MISOSYS

CATALOG OF SERIOUS SOFTWARE

CAT: 83-2

General Notes

MISOSYS specializes in quality tools and professional software to enhance your microcomputer system. All of the products in this catalog run under LDOS. In most cases, version 5.0.x and 5.1.x are supported. EDAS (and therefore LC) requires the use of the Extended Cursor Mode keyboard driver that first appeared in LDOS 5.1.1 (Model I) and 5.1.2 (model III). Many of our newer packages have been designed for use under both Model I/III LDOS and TRSDOS (Model I - Version 2.3 and Model III - Version 1.3).

We have developed a PROfessional series of software specific to Version 6.x LDOS/TRSDOS. This operating system is used on the Model 4 and other computers. All products for use with DOS version 6.x are identified with the prefix, "PRO". The PRO series is priced the same as our current series; however, since these packages are designed to run under different machines, please do not ask for any updates to the PRO series from 5.x.x packages.

There are many new products in this catalog. CMD-FILE/PRO-CESS Version 2 is an excellent load-module maintenance utility for the LDOS or TRSDOS user. MISOSYS picks up where Microsoft left off. The MLIB/PRO-MLIB librarian is available for the user of Microsoft packages producing /REL files. For the LDOS user, MACH2/PRO-MACH2 puts disk space allocation under your control while ZCAT/PRO-ZCAT helps you manage your collection of disks. We have enhanced versions of ZGRAPH and CONVCPM. We have improved the operation of ZSHELL, a command processor adjunct that adds more UNIX-like features to LDOS.

The order form now has a field to enter your machine type. Please enter the model number of the machine for which you are purchasing the software. In this way, we can be sure that you are ordering the proper package.

MISOSYS publishes "NOTES FROM MISOSYS" as a means of keeping our customers informed. NOTES is issued periodically and is free of charge. The contents will keep you up-to-date on the events happening here and keep you updated with changes to our packages (that means bug fixes). If you purchase one of our packages, it will be important to send in the registration card - that's how one gets registered [it's also important to enter the serial number found on a diskette label onto the registration card].

The information in this catalog represents all the printed matter we can send you concerning a software package. If this still does not answer all of your questions, please write or call.

THE C PROGRAMMING LANGUAGE

Yes, the book that brought the C language to the masses is now available from MISOSYS. THE C PROGRAMMING LANGUAGE, written by Brian W. Kernighan and Dennis M. Ritchie is "the Bible" of the C language. This book is a must if you are getting into C. This book is a must if you are purchasing or have already purchased our LC compiler. THE C PROGRAMMING LANGUAGE documents the C language as well as provides an extensive tutorial on writing programs in C.

THE BOOK, ACCESSING THE TRS-80 ROM, VOLUME II

This bOOk, written by Richard P. Wilkes and Stephen C. Hill with technical assistance from Roy Soltoff, Raymond E. Daly IV, and Thomas B. Stibolt, Jr., is a compendium of knowledge on the ROM Input/Output functions of the Model I TRS-80. Its eight chapters tell it all: Introduction to I/O on the TRS-80; The keyboard; The Video; The Printer; The Tape; ROM Disassembly: I/O; Other I/O Routines; Random Ramblings. Nine appendices include such topics as: Lower Case Driver, The TRS-80 Graphics, SET/RESET/POINT Routines, Lower Case Hardware Conversion, and Parallel Printer Driver. Although written specifically for the Model I, the textual information is also applicable to the Model III. THE BOOK, available at a special price while quantities last.

THE PROGRAMMER'S GUIDE TO LDOS/TRSDOS VERSION 6, by Roy Soltoff

The Guide is THE definitive answer to the solutions that programmers need in order to interface to LDOS/TRSDOS Version 6. This book is written by the primary author of the DOS. Every LDOS user has known him as the man behind RSOLTOFF! The Guide contains information never before seen in print. Six hefty chapters and a rich Appendix cover the entire gambit of topics from an overview of the system to an in-depth analysis of the file system. The Guide shows how to write disk drivers, device drivers, and device filters. The Guide provides entire SVC access protocols.

The six chapters cover: An Operating System Overview; Device Input/Output Interfacing; Disk Drive Input/Output Interfacing; The DOS Directory Structure; Disk File Access and Control; and Interfacing via SuperVisor Calls. The Appendix includes: Boot Initialization ICNFG interfacing; BREAK, PAUSE, ENTER Interrupt Latch Handling; Disk Load Module Format; Error Message Dictionary; Header Protocol of Memory Modules; Interrupt Task Processor Interfacing; Low Memory Details; Memory Bank Switching; Non-interrupt Background Task (KITSK) Interfacing; System Disk Boot Track; System Overlay Contents and Access; Using the System Parameter Scanner; and Sample Filters [TRAP, SLASHO, BOLDFACE]. The following paragraphs have been excerpted from the chapter on DEVICE I/O.

"It is interesting to observe that the process of removing the filters from the device chain is exactly the same as the process to add them into the chain. We can unhook the filters by exchanging the first three bytes of the DCBs in the order of last-in first-out (LIFO). Thus if you exchange the *PR and *BF Device Control Block TYPE and VECTOR fields, you will obtain the arrangement previously shown in figure 2-3. The RESET library command does this for the entire chain.

By now you should be able to notice that we could equally as well remove just the SLASHO filter if we swap the bytes associated with the *BF and *SO Device Control Blocks! All that is needed is a facility to do the following:

1. Identify what filter (by module name) is to be removed;
2. Locate the filter in memory via the @GTMOD SuperVisor Call;
3. Obtain the MODDCB pointer to its Device Control Block;
4. Scan through all DCBs to find the DCB pointing to the filter;
5. Then swap the three bytes."

CON80Z/PRO-CON80Z by Roy Soltoff

Quite often when you need a specific tool, it is unavailable. For the Z-80 assembly language programmer, the need arises to maintain or modify programs written in 8080 code using Intel mnemonics. Since 8080 code is a subset of Z-80 code, a useful approach is to translate the 8080 source file to Zilog mnemonics (Z-80) source code. You could hand translate your 8080 files to Z-80 files - a formidable task, indeed! An alternative would be to use a translator program. **CON80Z** should prove quite useful in such a task. The tool has been designed to facilitate the conversion of assembler source files written in 8080 Intel mnemonics to Z-80 Zilog mnemonics. **CON80Z** is a source translator to help you convert your 8080 files to Z-80 files - easily.

CON80Z consists of two programs: One, CON80Z/CMD, performs the necessary translations of code on a line by line basis. The translation is one-to-one. Each logical input line is replaced by one output line. The second program, UNNUMBER/CMD, is a preprocessor to CON80Z/CMD and is used to alter certain source files to conform with the requirements for the input file structure.

Although certain 8080 code sequences can be optimized if the Z-80 extensions are utilized, **CON80Z** does not perform optimizations. **CON80Z** does help to transform the source into a file structure that can be loaded by your assembler's editor. Most 8080 assembler source files are structured as pure ASCII files with each line terminated by a Carriage Return (CR) followed by a Line Feed (LF). Source lines are also generally not line numbered as are those used with most TRS-80 assemblers. **CON80Z** will expect the source file to be un-numbered. The line feed may or may not be present.

Some 8080 assemblers support a logical line ending character, such as the exclamation mark (!), to create multiple source statements on one physical line. By using the CR="c" parameter in the command line, the character "c" will be interpreted as a logical line end when found in the operand field of the source statement and not within single quotes.

Register nomenclature in 8080 code is always a single character. Eight-bit register references in 8080 assembler language are identical to Z-80 references [B, C, D, E, H, L]. The "(HL)" 8-bit memory reference is denoted in 8080 code as the single character, "M". The appropriate translation from "M" to "(HL)" will be made by **CON80Z** wherever necessary.

The 8080 16-bit registers available are denoted as B, D, and H with the OP code changed to "extended" to interpret the reference as 16-bit register use (e.g. LD changed to LDX). In addition, the Accumulator and FLAG register are referred to as "PSW" when used in PUSH and POP instructions (PSW is a carryover from main frames and stands for Program Status Word). CON80Z makes the appropriate translations on extended instructions and will translate B, D, H, and PSW to BC, DE, HL, and AF.

During the translation process, **CON80Z** will convert all comments in upper case characters to lower case characters except for the character immediately following the semicolon (;) comment indicator. **CON80Z** will also translate multiple blanks used as field separators to one tab (X'09').

CON80Z will perform translations on selected pseudo-ops where there is a similarity of usage on common TRS-80 assemblers. The following pseudo-op translations will be performed: <DB/DS/DW/SET> to <DEFB/DEFS/DEFW/DEFL>.

ORDERING INFORMATION:

CON80Z: For use with the TRS-80 Models I/III/4 under LDOS 5.x
PRO-CON80Z: For use with LDOS/TRSDOS Version 6 [i.e. Model 4].

CONVCPM/PRO-CURE (Version 2) by Roy Soltoff and Richard A. Deglin

This utility will allow you to transfer files from certain CP/M diskettes onto an LDOS (or TRSDOS 6) formatted diskette. Since there is no standard media format for CP/M 5" diskettes, MISOSYS has chosen to support some of the most popular formats. The CP/M formats supported are standard 8" Single Density (where your hardware permits 8" floppy drives), 5" Single Density single-sided 128-byte sectoring with 16 or 18 sectors per track (Omikron version and equivalent), 5" Single Density single-sided 256-byte sectoring with 10 sectors per track (Osborne format), 5" Double Density single sided 256-byte sectoring with 18 sectors per track (Model 4, Kaypro) and IBM PC media 512-byte sectors [Note: 512-byte sector support is limited to TRS-80 Model III and 4. Other machines may be utilized providing the disk controller driver used by the operating system supports reading 512-byte sectors.] Two drives are required.

The transfer utility will allow you to move all or groups of files from certain CP/M disks onto your LDOS disks. It provides many different parameters to choose the files to be moved. The file specifications on the CP/M disk should conform to LDOS file specification standards. The filename and extension must begin with an alphabetic character [A-Z[. Subsequent characters of the filename and extension must be either one of the alphabetic characters or a numeric [A-Z,0-9]. Any CP/M file not adhering to the LDOS standard will require a correct LDOS filename which will be prompted for prior to a transfer operation of the file.

The CONVCPM utility has been designed to aid in transferring data files and other files that are not directly executable under CP/M [COM files are directly executable under CP/M and although transferable with CONVCPM, they are not "loadable" under LDOS]. Once moved to an LDOS diskette, the transferred file is an exact image of the file as it appeared on the CP/M diskette. The LDOS end-of-file mark is established as if the moved file ended on a sector boundary.

CP/M generally uses a sector interleave translation scheme during disk I/O. The transfer utility has two sector interleave tables for commonly used CP/M formats. The single-density 8" diskette structure supported is the Digital Research standard. Each company implementing a version of CP/M on other than 8" single density media chooses their own sector interleave translation table. Thus, your version of CP/M that is on 5" media may or may not utilize the same translation table as that used in CONVCPM which is that implemented by Omikron, Osborne, Kaypro, or IBM. CONVCPM translation tables are Single Density 8" (1, 7, 13, 19, 25, 5, 11, 17, 23, 3, 9, 15, 21, 2, 8, 14, 20, 26, 6, 12, 18, 24, 4, 10, 16, 22) and Single Density 5" (1, 5, 9, 13, 17, 3, 7, 11, 15, 2, 6, 10, 14, 18, 4, 8, 12, 16). A parameter provides the means for entering a different translation table into CONVCPM.

The package includes a utility which can be used to remove line feeds after carriage returns. The LDOS end-of-line sequence is CR only while CP/M uses CR-LF. This utility will be helpful to remove the extraneous line feeds from text files by being invoked under your control!

ORDERING INFORMATION:

CONVCPM: For use with the TRS-80 Models I/III under LDOS 5.1.
PRO-CURE: For use under LDOS/TRSDOS Verison 6 [i.e. the TRS-80 Model 4].

DSMBLR/PRO-DUCE III - by Roy Soltoff

Programmers have probably been disassembling machine code programs since the time programs were being "hand" assembled. What is a disassembly? Simply the reverse process of assembly - taking a program or piece of a program and translating it back to an assembly language state. This disassembler is a tool for helping you with the process. **DSMBLR III** is a third generation product. This tool provides extensive capabilities such as direct disassembly from CMD disk files, automatic partitioning of output disk files, data screening for non-code regions, and full label generation. **DSMBLR III** even generates the ORGs and END statement - the complete ball of wax.

Surely a tool of this capability should be difficult to use. Not so! You will find that the use of this disassembler - even by a beginning assembly language programmer - will be paying handsome rewards with the ease of its use and clarity of the documentation. **DSMBLR III** is a professional tool for your use.

This program is a machine language labeling disassembler that produces an assembler source code using ZILOG mnemonics from either Z-80 machine language resident in memory or directly from a disk CMD-type file. Both products **[DSMBLR** and **PRO-DUCE]** continue the tradition that have made **DSMBLR** a standard for over four years. The disassembler operates in two passes in order to incorporate symbolic labels in the source output. The symbolic labels are generated for address and 16-bit references within the start-to-end disassembly request.

References preceding the START address or references that follow the END address are output as EQUates. Addresses between program segments such as would be referenced from DEFS-type instructions are also output as EQUates. EQUates are also generated for each symbolic label found in the symbol table that does not correspond to the start of an instruction. These labels are output in the form:

LABEL EQU $-n

where "n" is the offset to the label address from the current program counter. Equates are also generated for all address references that extend beyond the end of the target program. A reference is any relative instruction target address or a 16-bit target for load, call, jump, add, or subtract instructions.

Just about every program that you will disassemble has segments that are actual code and other segments that are data. We don't leave you out in the cold. The disassembler will assume that all segments are actual code unless told otherwise by means of a "screening data file". The "screening data" entry is the way in which you tell the disassembler what segments of the program are to be interpreted as data regions.

You arrive at the addresses of the "segments" by an analysis of the target program. For instance, a first disassembly to the video screen will easily identify literals since the ASCII equivalent of the object code is displayed. Make note of the address ranges on a piece of scrap paper to be used in building a text file of screening data fields. The "smarter" you are in assembly language, the easier it will be to identify word and byte data.

Data usually take one of three forms: literal fields commonly called strings (words that you can read - i.e. messages, prompts, etc.); byte fields of varying length with each byte a distinct value (tables, conversion codes, one-byte length specifiers), "confusion" bytes (hex values placed to alter the sense of following code depending on entry point; and "words" of varying length (16-bit values commonly used as pointers, arithmetic values, etc.).

The disassembler recognizes certain prefix specifiers to force the data generation to literal, byte, or word formats.

Where a literal has been forced, the disassembler maintains a range check for the characters. Valid literal characters are in the range X'20'-X'26' and X'28'-X'7E'. If a character value is outside this range, the decomposition will automatically revert to "xxH' format starting with that character and continuing until a character within the literal range is detected. For example, The byte sequence: "4C 61 6E 27 74 0D" will decompose to the pseudo-OP declaration: "DB<tab>'Can',27H,'t',0DH". The disassembler will output approximately 18 characters in the operand field of a line. If the screening range is such that the decomposition would exceed that limit, a subsequent line is generated. Also, any labels that would be addressed in the scope of the DB or DW field will be output immediately following the line. A sample screening data input is:

```
5228-5229,$5384-53aa,#5829-5832,5416
$5b20-5b3d,5f67-5f68.
```

During the disassembly, the byte value of instructions that have a byte-value operand will be displayed in either of two formats depending on the value of the byte. Bytes in the range X'20'-X'26' or X'28'-X'7E' will be output as literals in the form, "'c'". All other values are displayed in the form, "xxH", or if the value is in the range X'A0'-X'FF', "0xxH". The byte values of index instruction offsets are output in the non-literal format only. Also, the port number of IN and OUT instructions is kept non-literal. In most cases, this display format provides more meaningful information than displays strictly in the non-literal format. You will notice output displays such as:

```
100E FE45     02443      CP   'E'          .E
1010 2810     02444      JR   Z,M1022      (.
1012 FE44     02445      CP   'D'          .D
1014 280C     02446      JR   Z,M1022      (.
1016 FE30     02447      CP   '0'          .0
1018 28F0     02448      JR   Z,M100A      (p
```

which provide a greater ease in understanding the logic of a program.

Output routed to the CRT is displayed in screen-sized pages. The display will include the hexadecimal address, the machine code in hexadecimal, a sequential line number, the OP code, operand, and displayable ASCII characters equivalent to the disassembled instruction's hex code. A page advance is user controlled by key entry.

Output routed to the PRINTER is paged at 56 lines per page. Each page has column headings, supports a user-entered TITLE, and is numbered for producing sophisticated print-outs that look identical to an assembler listing. Columns include ADDRESS, HEX CODE, LINE NUMBER, OPCODE, OPERAND, and ASCII equivalent of the hex code.

For the Model I/III user, output can be routed to the TAPE CASSETTE in order to produce a source tape suitable for loading into the Radio Shack cassette Editor Assembler. The tape is generated in blocks of 256 lines of code. A tab character is used between line number, opcode, and operand for best Editor Assembler input.

Output routed to DISK produces a disk file suitable for loading into EDAS or PRO-CREATE, Disk-modified EDTASM, Radio Shack Series I EDTASM, or Microsoft's ALDS (M-80). The default file structure is neither line-numbered nor headered. Options provide for the addition of a file header, line numbers, and colons following labels in order to tailor the file to most

assemblers [MACRO-80 requires line numbers and colons, Radio Shack Series I EDTASM requires line numbers, EDAS supports files with all formats]. DSMBLR III automatically segments output source files into manageable file sizes. The maximum source file size is controlled by you. The disassembler will even prompt you to change the output file diskette when the disk becomes full.

Machine language programs that would overlay the disassembler can be relocated by BASIC or a utility and conveniently disassembled with proper address references by using the RELOC feature. The target program could also be disassembled directly from disk to avoid all memory conflicts.

Here is a sample of the output:

```
MISOSYS Disassembler - Disk Version 3.0  Partial ROM  PAGE  00001
ADDR CONTENTS LINE# LABEL  INSTRUCTION      ASCII

              00001        ORG  0000H
0000 F3       00002 M0000  DI                s
0001 AF       00003 M0001  XOR  A               /
0002 C31530   00004        JP   M3015          C.0
0005 C30040   00005 M0005  JP   M4000          C.@
0008 C30040   00006        JP   M4000          C.@
000B E1       00007 M000B  POP  HL              a
000C E9       00008        JP   (HL)           i
000D C31230   00009        JP   M3012          C.0
0010 C30340   00010        JP   M4003          C.@
0013 C5       00011 M0013  PUSH BC              E
0014 0601     00012        LD   B,01H          ..
0016 182E     00013        JR   M0046          ..
```

ORDERING INFORMATION:

DSMBLR III: For the TRS-80 Model I with LDOS 5.0, 5.1, TRSDOS 2.3 and the TRS-80 Model III with LDOS 5.1, TRSDOS 1.3
PRO-DUCE: For use under LDOS/TRSDOS Version 6 [i.e. Model 4].

EDAS/PRO-CREATE Version IV - by Roy Soltoff

EDAS is a powerful disk-based combined editor and Z-80 macro assembler. Among its features are direct assembly from one or more source disk files or memory buffer, conditional assembly, macro assembly, extensive cross reference listings, and a comprehensive line editor that supports upper and lower case text entry. **EDAS** was used to develop LDOS 5 & 6 and TRSDOS 6.

EDAS ASSEMBLER FEATURES

EDAS assembles absolute core-image object code to disk as a directly executable load module (CMD). Source code can exist in memory as well as included disk files when using the *GET assembler directive. *GET files can be nested to five levels. **EDAS** uses default file extensions of "ASM" for source and "CMD" for object code files to guard aginst inadvertant over-write of a source file with object code. **EDAS** also respects HIGH$.

A powerful "*SEARCH filespec" assembler directive will invoke automatic search of the Partitioned Data Set (PaDS) "filespec" containing a library of source code. The PaDS directory search will automatically GET any PaDS member that would resolve an undefined label reference. This process can be correlated to a relocating assembler's resolving references at link time. In EDAS, the source library is ISAM accessed for minimal I/O overhead. The PaDS or PRO-PaDS utility is required to construct your own subroutine libraries.

Conditional assembly is supported with seven pseudo-ops; IF expression; IFLT expression1, expression2; IFEQ expression1, expression2; IFGT expression1, expression2; IFDEF label; IFNDEF label; and IFREF label. Conditional assembly also supports the "IFx ELSE ENDIF" construct. Conditional expressions can be nested to 16 levels.

The expression evaluator supports left-to-right evaluation of the following operators: "+" addition; "-" subtraction; "*" 16-bit by 8-bit integer multiplication; "/" 16-bit by 8-bit integer division; ".MOD" modulo division; "<" shift; "&" or ".AND." logical AND; "!" or ".OR." logical OR; ".XOR." logical exclusive OR; ".NOT." unary one's complement; ".NE." logical not equal; and ".EQ." logical equal.

Pseudo-ops DEFB and DEFM are synonymous. **EDAS** also accepts DS, DW, DB, and DM as well as DEFS, DEFW, DEFB, and DEFM. **EDAS** provides for binary, octal, decimal, hexadecimal, and string constants. Constant declarations can be concatenated on one line, by separating terms with commas. This permits complex expressions such as:

```
DB  1,2,'Buckle your sho','e'.OR.80H,'I can''t'
```

Labels may be up to 15 characters long. Labels must start with A-Z, "@", or "$". Positions 2-15 may also use "?" and "_". The "*MOD" assembler directive is available to provide a unique character string substitution for the "?" character appearing in labels of all files accessed via *GET. The string value will increment each time *MOD is commanded. This will provide "local label" support for routines read off of disk.

A logical origin pseudo-op, LORG, will assemble load module files with the load addresses offset to the LORG address while execution addresses are based on the ORG address. When using **EDAS** to assemble applications that block move sections of code, the LORG can be used to assemble the entire job at once.

The **EDAS** assembler provides many switch options. These invoke: "-IM" assemble output to memory; "-LP" list to printer; "-NC" suppress false conditional blocks from listings; "-NE" suppress constant expansions on listing; "-NM" suppress listing of macro expansions; "-SL" suppress local labels from symbol table listings; "-WE" wait on error; "-WO" assemble with object code; "-WS" generate a sorted symbol table listing; "-XR" generate a cross reference data file for downstream processing by **XREF.**

Nested 7-level MACROs are supported with both positional parameters and parameters by keyword. Values can be applied to any parameter at MACRO definition time to allow for expansion time defaults if a parameter is omitted at the time a MACRO is referenced. MACROs can be defined in memory or source files but must be defined prior to being referenced. Local labels are supported with the provision of a string substitution for the "?" character in labels. The string will provide a unique value for each MACRO expansion. The MACRO "?" substitution takes precedence over any *MOD substitution.

Additional pseudo-ops are provided for enhanced operation: "COM" will allow a comment line to be written to the load module. These comment records will not be loaded when executing the module, but will merely provide an easy way to store such things as copyright messages in your object deck files; "TITLE" will paginate your listing with a title string including the current date and time, and an incrementing page number; "SUBTTL" lists the sub-title string after each title; "PAGE" ejects a listing to a new page; "SPACE" generates additional line feeds during listings for highlighting modules.

A sorted symbol table listing is available during the assembly. A complete CROSS REFERENCE listing is available by a downstream processing utility, **XREF.** Once an XR data file is generated, **XREF** will produce a listing identifying all defined labels, the line number containing the definition, its value, and the file name of the source file containing the definition ($CORE is used to designate labels defined in memory). For each defined label, all references to the label are listed by line number and source file containing the reference. **XREF** lists statistics on the quantity of defined labels and references. **XREF** can also be used to generate a file containing EQUates (or DEFLs) for all symbols or a subset of symbols (those including a special character). The EQU file is useful for interfacing separately executable modules to a resident module [such as in overlay applications].

EDAS EDITOR FEATURES

The **EDAS** editor operates on text in memory and uses a command syntax identical to BASIC for intra-line editing. Lines hacked to null length will be automatically deleted.

EDAS will "Load" and "Write" text buffers from/to disk with text file concatenation in memory. The standard source file will be un-headered and un-numbered which saves approximately 20% of disk file storage requirements. However, **EDAS** will AUTOMATICALLY recognize and properly read a file that is headered and/or numbered whether through "Load" or "*GET" input. Two switches are provided in the "Write" command to generate a header or line numbers when saving a text buffer to disk.

You can input text in upper or lower case. In the case-converted mode, all assembler source input is properly converted to upper case, AUTOMATICALLY. In the case consistant mode, text remains as it was input. Thus, the editor can be used for assembler source, or source for other languages such as PASCAL and C.

The editor supports relocating a block of lines with the "<M>ove start,end,to" command. Global changes to character strings can be made throughout the text buffer or to only a designated range of lines with the "<C>hange /string1/string2/start,end" command. Want to copy a block of lines? The "<C>opy start,end,to" command will duplicate the block numbered from "start-end" to follow the line numbered "to".

A "<F>ind string" command will search the text buffer starting from current line+1 for the next occurrance of "string". String may be up to 15-characters in length. If "string" is null, the next occurrance of the previous "find string" will be searched for.

Single line scrolling is supported with the <UP-ARROW> and <DOWN-ARROW> keys. The <SHIFT-CLEAR> key invokes a "warm-boot" which aborts the current operation, clears the screen, and re-initializes line numbering while maintaining the current text buffer.

A "<U>sage command displays buffer status (in use and remaining), and the first available in-memory address. The latter is useful for assembling into memory then executing a "<B>ranch" to the in-memory object program for debugging purposes.

EDAS provides MiniDOS-type directory "<Q>uery" and file "<K>ill" functions. A "<V>iew" command will list a source file to the screen without affecting the buffer contents. **PRO-CREATE** lets you invoke DOS commands from within the editor.

When all things are considered, if you are writing system software, support software, applications - big or small, **EDAS** will provide the power to make your assembly job easier, faster, and more worthwhile. It does everything but teach you how to program. **EDAS** comes complete in a three-ring binder with extensive documentation of over 100 pages of useful information (not OP-code explanations). A plastic Z-80 quick reference card is included.

ORDERING INFORMATION:

EDAS: For the TRS-80 Models I/III/4 under LDOS 5.1.
PRO-CREATE: For use with LDOS/TRSDOS Version 6 [i.e. Model 4].

GRASP - by Karl A. Hessinger and Scott A. Loomer

The GRAphics Support Package (**GRASP**) is a collection of programs, filters, and drivers that will enhance the capabilities of your Epson MX-80 Graftrax or MX-100 printer. **GRASP** implements customized character sets which include standard ASCII characters, TRS-80 graphics blocks, and Model III special character symbols.

A screen-oriented character editor makes it easy for you to modify or create any character font you desire up to a size of 16 vertical by 12 horizontal dots. If you use the double-character mode, your character font can occupy a width of up to 24 dots. The editor displays an individual character in a visual matrix made up of large graphics blocks. By manipulating the graphics cursor within the matrix, you control exactly what "dots" will be present in your character.

Filters are provided to toggle underlining and invoke selected double-width characters intermixed with standard width. Another filter gives you the capability of printing the Model III special characters with a minimum of high-memory usage.

A program is provided to easily set the custom functions of your MX-80G or MX-100 from the DOS Ready mode instead of having to write complex PRINT CHR$ instructions.

The **GRASP** package includes ALTCHAR/CMD, ALRCHAR/DVR, ALTLINE/FLT, ALTWIDE/FLT, MOD3CHAR/FLT, GPD/DVR, SETMX80G/CMD, SETMX100, and UNDRLINE/FLT - nine programs in all.

The ALTCHAR/CMD program is a special-purpose graphics editor for you to use in constructing and customing entire character sets for use with the ALTCHAR/DVR printer driver.

ALTCHAR comes supplied with seven already defined character sets which are: STD10/12 - a 10/12 pitch character set of "standard" characters, block graphics, and Model III special characters; TYPE10/12 - a 10/12 pitch set of typewriter like characters, block graphics, and Model III special characters; SCI10/12 - a 10/12 pitch set derived from STD10 which includes greek characters plus superscripted and subscripted numerals; and OLDENG - a 10 pitch double-width character set of Olde English characters.

ALTCHAR/DVR implements the printer support drivers that will use the character files to generate the customized character sets on your printer. The driver options include the following parameters: ADDLF will cause a line feed to be sent after each carriage return; SPACE will cause the output of an extra one-half line feed between each line of text; WIDTH establishes the number of characters to print on a line; DOUBLE will cause the interpretation of the character set as being "double-width". HIGH will allow the printing of only characters with an ASCII value less than or equal to the value specified. Only the necessary portion of the character data set will be read and stored in memory, thus allowing you to cut down on the ALTCHAR driver high memory requirements; LENGTH will set the page form length in one sixth inch lines.

ALTLINE is a filter to implement character underlining using a toggle character. The ALTLINE filter works in conjunction with the ALTCHAR driver to allow the printing of a continuous underline with little user intervention. Upon receipt of the switch toggle character, ALTLINE will underline all characters until either the end of the line is reached or the switch toggle character is detected. The toggle character is not printed.

This ALTWIDE filter provides the capability of printing selected characters in double width while all others are printed in standard width. It could be used, for instance, to print all capital letters in double width.

MOD3CHAR/FLT is a filter that adds the capability to print the special video characters as displayed on the Model III without the high memory overhead needed by ALTCHAR. If you only need the special characters, this filter will do it; however, ALTCHAR is still needed for your custom character sets.

GPD/DVR allows the use of all dot addressable graphics on the Epson printers. GPD/DVR replaces the printer driver routines located in the TRS-80 ROM. The TRS-80 ROM printer driver routines convert some characters and trap others. GPD/DVR eliminates this problem. When GPD/DVR is set, ALL codes will be passed unmodified to the printer.

The SETMX80G and SETMX100 utilities permit conviently setting the printer options for the Epson MX-80G or MX-100 printers. Command line options for MX-80G are:

RESET	- reset to defaults	RSmode	- Radio Shack mode
Paper	- paper transfer mode	Emph	- emphasized mode
Comp	- compressed mode	eXpand	- expanded mode
Italic	- italics mode	MSB	- MSB function
Double	- double strike mode	Space	- line spacing
Form	- form length in lines	Lines	- lines per inch
Margin	- restores PR/FLT left margin		

The SETMX100 program also supports the following:

US/FRench/GErman/ENglish/DAnish/SWedish/ITalian/SPanish			
SKip	- skip over perfs	COLumn	- column width

UNDRLINE/FLT is used to provide an easy means of underlining on any printer that will backspace (without erasing) and print an underline character (ASCII 95). This filter will work with the Epson MX-80 w/Graftrax but not with the Epson MX-100. The character specified by the parameter, CHAR, will be used to start and stop (toggle) underlining.

ORDERING INFORMATION:

GRASP: For use with the TRS-80 Models I/III/4 under LDOS 5.1 and Epson MX-type printers.

HELP by Scott A. Loomer

The **HELP** package provides prompting notes on the LDOS 5.1 system commands and syntax. It is supported under LDOS on lower case equipped Model I and Model III machines. **HELP** contains two types of files; HELP and SYN. The HELP files contain detailed descriptions of the system functions including explanation of parameters and their default values. The SYN files will be useful to the more experienced user. They consist of the syntax necessary to invoke the function without an explanation of terms. Both the HELP and SYN functions can be executed from LDOS Ready or from within LBASIC using a statement of the class: CMD"HELP_(command)".

The **HELP** programs provide adequate explanations of the specified commands. Each HELP file also contains a HELP function that will yield a menu of the helps within the particular HELP file.

The HELP and SYN files are implemented using the Partitioned Data Set (PaDS) utility. Each member of the data sets is an executable machine language program that consists of text that is loaded directly into screen memory, a short routine to position the cursor and a transfer to the LDOS Ready prompt. This manner of implementation allows quick access to any help member within the files with minimal memory requirements and rapid response time. When you type HELP_(command), only a small front end loader actually loads into memory. It, in turn, clears the screen then loads the HELP explanation directly into the screen memory.

A BASIC program is included with the **HELP** utility package that will assist you in creating your own "HELP" type files. The program, TEXTCMD/BAS, provides an easy means of converting text files created with a text editor into executable CMD files. You will need the PaDS utility in order to create new HELP files; however, PaDS is NOT needed in order to use any of the HELP files included in this package.

The **HELP** utility also comes with a Quick Reference Card. The QRC is a ten-panel foldout card that identifies all LDOS library commands, utilities, drivers, filters, Disk BASIC, and Job Control Language. With **HELP** at your disposal and the QRC at your side, keep you LDOS manual on the shelf and consult it when you need in-depth information. Over 90% of your reference needs could be rapidly satisfied with the **HELP** series. Do you need HELP?

PRO-HELP by Roy Soltoff

The **PRO-HELP** package is custom designed for TRSDOS 6.x! This package contains two PDS files packed with information. One file contains all of the mind joggers needed to use the DOS. The other places all of BASIC's reserved words at your fingertips. **PRO-HELP** automatically displays the names of all of its screens at your command so you do NOT even have to remember the names of commands or reserved words. Rather than provide you with two unmanageable disks of data files, **PRO-HELP** gives you the brief syntax necessary to invoke or use the system in two files that don't take up a whole drive! The DOS HELP file is 21.25K while the BASIC HELPB file is only 15K.

This package includes three programs: HELPGEN, HELPTXT, and HELPADD. These are all that you need to create your own partitioned data set HELP-type files for your application programs. Get HELP at your fingertips!

ORDERING INFORMATION:

HELP: For use with the TRS-80 Model I (lower/case) and III with LDOS 5.1.
PRO-HELP: For use with LDOS/TRSDOS Version 6.

LC C-COMPILER with MACRO ASSEMBLER - by Jim Frimmel and Roy Soltoff

One of the high-level languages getting a great deal of attention is the "C" language. This is due in part from the knowledge that UNIX, a powerful operating system for minicomputers, mainframes, and now micros, is written in the C-language. Why did they choose C? Because the UNIX designers realized that application software and system code could be both created and maintained more easily when written in the medium-level C-language. Another reason for C's growing popularity, is that it is a language rich in the use of expression operators, functions and structured code.

If you would like to get started in C, or you are a C expert just anxious to use a C-compiler on your micro, your waiting days are over. "LC" (pronounced 'elsie'), a C-language compiler, is available for use. The LC compiler provides a substantial subset of the C programming language as described in, "THE C PROGRAMMING LANGUAGE" by Kernighan and Ritchie. LC was written to be compatible with UNIX programs. LC programs using the standard library (supplied with the compiler) can be compiled and run under UNIX. Programs written under UNIX which use only statements supported by LC are also portable to LC. A large amount of existing software, both commercial and public domain, will be directly usable by LC owners.

C is a structured, portable language. A "C" program is a collection of functions arranged hierarchically. C functions can be recursive and re-entrant, as local variables are created and stored in a stack. All machine-dependent features needed, such as I/O, are not implemented in the language; rather, they are placed in the standard library. Thus, only the implementation of the standard library changes from installation to installation, and C programs are written in machine independent ways. The language itself provides ways of expressing program structure, and of giving arithmetic and logical expressions. C is known for having one of the most powerful expression capabilities available in any language. C statements supply the WHILE, DO-WHILE, FOR, IF, and SWITCH-CASE constructs. C also provides powerful pointer capabilities to enable direct access to memory and variable storage.

LC is an integer-only implementation of C which provides all C statements except "struct", "union", and "typedef". All data types except "float" and "double" are implemented; "long" and "short" declarations are accepted, but 16-bit fields are used for all integers. In LC, "char" variables are implicitly unsigned. Single-precision and double-precision floating point operations are supported via functions supplied in the FP/LIB library included with the LC compiler. LC accepts multiple input files, with four levels of nesting for "#include'd" files. The compiler generates an EDAS Version IV assembler source file which is then assembled with the standard library and any other libraries needed to resolve function references in order to generate the executable program. The value in generating assembler source is twofold. First, you can obtain a complete machine code source listing which could prove invaluable in debugging complex code. Second, local optimization of assembler source code can be performed as required by the experienced assembler programmer. The LC standard library provides such functions as standard I/O redirection, dynamic memory allocation, automatic standard I/O opening and closing, and program chaining. In addition, functions specific to the DOS and the hardware are supplied in an installation library, to provide access to such functions as graphics and system entry points.

LC supports separate compilation; programs may be compiled in segments, and frequently used functions can be pre-compiled. You can create your own library of commonly used functions with the Partitioned Data Set utility (PaDS is not included with LC but is available as a separate package). The assembler source code output by LC is designed to use the extensive SEARCH and conditional assembly support in EDAS Version IV. The assembler and companion assembler cross-reference utility are supplied with the LC package. You need nothing more to start writing and running C-language programs except your computer and a copy of "THE C PROGRAMMING LANGUAGE". LC requires two disk drives and lower/UPPER case.

Some highlights of the "Elsie" compiler are:

- o Integer subset of the C language.
- o Access to floating point routines via function calls.
- o All statements supported except STRUCT, UNION, TYPEDEF.
- o All operators supported except "->", ".", SIZEOF, and (TYPENAME).
- o UNIX-compatible standard I/O library.
- o Standard I/O redirection with complete device independence.
- o Input using FGETS or GETS functions support DOS Job Control Language.
- o Dynamic memory management (ALLOC, FREE, SBRK).
- o Sequential files open for READ, WRITE, and APPEND.
- o Generates Z-80 EDAS Version IV source code as output.
- o User libraries in Z-80 source ISAM-accessed PDS files.
- o Compact one-line invocation of the compiler.
- o LC's interactive friendly interface provides easy way to learn LC options.
- o Supports separate compilation of functions.
- o Compiled programs run under both Models I and III without modification.
- o Installation library gives access to graphics and DOS calls.
- o Supplied with example programs and utilities in source form.
- o The LC package includes the compiler, three libraries, the complete, macro assembler package, and more than 250 pages of documentation.
- o National LC Interest Group [LCIG] available with public domain LC compatible C source software.

As the C language is defined by Kernighan and Ritchie, LC supports all unary operators except "sizeof" and "casts". LC supports all binary operators. LC supports the conditional operator [? :]. LC supports all assignment operators. LC provides limited support of the comma operator within the "for" statement. LC supports all storage classes except "typedef". LC supports "char", "int", and "unsigned" type specifiers. LC does NOT support initializers but can initialize all statics to zero via an option. LC supports single dimensioned arrays. LC supports #define [without macro substitution], #include, #asm, #endasm, and #option.

The standard library, LC/LIB, includes the functions: alloc, atoi, exit, fclose, fgets, fopen, fprintf, fputs, free, getc, getchar, gets, isalpha, isdigit, islower, isupper, itoa, itox, move, printf, putc, putchar, puts, sbrk, toupper, tolower, and xtoi.

The installation library, IN/LIB, includes the functions: box [draw a box given two opposing corners], call [invoke a machine language routine with register linkage], circle [draw a circle or arc given radius and arc parameters], cmd [execute a DOS command and return], cmdi [exit the LC program then execute a DOS command], curpos [obtain the current cursor position], cursor [position the cursor to row,col], date [obtain system date string], fill [set a memory block to a value], freemem [obtain size of available memory], inkey [strobe the keyboard], inport [fetch input from a port], line [draw a line between two points], outport [output a value to a port], pixel [set, reset, or point a graphics pixel], ploc [alter the graphics plotting origin], pmode [set the graphics plotting quadrant], point

[obtain the state of a pixel], reset [turn off a pixel], set [turn on a pixel], strcat [concatenate two strings], strcmp [compare two strings], strcpy [copy one string to another], strepl [replace a substring with a string], strept [replicate a source string], strfind [find a substring within a string], stright [replace a destination string with the rightmost portion of a source string], strleft [replace a destination string with the leftmost portion of a source string], strlen [obtain the length of a string], strmid [relace a destination string with a substring], and time [obtain the system time string].

The floating point library, **FP/LIB and FP/OBJ**, includes the functions: **dadd, dabs, dcmp, ddiv, dfix, dint, dmul, dsgn, dsub, dtoa, dtoi, dtof, fpinit, fabs, fadd, fatn, fcmp, fcos, fdiv, fexp, ffix, fint, flog, fmul, fraise, frnd, fsgn, fsin, fsqr, fsub, ftan, ftoa, ftod, ftoi, itod, and itof.** These functions support single-precision floating point arithmetic including transcendental functions as well as double-precision arithmetic.

With LC, in no time at all you will be writing C programs such as:

```
#include stdio/csh /* standard I/O definitions */
/*  XFER - copy standard input to standard output  */
int  c, bytes, lines;
FILE *fp;
main()
{   while( (c=getchar()) != EOF)
    {   putchar(c);
        ++bytes;
        if (c == EOL) ++lines;
    }
    fp = fopen("*do","w");
    fprintf(fp,"%d characters , %d lines were copied", bytes, lines);
}
```

This program copies standard input (*KI) to standard output (*DO) while it counts the number of characters and lines. However, with LC's I/O redirection, input and/or output can be changed to any other device or file. Type directly into a file or copy a file to a printer!

Modernize your programming skills and begin writing maintainable applications. Get C - get LC!

ORDERING INFORMATION:

LC: For use with the TRS-80 Model I/III/4 under LDOS 5.1. Includes LC/CMD, LC/LIB, IN/LIB, FP/LIB, LC/ASM, LC/JCL, LCMACS/ASM, DO/FLT, STDIO/CSH, EDAS/CMD, XREF/CMD.

PRO-LC: For use with LDOS/TRSDOS Version 6 [i.e. Model 4]. Includes LC/CMD, LC/LIB, IN/LIB, FP/LIB, FP/OBJ, LC/ASM, LC/JCL, LCMACS/ASM, STDIO/CSH, PRO-CREATE, XREF/CMD, FPCAT/CMD.

MSP-01/PRO-GENY - By Roy Soltoff and Scott A. Loomer

This package is a collection of four utility programs to further enhance the use of your DOS. These programs are entitled: **DOAUTO, DOCONFIG, MEMDIR,** and **PARMDIR** [under LDOS/TRSDOS 6, **SWAP** replaces DOAUTO].

How many times do you suppress the execution of an "AUTO" command by holding down the <ENTER> key on boot-up only to later decide you want to execute the AUTO. However, you really don't remember the specific syntax of that complicated LBASIC command line that auto-executes. Wouldn't it nice to be able to command the AUTO to execute without having to type BOOT or hit the RESET button? The **DOAUTO** command, written by Roy Soltoff, is a short program that will execute the "AUTO" command buffer located on ANY drive - not just the SYSTEM drive. It's as easy as typing "DOAUTO :2".

DOCONFIG, written by Roy Soltoff, is a major enhancement of the SYSGEN configuration capabilities of your DOS. **DOCONFIG** works in one of two ways. You can SAVE the current configuration of your system to ANY file of your choice on any drive of your choice. You can also restore the machine's configuration at any time from any of the configuration files you created. The configuration file is constructed identically to the LDOS CONFIG/SYS file, except that now YOU control configurations without having to re-boot your machine [LDOS/TRSDOS 6 DOCONFIG files do not overwrite the user program region with drive control table information as is done under TRSDOS 6 and thus cannot be used as CONFIG/SYS files. This was done to permit you to save/restore the machine's state while within a running BASIC program].

DOCONFIG can even be executed from a Job Control Language file to either SAVE or RELOAD a configuration file while the JCL is executing. This will work even if a re-loaded configuration changes the drive assignment for the drive currently executing the JCL file - be it the system's SYSTEM/JCL file or your own execute-only JCL file. **DOCONFIG** is smart enough to correct the JCL interfacing being done by LDOS if drive assignments are switched. If the JCL is SAVING a configuration, the CONFIG file will not reflect JCL as being active. The use of **DOCONFIG** now gives JCL more power to run job streams that require revised high-memory configurations for selected applications. Wow, dynamic reconfiguration - on the fly!

Ever wonder what in the world was up in high memory when you execute a MEMORY command and it says HIGH$=X'E123'? Where did all that memory go? No need to wonder any more. **MEMDIR,** written by Scott Loomer, is here to give you a directory of high memory. It tells you what program/module is there, where it resides, and how long it is. MEMDIR makes use of the front end linkage protocol as documented by Logical Systems. For the LDOS/TRSDOS 6 user, **MEMDIR** even provides a directory of low-memory driver and module usage.

The biggest part of the package is **PARMDIR,** written by Roy Soltoff. This is a tough one to explain. Essentially, **PARMDIR** is a specific-purpose report writer that uses the on-line disk directories as a data base of information. PARMDIR was originally written to automatically generate Job Control Language files based on tests of data contained in the directory. For instance:

```
PARMDIR /DOC:3 REN:0 (A="RENAME ",X="/SCR")
```

will produce a JCL file containing an entry for all files on drive 3 that have an extension of "/DOC". Each JCL line of the file, REN/JCL, will appear as: "RENAME filename/DOC:3 /SCR". If the parameters were entered as "(A,X)", then each JCL line would appear as: "#A# filename/SCR #X#". Thus, at JCL compilation time, parameters may be substituted for "A" and "X".

However, **PARMDIR** goes light years beyond this simple example. You can have any of the parameters A,B,C,X,Y,Z be constructed with directory data

information for each filespec selected. The information is positioned according to key-word assignment within the parameter string. For example,

```
"(A="$NAM $EXT $LRL $REC")"
```

will recover in each output line, the file name, extension, logical record length, and number of records. Keywords are available also for protection level ($PRO), ending record number ($ERN), file date ($DAT), end-of-file byte location ($EOF), drive spec ($DRV), volume name ($VNM), volume date ($VDT), or the entire volume id ($VID).

Each of the keywords (except filename/ext) may be tested for value comparisons in order to select the directory record for output. The comparison is constructed as a complex "IF expression" syntax. For example:

```
IF="$LRL <= 18 & $REC < 3"
```

selects those directory records with a logical record length of from 1-18 only if the number of records is less than 3. If you make an error in the syntax of the expression, **PARMDIR** will tell you exactly what character was in error - that's friendly!

The output can be directed to any file or device and the output is SORTED by filename/extension. Since **PARMDIR** can make extensive use of parameters, you can enter parms in the command line OR from any file or device. You can create a PARMSLIB disk file that contains NAMED parameter procedures and refer **PARMDIR** to the specific procedure of parameters for a particular execution of **PARMDIR** - just like JCL can use a PROCLIB with named JCL procedures. **PARMDIR** even permits you to type in parameters from the keyboard at execution time if you select PARMS="*KI" as the parameter input device. There is no limit to the amount of parameters that can be entered from a parameter file or device input - only the command line limits its entry to 63 characters max.

When **PARMDIR** generates its JCL file, all of your parameter entries are written as comments to the output. You are even provided a parameter to suppress these "notes". **PARMDIR** can automatically generate a Partitioned Data Set (PaDS) "MAP" file as easy as "PARMDIR /TXT:2 ROYS/MAP (MAP)".

PARMDIR can access the directory information of a specific drive or all on-line drives. You can use **PARMDIR** to construct customized directory listings. Use it to mechanize your JCL file construction. In essence, **PARMDIR** is the most versatile program to come along that lets you tap the data contained in directories.

The fourth utility supplied with **PRO-GENY** is **SWAP.** The **SWAP** utility, written by Roy Soltoff, allows to to switch the logical-to-physical disk drive assignments for any two logical drives. This can be done at DOS Ready or while within executing Job Control Language. **SWAP** is useful for switching hard disk drive assignments from slots 4-7 to 0-3. Since the DOS always globally searches disk drives in the logical drive order 0-1-2-3-4-5-6-7, SWAP is useful to just change the search sequence to your requirements.

PARMDIR is worth the cost of the entire package. However, you get PARMDIR, MEMDIR, DOCONFIG, and DOAUTO/SWAP in the **MSP-01/PRO-GENY** package. You won't know how you ever got along without it.

ORDERING INFORMATION:

MSP-01: For use with the TRS-80 Models I/III/4 under LDOS 5.1.
PRO-GENY: For use with LDOS/TRSDOS Version 6 [i.e. the Model 4].

PaDS/PRO-PaDS - by Roy Soltoff

Katzan, in OPERATING SYSTEMS, A PRAGMATIC APPROACH, defines a Partitioned Data Set (PDS) as "a data file that is divided into sequentially organized members." Katzan further states, "Each PDS includes a directory that points to the beginning of each member. Data sets of this type are most frequently used to store object programs - each member corresponds to a single object program. The PDS as a whole is referred to as a library. Operating system libraries and user libraries are stored in this fashion." This definition describes exactly, the two LIB files in LDOS, SYS6/SYS and SYS7/SYS.

The PDS structure has provided a technique for combining separately executable object programs into one file thereby saving directory slots. It also saves time by not having to load an entire 10K-15K file just to get a few hundred bytes or a few thousand bytes of program loaded if all LIB commands were just one big file. The system overhead of having to read and search the member directory is minimal.

Up until now, only the system library has supported the PDS structure. Now, with this PaDS utility from MISOSYS, you can have "user" PDS structures. The PaDS command can be used to create custom libraries. A library could be a collection of a dozen utility programs - all stored under one name but directly executable by specifying the library name followed by the member name. Consider for a moment, that you have built a library containing CMDFILE, DSMBLR, FED, BINHEX, EDAS, and XREF. The library name MYLIB was chosen. You can then execute EDAS by entering:

MYLIB(EDAS)

at the LDOS ready prompt. If you wanted to build a custom LDOS command library, you could use CMDFILE to extract DIR, COPY, KILL, DEBUG, ROUTE, and RESET from SYS6/SYS and SYS7/SYS and build them into a user SYSLIB. Then you could kill off SYS6 and SYS7 which would save about 15K from your "custom" SYSTEM disk. When you want to do a directory, you would only need to type:

SYSLIB(DIR) :2 (A,I)

to achieve the same result as if you had typed DIR :2 (A,I) on a regular SYSTEM disk. Albeit you could have named your user library, "S" and save the entering of five characters each time you wanted to execute a member of the library. That would let you use "S(DIR)"! PDS also allows you to abbreviate the member's name to as few characters as uniquely identify it. If "DIR" was the only member starting with the letter "D", you could even have entered, "S(D)..."

The PaDS command is itself a Partitioned Data Set and supplies the following functions via installed members:

- o **APPEND** - Appends a new member to the existing PDS and updates the member directory and ISAM table records.
- o **BUILD** - Creates a new Partitioned Data Set. The PDS is composed of a Front End Loader program, a MEMBER directory, and an ISAM table.

- o **COPY** - Transfers an image of a PDS member from the PDS to a designated file.
- o **DIR** - Provides a directory listing for each member with its name, type, date of addition, and file space occupied.
- o **KILL** - Makes a member inaccessible for access.
- o **LIST** - Will list a specific member in standard hex format or ASCII.
- o **PURGE** - Removes killed member(s) from the PDS and compresses the file to reclaim the space previously occupied by the killed member(s).
- o **RESTORE** - Restores a killed file to accessibility.

Here is a sample PDS directory:

```
PDS: U/CMD          07/07/82     Size:  45K Members: 15/ 16
convcpm  P 13-Mar-82  1597     dct      P 13-Mar-82  3620
debugger P 13-Mar-82  2398     dircheck P 13-Mar-82  2137
dirlist  P 27-Mar-82   957     doconfig P 30-Apr-82   459
dsmblr2  P 13-Mar-82  5724     edas     P 13-Mar-82 10123
fed      P 13-Mar-82  7308     led      P 07-Apr-82  5699
monitor  P 13-Mar-82  1814     reformat P 13-Mar-82   614
strip    P 13-Mar-82   767     unhash   P 13-Mar-82   346
xref1    P 13-Mar-82  2127
```

Note that it is sorted, shows the size of each member (in bytes), and has the date that each member was added to the PDS. This is an excellent tool to organize your disk space and unclutter your directories!

ORDERING INFORMATION:

PaDS: For use with the TRS-80 Models I/III/4 under LDOS 5.0.3 or later.
PRO-PaDS: For use with LDOS/TRSDOS Version 6 [i.e. Model 4].

SOLE - by Roy Soltoff

LDOS is a sophisticated operating system. The folk's at Logical Systems have expended great efforts in producing such a powerful DOS for the TRS-80 users. Paramount in their implementation was the concept of standardization. LDOS makes every attempt at standardizing functions and media whenever possible. The media format chosen for double density operation on the Model I was an entire diskette formatted in double density. Since the TRS-80 Model I cannot begin to BOOT a diskette unless the BOOT sector (track 0, sector 0) is formatted in single density, the standard LDOS double density diskette cannot be BOOTed. As a result, some users have taken LSI to task for not providing a means of booting a double density disk on the Model I.

Operation of the RESET button causes the Z-80 CPU to begin execution at address 0. The Model I eventually finds its way to a ROM routine which attempts to read a disk booting routine stored on sector 0 in track 0 into a buffer at X'4200'. The problem here is that this ROM routine can only read the boot sector if it is in single density. Since LDOS has a double density track 0 when a disk is formatted in double density, the ROM cold start routine doesn't like it.

The disk boot routine is supposed to read in the resident system, known as SYS0/SYS. If SYS0 has been read successfully (that means no disk error in big letters), then the booting routine passes control to SYS0. The resident system initialization routine does its thing, loads in a CONFIG/SYS file if one is available, and finally brings in SYS1 to display the "LDOS Ready" prompt and await your command. A lot of work has been done to get to this point. If you are lucky to have a working double density adaptor, then you would have liked all of this work to take place on a double density diskette.

SOLE is an application to accomplish that goal. It will create a double density booting SYSTEM diskette for use with LDOS on a Model I. It essentially constructs a single density track 0 on a previously formatted double density diskette. It then proceeds to add a second BOOT routine and double density READ ONLY disk driver to be used to read SYS0. This **SOLE** BOOT routine and driver is what the sector 0 BOOT routine will read. Since the track 0 is single density, the ROM can read sector 0. The **SOLE** additions are also placed on track 0 so the BOOT routine in sector 0, which expects to see a single density diskette, actually winds up reading only single density sectors. The sector 0 BOOT passes control to the **SOLE** BOOT after it successfully loads the **SOLE** BOOT.

The **SOLE** BOOT routine interfaces with a double density driver that can do only one thing - read sectors. It reads the SYS0 which is obviously positioned on some double density track. After SYS0 is loaded and before passing control to SYS0, the **SOLE** BOOT slides its booting drive code table into the standard drive 0 position. Then when **SOLE** passes control to the SYS0 initialization, SYS0 is interfaced to the double density read-only disk driver. The requirement here is that an LDOS double density driver needs to be loaded. That is accomplished by having it in a configuration file. Thus, when the initialization part of SYS0 loads in the CONFIG/SYS file, the LDOS double density driver is loaded into high memory and the drive code table data is updated.

ORDERING INFORMATION:

SOLE: For use with the TRS-80 Model I [or work-alike] under LDOS 5.0 or 5.1 with either a PERCOM-type or the Radio Shack type double density adaptor.

ZGRAPH/PRO-ZGRAPH by Karl A. Hessinger

ZGRAPH is a graphics editor that allows creation of graphic images. ZGRAPH possesses two sets of commands, primary commands and secondary command functions. A 'help' list of commands at both levels is available by typing <H> for primary commands or <F><H> for secondary functions.

The video display screen of the TRS-80 models consists of memory arrayed as 16 rows of 64 columns or 24 rows of 80 columns. Each memory location is capable of displaying one ASCII or special character or any combination of the six graphic dots referred to as pixels. **ZGRAPH** allows any of the 160 (224 on the Model III/4) possible characters (ASCII, graphic and special) to be displayed at any point on the screen.

Cursor movement depends on the mode that **ZGRAPH** is in. In the graphics mode, movement is achieved using the number keys 1-4 and 6-9 or the ARROW keys. If you go off the screen to the left, you will reappear on the right. The same is true of the top and bottom.

In the DRAW mode, the cursor will leave a trail of bright graphic pixels everywhere it goes. In the ERASE mode, the graphic pixels will be turned off everywhere the cursor is moved. The MOVE mode is a non-destructive means of moving the cursor. While in the text INSERT mode, cursor movement is via the arrow keys. The cursor is non-destructive of both graphics and text. Simply move the cursor to the desired position and start typing text.

The entire screen can be reversed (graphic on/off) via the REVERSE command. Text will not be reversed. The XFLIP command will create a mirror image of the screen about the Y-axis. The graphics will be a true mirror image and the order of text characters will be reversed. The YFLIP is similar to the XFLIP except rotation is about the X-axis.

ZGRAPH uses all available memory (to HIGH$) as in-memory screen buffers in addition to the video display screen. All but one of these buffers are general purpose buffers and are available to the user to store displays. This is useful when creating a large graphic consisting of several **ZGRAPH** images or in creating those images using the MERGE function. **ZGRAPH** can also load and save images to disk files. All data moving to and from the disk passes through the primary video display. The reserved internal display buffer is used for error recovery in case you make a mistake (perish the thought!).

GET is the function for loading the video display screen from a disk file or one of the buffers. Any one portion of the screen can be saved to a buffer or file by using the SAVE command. MERGE allows you to superimpose one graphic image over another. If you want to exchange the screen display with a buffer, use the XCHANGE command.

The INPUT and OUTPUT functions reference all of the in-memory buffers for use as a multiple-image-file. Such a file is used with the BINPLAY program to provide slide-show type operations.

ZGRAPH provides functions to make your graphics generation easier. The DUPLICATE command replicates a block defined by markers to other areas of the screen. LINE will establish the best fitting line between the marker SET and the current cursor position. The marker position will be updated to the current cursor position after each line is drawn providing an easy way to construct lines connected end-to-end. The RECTANGLE command creates a rectangle with opposing diagonals being the SET marker and current cursor position. The FILL function completely whitens or darkens a display region enclosed by a boundary. This is similar to the "paint" command in other graphics systems. The CIRCLE function "rounds out" the **ZGRAPH** graphics functions by drawing a circle or an arc around the current cursor position.

You can also enlargen a particular rectangular area of the screen via the MAGNIFY function. The REDUCTION function can be used to shrink a rectangular area. Finally, the ZERO function whitens or darkens a marked block.

While in the WINDOW mode, the entire screen display will move in response to the arrow keys. Any part of the image moved off of the edges of the screen is erased. This command is very useful to reposition an entire image on the screen.

To allow **ZGRAPH** created displays to be used in other applications, the BINCONV post-processing program is provided. **ZGRAPH's** standard file format is a pure binary representation of the screen display. Each line of the screen memory is saved as the values of the memory bytes terminated by a carriage return. BINCONV converts its standard file formats to:

<1> - ZGRAPH to Load Module in order to create an executable /CMD file that will place your image on the screen;

<2> - ZGRAPH to Packed BASIC - creates a file of packed graphics strings with each line consisting of the string {ZG$(#)="packed value of one line of your image"} starting with an index (#) of 0, line number of 100 and line number increment of 10;

<3> - ZGRAPH to BASIC Data which creates BASIC data statements of 16 decimal numbers representing the sequential values of your screen image;

<4> - ZGRAPH to EDAS creates a file in assembler source format of DEFB statements with 16 decimal values per statement representing the values of the bytes of your image. This file may then be merged into an EDAS assembler program.

The **ZGRAPH** graphics package also includes a keyboard filter, DOSAVE, that is similar to the LDOS screen print function. However, where the screen print directs an image of the screen to the printer, DOSAVE will direct the screen image to a disk file specified by the user at the time you depress <CLEAR><SHIFT><S>. These screen files may be loaded into **ZGRAPH** for further operations. Also included is the BINPRINT program which provides the capability of printing a binary graphic file to a printer that supports compatible bit graphics (MX-80/Graftrax).

The EPBINCAT/CMD and RSBINCAT/CMD programs included with the **ZGRAPH** package provide for printing the graphic binary files to Epson or Radio Shack DMP printers respectively. Each program allows for the concatenation of more than one file to generate a large "picture" composed of many smaller ones. The xxBINCAT programs also provide for magnifying the printed image.

You get ZGRAPH, BINCONV, DOSAVE, EPBINCAT, RSBINCAT, BINPLAY, and BINPRINT in this graphics editor package. In no time at all, you will be creating complex graphics images.

ORDERING INFORMATION:

ZGRAPH: For the TRS-80 Models I/III/4 with LDOS 5.1.
PRO-ZGRAPH: For use with LDOS/TRSDOS Version 6 [i.e. Model 4].

ZSHELL - by Karl A. Hessinger and Roy Soltoff

A feature of the UNIX operating system that has made it famous is the SHELL, a command language interpreter. The "SHELL" generally provides for the parsing of arguments, the ability of redirecting standard I/O, and pipelines. LDOS already supplies a standard procedure for argument parsing (i.e. @FSPEC, @FEXT, and @PARAM). Now Karl Hessinger has written ZSHELL to supply standard I/O redirection, pipelines, and more.

What is standard I/O redirection? Simply this! Your standard input is the *KI device. Your standard output is the *DO device. LDOS already permits you to redirect all devices by using the ROUTE library command. Under the UNIX "SHELL" concept, standard I/O redirection permits you to temporarily route either standard input, standard output, or both - for the duration of the execution of a command. The original device linkage is restored AUTOMATICALLY. Programs written to use standard input and output can easily use any device by this temporary redirection - even files!

Many commercial programs are written that use the single-key inpput routine, @KEY or @KBD, to fetch keyboard input. You would just love to be able to run these programs from the sophisticated Job Control Language of LDOS. However, since JCL operates only from the line input routine, @KEYIN, you cannot "automate" your operation. With ZSHELL, ANY PROGRAM THAT USES THE LDOS KEYBOARD DRIVER CAN BE OPERTATED SO THAT ITS INPUT IS FETCHED FROM A DISK FILE. Furthermore, control is automatically passed back to the keyboard when the program's input gathering reaches the end of the "key" file. This redirection function will pay for the cost of ZSHELL many times by saving you work and time.

Piping, under UNIX, operates in a multi-processing environment. One executing program communicates its output to the input of another executing program. The connection is visualized as a "pipe". LDOS does not provide multi-processing; however, ZSHELL realizes the pipe as a temporary holding file that receives the output of the first program and uses this file as input to the second.

With **ZSHELL**, you can enter multiple commands on a single line. There is also a provision to designate the *PR device as "standard output" for redirection purposes. With ZSHELL, you will be entering commands such as:

```
LBASIC <TEMP/BAS
DIR :1 (A,I,S) | LSCRIPT
DEVICE (B=N) ; FREE
```

The first example has LBASIC getting its input from the file, TEXT/BAS, reverting to the standard *KI linkage upon reaching the end of the file. In the second example, the directory display is automatically loaded into LSCRIPT, while the third illustration is an example of multiple commands on a line.

For our LC users, ZSHELL provides a command line over-ride character which if entered, will inhibit ZSHELL from scanning for redirection. Thus, redirection inherent in an LC generated program can be utilized even when ZSHELL is installed and active.

ZSHELL is 100% compatible with Job Control Language. JCL job streams can be constructed with commands which use the power of ZSHELL's redirection capabilities.

ZSHELL is self relocating and requires less than 1500 bytes of upper memory. It functions only with LDOS 5.1.x versions. It provides I/O redirection of *KI [via "<"], *DO [via ">"], and optionally, *PR [via ">"] in

lieu of *DO. Standard output can be directed to overwrite or append to [via ">>"] a disk file. A program may be piped [via "|"] to another with the pipe file located on the drive of your choice. Multiple commands may be entered on a single line [via ";"]. Integrate these UNIX-inspired features into your LDOS and expand your system's capabilities.

Included with the **ZSHELL** package is **WC**, a wild-card "shell" processor, written by Roy Soltoff, that allows you to invoke compatible commands on all file specifications that match a wildcardspec entered on the command line. You enter the command once while the WILDCARD shell processor searches the designated disk drive(s) for files that match your wildcard specification. WILDCARD builds a Job Control Language file of your command line substituting each matching file specification for the wildcard specification on a separate command line. WILDCARD 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, WILDCARD 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.

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**

ORDERING INFORMATION:

ZSHELL: For use on the TRS-80 Model I/III/4 under LDOS 5.1.

CMD-FILE/PRO-CESS - by Roy Soltoff

The PRO-CESS/CMD-FILE tool is a powerful machine language program that has been designed to provide total maintenance of program load modules on a record basis. This means that it references the load module as a multi-record type, variable length record file - just like the operating system loader. By using the various commands identified in the menu, you can completely reorganize the load structure of a given module or modules in order to make them more efficient in terms of loading speed and occupied disk space.

This maintenance tool provides for file appending, mapping, sorting, packing, offsetting, library member and partitioned data set member extraction, as well as the specified deletion of any load module record. It can convert "CMD" files which contain various types of records to "CIM" files which are pure binary core-image constructs. It also provides the capability of converting "CIM" files to "CMD" files. It gives capabilities to load module maintenance never before possible.

By far, the most powerful function included in this tool is the reorganization capability of the PACK command. This versatile function converts any X-type patches to D-type patches [X-type patches are generated by the LDOS Version 5 or LDOS/TRSDOS Version 6 PATCH utility]. It then sorts the buffer by load address to construct sequential load records and generates a load module file that uses maximum sized (256-byte) load records. This feature is quite useful for reorganizing large inefficiently generated load modules such as Tandy's COBOL package. The PACK function is also useful for reorganizing the out-of-sequence load modules generated by the LC compiler.

All of the functions are immediately available through single letter commands. These commands are displayed in the MENU which looks like:

```
PRO-CESS 2.0 [Copyright (C) 1983 Roy Soltoff]

   <C>lear the buffer region
   <D>OS Command request
   <E>xit to DOS
   <I>mage file load/write
  X<L>oad a file into the buffer
   <M>ap the buffer records
   <O>ffset address from current load origin
   <P>ack the buffer records
   <R>emove a record from the buffer
   <S>ort the load records by address
   <U>n-remove a "removed" record
   <W>rite the buffer to a disk file

Buffer: Size 46802 Used 00002 Free 46800 Records 00000
 Module: Origin FFFF End 0000 Entry 0000 Offset 0000

 ..............................................
```

Each of the commands displayed in the MENU may be selected by depressing whatever key is contained within the angle brackets. A blinking "cursor" can be moved via the <UP-ARROW> or <DOWN-ARROW> keys. Alternatively, any command that is preceded by the graphics block can be selected just by depressing the <ENTER> key in lieu of the command letter.

The <C>lear command is simple enough - it restores the buffer as if you just executed the program. The <D>OS Command function provides access to

operating system commands from the MENU level. Your DOS requests should be limited to library commands [Model I/III CMD-FILE users can invoke any command]. The <E>xit command provides the means to terminate the maintenance session and return to DOS.

The <I>mage command provides two functions - both relating to core-image files. Use the <I>mage command to load a core-image file from disk into the buffer. Since core-image files have no loading information contained in the file, it is necessary to specify the origin address of the module. Use the <I>mage command to write the buffer out as a core-image file. The <IW> function will first scan the load records to ensure that they are in sequential load order, for a core-image file cannot be constructed if the records are out of order. It is not necessary for the load records to be contiguous. The <IW> function will generate null bytes, X'00', for all addresses interstitial to two adjacent non-contiguous load records.

The <L>oad function is used to read a load module file into the memory buffer. The file will be appended to any already contained in the buffer. If the file is recognized as an LDOS structured ISAM overlay file, you will be able to extract a single member by identifying the overlay number of the desired member. If the file is recognized as a PaDS file, you will be able to extract a single member by identifying the member name. The buffer and module status lines will be updated to reflect the revisions made to the buffer with the load of the file.

The <M>ap command provides the function of mapping the buffer contents by record. Records will be identified as to type: Module header, Yanked load block, Load, Transfer Address, and so forth. The address range of load records will also be displayed.

The <O>ffset command changes a load module so that it loads into memory at a location different from where it was assembled to execute.

The <P>ack operation is used to reorganize an object load module file so that it is most efficient in disk storage space and optimum for rapid loading by the operating system. Packing is a three-phase operation. The first phase identifies any LDOS X-type patch records and packs the object code revisions into the preceding load records wherever possible. The second phase then uses the <S>ort facility to sequence the load records by sequential load address. The third and final phase generates a new object load module file with maximum-sized load records. This is achieved by combining short contiguous load records wherever possible.

The <R>emove command can be used to delete an entire record from the load module. The <S>ort command reorganizes the buffer's load records so that they are in sequential load order. The <U>n-remove command is used to recover from inadvertantly removing the wrong record with the <R>emove command. Finally, the <W>rite command is used to write the buffer contents to a disk file. This function also provides the opportunity of changing the module's ENTRY address as noted in the MODULE's status display.

ORDERING INFORMATION:

CMD-FILE: For TRS-80 Model I under LDOS 5.1, TRSDOS 2.3 and
TRS-80 Model III under LDOS 5.1 and TRSDOS 1.3.
PRO-CESS: For LDOS/TRSDOS Version 6 [i.e. Model 4, etc.].

ZCAT/PRO-ZCAT by Karl A. Hessinger

We know you have built up a collection of hundreds of diskettes and need an easy way to locate that particular program and file. We know you want fast response and speed! So rather than waste time trying to determine what DOS each of your disks is compatible with, we recognize that the LDOS user stays with LDOS. ZCAT is a fast machine language program that creates and maintains a catalog of all files which reside on your LDOS-formatted diskettes.

This package is menu driven for ease of use. The initial "GETSPEC" menu, allows you to specify which drive contains your catalog files [a catalog file uses an extension of "/CAT" and is the data base for your disk directories]. Once you specify the drive number, ZCAT displays all files on that drive with a "/CAT" extension. You then enter the name of the catalog file you wish to read or create. Once the specified catalog file has been read or created, the master menu will be displayed.

```
*> M A S T E R   M E N U <*

    <A>dd disk to list
    <U>pdate disk in list
    <C>hange a diskname
    <R>emove disk from list
    <S>earch for a file
    <D>isplay directory
    <L>ist disks on file
    <P>rint files in list
    <E>xit to GETSPEC

        Selection ?


DIR file : MARC/CAT:0          Files cataloged :   324
Disks cataloged :    15        Maximum # files :  2226
```

The desired function may be selected by pressing the letter of the function which is bracketed between the '<>' symbols.

The <A>dd function is used to scan a new disk. ZCAT adds the disk name to the catalog list then adds all of the non-system, visible files to the list [invisible and/or system files may be included via command line options]. After the disk has been cataloged, the directory list is sorted. You can then insert another disk to catalog with one keystroke!

The <U>pdate function scans the directory of an ALREADY cataloged disk and updates the catalog file to reflect any changes in the free space on the disk or any changes in the contents of the disk.

The <C>hange function allows you to alter a disk's name - just like from DOS using the ATTRIB command.

The <R>emove function deletes all traces of a disk from the catalog list. ZCAT will display the names of all disks which are currently in the directory list to refresh your mind so that you can select the correct name.

The <S>earch function will allow you to rapidly locate a file or a group of files in the catalog list by means of your entered search string. The search string may be a filename, a partial filename, or an extension. The search string can also contain a wild card character ('$') which may be used to mark a position as "don't care".

The <D>isplay function lists all files which reside on a particular disk. ZCAT displays the diskname and available free space then displays an alphabetical list of all files on the disk. The information listed for each file includes: filename, protection level, logical record length, number of records, size in K, and modification date. A sample follows:

```
Diskname : MARC0037                                    Free Space :     0K
   Filespec    Prot     LRL    #Recs    Size    Mod  Date     Diskname
================================================================================
ATOD/ASM       Full     256        2      1K     27-Sep-82     MARC0037
CASSCO/ASM     Full     256       34      9K     18-Jun-82     MARC0037
CASSCO/CMD     Full     256        4      1K     18-Jun-82     MARC0037
CC2/CCC        Full     256       46     12K     22-Sep-82     MARC0037
CC3/CCC        Full     256       27      7K     22-Sep-82     MARC0037
CC4/CCC        Full     256       32      8K     21-Sep-82     MARC0037
CC6/CCC        Full     256       18      5K     31-Aug-82     MARC0037
CHGDATE/BAS    Full     256        4      1K     20-Oct-81     MARC0037
DABS/ASM       Full     256        2      1K     27-Sep-82     MARC0037
DADD/ASM       Full     256        3      1K     27-Sep-82     MARC0037
[ listing continues]
```

The <L>ist function displays all the disks which are currently in the catalog list. The catalog filename followed by a list of all the disknames on file with the free space available on each disk is included in the display. The following illustrates such a listing:

```
                These disks are in directory file : MARC/CAT:0

  Disk     Free     Disk     Free     Disk     Free     Disk     Free
========  ====  ========  ====  ========  ====  ========  ====
MARC0025    3K  MARC0026    3K  MARC0027    0K  MARC0028    0K
MARC0029    0K  MARC0030   33K  MARC0031    2K  MARC0032   15K
MARC0033    3K  MARC0034    9K  MARC0035   11K  MARC0036   32K
MARC0037    0K  MARC0038   84K  MARC0039   54K
```

The <P>rint function will allow you to produce a printed hardcopy of your directory list. You can print "<F>iles by disk order", "<E>xpanded file order", or "<C>ompressed file order".

The <E>xit function returns you to the GETSPEC menu to give you the opportunity to save or cancel all of your changes. At the GETSPEC menu, you can also change to a different catalog file.

This package is all you need to get a handle on your disk collection. Don't waste your time trying to DIR your disks to find a particular file. Catalog your collection, fast - with **ZCAT.**

ORDERING INFORMATION:

ZCAT: For use with the TRS-80 Model I/III under LDOS 5.1.
PRO-ZCAT: For use with LDOS/TRSDOS Version 6.

MLIB - by Richard N. Deglin

What user of Microsoft's MACRO-80 assembler has not reeled in frustration while trying to deal with its relocatable libraries? The frustration can now vanish with your use of MLIB - a module librarian. MLIB is a software tool which will aid you in construction of Microsoft-compatible relocatable object file libraries. These libraries are supported by the Microsoft language products MACRO-80, BASIC-80, FORTRAN-80, and LINK-80. MACRO-80 and FORTRAN-80 were formerly sold by Radio Shack as the Model I Disk Editor/Assembler and FORTRAN packages, while the BASIC-80 compiler is often known as BASCOM. FORLIB/REL, which is an integral part of the FORTRAN-80 package, is an example of such a library.

MLIB is menu driven and easily controlled by single-letter commands. A command menu similar to the following is your master control panel:

```
         _____   ________________________________________________
      ___|__ |TM                                                 |
     |   |  | |  MLIB 3.1L - Relocatable Object File Librarian   |
     | Ri|cl|in| (C) Copyright 1983 Riclin Computer Products     |
     |   |__|__|                                                 |
     |_____|                                                     |
           X Load library              Save Library              |
     |       Add module                Module map                |
     |       Purge module              Disk Directory            |
     |       Replace module            Kill disk file            |
     |       Extract module            Clear buffer              |
     |       Insert before module      eXit program              |
     |                                                           |
     |             Bytes used:  0, free: 29736                   |
     |___________________________________________________________|
```

MLIB is menu-driven for your convenience. To execute a command, enter the first letter of the command. Alternatively, you may find it simpler to move the blinking cursor until it is next to the desired command, and then depress <ENTER> to execute that function. This is accomplished by using either the <DOWN-ARROW> and <UP-ARROW> keys. The blinking cursor will reappear as a solid block next to the selected command, as a visual indication of which command is executing.

If an error occurs, MLIB will "beep" the console speaker if your machine is so equipped (on a Model I/III computer, a short beep tone will be directed to the cassette port which can be audible if you have an audio amplifier and speaker hooked up). An error message will appear at the bottom of the screen, and will remain there until you enter any keystroke, indicating that you have acknowledged the error.

The <A>dd command adds a new module to the library in memory from a disk file. This is the primary method of building a new library or adding to an existing one. The <C>lear command resets all internal pointers, thus effectively destroying what is currently in the memory buffer. The <D>irectory command displays an unsorted directory of visible files for the selected drive. The <E>xtract command takes a single module from the library in memory and writes it to disk as a stand-alone /REL file. The <I>nsert command places a new module from disk into the library in memory. The <K>ill command deletes a file from disk. The <L>oad command reads an existing library into memory from a disk file.

The <M>ap command displays the attributes of the library which is currently in memory. This listing can be in either detailed or summary format. The summary format option lists all modules, by name only, in the order they exist in memory. The summary would look like this:

```
RAN      INT4     DSQRT    DMOD     DSIGN    DABS     DATAN2
DATAN    DLOG10   DCOS     DSIN     DBLEXP   DEXP     DLOG
DMIN1    DMAX1    DCOMP    DBLDIV   DBLFLR   DBLPLY   DBLUTL
DMLDV    DSHR     DBLCON   MFM      AMINO    MINO     AMAXO
TANH     SQRT     MIN1     MAX1     MAXO     IFIX     FLOAT
EXP      DIM      COSIN    ATAN2    ATAN     AMOD     AMIN1
AMAX1    AINT     ABS      ALOG10   ALOG     MOD      RIEXP
NEGF     RREXP    NEG      NEG      EXPB     LOG2     POLY
FADD     IIEXP    FDIV     STOP     FMUL     ADE      FLR
FCP      FLT      CMPGTO   IDIV     SIGN     RAT      DAT
ISIGN    IABS     NEGATE   XAR      IDIM     FAT2     IMUL
POKE     UNPACK   PSPLAC   ICP      FORMIO   NORM     SHIFT
RNDOVF   ZAC      IOINIT   LUNTB    IAT2     LODSTR   SAF

      Hit <ENTER> for next screen, Hit <BREAK> to exit
```

The detailed format option displays the attributes of each module as shown in the following illustration:

```
Module name    Module size    Program size    Data area size
  DSKDRV          1417            996              76

Entries:   $FLBUF   $FLCNT   $FLFCB   $FLFLG   $GTFCB   $GTFLG
           0000"    0014"    0032"    0028"    0024"    0018"
           $MEMRY   DSKDRV   OPEN
           0048"    000A'    033C'

Externals: $BF      $BL      $CLSFL   $ERR     $IOERR   $IOINI
           $LUNTB   $REC     $UN

Commons:

      Hit <ENTER> for next screen, Hit <BREAK> to exit
```

The <P>urge command removes a module from the library in memory. If the module has multiple entry points, all will be deleted. The <R>eplace command replaces an existing module in memory with a new version from a disk file. All aliases will be replaced or deleted. The new module may be longer, shorter, or the same lengtn as the old one - MLIB will compress, overlay in place, or expand the library in memory to fit the new length. This command will abort on module not found, out of memory, symbol table overflow, or disk error. The <S>ave command writes the entire library in memory to a disk file. The e<X>it command returns you to "DOS Ready". If there is data in the buffer that has not yet been saved to disk, you have a chance to abort the exit and return to the main menu.

ORDERING INFORMATION:

MLIB: For use on the TRS-80 Model I under LDOS 5.1 or TRSDOS 2.3 and, the TRS-80 Model III under LDOS 5.1 or TRSDOS 1.3.
PRO-MLIB: For use under LDOS/TRSDOS 6.x.

MACH2/PRO-MACH2 by Karl A. Hessinger

The LDOS file system allocates space for your files in one of two ways. It either assigns space randomly [as used in LDOS 5.0.x, 5.1.0, 5.1.1, 5.1.2, 5.1.3, 6.0] or assigns space automatically starting from cylinder one [5.1.4, 6.1]. Since the access of disk files requires a disk drive to step from track to track [which is one of the slowest operations of a disk drive], the most efficient placement of files is one that minimizes this stepping operation. Either method may prove inefficient in particular cases since the optimimum arrangement is dependent on the specific function and access sequence of all files on the disk. Once and for all, MISOSYS now puts you in the driver seat when it comes to DOS allocation of disk space. Whether you are a commercial programmer wanting to construct "optimized" master diskettes for distribution purposes, whether you are a sophisticated hacker wanting to squeeze every bit of performance out of your system, or whether you just want to create most contiguous files where you want them, **MACH2** is for you.

MACH2 is a collection of four utilities that were designed to work together with such ease, that you will be amazed at how easy direct control of space allocation can be. The **MACH2** utilities are friendly. The **MACH2** utilities are flexible. Finally, the **MACH2** utilities are powerful. You are not limited to floppies, **MACH2** works just as well with hard drive systems.

First, the **MAPPER** provides a diskette map by granule by file. It produces a screen or printer listing such as the following [some lines have been deleted to abbreviate the listing]:

```
Diskname : TESTDISK                          Free Space :   108K
Sides : 1              Density : Double        Cylinders :  40
-------------------------------------------------------------
  0  BOOT/SYS            M80/CMD              M80/CMD
  1  M80/CMD             M80/CMD              M80/CMD
  2  M80/CMD             M80/CMD              TEST/CMD
  3  TEST/CMD            TEST/CMD             TEST/CMD
 19  ** Empty **         ** Empty **          ** Empty **
 20  DIR/SYS             DIR/SYS              DIR/SYS
 24  ** Empty **         ** Empty **          ** Empty **
 25  ** Locked **        ** Locked **         ** Locked **
 26  ** Empty **         ** Empty **          ** Empty **
 27  M80PACK/CMD         M80PACK/CMD          M80PACK/CMD
 28  M80PACK/CMD         M80PACK/CMD          M80PACK/CMD
 37  ** Empty **         ** Empty **          ** Empty **
 38  M80/CMD             M80/CMD              M80/CMD
```

You can use the **MAPPER** just to get a look at what files are where. This is great for other uses [such as reconstruction of damaged cylinders]. Use the **MAPPER** with the BLANK option on a freshly formatted [or otherwise disk with available free space] and all of the "** Empty **" denotations are blanked out and bracketed giving you a worksheet for preparing your optimum arrangement of files.

Second, the **CALC** utility can process a disk's files to let you know exactly how many granules each file would take up on every media format supported by LDOS. This tool is great for taking a disk full of programs that are on one media type and ascertaining the granule requirements for a different media type. As easy as **CALC :1 (P)**, you get the following printout:

Filename	5" single	5" double	8" single	8" double	4 SPG	16 SPG	32 SPG
BOOT/SYS	1	1	1	1	2	1	1
DIR/SYS	4	3	3	2	5	2	1
M80/CMD	15	13	10	8	19	5	3
M80PACK/CMD	15	12	9	8	18	5	3
TEST/CIM	5	4	3	3	6	2	1
TEST/CMD	15	12	9	8	18	5	3
TEST/DAT	0	0	0	0	0	0	0

Pick your media and use your MAPPER-generated worksheet to develop your customized layout of files - or plunge right in to ALLOC since it gives you a dynamic free-space map on-line!

The allocation tool, **ALLOC,** lets you tell the DOS where you want a file placed. ALLOC's screen display conveniently presents the controls. All you need do is specify the file specification, and for each directory extent, the starting cylinder, starting granule, and number of granules. Once you identify the file spec, **ALLOC** gives you a three-line scrolling free-space map so that you can see what granules have already been allocated on the disk in question. All you need do is allocate the space for your files then copy them to the "optimum" disk. Look at the useful **ALLOC** screen display:

```
                ALLOC - LDOS File Space Allocator

 6- 11   xx.      ...      ...      ...      xxx      x..
12- 17   ...      ...      ...      ...      ...      ...
18- 23   ...      ...      xxx      ...      ...      ...

                    Allocating file : TEST/DAT:3

                 Starting          Starting          Granule
Extent           Cylinder          Granule           Count
======           ========          =======           =====
 1               12                0                 ..
```

Want to just get a large block of contiguous space for that data base? **HANDY** is just handy for those non-critical allocation jobs. **HANDY** will easily allocate the most contiguous extent of space in up to four extents - the number of extents controlled by you!

We know that you have been asking for better control over space allocation for a long time. We know that you have needed **MACH2** for a long time. All that LSI could come up with was more stringent DOS control over file placement. Now you can take control! MISOSYS delivers with **MACH2!**

ORDERING INFORMATION:

MACH2: For TRS-80 Model I/III under LDOS 5.1.
PRO-MACH2: For use with LDOS/TRSDOS 6.x [i.e. TRS-80 Model 4].

VRHARD Hard Disk Driver by Roy Soltoff

This package provides driver support for the HARD DISK III type of hard drive manufactured by the V R Data Corporation and using the hard disk controller manufactured by the XEBEC corporation. The VRHARD package provides for the operation with LDOS 5.x or LDOS/TRSDOS 6.x of one or two IDENTICAL hard drives [i.e. two 5 meg, two 10 meg, etc.].

The driver supports partitioning of the hard drive media by both number of heads and number of cylinders. Up to eight heads and up to 999 cylinders are supported. The driver initializer is very friendly and virtually failsafe in inhibiting you from making conflicting partitioning requests during the partitioning operation.

The driver supports both the fixed media and removable cartridge media type drives. Your configuration of two drives may be mixed as to type.

The package includes MOVEFILE, a large file segmentation backup utility. This machine-language program gives you the capability of making backups up the large file by segmenting it into floppy-sized sub-files. MOVEFILE also provides for the restoral of the large file from the sub-files.

ORDERING INFORMATION:

VRHARD: For use with TRS-80 Model I/III under LDOS 5.1 and
for use with LDOS/TRSDOS 6.x [i.e. Model 4, etc.].

Note: MISOSYS can custom fit our XEBEC controller driver to your non-VR hard drive. Contact MISOSYS for consultation.

M I S O S Y S   O R D E R   F O R M

=*=

Company Name: ______________________________ Date: __________

Individual: ______________________________ P.O.#: __________

Address: ____________________ Apt#: _____ Phone: ___+___-_____

City: ____________________ State: ________ ZIP: __________

=*=

Payment: Check/MO { } MC/VISA { } Signature _______________ COD { }

Credit Card #: ______________________________ Expires: _____

=*=

Qty	Description	Net Each	Total
	Compiler/Macro-assembler - LC { } PRO-LC { }		
	Macro-assembler - EDAS-IV { } PRO-CREATE { }		
	Disk disassembler - DSMBLR { } PRO-DUCE { }		
	L/M reorganizer - CMD-FILE { } PRO-CESS { }		
	Partitioned Data Set - PaDS { } PRO-PaDS { }		
	REL Librarian - MLIB { } PRO-MLIB { }		

Send to: MISOSYS   [703+960-2998]   Sub-Total | |
P.O. Box 4848
Alexandria, VA 22303-0848   VA Residents Sales Tax | |

Specify computer: Model I { } Model III { }   Shipping Charge | |
MAX-80 { } Model II { } Model 4 { }
Other ____________________ { }   Total | |

M I S O S Y S P R I C E L I S T

Effective November 15, 1983

CMD-FILE/PRO-CESS Version 2	$40	
CON80Z/PRO-CON80Z - Translate 8080 to Z-80	$40	
CONVCPM/PRO-CURE - Transfer CP/M files to LDOS	$50	&&
DSMBLR-III/PRO-DUCE - Disk disassembler	$40	
EDAS/PRO-CREATE - Version IV Macro assembler	$100	+
GRASP - Graphics Support Package	$50	
HELP/PRO-HELP - Help for LDOS 5.1/TRSDOS 6.x	$25	
LC/PRO-LC - Compiler/Macro assembler	$150	*
MACH2/PRO-MACH2 - File allocation package	$40	##
MLIB/PRO-MLIB - /REL Librarian	$50	
MSP-01/PRO-GENY - DOAUTO/SWAP, DOCONFIG, MEMDIR, PARMDIR	$40	
PaDS/PRO-PaDS - Partitioned Data Set utility	$40	
SOLE - DDEN booting LDOS Model I	$25	
THE BOOK, ACCESSING THE TRS-80 ROM, VOL II	$5	
THE C PROGRAMMING LANGUAGE by Kernighan & Ritchie	$18	
THE PROGRAMMER'S GUIDE TO LDOS/TRSDOS VERSION 6	$20	
VRHARD - Hard disk driver package	$75	
ZCAT/PRO-ZCAT - LDOS Catalog utility	$30	##
ZGRAPH/PRO-ZGRAPH Version 5 - Graphic Screen Editor	$50	
ZSHELL - Command processor for LDOS 5.1.x	$40	

Shipping: Items marked "*", add $5. Items marked "+", add $4
All others, add $2.00 plus $0.50 each additional item.
COD add $1.50. Shipments > 12 oz or > $40 are via UPS.
VA residents, please add 4% Sales Tax.
Outside US, Canada, and Mexico multiply shipping by 4
for AO Air or Air Parcel Post.

Available 11/30/83

&& Available 12/15/83

MISOSYS
P.O. Box 4848
Alexandria, VA 223Ø3-Ø848

15% OFF UNTIL 12.31.83

Contents: Printed Matter