior to the command's completion. For instance, you can have ZSHELL exit to the @ABORT vector which immediately ceases the execution of the command. Alternatively, you can have ZSHELL return control to the standard device (the keyboard) and automatically indicate a condition as if the BREAK key was depressed. The normal method of handling the EOF condition is to restore control to the keyboard. Thus, ZSHELL is designed to gracefully return the keyboard to you.

This one ZSHELL function - that of controlled redirection of character input - will provide handsome rewards to your total LDOS operation once you purchase and start using ZSHELL.

CONTRIBUTIONS

The following program can be used by LDOS 5.1.x users to transfer files from NEWDOS80 or DBLDOS double density media. In the case of NEWDOS80, source diskettes cannot have extended directories. The program acts as a filter on a designated disk drive. Once the program is invoked, the referenced drive can only be used to read files from the "alien" diskettes. The command procedure to invoke the drive filter is:

```
CONVDOS :D ({NEWDOS},{DBLDOS},{REMOVE})
```

Specify "NEWDOS" or "N" if you are going to transfer files from NEWDOS80 double density data diskettes (when track zero is double density). Specify "DBLDOS" or "D" if you are going to transfer files from Percom's DBLDOS, or NEWDOS80 Model I system disks (when track zero is single density). Specify "REMOVE" or "R" when you are finished transferring files and wish to eliminate the filter. A warning... DBLDOS data diskettes do not store the directory track number in byte 2 (relative from 0) of the BOOT sector. Therefore, problems can develop when transferring files from DBLDOS data diskettes.

To capture the following program, enter the data as ASCII and use any of the public domain BINHEX conversion programs.

```
0506434F4E56444F01020052E53A2501FE492027215444225852218A4222
6E53214E4422615521514422 6E5521114422815222955222E852220E5321
7353CD6744E17E23FE2028FAFE3AC262537E23E607CA6A534FCD8F47FD7E
00FEC9CA5A53111055CD7644C26A53010000078B12839FDE5E1CD3853C25E
53FD5E01FD5602213C00194E2346FD7101FD70022A494023AFED52212154
C23253EB23235E2356ED5349402107 54C3325301000011000078B1B283CA
66537AB32804AF327255FDE5E1CD3853CA5653117C55010A00EDB0DD217C
55DD7E03E607F6C0DD77U3DD7E04E60FF640DD7704DD360622DD360711DD
36U9FF2A4940224355 1185 55D5E5B7ED52444D2A5755092257552A760102
00535509227655D1E1014500EDB8ED53494013FD7301FD7202FD360340FD
360440FD36063CFD360709FD360824FD36091121EB53CD6744C32D40E523
7E23666F232323231145551ABE200B2313471ABE2004132310F8E1C921ED
54DD21AD54DD21CE54DD214954DD219554DD216754CD7B44C33040434F4E
56444F53202D20436F6E76657273696F6E207574696C69747920666F7220
73656C656374656420444F5373657320 2D2056657273696F6E20312E320A
436F7079726967687420286329203139383 22C20627920526F7920536F6C
746F66662C20416C6C20726976874732072657365727665640A0D436F6E
76657273696F6E2066696C74657220696E7301020054746 16C6C65640D43
6F6E76657273696F6E2066696C7465722072656D6F7665640D52656D6F76
6564206275742063616E6E6F74207265636C61696D2068696768206D656D
6F7279210D447269766520737065636966696361746 96F6E207265717569
726564210D506172616D6574657220657267 2D2074727903C4E45
57444F532C44424C444F532C52454D4F56453E210D444F53207061726 16D
6574657220726571569726564210D526571756573746564206472697665
20736C6F74206E6F7420696E20757365210D44657369676E617465642064
72697665206E6F742066696C7465726564210D447269766520616C726561
64792066696C7465017E0055726564207769746820444F534356210D4442
4C444F539E52442020202020 9E524E4557444F53A1524E2020202020A152
52454D4F56455E525220202020205E5200180A000007434F4E56444F5378
FE0B3E0FD0FDE5FD217C55E5D53E0A6A2600CDC144D1D565836F8C95673E
12CDC4445F5514E1E3CD7C55D1FDE1C902020052
*72
```

MISOSYS
P.O. Box 4848
Alexandria, VA 22303-0848

BULK RATE
U. S. POSTAGE
PAID
ALEXANDRIA, VA
PERMIT NO. 916

Contents: Printed Matter