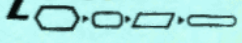
LDOS

VERSION 5.1

THE TRS-80™ OPERATING SYSTEM

MODEL I AND III

LOGICAL SYSTEMS INC.



T A B L E O F C O N T E N T S

TABLE OF CONTENTS

Second Edition LDOS 5.1.x Combined Model I/III

Copyright 1982 by LOGICAL SYSTEMS, INCORPORATED
All rights reserved
LDOS is a Trademark of LOGICAL SYSTEMS, INCORPORATED
TRSDOS and TRS-8Ø are Trademarks of Tandy Corporation

ACKNOWLEDGEMENTS

The LDOS product is the property of LOGICAL SYSTEMS INCORPORATED and is copyrighted in its entirety. No portion of the system code or documentation may be reproduced in any manner, for any purpose. Copies of the serialized master LDOS disk may be made by the registered owner of that disk for archival purposes only. Logical Systems, Incorporated, does not authorize any LDOS owner or any other person to duplicate or distribute LDOS or any portion of LDOS for purposes other than the creation of reserve (archival) copies for the original purchaser's personal use.

The following persons were members of the team that made LDOS possible:

Bill Schroeder
PROJECT LEADER

Roy Soltoff
SYSTEMS ANALYST

Chuck Jensen

Doug Kennedy

Dick Konop

Tim Mann

The LDOS development team would like to thank the many people that provided suggestions and support during the creation of the LDOS products.

INTRODUCTION TO LDOS

These first few pages will introduce you to the basic principles of a disk operating system, and list some of the LDOS commands and utilities you will be using in your day to day activities. You can use this as a convenient reference until you become familiar with operating your computer under the LDOS system.

What is a Disk Operating System (DOS)?

Without a DOS, your computer is controlled by routines stored in ROM (read only memory). The ROM handles all I/O (input/output) to your system devices - the keyboard, cassette, video, printer, etc. It does this from within a BASIC language environment. A DOS takes over control of the I/O processing, and adds its own routines to handle the disk drives. Instead of starting out in BASIC, you will be in a new environment known as the "DOS Level". Now, any commands you give are interpreted by the DOS, rather than by the BASIC stored in the ROM.

What does this mean to you, the user? First of all, every DOS, including LDOS, has a given set of commands, different than the ones available from the BASIC level. Commands may be entered when the "LDOS Ready" prompt is on the screen. The commands are not programs that you would use to perform "application" type functions such as accounting or word processing. Rather, they are the means to load and execute applications programs, and then maintain those programs and the data created by those programs. LDOS commands also let you modify the way certain devices work, by providing things such as type ahead for the keyboard, a user-definable blinking cursor, and an output format program for a line printer.

As you can see from the table of contents, there are several different groups of program files provided on your LDOS disk. Before explaining their function, you should know how any information, including the LDOS programs, are stored on a disk.

Disk organization

Your disk drives store data in the form of magnetic pulses on the diskette media. In order for a disk operating system to access this data, the diskette obviously must be organized in some manner. The LDOS FORMAT utility program will write all necessary information to a diskette to organize it into cylinders, tracks and sectors so it may be accessed in a structured manner. Only after a diskette is formatted may it be written to or read from by LDOS.

The structure of a diskette surface may be thought of as a series of concentric circles, starting at the outer edge of the diskette media and moving in towards the center. These circles of information are referred to as tracks, starting with track number Ø at the outer edge. The number of tracks available on a diskette will depend on the type of disk drive you have purchased.

The term "sector" refers to a space on a track that can hold 256 characters of data. You will find the term "byte" is commonly used to refer to one character, and thus a sector is referred to as 256 bytes. The number of sectors available per track is dependent on the size (5", 8" or hard) of the diskette, and on the density specified when formatting the diskette. With LDOS, a double density diskette will have approximately 8Ø% more space available compared to a single density diskette with the same number of tracks. Four sectors, or 1Ø24 bytes, will be referred to as "1K" (standing for about one thousand bytes or characters).

The minimum allocation of disk space made when storing a file on a diskette is called a "granule" or "gran". The number of sectors that make up a gran will vary depending on the size and density of the diskette and the type of drive. This basic format structure is used by LDOS for any type of drive it supports, 5", 8" or hard disk. Diskettes formatted by other operating systems may not be directly readable by LDOS.

To keep track of the various file names on the disk, and to record the tracks and sectors where a file is stored, LDOS uses an area of the disk called the DIRECTORY. This directory is always placed on the center track of the diskette during the formatting process.

FILES - How information is stored

The most important thing to remember about disk storage is the term "file". Any information, be it a program or data, is stored as a file. This means that the information is written to the data areas of the disk, and the name of the file and the actual location of the data are stored in the directory. No matter what kind of file you are dealing with, the file name must use the following format:

FILENAME/EXT.PASSWORD:Ø

FILENAME - Up to 8 alphabetic or numeric characters, the first of which must be alphabetic. **All files must have a filename.**

/EXT - An optional field called the file extension. If used, it can contain up to 3 alphabetic or numeric characters, the first of which must be alphabetic. An extension can be of great use to identify and deal with certain types of files, and it is strongly recommended that all files you create be given extensions.

.PASSWORD - This is an optional field that will assign a protection status to a file. If used, it can be up to 8 alphabetic or numeric characters, the first of which must be alphabetic.

:Ø - This is an optional field called the drive specification. It is used to specify the particular drive number the file is on. It can be any number Ø through 7, depending on the number of drives in your system.

These four parts comprise the complete file specification, which will be referred to as a "filespec". All of the LDOS files are listed in the section beginning on page 1-13.

LDOS organization and files

LDOS is organized into different groups of files. The first group is the SYSTEM files, containing the necessary information to control the disk drives and your other devices. There are two special system files known as the LIBRARY. They contain the programs most commonly used to manipulate your files and devices. Another group is known as the UTILITY files. These provide added means of handling your program and data files. The file group containing DRIVER and FILTER files gives added flexibility to your system devices. There is a LANGUAGES section containing JCL and LBASIC. JCL stands for Job Control Language, and is a very powerful chaining language. Also provided is LBASIC, which is an extension of the BASIC language provided in the ROM.

There are two types of disks that can be used with LDOS. The first type is called a SYSTEM DISK. This is a disk that contains the LDOS system files.

A system as complex and flexible as LDOS would occupy considerable memory space to be able to provide all of its features. LDOS, however, makes extensive use of overlays in order to minimize the amount of memory reserved for system use. An overlay is a module that loads into memory, overlaying anything which was loaded there previously. In this manner, many functions can occupy the same area of memory, being loaded and used only when specifically needed. The compromise in using an overlay driven system is that while a user's application is in progress, certain disk file activities requested of the system may require the operating system to load different overlays to satisfy the request. This could cause the system to run slightly slower than a less sophisticated system which has more of its file access routines always resident in memory. The use of overlays also requires that a SYSTEM diskette be available in drive Ø - the system drive.

The second type of disk will be called a DATA DISK. This is a disk that has been formatted, but contains no LDOS system files. This would be the type of disk you would normally use in a drive other than drive Ø.

No matter which type of disk you are using, the formatting process will put two files on the disk; BOOT/SYS and DIR/SYS. These files contain information about the type of disk and the disk directory, and are normally invisible to the user. Under no circumstances should you ever copy these files from one disk to another, or attempt to kill them. Doing so can render the disks involved totally useless! LDOS automatically will take care of updating any information in these two files.

LDOS and Devices

Devices are generally thought of as a physical piece of your computer hardware; the video display, keyboard, printer, etc. The routines that control the I/O to these devices can be the ones provided in the ROM, or can be ones provided by LDOS. In either case, there is a small section of memory set aside as a control block for each device. With LDOS, you will have a certain amount of "device independence". Device independence will allow you to treat each of your devices individually. In fact, certain of the Library commands will let you move data directly from a device to a disk file, or vice versa.

As with files, LDOS uses a definite specification when accessing devices, called a "device specification", or "devspec". A devspec is very easy to understand. It consists of an asterisk followed by two alphabetic characters. For example, your keyboard devspec is *KI (Keyboard Input), the video is *DO (Display Output), and the printer is *PR (PRinter).

There are programs provided that will modify the I/O routines for certain devices. The DRIVER and FILTER section of the manual explains the functioning of these routines. You can read that section and determine if you need the extra features provided by those programs.

Using the LDOS files

Now that you know what file groups are on an LDOS disk, let's discuss some of the more important LDOS commands and how to use them. The following descriptions will be general in nature, more to give you an idea of which commands and utilities do what than to explain them in detail. You should refer to the proper section of the manual for in-depth explanations.

Viewing files

To see the files in a disk directory, you should use the DIR Library command. This will show you any LDOS files on a disk, as well as any program or data files you have created. The Library command LIST will allow you to inspect the contents of any individual file, sending the display to either the video or the printer.

Moving files

Files may be moved individually from one disk to another with the COPY Library command. The BACKUP Utility lets you automatically move any or all files from one disk to another. Of course, the disk to receive the files must have been previously formatted.

Removing unwanted files

Any file may be removed from a disk with the KILL Library command. This will remove the information from the directory, and free up the data storage space previously assigned to that file.

Changing file names

The RENAME Library command will let you change the filename or extension of any file in the directory. The ATTRIB Library command will let you apply or change a file's password. Also, a file specification may be changed during the COPY process.

Viewing devices and disk drive parameters

The DEVICE Library command will let you see what devices you have active in your system. You will also see the information that LDOS has stored in memory about the number of disk drives and the types of disks you are using. Certain disk drive information may be changed with the SYSTEM Library command.

Establishing or Removing devices

The SET and ROUTE Library commands will let you establish devices. The LINK Library command will also let you link multiple devices together. The RESET and KILL Library commands can be used to remove unwanted devices.

M A K I N G A B A C K U P

Now that you have read the introduction, you should follow the next set of instructions. They will tell you how to make a "backup", which will be an exact duplicate of your master LDOS disk.

1) Your LDOS master disk is WRITE PROTECTED with a small adhesive tab. DO NOT REMOVE THE WRITE PROTECT TAB.

2) Power up your computer system and all peripheral hardware. Place the LDOS Master diskette in drive Ø and press the RESET button to boot the LDOS diskette into the system. The LDOS logo will now appear on the screen. Enter in the correct date (mm/dd/yy), and the message LDOS READY will appear.

3) The name of your Master diskette will be displayed in the center of the screen above the LDOS logo. It will probably appear as something like LDOS-513. Write this name down, as you will need it in the following procedure.

4) Now you are ready to make several BACKUPS of your LDOS Master diskette. Follow the step by step procedures listed below.

After a backup is complete, you will see the message "CANNOT CLEAR MOD FLAGS - SOURCE DISK IS WRITE PROTECTED" on the screen. This is just an informative message, and is normal when there is a write protect tab on the source disk. A complete explanation of "Mod flags" can be found in the DIR Library command section.

**** CAUTION: The default drive step rate will be 6ms for the Model III. If this is too fast for your disk drives, use the additional parameter STEP=3 inside the parentheses in the following FORMAT commands.**

FOR SINGLE DRIVE OWNERS:

Type in the command: FORMAT :Ø (NAME,Q=N)

The screen will clear and the LDOS disk FORMAT utility will be loaded. You will see the following prompt appear:

DISKETTE NAME ?

Answer the prompt with the disk name from step 3. You will then see the message:

LOAD DESTINATION DISK AND HIT <ENTER>

At this point, insert a new, blank diskette in drive Ø and press <ENTER>. After the FORMAT is complete, this message will appear:

LOAD SYSTEM DISK AND HIT <ENTER>

Put the LDOS Master disk back in drive Ø and press <ENTER>. Now type in the command:

BACKUP :Ø :Ø

The message INSERT SOURCE DISK (ENTER) will appear on the screen. Since your LDOS disk is the SOURCE disk, simply press <ENTER>. The message INSERT DESTINATION DISK (ENTER) will now appear on the screen.

Put the disk you have just formatted into drive Ø and press <ENTER>. You will be prompted several times to swap the Source and Destination disks until the BACKUP is completed. At that point, the message INSERT SYSTEM DISK (ENTER) will appear. Place the Master in drive Ø and press <ENTER>. The BACKUP is now complete.

FOR MULTIPLE DRIVE OWNERS:

Place a new, blank diskette in drive 1 and type in the command:

FORMAT :1 (NAME,Q=N)

The screen will clear and the LDOS disk FORMAT utility will be loaded. The following prompt will then appear:

DISKETTE NAME ?

Answer the prompt with the disk name from step 3. LDOS will now FORMAT the disk in drive 1. When it is finished, the prompt LDOS READY will appear. To make the BACKUP, type in the command:

BACKUP :Ø :1

LDOS will now make a BACKUP copy of itself on drive 1.

MODEL I - Backing up the LDOSXTRA disk.

To make a copy of the LDOSXTRA disk, you must again format a disk. Use the same format instructions as for the Master disk, except answer the "DISKETTE NAME ?" prompt with LDOSXTRA. To make the backup on a single drive system, type in the command BACKUP :Ø :Ø. You will now be prompted to swap the source disk (the LDOSXTRA) and the destination disk (the one just formatted) until the backup is completed. On a multiple drive system, give the command BACKUP :Ø :1 (X), and you will be prompted to insert the source disk (the LDOSXTRA) in drive Ø. Do so, being sure that the newly formatted disk is in drive 1, and then press <ENTER>. The backup will now begin. When prompted to insert a system disk, place the LDOS system disk back in drive Ø and press <ENTER>.

After the Backups are completed:

After the initial backups of your LDOS disk are completed, remove the LDOS master diskette from drive Ø and put it in a safe place. Be sure to leave it in its original jacket to protect it from dust and other contamination.

Label the backup copies of the diskettes as original backups of the LDOS master diskette. You should use these diskettes to make any other backups you require. Do not use the master diskettes except to create a backup as just described.

It is extremely important that you now completely read the next section of the LDOS user's manual. This section contains an overall view of the operating system as well as explanations of certain terms and conventions.

U S I N G T H I S M A N U A L

The LDOS User's manual is set up to be easily used. It is divided into several different sections, each containing information about a specific group of commands. These sections can be identified by section identifier title blocks printed directly above the page numbers.

SECTION...1> is made up of general information about the LDOS system. It contains the introduction to LDOS, as well as descriptions of the commands and files available.

SECTION...2> contains the LDOS LIBRARY COMMANDS. These commands are the heart of the operating system, and provide the link between the user and the computer. They will be listed in alphabetical order

SECTION...3> contains information on the LDOS UTILITY programs.

SECTION...4> contains device DRIVER and FILTER programs for some of the devices available under LDOS. Programs include provisions for such features as keyboard type ahead and formatted line printer output.

SECTION...5> contains detailed operating instructions and information on the LDOS Job Control Language (JCL) and information on the enhanced LBASIC, a Microsoft compatible Disk BASIC language.

SECTION...6> contains detailed TECHNICAL information about the LDOS operating system, including important addresses and system routines.

SECTION...7> contains the GLOSSARY, WARRANTY, and INDEX information.

To locate the section of the manual you wish to access, refer to the tab insert sheets, Table of Contents or the Index. All commands or programs in each section will be in alphabetical order. Any time you encounter an unfamiliar word or definition, refer to either the Glossary or the information in Section 1 of this manual.

PAGE NUMBERING

The pages in the LDOS manual will be numbered consecutively within sections. In addition, each section or command will have it's name printed directly above the page number. For example, the LIBRARY COMMAND section page numbering will show something like this:

APPEND - LIBRARY COMMAND
Page 2 - 1

This would let you know that the page you are on deals with a Library command called APPEND, and it is the first page in section 2, the Library command section.

Special Addenda

Any special addenda will be found at the very end of the manual. These will generally deal with LDOS patches to applications programs. Because support for new products is an ongoing process, these addenda may not all be listed in the Table of Contents or Index.

E N T E R I N G L D O S C O M M A N D S

Looking through this manual, you should notice a very distinct structure regarding the command syntax of the LDOS system.

Each Library command, utility, or program section will begin with a very brief description of the function involved. Immediately following will be a "syntax block". This block will be laid out to show the command syntax, allowable parameters, and abbreviations, if any. A typical block might show the following:

```
================================================================
| THE COMMAND any files or devices (parameters)                |
|  "     "     "    "    "    "          "                    |
|                                                              |
| FILES/DEVICES DESCRIPTIONS                                   |
|                                                              |
| PARAMETER DESCRIPTIONS                                       |
|                                                              |
| ABBREVIATIONS                                                |
================================================================
```

The first line(s) in the block will show the allowable command structure. In some cases, more than one command structure will be shown. Throughout this manual, several words may be used as prepositions separating commands and/or parameters. They are:

TO ON OVER USING

The use of these prepositions is always optional; the LDOS command will function the same whether they are used or not. They are merely a convenience to allow the user to enter a command in more conversational syntax. If a preposition is not used, a single space must be used between words.

Throughout this manual, you will see references to "filespec" and "devspec". These are the abbreviations for "file specification" and "device specification". The INTRODUCTION TO LDOS described what filespecs and devspecs are. Due to the device independence of LDOS, it is possible to interchange these two specifications in some Library commands. For example, you can copy your keyboard to your line printer, or to a disk file. You can even append information from a device onto the end of a disk file! Each Library command will give detailed instructions and examples of interchanging filespecs and devspecs, if applicable.

Certain LDOS Library commands and utilities allow the use of "partspecs" (partial file specifications) and "not-partspecs". A partspec is any or all parts of a filespec, generally excluding the password. For example, the full filespec for the LDOS utility REPAIR is:

```
REPAIR/CMD.RRW3:Ø
```

Some examples of partspecs would be:

```
REPAIR/CMD:Ø   REPAIR        /CMD          REP/C         REP:Ø         R/C
```

A not-partspec is simply a partspec preceded by a dash character, such as -REPAIR, -/CMD, etc. Also, a not-partspec would be used to exclude a certain file or group of files from a command, while a partspec is used to include a file or group of files. For example, using a partspec of REP would find a match with all of the following files:

```
REPAIR/CMD     REPAIR/BAS     REPAIR/ASM     REPEAT/BAS     REPRESEN       REP1Ø2:3
```

Since some of the LDOS Library commands and utilities allow the use of partspecs, you can use the filename and extension fields to create files with common attributes, and then access them as a group. LDOS also creates or uses default extensions during some operations. Other operations can then use these default extensions when searching for a file.

The parameters section of the syntax block will give a very short description of the allowable parameters for the command. This description will generally be very brief, as a complete explanation will be given in detail in the text of that section.

Please note that many command parameters may have a default value if they are not specified. This may not be readily apparent, as many operating systems do not allow any parameters for these commands. For example, the DIR Library command used to view a disk's directory has a parameter called SORT. The default of this parameter is ON, so the directory display will automatically be in sorted alphabetic order.

The abbreviations line will list all acceptable parameter abbreviations for the function. Note that (ON,YES and Y) and (OFF, NO and N) are completely interchangeable in most commands in the LDOS system.

SPECIAL COMMAND SPECIFICATIONS

Drivespecs must always be preceded by a colon, whether used as part of a filespec or as a stand alone parameter, except in the DIR Library command.

The closing parenthesis may be omitted from any LDOS command.

It is totally acceptable to enter any filespec, command or parameter in either upper or lower case. As an arbitrary convention, this manual shows most command lines and error messages in upper case. The actual LDOS messages will be displayed in upper and lower case, assuming your hardware is capable of lower case display.

Numeric values for any parameter may be entered in either decimal or hexadecimal. Decimal numbers are entered in normal notation, such as PARAMETER=32768. Hexadecimal numbers are entered as X'value', such as X'FØØØ', X'ØD', etc. Using PARAMETER=X'8ØØØ' would produce the same value as the previous example of PARAMETER=32768.

HARDWARE RELATED FEATURES

Your LDOS Disk Operating System is a user-oriented, device independent system. LDOS provides compatibility between the TRS-8Ø Models I and III, so your data files and most BASIC programs will be truly transportable. Assembly language programmers will find the LDOS SVC table provides direct transportability of programs between LDOS supported computers. LDOS also contains many features that have never before been found in a micro-computer operating system. New users will discover that the LDOS manual will answer most questions about their computer's operation. Those familiar with the TRSDOS-like disk operating systems should be able to step right up to the LDOS system, as much of the command structure and syntax is similar. However, to get the greatest value out of the system, it will be necessary to read and study the user's manual. This section will deal with generalized conventions that exist throughout the operating environment. It will also give an overall view of the total LDOS system.

Let's start by listing some of the hardware related features that you will find when using LDOS.

HARDWARE RELATED FEATURES

1) THE KEYBOARD will originally use the ROM driver. On the Model I, this will not provide key debounce, key repeat, type ahead, or any other advanced feature. The Model III has debounce, key repeat, and screen print built into the ROM driver.

LDOS comes with a keyboard driver program called KI/DVR. The use of this driver is mandatory if functions such as key repeat, type ahead, screen print, printer spooler, KSM, MiniDOS, LCOMM, or the SVC table are to be used. **It is strongly recommended that the KI/DVR program with the TYPE option be active in your runtime system.** It requires very little memory space and will make using LDOS even more pleasant. Use of the KI/DVR program will enable you to easily type in either upper or lower case. It also establishes the <SHIFT><Ø> key as a CAPS lock key.

Once the KI/DVR program has been set, shifting between the CAPS lock mode and the normal upper/lower case mode can be accomplished by pressing the <SHIFT><Ø> keys. In the normal upper/lower case mode, unshifted alphabetic keys are entered as lower case, and shifted keys as upper case, the same as on a standard typewriter. In the CAPS lock mode, any alphabetic character will be displayed as upper case, whether the <SHIFT> key is held down or not. On the Model III, you will initially find that all keyboard entries will be in lower case. The Model I will initialize in the CAPS lock mode. This may be changed on either machine, as described in the SYSTEM CONFIGURATION section, page 1-24.

When using the KI/DVR program, the KSM and MiniDOS functions may also be used. Keys may be assigned special commands, functions, or characters with the KeyStroke Multiply (KSM) feature. These associated functions are then available when the <CLEAR> and desired unshifted key are pressed together. Due to this, it is necessary to press the <SHIFT><CLEAR> to clear the screen when the KI/DVR program is used. The MiniDOS filter program will give you immediate access to certain LDOS functions, such as a disk directory or amount of free space, a line printer top of form command, repeat the last DOS command, and a disk file kill command.

The <SHIFT><BREAK> key will re-select a 5" disk drive that has "timed out" and hung up the system. This may happen if you attempt to access a drive with the drive door open, or if there was no diskette in the drive, etc. It should prevent having to reset the entire system. Ready the drive for access and then press the <SHIFT><BREAK> keys to complete the operation. Do not press the <SHIFT><BREAK> keys if the system is currently active!

2) THE DISK DRIVES in LDOS can be 5", 8", or hard disk. LDOS will support a total of 8 disk drives. The drives may be double/single sides and density, and up to 96 tracks on floppy disks. Hard disk support will be determined by the manufacturer/distributor of the hardware. At present, no more than four of any one drive type (5 or 8 inch floppy, or hard drive) may be accessed. Of course, you must have the appropriate hardware and drivers for this. If you have purchased a supported hard disk system, driver programs and documentation for hard disk will be provided separately.

3) THE VIDEO display will allow display of upper and lower case characters, assuming your hardware is capable of lower case display. If you have a Model I with the switch type of modification, be sure to have the switch in the lower case position when booting. Keyboard entries will normally be displayed in all upper case unless the KI/DVR program has been set. If KI/DVR is used, keyboard entries will be displayed as determined by the mode (normal or caps lock) set with the <SHIFT><Ø> function.

4) ALL SYSTEM HARDWARE DEVICES are totally independent of the normal routing structure found in most operating systems. Your system devices such as the video display and printer can be routed or linked almost anyway you could desire - to each other, to a disk file, to another device, etc. You can even create your own logical devices!

5) THE CASSETTE on the Model III can be used in either the 5ØØ or 15ØØ baud mode. Use of the high speed (15ØØ baud) mode requires the use of the HITAPE/CMD program. Both the LBASIC and CMDFILE utilities allow high speed tape operation.

Once you have powered up your system, you can control the boot sequence to some extent. Note that if the <BREAK> key is held down during power up or reset, the computer will immediately enter ROM BASIC. Otherwise, you may be prompted to enter the date and/or time. After answering these prompts, there are several keys that will modify the remaining boot sequence if held down. They are:

<CLEAR> This key will prevent any configuration file stored on the disk from being loaded. The configuration would have been created and stored with the SYSTEM (SYSGEN) library command.

<D> This key will cause the debugger (non-extended) to be loaded and executed. No configuration file will be loaded, and all memory above X'52ØØ' will be untouched. Use of this debug function is explained under the library command DEBUG.

<ENTER> This key will prevent the execution of any breakable AUTO commands from taking place. Refer to the library command section AUTO.

<RIGHT ARROW> ** MODEL III ONLY ** This key will prevent the LDOS video driver from being loaded. The system will use the ROM video driver instead. This may be necessary for certain machine language programs. CAUTION: Using the ROM video driver will cause problems with Type Ahead, Lcomm, the Spooler, and any other LDOS function that uses interrupt processing, and should NOT normally be done!!

Once the system has booted and displayed the message "LDOS READY", it is ready to accept a command from the user.

LDOS FILE DESCRIPTIONS

Throughout the manual, you will see references to "filespec" and "devspec". These are abbreviations for "file specification" and "device specification". Due to the device independence of LDOS, it is possible to interchange these two terms in most library commands. For example, you can copy your keyboard to your line printer, or to a disk file. You can even append information from a device onto the end of a disk file! Each library command will give detailed instructions and examples of the interchanging of filespecs and devspecs (if applicable).

Certain LDOS library commands and utilities allow the use of partspecs (partial filespecs). This will allow you to use the filename and extension fields to create groups of files with common filespecs, and then access these files as a group. LDOS creates "default" extensions in the filespec during some operations. Other operations will use these default extensions when searching for a file. Following is a list of LDOS default extensions along with suggestions for others that may help you "standardize" your file access.

ASM - The extension used by some Editor/Assembler programs for source files.

BAS - LBASIC default for programs. Also used by some BASIC compilers.

CIM - LDOS default for DUMP command. It stands for Core IMage.

CMD - LDOS default for LOAD and RUN commands, and PATCH and CMDFILE utilities. Used to indicate load module format files.

COM - Used by some systems to indicate COMpiled object code.

DAT - Possible extension for data files.

DCT - LDOS default for the SYSTEM (DRIVER) command (Drive Code Table).

DVR - LDOS default for the SET command. Usually indicates a "driver" program.

FIX - LDOS default for files to be used by the PATCH utility.

FLT - LDOS default for files used with the FILTER command.

JBL - LDOS default for Joblog files.

JCL - LDOS default for the DO command. Stands for Job Control Language.

KSM - LDOS default for KSM Utility. Stands for KeyStroke Multiply.

OVx - LBASIC extension for Overlay files (Overlay "x").

REL - Used by some systems to indicate relocatable object code.

SCR - LDOS default for Scripsit text files.

SEQ - Possible extension for sequential files.

SPL - LDOS default for the SPOOL command.

SYS - LDOS SYStem files only. Do not use for your own files!

TXT - LDOS default for the LIST and DUMP (with the ASCII parameter) command. Stands for TeXT file.

This next section will describe the various files found on your LDOS master diskettes, and explain their functions. It will also describe how to construct a minimum system disk for running applications packages.

FILE GROUP - SYSTEM FILES (/SYS)

LDOS's use of overlays requires that a SYSTEM diskette usually be available in drive Ø - the system drive. Since the diskette containing the operating system and its utilities leaves little space available to the user, it is useful to be able to remove certain parts of the system software not needed while a particular application is running. In fact, you will discover that your day-to-day operations will only need a minimal LDOS configuration. The greater the number of system functions unnecessary for your application, the more space you can have available for a "working" system diskette. The following will describe the functions performed by each system overlay, identified in an LDOS DIR command (using the SYS parameter) by the file extension, /SYS. There are two system files that are put on the disk during formatting. They are DIR/SYS and BOOT/SYS. These files are NEVER to be copied from one disk to another! LDOS automatically updates any information contained in these files.

SYSØ/SYS

This is not an overlay. It contains the resident part of the operating system (SYSRES). Any disk used for booting the system MUST contain SYSØ. It may be removed from disks not used for booting.

SYS1/SYS

This overlay contains the LDOS command interpreter, the routines for processing the @FEXT system vector, the routines for processing the @FSPEC system vector, and the routines for processing the @PARAM system vector. This overlay must be available on all SYSTEM disks.

SYS2/SYS

This overlay is used for opening or initializing disk files and logical devices. It also contains routines for checking the availability of a disk pack (services the @CKDRV system vector), and routines for hashing file specifications and passwords. This overlay must also reside on all SYSTEM disks.

SYS3/SYS

This overlay contains all of the system routines needed to close files and logical devices. It also contains the routines needed to service the @FNAME system vector. This overlay must not be eliminated.

SYS4/SYS

This system overlay contains the system error dictionary. It is needed to issue such messages as "File not found", "Directory read error", etc. If you decide to purge this overlay from your working SYSTEM diskette, all system errors will produce the error message, "SYS ERROR". It is recommended that you not eliminate this overlay, especially since it occupies only one granule of storage.

SYS5/SYS

This is the "ghost" debugger. It is needed if you have intentions of testing out machine language application software by using the LDOS DEBUG library command. If your operation will not require this debugging tool, you may purge this overlay.

SYS6/SYS

This overlay contains all of the algorithms and routines necessary to service the library commands identified as "Library A" by the LIB command. This represents the primary library functions. Very limited use could be made of LDOS if this overlay is removed from your working SYSTEM disk.

SYS7/SYS

This overlay contains all of the algorithms and routines necessary to service the library commands identified as "Library B" by the LIB command. A great deal of use can be made of LDOS even without this overlay. It performs specialized functions that may not be needed in the operation of specific applications. Use the PURGE command to eliminate this overlay if you decide it is not needed on a working SYSTEM diskette.

SYS8/SYS

This overlay is needed to dynamically allocate file space used when writing files. It must be on your working SYSTEM diskettes.

SYS9/SYS

This overlay contains the routines necessary to service the extended debugging commands available after a DEBUG (EXT) is performed. This overlay may be purged if you will not need the extended debugging commands while running your application. In addition, if you purge SYS5/SYS, then keeping SYS9/SYS would serve no useful purpose.

SYS1Ø/SYS

This system overlay contains the procedures necessary to service the request to kill a file. It should remain on your working SYSTEM diskettes.

SYS11/SYS

This overlay contains all of the procedures necessary to perform the Job Control Language execution phase. You may remove this overlay from your working disks if you do not intend to execute any JCL functions. If SYS6 has been purged (containing the DO command), keeping this overlay would serve no purpose.

SYS12/SYS

This overlay contains the routines to service the @DODIR and @RAMDIR directory vectors. These routines are used by the MiniDOS filter and may also be used by other applications programs that provide a directory display.

FILE GROUP - UTILITIES (/CMD)

BACKUP - Used to duplicate data from one disk to another.

CMDFILE - A disk/tape, tape/disk utility for machine language programs.

CONV - Used to move files from Model III TRSDOS to LDOS 5.1.x.

FORMAT - Used to put track, sector, and directory information on a disk.

HITAPE - Used for 15ØØ baud cassette operation on the Model III.

LCOMM - A communications package for use with the RS-232 hardware.

LOG - Used to log in a double sided diskette in drive Ø. Also updates the drive code table information the same as the DEVICE library command.

PATCH - Used to make minor changes to existing disk files.

PDUBL - Used with non-Radio Shack double density modifications on the Model I. Also provides double sided disk support.

RDUBL - Used with the Radio Shack double density modification on the Model I. Also provides double sided disk support.

REPAIR - Used to correct certain information on non-LDOS formatted diskettes.

TWOSIDE - Used with single density, double sided drives on the Model I.

FILE GROUP - DEVICE DRIVERS (/DVR)

JL - The LDOS JobLog driver program.

KI - The LDOS Keyboard driver, providing Type Ahead, Screen Print, and special <CLEAR> key functions.

RS232R - Used to access the RS-232 hardware, Model I.

RS232T - Used to access the RS-232 hardware, Model III.

FILE GROUP - FILTER PROGRAMS (/FLT)

KSM - Establishes the KeyStroke Multiply feature, which allows assigning user determined phrases to alphabetic keys.

MINIDOS - Establishes certain immediate functions to shifted alphabetic keys.

PR - Allows the user to format printed output.

FILE GROUP - BASIC AND BASIC OVERLAYS

BASIC/CMD - Translates a BASIC command into the equivalent LBASIC command.

LBASIC - Enhanced Disk Basic program.

LBASIC/OV1 - Renumber overlay used with LBASIC's CMD"N" feature.

LBASIC/OV2 - Cross reference overlay used with LBASIC's CMD"X" feature.

LBASIC/OV3 - Error display and Sort routines for LBASIC.

MISCELLANEOUS FILES

EQUATEx/EQU - A file of LDOS system entry points and storage locations for use by assembly language programmers (the "x" will be a 1 for Model I and a 3 for Model III).

MODx/DCT - Used by hard drive systems to set up the floppy disk drives.

COPY23B/BAS - Used to move files from Model I TRSDOS 2.3B or later.

CREATING A MINIMUM CONFIGURATION DISK

All files except certain /SYS files may be removed from your run time drive Ø disk. Additionally, if the needed /SYS files are placed in high memory with the SYSTEM (SYSRES) command, it will be possible to run with a minimum configuration diskette in drive Ø after booting the system.

For operation, SYS files 1, 2, 3, 4, 8, and 1Ø should remain on drive Ø or be resident in memory. SYS 2 and SYS 8 must be on the boot disk if a configuration file is to be loaded. SYS 12 can be removed if the two "mini" directory routines are not needed. SYS 11 must be present only if any JCL files will be used. Both libraries (SYS 6 and SYS 7) may be removed if no library commands will be used. SYS 5 and SYS 9 may be removed if the system debugger is not needed. SYS Ø may be removed from any disk not used for booting. Note that SYS 11 (the JCL processor) and SYS 6 (containing the DO library command) require that both files be on the disk if the DO command is to be used - if you kill SYS 6, you may as well kill SYS 11.

When using LBASIC, the OV3 overlay must also be present. OV1 and OV2 may be removed if no renumbering or cross referencing will be done.

The presence of any utility, driver, or filter program is totally dependent upon the user's individual needs. Most of the LDOS features can be saved in a configuration file with the SYSTEM (SYSGEN) command, so the driver and filter programs won't be needed on run time application disks.

The passwords for LDOS files are as follows:

System files	(/SYS)	Update password = SYSTEM
Filter files	(/FLT)	Update password = GSLTD
Driver files	(/DVR)	Update password = GSLTD
Utility files	(/CMD)	Update password = RRW3
LBASIC		Update password = BASIC
LBASIC overlays	(/OV$)	Update password = BASIC
CONFIG/SYS		Update password = CCC

LDOS SYSTEM DEVICES AND DISK DRIVES

LDOS SYSTEM DEVICES

The LDOS operating system is a truly "Device Independent" system. Each device the system uses has its own "control area" of memory, called a DCB (Device Control Block). This is true for hardware devices as well as any "phantom" devices created by the user. Each device has its own driver routine, whether located in the ROM or in RAM.

An LDOS device is used or created by specifying an asterisk followed by two alphabetic characters. Your original LDOS master diskette is configured with six devices in the device control table. To view these devices along with the currently enabled disk drives, use the DEVICE library command. You will see the devices as listed:

*KI This is the Keyboard Input (your keyboard).

*DO This is the Display Output (your video screen).

*PR This is the PRinter output (your parallel printer).

*JL This is the Job Log (an output log of commands with a time stamp).

*SI This is the Standard Input (presently unused by LDOS).

*SO This is the Standard Output (presently unused by LDOS).

Note that these are just the LDOS system supplied devices; it is possible for you to create YOUR OWN DEVICES!

There is another LDOS device that is referenced in this manual, even though not shown in a "normal" device table. This device is the Comm Line (*CL), and will also be explained in this section.

The LDOS device independence makes it possible to route devices from one to another, or even to a disk file. It allows setting the device to a totally different driver routine. It makes possible single or multiple links of devices to other devices or to disk files. In other words, you may re-direct the I/O between the system, its devices, and the disk drives in almost any manner.

**** N O T E ****

Once a normal LDOS device has been pointed away from its original driver routine, it may be returned to its normal power up state with the RESET Library command. A user created device may be either disabled or completely removed via the RESET and KILL library commands. Refer to these two commands for examples.

Besides just sounding impressive, this device independence feature has many practical aspects. For example, the line printer is normally controlled by a very simple driver routine. However, the printer output may be filtered with the PR/FLT program supplied with LDOS.

This filter program allows you to set parameters such as the number of characters per line, the indent on line wrap around, the lines per page, the page length, etc. If you don't have a printer, simply use the ROUTE library command to route the printer to a disk file and all printing will be saved, enabling you to print it later, on a system with a printer. You could also route the printer to the display, and the characters will appear on the video instead of going to the printer.

Throughout this manual, you will see reference to the terms "device" (or "devspec") and "logical device". These terms represent the six system devices plus any devices the user has created. To create a device, please refer to the library commands LINK, ROUTE, and SET. It is possible to use certain library commands involving data I/O such as APPEND, COPY, and KILL with device specifications (devspecs) as well as file specifications (filespecs). It is not possible to imagine or describe all situations involving the possible uses and creation of devices. What this section will do is to explain the six LDOS system devices. Any other device interaction or creation will be determined by the individual needs, sophistication, and imagination of the user.

***KI - The Keyboard**

The *KI device is the keyboard. The power up *KI driver will be the ROM driver, as explained earlier. Using the KI/DVR program will give you features such as type ahead and adjustable key repeat. It also establishes the <CLEAR> key as a special control key. This will be used by programs such as KSM and MiniDOS filters, and the LCOMM communications utility. Certain other programs and commands, such as the SPOOL library command, require that KI/DVR be set for proper operation. The individual program sections will note if it is necessary to have the KI/DVR program set. If in doubt, check the appropriate command section of the manual. It is strongly recommended that the KI/DVR program be active during normal operation. It requires very little memory space, and enables many functions not available if KI/DVR is not used. The address of the *KI driver routine as shown with the DEVICE library command will be changed to a location in high memory if any of these functions are used, or if *KI is routed, set, or linked. The DEVICE command will also show the currently selected keyboard options.

You are advised not to route or link the *KI device unless you are extremely careful. You may inadvertently remove all input to the system or introduce totally unwanted characters. To send the *KI characters to a specific device or file, see the library commands APPEND and COPY.

***DO - The Video Display**

The *DO device is the video display. If your computer is equipped with a lower case modification, the power up video driver routine will include a lower case driver. Note that the *DO driver routine address will also change if *DO is involved in a route, set, or link. If you wish to send a screen display to a disk file, there are some simple ways to accomplish this. You can route the printer (*PR) to a disk file, and then link *DO to *PR. This will send all screen displays (including errors - backspace characters, etc.) to the printer, which is routed to a file so the output will really go to the disk. You could also enable the screen print function with the KI/DVR program and route the printer to a disk file. By pressing the <SHIFT><DOWN ARROW> and the <*> keys, the current screen display will be sent to the printer, and actually be written to a disk file. The *DO may also be linked to a disk file using a "phantom" user device (see the LINK library command).

***PR - The Line Printer**

The *PR device is the line printer. This device may be set to other drivers or routed to disk files very easily in the LDOS system. A printer filter program is supplied on your LDOS Master diskette, and is called PR/FLT. This program will allow you to set page size, line length, line indent on wrap around, and many other parameters. The operation of this driver routine is detailed in the DRIVERS/FILTERS section of the manual.

For serial printer owners, the supplied RS232/DVR program should enable you to interface with your printer. Use the SET library command as follows:

SET *PR TO RS232x (parameters)

with the "x" being either the letter "R" or "T" for the Model I or III. The (parameters) portion of the command will be determined by the internal settings of your particular printer. It should be noted that the BUSY or CTS line of your printer will usually have to be connected to the CTS line of your RS-232 cable to provide proper handshaking between the printer and the RS-232 hardware.

You may also use the SPOOL library command to create disk and/or memory buffers to store information being sent to *PR and spool it out as *PR becomes available (i.e. not in a busy state, such as printing a line).

If *PR is routed to a disk file or another device, it will not be necessary to have a line printer physically hooked to the system. All I/O to the printer will be sent to the appropriate device or file, and no lockup will occur. Using the PR/FLT program will also prevent the system from locking up if a print command is given with no line printer available.

***JL - The Job Log**

The *JL device is the system's Job Log. This unique feature will keep a log of all commands entered or received and most system error messages, along with the time they occurred. The time is determined by the setting of the real time clock, and may be set or changed with the TIME library command. To enable *JL, use the SET command to set *JL to its driver program called JL/DVR (see the section DRIVERS/FILTERS). Every command or request processed through the LDOS command line interpreter will then be logged in the *JL file, along with the time of the request. You may also enter comments or data directly into the Job Log by using an LDOS comment line (any line beginning with a period), or by opening a Job Log file from BASIC and writing to it. The Job Log may be turned off by using the RESET library command to reset *JL, which will also close the associated disk file.

***SI - The Standard Input**
***SO - The Standard Output**

The *SI and *SO devices are system generated devices provided by LDOS, although they are not presently used by the system. Both devices will initially be shown pointed (NIL). These devices are available to the user.

***CL - The Comm Line**

*** NOTE ***

This device has been designated *CL strictly for the purpose of standardizing examples throughout this manual, although any other available devspec could have been used (such as *DK, *CJ, *RS, *TM, etc).

The *CL device stands for the Communication Line. It is not an actual physical piece of hardware, but an area of memory used to talk to the RS-232 hardware. This device will allow characters to be sent and received using the RS-232 interface. The *CL will not be shown in the device table unless *CL is set to an RS-232 driver program, but is always available to the user. To enable *CL, simply use the SET library command to set it to an appropriate driver program. There is an RS232/DVR program supplied on your LDOS disk for this purpose. Please note that the LCOMM/CMD program examples also use *CL as its RS-232 link (see the LCOMM utility program).

Whenever I/O is needed via the RS-232 interface, the *CL will provide it. The RS-232 driver program allows *CL to interface between the LDOS system and external devices such as a serial line printer, an acoustic coupler (commonly called a modem), a hard wired telephone data set, a paper tape reader, etc. Please refer to the RS-232 DEVICE DRIVER section for a complete description of the allowable configurations of the RS-232 hardware.

LDOS even provides a method to put your TRS-8Ø into a "host" mode for access by a remote terminal. To do this, set *CL to the RS-232 driver program. It may be necessary to specify the RS-232 word parameter as "WORD=8" and the parity parameter as "PARITY=OFF", depending on the terminal you are using. Then, issue the following library commands:

```
LINK *KI *CL
LINK *DO *CL
```

This will link the display and keyboard to the RS-232 interface, and allow inputs to be taken from and output to be sent to a remote station via the RS-232 hardware.

LDOS DISK DRIVE ACCESS

Your LDOS master diskette comes configured to access four 5" floppy disk drives. The initial default drive configuration comes on the master diskette, and will remain consistent on any backup copy made from the master. To view your initial configuration, type in the command DEVICE at the "LDOS Ready" prompt. Be sure no configuration file has been loaded. Do not have diskettes other than your master LDOS diskette in any drive at this time, or you will not get a true picture of the default drive configuration.

LDOS reserves a certain area of memory called the Drive Code Table (referred to as the DCT) to store information about the disk drive configurations. This drive code table is used by LDOS any time access is attempted to a disk drive. The exact details of the drive code table will not normally concern the average user. However, detailed information on disk drive access can be found in the Technical section under the heading DRIVE CODE TABLE, which includes information on suggested hard drive setup.

LDOS provides features such as type ahead, the spooler, and LCOMM that use the hardware interrupt clock. Because a disk rotational speed of 3ØØ RPM is evenly divisible by the interrupt clock rate, from time to time it may appear that the disk drives "go to sleep" for 1Ø to 3Ø seconds. To avoid this, we recommend that the drives be set to run at 3Ø2-3Ø3 RPM. This will assure optimum disk operation without degrading the overall reliability of the system.

As stated above, issuing a DEVICE library command will show all currently enabled disk drives. Two of the display areas are very important to understand - the "logical" and the "physical" drive numbers. Refer to the following table:

"LOGICAL DRIVES" - any number between Ø and 7

Any time the LDOS manual refers to a "drivespec", it will be referring to the logical drive number. The logical drive number will be shown in examples preceded by a colon. ":Ø" means drive zero, the first drive - drive ":1" means drive 1, the second drive.

"PHYSICAL DRIVE" - shown as 1, 2, 4, or 8

The physical drive number refers to a drive's position on the drive cable. For floppy drives, the numbers 1, 2, 4, and 8 will be shown by the DEVICE Library command, corresponding to the first, second, third, and fourth positions on a cable. When using special hardware such as a hard disk, an appropriate disk driver program will be supplied so you can set the logical and physical drive locations to match your actual needs. Since there are only 4 physical locations available, no more than 4 of any single drive type may be used.

The DEVICE command display also shows other drive information - the number of cylinders, the density, number of sides, step rate, and delay (5" floppies only). This information will be divided into two groups for ease of explanation. Group one contains information about the diskette in the drive, namely the cylinders, density, and side information. Group two contains information about the disk drive itself; the step rate and delay.

Diskette parameter information

All three of the diskette parameters shown in the device display (cylinder, density, and sides) are dependent on the diskette that was in the drive when it was last accessed. These parameters are established when the disk was initially created with the FORMAT utility. You will notice that the device display will show information for an enabled disk drive even if there is no diskette in the drive. This information will reflect the diskette that was last accessed in that drive.

DENSITY will be shown as SDEN (single density) or DDEN (double density). If you have a Model III, or have an appropriate double density board and are using the PDUBL or RDUBL driver for the Model I, you will see the prompt "Single or Double density ?" when formatting a diskette. The density of the diskette will be determined by how this question is answered.

CYLINDER and SIDES are interrelated terms. Most TRS-8Ø disk operating systems use the term "tracks" when referring to a diskette. A track is limited to one side of a diskette. The term "cylinder" refers to a track number on all surfaces of a diskette. On a single sided diskette, a track is the same as a cylinder. On a double sided diskette, there are two tracks with the same number for each cylinder, one on the front and one on the back of the diskette. A multi-platter hard drive may have as many as 8 tracks per cylinder when using LDOS. Again, the number of sides and cylinders are established by the hardware capabilities and by answering questions when formatting the diskette.

Drive parameter information

Two of the drive parameters shown in the device display may be adjusted with the SYSTEM library command. They are the drive step rate and the access delay for 5" floppy drives.

STEP RATE refers to the speed of the disk head movement when moving from one cylinder to another. To assure compatibility with all drive types, the LDOS master disk comes with the step rate set to the slowest rate. The step rates for all drives can be seen with the DEVICE Library command. If desired, they may be changed with the SYSTEM (DRIVE=,STEP=) library command. The manufacturer of your disk drives should provide the maximum step rate the drive can handle. If you specify too fast a step rate, you will not be able to access the disk. You will also be asked to set a "bootstrap step rate" when formatting a diskette. This is the step rate that will be used for booting if the disk will be used as a system disk in drive Ø. Again, too fast a step rate will keep the system from booting, so be sure to check out the fastest rate your drives can handle.

The bootstrap step rate will have no effect on any drive except drive Ø - you must use the SYSTEM library command as previously explained to adjust your other drive step rates.

DELAY refers to the amount of time between the drive motor startup and the first attempted access. It is valid for 5" floppy drives only, as 8" floppies and hard drives motors are always running. Some delay is necessary to allow a 5" drive motor to come up to its normal speed. The default will be 1 second for the Model I, and .5 seconds for the Model III, and may be changed with the SYSTEM (DRIVE=,DELAY=) Library command. The drive manufacturer should provide information on the minimum delay time you can use.

Special Floppy disk drive types

The "standard" LDOS floppy disk types are:

40 track, double density, single side - Model III
35 track, single density, single side - Model I

LDOS will also support any track count up to 96, double density, and double sided drives PROVIDED YOU HAVE THE CORRECT HARDWARE! For the Model III, all that is needed to add double sided support is a double headed drive and the proper drive cable. No special driver programs are required.

For the Model I, double density requires a double density board and use of the PDUBL or RDUBL driver program. These drivers also provide support for double headed drives. For single density, double sided users, the TWOSIDE program must be used. Also, any time double headed drives are used, an appropriate drive cable is needed. The standard drive cable will NOT work.

S T A N D A R D D I S K F O R M A T S

This table reflects the format used on standard LDOS data diskettes.

Size	Density	Tracks	Sides	Sectors/ Granule	Granules/ Cylinder	Directory Sectors	Total Files	User Files	Free Space
5"	Single	35	1	5	2	8	64	48	84K
5"	Single	35	2	5	4	18	144	128	169K
5"	Double	35	1	6	3	16	128	112	152K
5"	Double	35	2	6	6	32	256	240	305K
5"	Single	40	1	5	2	8	64	48	96K
5"	Single	40	2	5	4	18	144	128	194K
5"	Double	40	1	6	3	16	128	112	174K
5"	Double	40	2	6	6	32	256	240	350K
5"	Single	80	1	5	2	8	64	48	196K
5"	Single	80	2	5	4	18	144	128	394K
5"	Double	80	1	6	3	16	128	112	354K
5"	Double	80	2	6	6	32	256	240	710K
8"	Single	77	1	8	2	14	112	96	302K
8"	Single	77	2	8	4	30	240	224	606K
8"	Double	77	1	10	3	28	224	208	568K
8"	Double	77	2	10	6	32	256	240	1138K

MEMORY USAGE AND CONFIGURATION

Certain features of LDOS are user selectable (i.e. the keyboard driver KI/DVR, the printer filter PR/FLT, Model I double density drivers, etc). To implement these features, LDOS will load the necessary program into high memory. There is one term that is very important in the LDOS operating system - **HIGH$**.

This term is pronounced "High dollar", and refers to a location that holds the address of the highest unused memory location. If LDOS is using no high memory, then HIGH$ will contain X'7FFF', X'BFFF', or X'FFFF' for 16K, 32K, and 48K machines, respectively. To see the current HIGH$ value, use the MEMORY Library command. When LDOS needs to use high memory, it does so in the following manner:

1) Find the highest unused memory address by looking at the value stored in the HIGH$ location.

2) Install the necessary code in memory below the current HIGH$ value.

3) Lower the HIGH$ value to protect the added program code.

Any code that LDOS stores in high memory is written to be relocatable. This means that it can load anywhere in memory, and is not restricted to a specific area. Since LDOS always respects the HIGH$ value, it will never attempt to overlay any programs loaded and protected by using the HIGH$ value in this manner.

Unfortunately, other operating systems and/or applications programs do not always respect the HIGH$ value. As a result, programs or BASIC USR routines that load in high memory are not always written in a relocatable manner. They have a fixed load address, and MUST be loaded there to execute properly. If LDOS has previously put program code in that memory area, it will be overwritten. This results in what is called a "memory conflict" - two pieces of program code that want to use the same memory area at the same time. When the LDOS code is something like the KI/DVR program, this usually results in an immediate system crash.

Fortunately, it is possible to get around this problem by using the MEMORY Library command. To resolve a memory conflict, you need only to know the load address and length of the unrelocatable code. We will consider two cases - when the code loads at the very top of memory, and when it loads at some other point.

When the conflicting code loads at the very top of memory, it is very easy to resolve the problem. Since you know the load address of the code, use the MEMORY Library command to change the HIGH$ value to one byte below that address. For example, if a piece of code loads from address X'F900' and goes to the top of memory, you would issue a MEMORY (HIGH=X'F8FF') command. LDOS will now put any of its own high memory code below X'F900', protecting the module that will load there.

When the conflicting code does not load at the top of memory, you can use the same method just described to protect it. However, this will waste any memory between the end of the program and the top of memory. Let's consider the case where a module loads at X'F200' and extends to X'F3FF'. There is 3K of space between the end of the module and the top of memory. To avoid wasting this space, use the following procedure.

1) Load an LDOS module into high memory (i.e., SET KI/DVR, install a filter, etc).

2) Type in the command MEMORY with no parameters to see the current HIGH$ value.

3) If the HIGH$ value is above X'F3FF', repeat steps 1 and 2. If the value has gone below X'F3FF', you will need to start over, stopping before you load the module that caused the HIGH$ value to go below X'F3FF'.

4) Now, issue a MEMORY (HIGH=X'F1FF') command. This will protect the block of memory that will be needed by the unrelocatable module.

5) Continue to load any other LDOS modules as desired.

LDOS has a command that will let you save your current memory configuration to a disk file, and have it load automatically every time you power up or reset the computer. This will let you store the memory allocation you have created, so you can "permanently" resolve any memory conflicts. Refer to the next section, System Configuration.

SYSTEM CONFIGURATION

Certain LDOS features can be configured by the user and stored in a disk file. They will automatically be loaded each time the system is powered up or rebooted. The Library command SYSTEM gives a description of the configuration procedure in its (SYSGEN) parameter section. Using the DEVICE Library command will show the current system configuration, including active disk drives and drive parameters, device I/O paths, and some user selected options currently active. Any high memory driver or filter programs will be saved in the configuration file. Be aware that all memory from the physical top to the current value stored in the HIGH$ pointer will be written to the disk. Be sure there is enough room on the disk to store the configuration file, or a "Disk Space Full" error may occur. Once saved on disk, any configuration may easily be changed by setting the desired parameters and doing another SYSTEM (SYSGEN) Library command. A configuration file may be deleted with the SYSTEM (SYSGEN=OFF) command, or may be bypassed by holding down the <CLEAR> key when resetting or powering up the system.

COMPATIBILITY WITH OTHER OPERATING SYSTEMS

To use files created on other operating systems, it will be necessary to move them onto diskettes that have been formatted by LDOS. The LDOS utilities BACKUP, REPAIR, and CONV, along with the COPY, and COPY (X) Library commands and the COPY23B/BAS program will usually provide the means to transfer your program and data files onto LDOS formatted diskettes.

Under NO circumstances should you use files on other than LDOS formatted diskettes in your actual day to day operation. At press time, LDOS provides either direct access or utility programs to read data from Model I TRSDOS 2.1, 2.2, 2.3, and 2.3B, single or double density DOSPLUS, Model III TRSDOS 1.2 and 1.3, and single density DOUBLEDOS, NEWDOS and NEWDOS8Ø.

THE LDOS LIBRARY

Your LDOS operating system contains a set of commands which direct the overall operating environment. These commands are called LIBRARY COMMANDS. They interface between the extremely complex code of the operating system and the simple one line commands entered by the user.

These LIBRARY COMMANDS are listed in the Table of Contents, and also by issuing the Library command LIB. They are divided into two groups, the PRIMARY and the SECONDARY (or Extended) Library commands. Each of these Library sections resides in a different module of the operating system. In this manner, you may delete the module containing the particular group of Library commands (if they are not needed), thereby freeing extra space on your diskette.

The Primary Library commands are contained in the module SYS6/SYS. It may be deleted if none of the commands will be used during operation.

The Secondary (or Extended) Library commands are contained in the module SYS7/SYS. If the commands contained in the Extended Library are not needed, you may delete this module.

The next section of the manual gives detailed descriptions of all Library commands. The name of the command can be found directly above the page number on the bottom of each page.

NOTE: Those commands preceded by an asterisk are Extended Library commands.

A P P E N D

This command lets you append (add) one file onto the end of another. Its primary use is with data files or ASCII-type text files. Files that are in load module format such as "CMD" or "CIM" type files, cannot be appended properly using the APPEND command. (To append these types of files refer to the CMDFILE utility in the UTILITY section of the manual). BASIC programs cannot be appended unless saved in the ASCII mode. The syntax is:

```
APPEND filespec1 TO filespec2 (STRIP)
APPEND devspec TO filespec (ECHO,STRIP)

        filespec1 and filespec2 are valid LDOS file
        specifications, including drivespecs.

        devspec is any valid, active device capable
        of generating characters.

 ECHO   is an optional parameter that will echo the
        characters to the screen when appending a
        device to a file.

 STRIP  is an optional parameter that will backspace
        the destination file 1 byte before the append
        begins.

 abbr: ECHO=E
```

APPEND copies the contents of file1 onto the end of file2. File1 is unaffected, while file2 is extended to include the contents of file1. The files must each have the same LRL (Logical Record Length) or the append will be aborted with the error message "FILES HAVE DIFFERENT LRLS" and neither file will be touched.

For example, suppose you have two customer lists stored in data files WESTCST/DAT and EASTCST/DAT. You can add the WESTCST/DAT file onto the end of EASTCST/DAT file with the command:

```
APPEND WESTCST/DAT:1 TO EASTCST/DAT:Ø
```

EASTCST/DAT will now be extended to include WESTCST/DAT, while WESTCST/DAT will remain unchanged.

You can also append a device (capable of sending characters) to a file. For example:

```
APPEND *KI TO WESTCST/DAT:2
```

This command will cause characters that are input on the keyboard to be appended to the file WESTCST/DAT on drive 2. Depressing the <BREAK> key at any time will terminate this type of append. Note that the keystrokes will not be shown on the display during this append, as the ECHO parameter was not specified.

APPEND *KI TO WESTCST/DAT:2 (ECHO)

This example will perform identically to the last, except that any key typed will also be echoed to *DO (the video screen).

APPEND PAGE2/SCR:Ø TO PAGE1/SCR:Ø (STRIP)

This example would append PAGE2/SCR to the end of PAGE1/SCR in the following manner. PAGE1 would be backspaced 1 byte, in effect allowing the first byte of PAGE2 to overwrite the last byte of PAGE1. This would be necessary when appending files such as SCRIPSIT files that have an internal end of file marker in the file. If the STRIP parameter was not used, SCRIPSIT would load the appended file only up to the first end of file marker, and ignore the appended PAGE2 section of the file.

* A T T R I B

This command allows you to alter or remove the protection status of a file by changing passwords and/or the degree of access granted by a password. ATTRIB also allows the defining of whether a filename will be visible or invisible when a normal directory of the disk is displayed. ATTRIB will also allow you to alter the diskette name, master password, and lock or unlock all visible, non-SYStem files. The syntax is:

```
ATTRIB filespec.password:d (ACC=a,UPD=b,PROT=c,VIS/INV)
*ATTRIB :d (LOCK,UNLOCK,MPW="aa",NAME="bb",PW="cc")

For filespec ATTRIBs, use the following parameters:

password = update password, used only if a password
           already exists.

ACC=       access password

UPD=       update password

PROT=      the protection level

VIS        visible file in directory

INV        invisible file in directory

abbr: ACC=A, UPD=U, PROT=P, VIS=V, INV=I

*For disk ATTRIBs, use the following parameters:

:d      is an optional drivespec, defaults to Ø.

LOCK    Locks all visible non-system files not currently
        protected by changing their access and update
        passwords to the master password of the disk.

UNLOCK  This parameter removes the access and update
        passwords from visible, non-system files, as
        long as their passwords match the master
        password of the disk.

MPW=    Allows passing the disk's current master
        password in the command line.

NAME=   Allows changing the disk name.

PW=     Allows changing the disk master password.

abbr: NONE
```

This section will deal with attribing a filespec.

The levels of protection associated with the passwords are as follows:

LEVEL	PRIVILEGE
EXEC	execute only
READ	read, execute
WRIT	write, read, execute
NAME	rename, write, read, execute
KILL	All access except re-attrib
ALL	Allows total access
FULL	Same as ALL

The protection levels form a hierarchy, with the highest protection level (EXEC) allowing the least amount of access.

When you create a file, the password you specify becomes both the access and update password. If you don't specify a password, a string of 8 blanks is assigned as a default password for both access and update, in effect creating NO password.

The parameters UPD, ACC, PROT, VIS and INV may be abbreviated to their first character U, A, P, V, and I respectively. The levels of protection (abbreviated P) may be abbreviated to their first TWO characters; KI used instead of KILL, EX used instead of EXEC, etc.

The word which follows the "ACC=" is the access password, and will grant access up to and including the level of protection that is specified. The password that follows the "UPD=" is the update password and always allows complete access to a file.

If the VIS or INV parameters are not specified in an ATTRIB command, they will remain unchanged. If the file is currently visible, it will remain so, and vice versa.

Attrib sets or changes the protection of a file which already exists on a disk. There are several ways to use this feature. Here are some examples of the ATTRIB command:

```
ATTRIB CUSTFILE/DAT:Ø (ACC=,UPD=BOSSMAN,PROT=READ,VIS)
ATTRIB CUSTFILE/DAT:Ø (A=,U=BOSSMAN,P=RE,V)
```

This will protect the file CUSTFILE/DAT on drive Ø so that it can only be read by a file read routine. No password will be required to open and read the file because the access password has been set to "null" by placing no password after "ACC=". It can't be changed or written to in any way unless the update password (BOSSMAN) is used when specifying the file, in which case full access would be given. Notice that the file will be visible in the directory.

```
ATTRIB ISAM/BAS:Ø (ACC=,UPD=SECRET,PROT=EXEC,INV)
ATTRIB ISAM/BAS:Ø (A=,U=SECRET,P=EX,I)
```

After execution of this command no one will be able to list this program when it is brought into LBASIC, because the protection level for the access password has been set to EXECute only. The only way this file can be read into the computer is with the RUN command in LBASIC (RUN "ISAM/BAS"). Notice no password is needed to run the program as none was set with the "ACC=" parameter of the attrib command. This file cannot be loaded, listed or printed without using the update

ISAM/BAS.SECRET because the update password has been given. Remember that the update password will allow complete access regardless of the protection level that has been set. Notice that this file will be invisible in the directory because the INV parameter has been specified.

EXAMPLE: RUN"ISAM/BAS"

Any attempt to look at this program after it is running will cause the program to be deleted from memory. Listing or Llisting the program in LBASIC cannot be done in the normal manner unless the program is loaded using the password SECRET.

```
ATTRIB ISAM/BAS.SECRET:Ø (ACC=NOWAY,UPD=SECRET,PROT=EXEC,INV)
ATTRIB ISAM/BAS.SECRET:Ø (A=NOWAY,U=SECRET,P=EX,I)
```

This command will do the same thing to this LBASIC file except that now the only way to get the program into memory, even to run it, is to know the access password of NOWAY.

EXAMPLE: RUN"ISAM/BAS.NOWAY"

It will now be brought into memory and executed but it cannot be listed. Any attempt to interrupt the execution of the program will cause the program to be erased from memory.

```
ATTRIB ISAM/BAS.SECRET:Ø (ACC=,UPD=,VIS)
ATTRIB ISAM/BAS.SECRET:Ø (A=,U=,V)
```

This command will get rid of all passwords and make the file ISAM/BAS visible in the directory. Notice that the update password of SECRET was required to re-attrib the file.

```
ATTRIB HOST/CMD:Ø (INV)
ATTRIB HOST/CMD:Ø (I)
```

This command will make the file invisible to the normal directory command DIR, without assigning any passwords to the file. To see invisible files, use the (I) parameter of the DIR command.

The following section deals with attribing a disk.

The ATTRIB command will allow you to change the disk name, the disk master password, and the password protection of all visible and non-system files. Any time the ATTRIB command is used, you will be prompted for the disk's master password if it is other than PASSWORD and not specified with the MPW= parameter.

```
ATTRIB :Ø (UNLOCK, NAME="MYDISK")
```

This command will remove all access and update passwords from the user's visible non-system files on drive Ø, as long as the files' current password matches the master password of the disk. It will also change the disk's name to MYDISK. Since the current master password was not specified with the MPW= parameter, you will be prompted for it before this command is actually executed, if it is other than PASSWORD.

ATTRIB :1 (NAME="DATA",PW="SECRET",MPW="BOSSMAN")

This command will change the name of the disk in drive 1 to DATA. It will also change the master password to SECRET. Note that the current master password was specified as BOSSMAN with the MPW= parameter.

ATTRIB (LOCK)

This command will first prompt you for the disk's master password, if other than PASSWORD. It will then change the access and update passwords of all the user's visible, non-system, non-password protected files to the disk's current master password. This command will be carried out on drive Ø, as no drivespec was used.

ATTRIB :1 (NAME)

This command will first prompt you for the drive 1 disk's master password, unless it is PASSWORD. It will then prompt you for the new name to be given to the disk.

* A U T O

The AUTO command lets you modify the power up sequence, by specifying a command to be executed immediately after power-up, reset or reboot. The syntax is:

```
AUTO *dos-command

  * is optional and if used will disable the ability
    of <ENTER> to suspend the execution of the AUTO
    dos-command and also disable the <BREAK> key.

  dos-command can be any executable LDOS command
    with or without parameters up to 31 characters
    in length.

abbr: NONE
```

If the AUTO *dos-command has disabled the <BREAK> key, it is possible to re-enable the <BREAK> after the AUTO command has finished execution. See the SYSTEM (BREAK=ON) library command and the LBASIC CMD"B" command for complete instructions.

Here are some examples of the use of the AUTO command.

AUTO LBASIC

Will write the command LBASIC as an "automatic key-in" on the drive Ø diskette, replacing any previous auto command. From that point on, every time you power up or press the reset button with that diskette in drive Ø, LBASIC will automatically be loaded into memory and executed. An AUTO command takes the place of a keyboard input, just as though the command had been typed in and <ENTER> had been pressed.

AUTO *DO INIT/JCL:Ø

After this has been written to the drive Ø disk, power-up or pressing the reset button will cause the DO file INIT/JCL:Ø to be executed, which will allow several commands to be executed automatically (see DO command and JCL). Note the asterisk immediately preceding the command. This is optional; when used it will disable the ability of the <ENTER> key to halt the auto command. The <BREAK> key will also be disabled from this point.

To restore the power up sequence to the normal LDOS READY, type:

AUTO

This will eliminate any stored automatic key-in by removing it from its storage place on the disk.

If a disk has an active auto command, it will be carried over to the destination disk when doing a mirror image backup.

With the SYSTEM (SYSTEM=) command it is possible to assign a drive other than physical floppy drive Ø as the system drive. This changed configuration can be saved in a configuration file and be executed on every power up or reset. However, any AUTO command will be taken from the boot disk, not from the re-assigned system disk. Any time you are using a disk other than physical floppy drive Ø as your normal system (drive Ø) disk, this will be the case. To change an AUTO command in this situation, it will be necessary to use the following procedure. Power up or reset the system, holding down the <CLEAR> and <ENTER> keys. This will prevent the configuration file from switching the drive Ø disk assignment and also stop any previously issued AUTO command, leaving the booting disk as drive Ø. Now, type in the new AUTO command. You can now reset the machine and let the configuration and the new AUTO command take place.

*** N O T E ***

You can override any breakable AUTO command during power up or reset by holding down <ENTER> during initialization. This may be your only way of regaining control of the system, if the dos-command is not a working program. If the AUTO command disables the <BREAK> key and the program is non-functional, it may seem impossible to regain control of that disk. Should this occur, simply boot another (non-AUTOed) disk in drive Ø. When the LDOS READY appears, place the non-functional disk in drive Ø, type AUTO, and press <ENTER>. The runaway AUTO command will then be removed from that disk.

* B O O T

This command causes the disk in drive Ø to be booted into the system. It has the same effect as pushing the reset button or a power up condition. The syntax is:

```
BOOT  <CLEAR> <RIGHT ARROW> <ENTER> <D>

Holding down the indicated key during the BOOT will
result in the following actions:

<CLEAR>        No sysgened configuration will take place.

<ENTER>        No breakable AUTO commands will be done.

<D>            The system debugger will automatically
               be entered. Note that no sysgened
               configuration will be loaded.

<RIGHT ARROW>  For the Model III only, the video driver
               will be the ROM driver, not the normal
               LDOS driver.

abbr: NONE
```

NOTE: Only single density diskettes may be used to boot the Model I LDOS operating system when using the Radio Shack expansion interface. Only double density disks may be used to boot the Model III.

On the Model III only, holding down the <RIGHT ARROW> key during booting will prevent the LDOS front end to the video driver from being loaded. The system will use the ROM video driver instead. This may be necessary for certain machine language programs. CAUTION: Using the ROM video driver will cause problems with Type Ahead, Lcomm, the Spooler, and any other LDOS function that uses interrupt processing, and should NOT normally be done!!

You may be prompted for the date and time when powering up or booting. These prompts may be enabled or disabled with the SYSTEM library command.

By typing in the BOOT command, the LDOS system disk in drive Ø is booted back into the system. All devices will be returned to their normal power up configuration as if the system had been turned off and then turned on again. Any required filtering, linking, routing, or setting of the SYSTEM library command parameters must be done again at this point, unless a SYSTEM config file has been generated on drive Ø by the use of "SYSTEM (SYSGEN)" (see SYSTEM library command). If the system has been sysgened, the user configuration will be loaded and executed at this time, and any AUTO command will be done.

* B U I L D

This command allows the user to build a file of desired character strings and save this file under any valid filespec. BUILD is in the system mainly to build ASCII files for use with the DO, KSM and PATCH features of LDOS, although you may build files containing any characters X'ØØ to X'FF' with the HEX parameter. The syntax is:

```
BUILD filespec (HEX,APPEND)

filespec  is any valid LDOS filespec

HEX       optional parameter allowing a "packed"
          hexadecimal format only.

APPEND    optional parameter that allows appending
          the BUILD data to the end of existing files.

abbr: NONE
```

The BUILD command is used to create a file (or append to an existing file), a series of commands, comments, or character strings entered from the keyboard. If the filespec does not contain a /ext (extension), the system will automatically assign a default extension of /JCL, for Job-Control-Language (see DO and JCL). If a file with the identical name exists, the BUILD command will abort with the error message "File already exists", unless the APPEND parameter has been specified.

The APPEND parameter will allow you to add to the end of an existing file. Be aware that some programs like the SCRIPSIT word processor place their own end of text marker at the end of a file. To properly extend this type of file, use the BUILD command to create a new file consisting of the information you wish to append to the existing file. Then use the APPEND library command with the STRIP parameter to properly append the new information to the file.

Should the user wish to create a KSM type file (see KSM filter), the file extension should be /KSM. This will prompt you with each key identifier as you enter what you wish that key to represent. This type of build is detailed in the section on the KSM utility.

After the file has been opened, all characters that are typed will be placed in the file just as they appear on the video. Lines are limited to a length of 255 characters. Each line that is entered should be terminated by pressing the <ENTER> key. The build will end and the file will be written to the disk when the <BREAK> key is pressed as the first character of any new line.

NOTE: If you are building a /JCL file, lines will be limited to 63 characters in length.

The HEX parameter will allow you to enter characters other than those directly available from the keyboard. Any one byte character value may be entered in the HEX format "nn". The line length during a hex build will be 254 characters, allowing 127 hexadecimal characters to be entered. The HEX parameter uses a "packed" format, with no spaces or delimiters between bytes.

For example, you could create a character string containing graphics characters in the following manner:

818A9ØA1ØD

This line contains the hexadecimal bytes 81, 8A, 9Ø, and A1. Note that the byte values are entered "packed" together, with no spaces or other delimiters between them. One of the possible uses for this format may be to build graphics strings to be used with the KSM function. If a file is to be used with the KSM function, do not embed the bytes ØD or 3B in the string unless you actually intend for these characters to be present, as these represent the Carriage Return and Semi-colon characters. They will be acted upon by the KSM file as end of line (ØD) and embedded <ENTER> character (3B). Note that each logical line must be terminated with a ØD. Therefore several "logical lines" may appear on each "physical line". Each logical line is terminated with a ØD in the entered string, and each physical line terminated by pressing <ENTER>. The <ENTER> does not terminate the logical line.

EXAMPLE: F5ØDF1ØDFAØD<ENTER>

This would represent three logical lines in a KSM type file. Notice the three ØD's in the string.

IMPORTANT: The HEX parameter will not cause the file to be stored in load module format; it will remain a normal ASCII image type file, even though some of the characters may be well out of the pure ASCII range.

When building files other than KSM or HEX, the line input length should be limited to 63 characters (for clarity). The build will be terminated when the <BREAK> key is entered as the first character in a line.

Following are some examples of the BUILD command.

BUILD THISFILE:2

This will check drive 2 for a file named THISFILE/JCL. If it exists, a "File already exists" error will occur. Otherwise, the file will be opened on drive 2. Note that the default extension /JCL was used, as no extension was specified in the command line. A /JCL file will not allow more than 63 characters per line to be entered.

BUILD MYKEYS/KSM

This command will search all available drives for a file named MYKEYS/KSM. If the file exists, a "File already exists" error will occur. Otherwise, this file will be created on the first available drive. Since the extension was specified as KSM, the prompts A>, B>, C>, D>, etc. will appear one at a time so each of the alphabetic characters may be assigned the character string(s) they are to represent (see the KSM filter). This build will terminate after the letter Z, or when a <BREAK> is used as the first character of a line.

BUILD SPECIAL/:Ø

This will build a file using the name SPECIAL with no extension. Using the "/" with no following characters is the only way to build a file without an extension (overriding the default /JCL extension). Note that the file SPECIAL cannot already exist, or an error will be generated.

BUILD MYJOBS/JCL (APPEND)

This command would search all available drives for a file named MYJOBS/JCL. If not found, it would be created on the first available drive. If the file already existed, any input from the build would be appended onto the end of the file. This is the way, for example, to extend an incomplete JCL file.

BUILD DISPLAY/BLD (HEX)

This command would build a file allowing the use of the "packed" HEX format. The file must not already exist, or an error will be generated. Information may be entered into this file as hexadecimal bytes, and will be stored as a normal ASCII format file. This format will allow 127 hex byte representations per physical line. Logical lines may continue on more than one physical line as long as a ØD does not appear, which would terminate the logical line. The <ENTER> is used to terminate a physical line.

If a non-hex digit is entered, the error message "BAD HEX DIGIT ENCOUNTERED" will be displayed, and the build will abort.

BUILD MYPROGA/FIX:Ø

This would build a file with the desired extension of /FIX for use with the PATCH utility program.

* C L O C K

This command turns on or off the screen display of the real time clock. The syntax is:

```
CLOCK (switch)

switch   The switch ON or OFF

abbr: ON=Y, OFF=N
```

When you enter in this command it will activate a background task and display the "real time" clock in the upper right corner of the screen, at print locations 54 to 61. This clock is under software control and is fairly accurate. The clock will only run in the 24 hour mode. The date can automatically be updated when the clock passes midnight on the Model I by using the SYSTEM (UPDATE) library command. The Model III time and date routines are in ROM, and cannot be made to update the date automatically. The initial date value is normally prompted for when powering up the system. The time and date values may also be set with the TIME and DATE library commands, or the values may be poked into memory from LBASIC.

The real time clock will be turned off while the LDOS system is doing some of its disk I/O functions, such as when using the BACKUP and FORMAT utilities. You will be notified of this by the message:

NOTE: REAL TIME CLOCK NO LONGER ACCURATE

This message notifies the user that the real time clock has lost several seconds or more, because the system had to turn off the hardware clock during certain critical functions.

The CLOCK display may also be turned on and off with the <CLEAR><SHIFT><C> keys if the MiniDOS filter is active. It should be noted that the CLOCK in the upper right hand corner of the screen will take precedence over whatever LDOS or LBASIC may attempt to print at the screen locations occupied by the display.

** NOTE **

The clock on the Model III is NOT accurate when running with 5Ø Hertz AC power.

C O P Y

Copies data from one file or device to another file or device. The syntax is:

```
COPY filespec1 TO filespec2 (LRL=nnn,CLONE=on/off)
COPY filespec1 TO partspec (LRL=nnn,CLONE=on/off)
COPY filespec1 TO :d (LRL=nnn,CLONE=on/off)
COPY filespec:d (X)
COPY devspec TO filespec (LRL=nnn,ECHO)
COPY devspec TO devspec (ECHO)
COPY filespec TO devspec

LRL     is an optional parameter, where nnn= the
        logical record length at which filespec2
        is to be set (1 to 256).

CLONE   indicates the desire for an exact duplicate
        of the directory entry of filespec1. All
        ATTRIButes will be copied with the file. The
        CLONE parameter defaults to ON.

ECHO    will cause any characters copied from a
        devspec to be echoed to the screen.

X       is an optional parameter that allows a single
        drive copy.

abbr: LRL=L, CLONE=C, ECHO=E
```

IMPORTANT

COPY should NOT be used to move System (/SYS) files from one disk to another. The BACKUP utility must be used for this purpose.

The COPY command in LDOS is greatly enhanced and expanded over similar commands in other systems. The user of LDOS should become familiar with this important command as it is used in the LDOS system, so the full power of this feature can be utilized. Special attention should be given to the ECHO, LRL and CLONE parameters, which are totally new to the COPY concept.

LRL is a parameter that allows the establishment of a new logical record length for a file during the copy process. If not specified LRL will default to the LRL of the file being copied. This can be very useful when restructuring data files and for changing ASCII type files to be compatible from one application to another. It may also be needed when converting a source file from one language to another to allow the file to be read by another application language.

CLONE provides a feature that has never before been available to the TRS-8Ø user. With CLONE, the copy will not only duplicate the contents of the file but will also duplicate the directory entry. The access and update passwords will be copied as well as the assigned protection level, the visibility, the create flag, and the modified status of the file. Files that are copied with the CLONE parameter will not have their date touched. The same last-written-to date that appeared in the original will be moved to the CLONE-copied file. If the CLONE parameter is turned off, the date that was set as the system date will be written to the directory as the last-written-to date for the copied file.

If CLONE is not used, and an existing destination file was being copied over, the attributes of the destination file (except for the date) will be unchanged. If the COPY command creates a new file, any password included will become both the access and update password of the destination file, and the file will be visible, even if the file it was copied from was invisible. See the ATTRIB library command for more on file attributes.

ECHO can only be specified when the source for the copy is a device. If specified, all characters will be echoed to the screen as they are sent to the destination file or device.

The X parameter provides a means of copying between two non-system diskettes. During this copy, the user will be prompted to switch disks as necessary. See the example of an X parameter copy at the end of this section.

In the following examples, the use of the word TO between filespecs or devspecs is optional. Spec1 and spec2 need only be separated by a space.

EXAMPLES OF COPYING: filespec1 TO filespec2

Note that when copying files, if filespec2 already exists on the destination drive, it will be overwritten by the copy.

When copying files, the filename, extension, and password of filespec2 will automatically default to those of filespec1 if they are not specified. See the following examples.

```
COPY TEST/DAT:Ø TO TEST/DAT:1
COPY TEST/DAT:Ø TO /DAT:1
COPY TEST/DAT:Ø TO TEST:1
COPY TEST/DAT:Ø :1
```

These four commands will execute identical copies. All parts of filespec2 will default to those of filespec1 if not specified. The use of the word TO is optional in any copy command.

```
COPY TEST/DAT TO :1
```

This command will search the disk drives until it finds a file named TEST/DAT and then copy it onto drive 1, using the filespec TEST/DAT.

```
COPY TEST/DAT.PASSWORD:Ø TO :1
```

This command would copy the password protected file TEST/DAT.PASSWORD from drive Ø to drive 1. The file on drive 1 will be named TEST/DAT.PASSWORD. Remember that all parts of filespec2 including the password will default to those of filespec1 if they are not specified.

```
COPY TEST/DAT:Ø TO TEST/DAT.CLOSED:1 (C=OFF)
COPY TEST/DAT:Ø TO .CLOSED:1 (C=OFF)
```

These commands will copy the file TEST/DAT from drive Ø to drive 1. The file on drive 1 will be named TEST/DAT, and have the update and access passwords set to CLOSED. Notice that the second command dynamically assigned the name and

extension of filespec1 to filespec2 and then added the password CLOSED. If a password exists on the file being copied it cannot be changed during a copy. Also, it will be necessary to turn off the CLONE parameter when assigning a password with the copy command. To change a password see the ATTRIB library command.

COPY TEST/DAT:Ø TO MYFILE:1

This command would copy the file TEST/DAT from drive Ø to drive 1, with the file on drive 1 named MYFILE/DAT. Notice that the extension of filespec2 was not specified and defaulted to /DAT.

COPY TEST/DAT:Ø TO MYFILE/:1

This command will copy the file TEST/DAT from drive Ø to drive 1, with the file on drive 1 named MYFILE. There will be no extension on MYFILE because the "/" with no other characters was specified in filespec2.

COPY DATA/NEW:Ø TO /OLD:Ø

This command will copy the file DATA/NEW from drive Ø to a file named DATA/OLD on drive Ø. The filename was not specified for filespec2 and defaulted to that of filespec1 (DATA).

COPY DATA/V56:Ø TO DATA/V28:1 (LRL=128)

This command will copy the file DATA/V56 on drive Ø to a file called DATA/V28 on drive 1. These two files will contain the same data but the logical record lengths will not be the same. We will assume that the original file had a record length of 256. This would be a normal TRS-8Ø "random" type data file. The file DATA/V28 will be created by the copy and will have a record length of 128 bytes. This ability to reset the LRL of a file is very useful when converting data to be used by a BASIC that can deal with blocked files (record lengths less than 256), such as the LBASIC you run with LDOS. This function is also necessary when you wish to append two files but cannot because they have different logical record lengths. By copying one of these files and setting the LRL parameter to the desired length, the record length can be adjusted and the APPEND library command will then function.

COPY MANUAL/TXT.JWY:Ø TO :1 (L=128)

This command will copy the file MANUAL/TXT with the password JWY from drive Ø to drive 1. In the process of doing the copy the LRL will be changed from whatever it was to 128. Note that the LRL parameter was abbreviated to L in the above example.

COPY CONTROL:Ø /ASC:1 (LRL=1)

This will copy the file CONTROL to CONTROL/ASC on drive 1. The LRL of the file will be changed from whatever it was in CONTROL to 1 byte in length. This is an excellent way to convert a data file to a file that could be handled by a word processor (providing the data file was ASCII to start with).

EXAMPLES OF COPYING: devspec to devspec

When copying from devspec to devspec, it is very important that all devices specified be assigned and active in the system. Any routing or setting that has been done to the devices may affect the copy. Some caution is necessary when copying between devices, as non-ending loops can be generated and lock up the system. In other words, PLEASE do not involve devspecs in your copies unless you thoroughly understand the procedures and constraints involved. Destruction of files and/or locking up the system could easily result from lack of user understanding when using this complex structure of the copy command. Following are a few examples of possible devspec to devspec copies. The results produced by these copies may possibly be duplicated through the use of other LDOS commands (such as ROUTE and/or LINK).

COPY *KI TO *PR

This command will copy the keyboard to the printer. As keys are pressed, they will be sent to the line printer. Depending on the printer, the characters may be printed immediately or may require that a linefeed/carriage return be sent before printing. The keystrokes will not be visible on the video because the ECHO parameter was not specified. Hitting <BREAK> will terminate the copy.

COPY *CL TO *PR (E)

This command will COPY *CL (the RS-232 Comm Line) to *PR (the line printer). Each character that is received by the RS-232 will be processed by the RS-232 driver and then presented to the line printer. Since the ECHO parameter (E) was specified, each character will also be echoed to the screen. Prior to executing this command, the *CL device must have been set to an appropriate RS-232 driver.

EXAMPLES OF COPYING: filespec/devspec TO devspec/filespec

COPY *KI TO KEYIN/NOW:Ø

This would allow the sending of all keyboard entries to the disk where they would be stored in a file named KEYIN/NOW. If the file already exists it will be written over. Because the characters that are typed are going directly to the file, they will not appear on the screen. To view the characters, specify the ECHO parameter. To terminate this copy you should depress the <BREAK> key. The file will then be closed and LDOS Ready will appear.

COPY ASCII/TXT:Ø TO *PR

This command will copy the contents of the file ASCII/TXT to the line printer. Although this command is functional, it would give the same output as would the LIST library command with the (P) parameter. The copy in this example will terminate automatically when the end of the file is reached.

EXAMPLES OF COPYING with the X parameter:

The command COPY filespec:d (X) is similar to a regular copy, except that the X parameter will allow transferring a file from one diskette to another without requiring an LDOS system present on any disk involved in the copy.

The colon and drive number are optional so that you can choose to copy a file on some drive other than drive Ø. This command requires swapping diskettes several times in order to utilize the LDOS operating system modules to perform the transfer. The number of disk swaps can be kept to a minimum by having system modules 2, 3, and 8 resident in memory (see the SYSTEM (SYSRES) library command).

You will be prompted for the correct diskette and when to insert it into the drive doing the copying. The prompts are as follows:

INSERT SOURCE DISK (ENTER) = The disk that contains the file to be copied.

INSERT SYSTEM DISK (ENTER) = This is any LDOS SYSTEM diskette. If the diskette which is currently in drive Ø contains the complete system, just press <ENTER>. If the proper system modules (1, 2, 3, 8, and 1Ø) are resident in memory, you may press <ENTER> without actually inserting a system disk.

INSERT DESTINATION DISK (ENTER) = This is the diskette to receive the file. You must have enough space left on that diskette to contain the entire file to be copied. Under certain conditions, this prompt may appear twice in a row.

The disk swap prompts will be repeated as many times as necessary until the copy is complete.

You cannot COPY (X) logical devices, only disk files. The disk files can be any type file made with any LDOS compatible operating system. Note that the source and destination disks MUST have different pack IDs (disk name and/or master password or date).

After the COPY (X), the destination disk file will have its attributes set as follows:

The file will be visible, whether the source file initially was or not.

If a password is entered on the command line, the destination file will automatically have this password set as its update and access passwords. If no password is specified for the source or destination file, then the destination will have no password protection set.

The correct attributes for the destination file may be re-applied with the ATTRIB library command.

* C R E A T E

This command allows for the creation of a file of the type and size that is requested by the parameters. The syntax is:

```
CREATE filespec (LRL=aaa,REC=bbbb,SIZE=cccc)

LRL=  This is the Logical Record Length to be used.
      It must be an integer in the range 1 to 256
      (default LRL=256).

REC=  This is the number of records of length LRL
      to be allocated to the file.

SIZE= This is the amount of space in K (1Ø24 byte)
      blocks that the file is to be able to hold.
      SIZE may not be specified if LRL or REC are.

abbr: LRL=L, REC=R, SIZE=S
```

The CREATE command is used to pre-create a file of a specified type and size. This allows the file to be as contiguous as possible on the disk and limits the number of disk accesses that must be performed when dealing with the file. This pre-create of a file also assures that the expected amount of space on the disk will be available for use by this file. This file will be dynamically expanded if the created size is exceeded, but it will never decrease in size below its current size. A file cannot be created that would require more space than is available on a disk. Remember, if a drivespec is not used, the system will attempt to create the file on the first available drive.

NOTE: If a file has been created, doing a DIR library command with the (A) switch set will show:

```
For a CREATEd file -   S: nnK
For a normal file  -   S= nnK
```

The normal equal sign (=) will be replaced by a colon (:) to indicate that the file length is the result of a CREATE rather than the actual size of the data in the file (see DIR library command for a complete display example).

CREATE NEWFILE/DAT:Ø (LRL=128,REC=1ØØ)

This command will create a file named NEWFILE/DAT on drive Ø. It will have enough space allocated to accommodate 1ØØ records of 128 bytes each.

CREATE ASCII/DAT:2 (LRL=1,REC=512Ø)

This command will create a file named ASCII/DAT in which records will have a length of 1 byte, and there will be space taken on the disk for 512Ø (5K) of these one byte records.

CREATE MAIL/DAT:3

This command will create the file MAIL/DAT on drive 3. There will be no space assigned to the file at this time. The file name is merely placed in the directory. This is very useful as it allows the placement of a yet-to-be-used file on a designated drive. Since it was not specified, the LRL of this file will be 256.

CREATE GOOD/DAT (REC=5Ø)

This will create a file named GOOD/DAT on the first available drive. There will be space taken for 5Ø records of 256 bytes each, since the default LRL is 256.

CREATE SMALL/FIL:1 (SIZE=1)

This command will create a file SMALL/FIL on drive 1 and take 1,Ø24 bytes of space for the file (in actuality 1 gran will be taken as this is the smallest unit of allocation the system can deal with).

If the file already exists

It is acceptable to create a file larger than it already exists. This is the way that you would permanently assign additional space to a file. Once a file is created, any attempt to create it smaller will not be allowed. For example, a file named TEST/DAT was created with space for 1ØØ records, and a LRL of 256. This file will take up approximately 25K of space on the disk.

As long as no records are written to the file, trying to create the file smaller will merely be ignored. Once records have been written to the file, any attempt to create the file smaller than the space used by the records will give a "File exists larger" error. For example, 5Ø records have been written to the file, and are using approximately 12.5K of space. Trying to create the file to a size of 1ØK will show the error message and the file will be untouched. Trying to create the file to some size between 12.5K and 25K will simply be ignored, as the 25K has already been set aside by the original create command. Creating the file to more than 25K will be allowed, and will permanently assign the extra space allocation to the file.

* D A T E

The DATE command is used to set the month, day, and year for use with your applications programs and by LDOS as it creates and handles your disks. The syntax is:

```
DATE mm/dd/yy
DATE

    mm      is the 2 digit month, Ø1 to 12

    dd      is the 2 digit day, Ø1 to 31

    yy      is the 2 digit year, 8Ø to 87


DATE        with no parameters specified will return
            the current DATE.

abbr: NONE
```

It is more important to set the date with LDOS than other systems, because LDOS uses the date when creating and accessing files, and when making backups and formatting disks. If the date is not set, LDOS will make no date entries in the directory, when it would be useful to see the actual date. When looking at the directory you are able to see the date when a file was created or last written to. If the date was not set, this "last-written-to date" will not be updated, and the file would not show the true last-written-to date. When doing backups or formatting, LDOS will use ØØ/ØØ/ØØ if the date has NOT been set. Because the date is so important in the LDOS system, you will be prompted for it on power-up.

Because LDOS will generate the day of the week and day of the year, memory restrictions to hold this data limit the date to the range Ø1/Ø1/8Ø - 12/31/87. Entering a date outside of this range will not be permitted.

In the lower center of the screen at power-up the system will prompt for the date to be entered. You should answer this prompt with a valid date string. The prompt and the date you entered will then be erased and replaced by a display showing the day of the week in the standard three character abbreviation, the name of the month (also abbreviated), the day of the month, and the year expanded as 19xx. If desired, the date prompt can be removed from the boot sequence with the SYSTEM (DATE=NO) library command. If you do disable the date prompt, the date will not be initialized although it may later be set.

It should be noted that LDOS will store two other numbers calculated from the current date. These are day of the year, and day of the week. These values will be placed in memory, and will remain constant unless the date is reset. The RAM STORAGE ASSIGNMENTS in the Technical section details the exact locations and patterns of these values.

It should be noted that the date will stay set as long as the computer has power applied to it, providing the date storage area is not overwritten by user applications. Therefore when resetting or executing the BOOT library command after the date has been set, the system will automatically recover the date that was last set and will not bother with prompting for the date. The date that had been set will be displayed and the system will continue.

Should you wish to examine the date that is set in the system simply type DATE and press <ENTER>. If the date has been properly set, the system will return the currently set date in day-of-week, month, day-of-month, year format. The current date will also be sent to the Job Log, if active. If you have disabled the initial date prompt, the message "Date not in system" will be displayed.

Should you wish to set the date to one other than the system is currently using, simply enter:

DATE mm/dd/yy

The new date will be set by the system.

EXAMPLE:

DATE Ø1/Ø4/8Ø

Sets the date for the first month (January), the 4th day and the 8Øth (198Ø) year.

On the Model I, the SYSTEM (UPDATE) library command will allow the date to change when the system's real time clock rolls past midnight. Due to hardware restrictions, this is not possible on the Model III. Because the real time clock is turned off during certain I/O functions (most notably during the BACKUP and FORMAT utilities, and sometimes during other disk I/O), the time and date may not remain accurate. If the computer is kept in a constant power on state from day to day, do not depend on the system clock for exact timing functions.

* D E B U G

The DEBUG command turns the LDOS system's debugging utility on or off. The syntax is:

```
DEBUG (switch,EXT)

switch  is the switch ON or OFF. If not specified,
        ON is assumed.

EXT     optionally turns on the extended debugger

abbr:   EXT=E, ON=Y, OFF=N
```

Unlike the other library commands, the DEBUG command does not immediately produce a visible result. It loads the system debugger into memory and then waits to be activated. The extended debugger also loads a separate block into high memory, and protects this area by decrementing the HIGH$ value.

Once the debugger has been turned on, it will be entered when one of the following occurs:

1) The <BREAK> key is pressed.

2) After a program has been loaded, before the first instruction in the program is executed, as long as the file's protection level is not execute only.

The debugger may also be automatically activated by holding down the <D> key during the bootstrap operation.

The debugger will be disabled during the execution of any programs with an execute only password protection.

Refer to the following examples to turn the debugger on or off.

DEBUG (ON) Turns on the standard debugger.

DEBUG (E) Turns on the extended debugger

DEBUG Turns on the standard debugger

DEBUG (OFF) Turns off the debugger, standard or extended.

Once the debugger is turned on, it will remain active until it is turned off, until an execute only program is executed, or until the system is booted. Detailed examples of interaction between the debugger and program modules will be given later in this section.

Once in the debugger, you can return to the LDOS Ready prompt with the command G4Ø2D<ENTER>. From the extended debugger, the command O<ENTER> may be used instead. If you entered the debugger from LBASIC with a CMD"D", a G<ENTER> will return you to LBASIC.

Following is a sample display of the debugger screen.

```
AF = ØD 2C --1-1P--
BC = ØD 61 =>  79 9E 77 23 Ø5 2Ø F9 C9   71 E5 D6 Ø8 38 ØE E1 E5
DE = Ø1 Ø4 =>  1A 4D 45 4D 2Ø 53 49 5A   45 ØØ 52 2F 53 2Ø 4C 32
HL = ØØ 54 =>  Ø1 Ø1 5B 1B ØA ØØ Ø8 18   Ø9 19 2Ø 2Ø ØB 78 B1 2Ø
AF'= ØØ 54 -Z-H-P--
BC'= 51 BØ =>  29 29 29 29 B5 6F CD 8A   51 2Ø EF 1F CE 81 C9 D6
DE'= Ø6 Ø1 =>  Ø9 28 42 FE 19 28 39 FE   ØA CØ D1 77 78 B7 28 CF
HL'= 51 ØØ =>  Ø2 C7 C6 Ø2 FF CB Ø2 F7   1Ø 32 E7 2Ø 32 Ø1 C7 43
IX = 4Ø 15 =>  Ø1 9C 43 ØØ 9A ØØ 4B 49   Ø7 C2 FE 31 3E 2Ø 44 4F
IY = ØØ ØØ =>  F3 AF C3 74 Ø6 C3 ØØ 4Ø   C3 ØØ 4Ø E1 E9 C3 9F Ø6
SP = 41 CA =>  52 Ø4 C3 4B DD Ø3 15 4Ø   5D 45 18 43 3F 3F 4C ØØ
PC = ØØ 62 =>  B1 2Ø FB C9 31 ØØ Ø6 3A   EC 37 3C FE Ø2 D2 ØØ ØØ
     3EØ4 =>  2Ø 34 3Ø 2Ø 31 35 2Ø 3D   3E 2Ø 2Ø 3Ø 31 2Ø 39 Ø3
     3E14 =>  2Ø 34 33 2Ø 3Ø 3Ø 2Ø 39   Ø1 2Ø 3Ø 3Ø 2Ø 34 Ø2 2Ø
     3E24 =>  34 39 2Ø 2Ø 3Ø 37 2Ø Ø3   32 2Ø Ø6 Ø5 2Ø 33 31 2Ø
     3E34 =>  33 Ø5 2Ø 32 3Ø 2Ø 34 34   2Ø 34 Ø6 2Ø Ø9 19 2Ø 3D
```

The debug display contains information about the Z-8Ø microprocessor registers. The display is set up in the following manner:

The register pairs are shown along the left side of the display, from top to bottom. The current contents of each register pair is shown immediately to the right of the register labels.

The AF and AF' pairs are followed by the current status of the flag registers to the right of the register contents. The other register pairs will be followed by the contents of the 16 bytes of memory they are pointing to.

The PC register will show the memory address of the next instruction to be executed. The display to the right of that address shows the contents of that address to that address + X'ØF'.

The bottom four lines of the screen show the contents of the memory locations indicated by the address at the left of each line. Refer to the list of debug commands for information on use of the debugger.

Note that in all examples, any parameter dealing with an address or a quantity must be entered as a hexadecimal number. The debugger will not accept the backspace key. If an incorrect command has been entered it may be cancelled by pressing the X key until the command line is cleared. Also, debug only looks at the last four numbers entered in response to any address question.

In the following examples, the first line following a command will give the syntax of the command. There are 3 ways the debug commands can be entered. The first requires the <ENTER> key be pressed. The second requires the <SPACE> bar be pressed. The third type is immediate and will execute whenever the command key is pressed as the first character in the command. The following commands are allowed in both the regular and the extended debugger.

COMMAND: A

The command syntax is: A

This command will set the display to the alphanumeric mode. Characters outside of the displayable range (X'2Ø'-X'BF') will be displayed as periods.

COMMAND: C

The command syntax is: C

This command will single step through the instructions pointed to by PC (the Program Counter). If any CALL instruction is encountered, the routine called will execute in full. The destination of any jump instruction encountered must be in RAM memory, or the C command will be ignored.

COMMAND: D

The command syntax is: Daaaa<ENTER>

This command starts the memory display from the address aaaa.

COMMAND: G

The command syntax is: Gaaaa<ENTER>
Gaaaa,bkp1,bkp2<ENTER>

This command goes to a specified address and begins execution. The parameters are:

aaaa.....specified address. If omitted, the PC contents are used.

bkp1,2...optional breakpoint addresses. They will cause execution to stop at the specified breakpoint. One or both may be specified. The breakpoints must be in RAM memory. All breakpoints are automatically removed whenever you return to debug. No more than 2 breakpoints are allowed at the same time. If you have entered DEBUG from LBASIC, doing a <G><ENTER> will return you to LBASIC. Doing a <G4Ø2D><ENTER> will return you to the LDOS ready prompt.

COMMAND: H

The command syntax is: H

This command sets the display to show hexadecimal format. This is the format used in the earlier debug example display.

COMMAND: I

The command syntax is: I

This command causes the debugger to execute the command at PC and single-step to the next instruction. This command is identical to the C command except that any CALLS encountered will not automatically be executed in full; they must be stepped through instruction by instruction. Any jump or call instruction encountered must have its destination in RAM, or the I command will be ignored.

COMMAND: M

The command syntax is: Maaaa<SPACE>

This command will allow modification of a specified memory address aaaa. If the display is set to include the specified address, you will see vertical bars

around that byte of memory. The address and current byte will appear in the lower left of the screen. To modify the byte, enter in the desired characters. Pressing the <SPACE> bar will modify the byte and move to the next address. Pressing the <ENTER> key will modify the byte and exit from the M command. Pressing the <X> key will exit from the M command without modifying the current byte. If aaaa is omitted, the current memory modification address (shown by the vertical bars) will be used.

COMMAND: R

The command syntax is: Rrp dddd<ENTER>

This command will modify the contents of a specified register pair.

rp.....represents the register pair to be modified.

dddd...represents the new register contents.

The contents of the registers can be seen while in the register display mode. The PC register may not be altered with this command.

COMMAND: S

The command syntax is: S

This command changes the display format from the register display mode to the Full Screen mode. The full screen mode will display a page of memory (256 bytes) with the current display address being contained in the display (see the D command). The register pairs will not be shown.

COMMAND: U

The command syntax is: U

This command will dynamically update the display, showing any active background tasks. It may be cancelled by holding down any key.

COMMAND: X

The command syntax is: X

This command will return the display to the normal register display mode.

COMMAND: ;

The command syntax is: ;

This command advances the memory display 64 bytes in the register mode and 256 bytes in the full screen mode.

COMMAND: -

The command syntax is: -

This command decrements the memory display by 64 bytes in the register mode or by 256 bytes in the full screen mode.

THE FOLLOWING COMMANDS ARE FOUND ONLY IN THE EXTENDED DEBUGGER.

EXTENDED COMMAND: B

The command syntax is: Baaaa,bbbb,nnnn<ENTER>

This command will move a block of memory from one location to another.

aaaa...is the starting address of the memory block to move

bbbb...is the destination address of the memory block.

nnnn...is the number of bytes to move. Entering a Ø will move 65535 bytes, and will cause the extended debugger to function improperly.

EXTENDED COMMAND: E

The command syntax is: Eaaaa<SPACE>

This command allows you to enter data directly into memory, starting at address aaaa. The contents of memory address aaaa will be displayed and you may then type in 2 hex characters to replace the current contents. Pressing the space bar will then enter the new characters into memory. This operation will automatically advance to the next memory location, and allow you to continue entering characters until the <ENTER> or <X> key is pressed. If aaaa is omitted, the current memory modification address will be used.

EXTENDED COMMAND: F

The command syntax is: Faaaa,bbbb,cc<ENTER>

This command will fill a block of memory with a specified byte. The parameters are:

aaaa...The first address to be filled.

bbbb...The last address to be filled.

cc.....The specified byte to fill the locations with.

EXTENDED COMMAND: J

The command syntax is: J

This command will Jump over the next byte, in effect incrementing PC by 1.

EXTENDED COMMAND: L

The command syntax is Laaaa,dd<ENTER>

This command will locate the first occurrence of the byte dd, starting the search at address aaaa. If aaaa is not specified, the current memory modification address will be used. If dd is not specified, the last byte given in a previous L command will be used.

EXTENDED COMMAND: N

The command syntax is: Naaaa<ENTER>

This command will position the vertical cursor bars to the next load block. This instruction is used to move logically through a block of memory that has been loaded directly from disk using DEBUG. To use this instruction you must position the location bars over the file type byte at the beginning of any block. Press N<ENTER> and DEBUG will advance to the next load block header. The position of this header is determined from the length byte of the load block.

EXTENDED COMMAND: O

The command syntax is: O<ENTER>

This command is the normal way to return to LDOS READY.

EXTENDED COMMAND: P

The command syntax is: Paaaa,bbbb<ENTER>

This command will Print a block of memory from address aaaa to bbbb inclusively. The output will contain both HEX and ASCII formats in the following manner:

aaaa bb bb ... bb cccccccccccccc

aaaa....represents the current line address.

bb bb...represents a line of 16 locations in HEX notation.

cccc....represents the ASCII equivalents of the locations.

EXTENDED COMMAND: Q

The command syntax is: Qii<ENTER>

This command will display the byte at the input port ii.

EXTENDED COMMAND: Q

The command syntax is: Qoo,dd<ENTER>

This command will send the data byte dd to output port oo.

EXTENDED COMMAND: T

The command syntax is: Taaaa<SPACE>

This command will allow you to type ASCII characters directly into memory, starting at address aaaa. The current contents of the address will be shown, and the command will wait for the next keyboard character. After the character is entered, you will advance to the next memory location. To exit this command, use the <ENTER> key. The <SPACE> character cannot be entered with this command. Pressing the <SPACE> will advance one memory location without changing the contents of the current location. If aaaa is omitted, the current memory modification address will be used.

EXTENDED COMMAND: V

The command syntax is: Vaaaa,bbbb,nn<ENTER>

This command will compare the block of memory starting at aaaa to the block of memory at bbbb. The compare will be for nn bytes. If the display is in the register mode, the first byte of memory displayed will be set to the first location in the block starting at aaaa which does not match the block at bbbb. The current memory modification address used by the M, E, and T commands will be reset to the corresponding byte in the second block.

EXTENDED COMMAND: W

The command syntax is: Waaaa,dddd<ENTER>

This command will search memory for the WORD specified with dddd (entered in lsb, msb format). The search will start at memory location aaaa. If aaaa is not specified, the current memory modification address will be used. If dddd is not specified, the last word given in a previous W command will be used. The memory display will automatically be set to show the address where dddd was located, with the vertical bars one byte before the requested word.

EXTENDED COMMAND: DISK READ/WRITE UTILITY

The command syntax is: a,b,c,d,eeee,f

a is the desired disk drive number.

b is the desired cylinder.

c is the first sector to read or write.

d is the operation, R for Read, W for Write, * for a Directory Write.

e is the starting address in memory where the information read from the disk will be placed, or where information written to the disk will be taken from.

f is the number of sectors to read or write.

CAUTION - Be sure to log in the disk to be accessed with either LOG/CMD or the DEVICE Library command before using this option! If the disk will not log in properly, insert another disk of the same density and number of tracks, and log that disk instead.

If the cylinder is not specified, the DIRectory track will be used. If the number of sectors is not specified, a full cylinder will be read. If the starting sector is not specified, sector Ø will be the default.

If an error is encountered during a disk function, the error number will appear on the screen, surrounded by asterisks. The error indication will repeat each time another error occurs. To abort the disk function, hold down the <ENTER> key.

Example: Ø,,,R,7ØØØ

This example would read the directory track from the disk in drive Ø. The information read would be stored in memory starting at location X'7ØØØ'.

Example: Ø,,,*,7ØØØ

This command would write the directory track on drive Ø, using the information stored in memory starting at location X'7ØØØ'. The "*" assures that the proper Data Address Mark will be written to the directory cylinder.

Example: 1,Ø,Ø,W,7ØØØ,5

This example would write to drive 1, cylinder Ø, starting with sector Ø, and writing 5 sectors. The information written to the disk would be taken from memory starting at location X'7ØØØ'.

D E V I C E (and Drive LOG-ON)

This command will display all logical devices which are in use and the devices and files to which they are currently pointing and/or attached to. It will also "log on" the diskettes currently in the available disk drives by updating the Drive Code Table to show the number of cylinders, sides, the density, and the location of the directory track. The syntax is:

```
DEVICE (parm,parm,...)

Allowable parameters are:

B=ON/OFF   If OFF, suppresses the "byte I/O" portion
           of the display. The default is ON.

D=ON/OFF   If OFF, suppresses the drive portion of
           the display. The default is ON.

S=ON/OFF   If OFF, suppresses the options status portion
           of the display. The default is ON.

P=ON/OFF   If ON, duplicates the display to the screen
           and the printer. The default is OFF.
abbr: NONE
```

A typical device display might look like this:

```
:Ø   5" Floppy #1 Cyls= 4Ø, Dden, Sides=1, Step=12 ms, Dly= 1 s
:1WP 5" Floppy #2 Cyls= 35, Sden, Sides=1, Step= 6 ms, Dly=.5 s
:2   5" Floppy #4 Cyls= 8Ø, Dden, Sides=2, Step= 6 ms, Dly=.5 s
:3   5" Rigid  #Ø Cyls=153
*KI <=  X'FC11'
*DO <=> X'4DC2'
*PR <=> X'41E5'
*JL  =  Nil
*SI  =  Nil
*SO  =  Nil
*UD <=> TEXTFILE/TXT:1
Options: Type, KI, JKL
System modules resident: 2,3,8
```

For reference purposes, the display will be considered as having three parts. The first is the DRIVE section, showing the current configuration of the disk drives. Second is the BYTE I/O section, showing the devices (in this example, *KI through *UD). Last is the STATUS section, displaying the status of the user selected options.

The information on the type and configuration of each drive in the system is found at the top of the device display. There are several fields in each line which explain what the system sees for disk storage. Here is a typical line of information pertaining to one drive and the explanation of the fields it contains.

```
:1WP 5" FLOPPY #1 CYLS= 4Ø, DDEN, SIDES=1, STEP=6  MS, DLY= 1 S
---- - ------ -- -------- ---- ------- ---------- --------
aabb c  dddddd ee ffffffff  gggg  hhhhhhh  iiiiiiiii  jjjjjjjj
```

aa This is the logical drive number the line deals with.

bb This is the diskette write protect status, with WP = Write Protected. See the SYSTEM (DRIVE=,WP) command.

cc This is the size of the floppy or hard disk. 5", 8", or 14" will appear in this field.

dd This is the type of drive, floppy or rigid (hard) shown.

ee For floppy drives, this is the physical binary location of the drive on its cable. 1, 2, 4 or 8 will appear here. For hard drives, this will be the starting head number.

ff This is the number of cylinders on the disk that was in the drive when it was last accessed.

gg This shows the density of the last disk accessed in the drive and will show either DDEN or SDEN (double/single density).

hh This shows the number of sides on the last disk accessed by the drive and will be a 1 or a 2.

ii This shows the step rate in "ms" (milliseconds) for the drive.

jj This shows the delay time that will be imposed when accessing a 5" minifloppy drive. It refers to the time the system will wait after starting the drive motor before it attempts to access the disk. It does NOT refer to the time the drive will stay on after an access.

The following briefly describes what each of the *xx devspecs refer to. Each device will be shown as an asterisk followed by the 2 letter device name, its I/O symbol, and an address. The DEVICE command will use special symbols to show the I/O (input/output) paths for each device.

<= will indicate an input device.
=> will indicate an output device.
<=> will indicate a device capable of input and output.

For additional information see the section on SYSTEM DEVICES AND DISK DRIVES in Section I of this manual.

*KI This device is the Keyboard Input which is controlled with the keyboard driver. This may be the ROM driver routine or the KI/DVR driver program that allows key repeat, type ahead, screen print, and special <CLEAR> key recognition.

*DO This device is the Display Output (video).

*PR This device is the line printer, normally accessed with the TRS-8Ø parallel printer connection.

*JL This is the JOBLOG control device which is shown NIL on power up, and must be set to its driver (JL/DVR) to be used.

*SI This is the Standard Input device which will normally be pointed (NIL).

*SO This is the Standard Output and will normally be pointed (NIL).

*UD This may be any user created device. It must be an asterisk followed by any two alpha characters. Setting up "phantom" or "real" devices is a very important part of the device independence of LDOS (see the FILTER, LINK, ROUTE, and SET library commands).

These device relationships and specifications will change from time to time depending on the use of the FILTER, ROUTE, SET, LINK, SYSTEM and RESET library commands. Note that the *UD device shown in the display is a "phantom" device that has been routed to a disk file. A phantom device refers to a device specification that is not an actual piece of hardware - it is merely a way to link to another file or device. The filename of the file is shown after the device that is providing the input to that file. After you have routed or set a device and/or driver, you may use the DEVICE library command to see how the different devices have been affected. After any changes are made to the system, the DEVICE command will show the memory address where each of the hardware devices is going to enter the first driver or filter dealing with that device, as well as the interaction between devices and/or files.

Issuing the DEVICE command will also update disk information in the drive code table in the following manner. Each diskette in a currently enabled disk drive will be examined for number of cylinders, sides, density, and location of the directory cylinder. If a drive is enabled but contains no diskette, the device table entry for that drive will not change. The LOG utility program will perform the same function, but for a single drive only.

If the device command should "hang up" or return totally incorrect drive information, it is because the drive code table does not contain the proper size and location of your drives. The physical location and size (5", 8", or 14") must be correct for the device command to log on the drives. If hard drives are enabled in your system, the number of heads and the starting head number for each logical drive must have been properly set with the appropriate driver program. Should you have a problem when executing the device command, reboot LDOS with the <CLEAR> key held down. If using special hardware, use the SYSTEM (DRIVE=,DRIVER) command to set the proper drive configuration for your system. After assuring the drive information is correct with the device command, save this configuration with the SYSTEM (SYSGEN) command.

The "Options:" line will show you the system options currently active. These options are usually established with the FILTER, LINK, ROUTE, SET, SPOOL, and SYSTEM library commands.

You will also see which system overlays are currently resident in high memory. See the SYSTEM (SYSRES=) library command for details on how to reside these overlays.

D I R

This is the command which allows the examination of a disk directory. Several parameters are allowed to set the type of data that will be displayed. The syntax is:

```
DIR :d (parm,parm,parm)
DIR filespec:d (parm,parm,parm)
DIR partspec with wcc:d (parm,parm,parm)
DIR -partspec with wcc:d (parm,parm,parm)

wcc         WildCard Character <$> used as
            required for masking characters.

:d          Optional drive specification.

Allowable parameters are as follows:

A       Display the directory in full Allocation format.

INV     Display the Invisible files.

MOD     Display Modified files.

N       Non-stop display mode (will not pause after
        each 15 lines). Assumed if "P" is selected.

P       Direct output to the Printer.

SYS     Display the System files.

DATE=   "M1/D1/Y1-M2/D2/Y2" will display those files
        whose mod dates fall between the two dates
        specified, inclusive.

        "M1/D1/Y1" will display those files whose mod
        dates are equal to the date specified.

        "-M1/D1/Y1" will display those files whose mod
        dates are less than or equal to the date
        specified.

        "M1/D1/Y1-" will display those files whose mod
        dates are greater than or equal the date
        specified.

SORT=   the switch YES or NO. YES is the default.

abbr: DATE=D, INV=I, MOD=M, SYS=S YES=Y, NO=N, SORT=O
```

LDOS reserves a certain portion of every disk to keep information about the files and free space available on a disk. This space is called the disk's directory. The maximum number of files a directory can hold, as well as the available free space will be determined by the number of sides, the density, and the number of cylinders on the disk. LDOS will always reserve certain portions of the directory space to store its own operating system files. The following chart shows the maximum number of files available to the user with different disk types.

Size	Density	Sides	User Files Available
5"	Single	1	48
5"	Single	2	128
5"	Double	1	112
5"	Double	2	24Ø
8"	Single	1	96
8"	Single	2	224
8"	Double	1	2Ø8
8"	Double	2	24Ø
5"	Hard		24Ø
8"	Hard		24Ø

There are 16 additional file spaces available in the directory, but are reserved for system (/SYS) files used by LDOS. This will be true even if there are no system files present on the disk.

The maximum storage on a disk is determined by two things - the amount of free space and the number of directory records. Reaching the maximum on either will prevent any more information from being written to the disk. It will be necessary to remove existing files before anything more can be written to that disk. Both the number of remaining files and the free space on a disk can be seen with the FREE library command. Files can be removed with either the KILL or PURGE library command.

There will be three main classifications of files used when discussing a disk's directory. They are SYStem files, INVisible files, and VISible files.

System files contain the instructions used by LDOS to perform most of its basic operations. They are identified by the extension /SYS. These files will not normally be seen when issuing a DIR command.

Invisible files can be any files that you do not wish to normally see with a DIR command. Most LDOS utility files are set to invisible on your master disk. The library command ATTRIB allows changing the visibility of a file.

Visible files are those seen when doing a simple DIR command. These are usually your program and data files.

The reason for having methods of keeping files from being displayed by a simple DIR command is one of readability. It is much easier to find program and data files on a disk if you do not have to search through all the different system and utility filenames. Parameters are provided to allow all files on a disk to be displayed.

Directory parameters

The first parameter discussed will be the drivespec. It is generally entered as a colon followed by the desired drive number. The command DIR :Ø would display the directory of logical drive Ø, and DIR :5 would do the same for drive 5. The command DIR with no drivespec would display the directories of all enabled drives. Specifying a drive that is not enabled will cause an "Illegal drive number" error message to appear. If you are doing a DIR command, you may omit the colon if you are not specifying a filespec or partspec. The command DIR Ø would be the same as DIR :Ø. The parameters to include the system and invisible files are S and I. Visible files will always be included in any display. The command DIR :Ø (I) would display all visible and invisible files on drive Ø, the command DIR :Ø (S) would display all visible and system files, and the command DIR :Ø (I,S) would display all files.

The directory display will normally show files sorted alphabetically in three columns across the screen. A typical display of an LDOS disk done with the command DIR :Ø may appear as follows:

```
Free space=    3.8 K  Drive 1  LDOS-5.1 -- 1Ø/19/81

KI/DVR P             KSM/FLT P            MINIDOS/FLT P
PATCH/CMD P          RDUBL/CMD            PR/FLT P
RS232R/DVR P
```

Including the S and I parameters will show all files in the directory. The command DIR :Ø (I,S) may produce a display such as:

```
Free space=    3.8 K  Drive 1  LDOS-5.1 -- 1Ø/19/81

BACKUP/CMD IP        BOOT/SYS SIP         CMDFILE/CMD IP
DIR/SYS SIP          FORMAT/CMD IP        KI/DVR P
KSM/FLT P            LBASIC/CMD IP        LBASIC/OV1 IP
LBASIC/OV2 IP        LBASIC/OV3 IP        LCOMM/CMD IP
MINIDOS/FLT P        PATCH/CMD P          RDUBL/CMD
PR/FLT P             RS232R/DVR P         RS232T/DVR P
SYSØ/SYS SIP         SYS1/SYS SIP         SYS1Ø/SYS SIP
SYS11/SYS SIP        SYS2/SYS SIP         SYS3/SYS SIP
SYS4/SYS SIP         SYS5/SYS SIP         SYS6/SYS SIP
SYS7/SYS SIP         SYS8/SYS SIP         SYS9/SYS SIP
```

The A parameter will show a disk's directory in allocation format. The command DIR :Ø (I,S,A) may produce a typical display as follows:

```
Filespec Attributes Prot / LRL #Recs / Ext File Space  Mod Date

BACKUP/CMD IP       EXEC / 256    21 /  1  S=    6.2K 1Ø-Sep-81
BOOT/SYS SIP        EXEC / 256     5 /  1  S=    1.2K
CMDFILE/CMD IP +    EXEC / 256    12 /  1  S=    3.8K 12-Sep-81
DIR/SYS SIP         READ / 256    1Ø /  1  S=    2.5K
FORMAT/CMD IP       EXEC / 256    19 /  1  S=    5.ØK 24-Aug-81
KI/DVR P            EXEC / 256     5 /  1  S=    1.2K 19-Oct-81
LBASIC/CMD IP       EXEC / 256    2Ø /  1  S=    5.ØK 19-Oct-81
------------------- ----   ---    --   --  ---------- ---------
aaaaaaaaaaaaaaaaaaa bbbb   ccc    dd   ee  fffffffff ggggggggg
```

As you can see from the previous displays, every directory command shows more than just the filenames. The first line gives the amount of free space on the disk, followed by the drive number, the disk name and the date of creation. The letters P,I, and S, and the plus sign (+) appear after certain filenames. The fields a-g shown in the allocation example describe all information about a directory entry.

aa The first information in this field will be the filename and extension. It may be followed by certain letters or a plus sign, indicating:

- I indicates the file has been declared invisible.
- P indicates the file has an UPDate password.
- S indicates the file is a system file.
- \+ is called the "mod flag", and indicates the file has been modified since it was last backed up.

The letters I, P, S and the mod flag may appear separately or in any combination to show the file's actual status.

bb This field shows the protection level of the file, and can be set or changed with the ATTRIB library command.

cc This is the length of each logical record in the file.

dd This is the number of logical records in the file.

ee This is the number of extents (non-contiguous blocks of space) in which the file is stored.

ff This is the amount of space in K (1K = 1Ø24 bytes) that the file takes up on the disk. If the file has been created with the CREATE library command, the "S=" will appear as "S:".

gg This is the date that the file was created or last written to. If you have used the SYSTEM library command to disable the initial DATE prompt when powering up the system, this date cannot be established or updated. If date is not set and you write to a dated file, plus signs will be inserted into the date field, producing a date such as 1Ø+Oct+81. It is strongly recommended that the initial DATE prompt never be disabled, as a file's date can be used in many different ways. LDOS 5.1 can use dates between Ø1/Ø1/8Ø and 12/31/87.

The directory display will normally go to the video display (the *DO device). It will automatically pause after every 15 lines of display. Pressing <BREAK> will terminate the display, while pressing any other key will continue with another 15 lines. The display may be made to scroll without pause if the N parameter is specified in the DIR command. If the DIR command is executed from a JCL file, the N parameter will automatically be set. You may use the <SHIFT><@> keys to pause the display if this is the case.

The P parameter will send the display to the line printer (*PR device) as well as the video. The P parameter will automatically set the N parameter, and will print the entire directory without pause.

The DATE or D parameter is used to view files whose mod date match a certain date or fall within a specified range of dates. The current LDOS release requires dates to be within the range Ø1/Ø1/8Ø to 12/31/87. The PURGE library command and the BACKUP utility also have provisions for mass manipulation of files dependent upon their dates.

The MOD or M parameter is used to display only those files that have been modified since the last backup.

The directory display will normally be shown sorted in alphabetical order. To disable this feature, specify SORT=NO as a parameter when issuing a DIR command. The PURGE library command and the BACKUP utility access files in their unsorted order. You may see the same order of unsorted access by specifying the SORT=NO parameter in a DIR command.

Using filespecs and partspecs

Along with the previous parameters, LDOS provides other methods for locating files in a disk directory. Three terms will be used when discussing these parameters - "filespec", "partspec", and "wcc" (WildCard Character). Filespec refers to a file's name and extension. For example, the filespec BACKUP/CMD has the filename BACKUP and the extension /CMD. A partspec would be any part or parts of a filespec. Wcc means a special symbol (the dollar sign "$") used in place of characters in a filespec or partspec.

For example, a command using a partspec is:

```
DIR /CMD:Ø
```

This would show only visible files with the extension /CMD on drive :Ø. You can always include any of the A, I, N, P, S, DATE, or SORT parameters whenever using any filespec, partspec, or wcc.

You may use a filename, a file extension, or both together in any DIR command. It is not necessary to use the complete name or extension. The wcc mask character ($) can be used to mask out certain groups of characters when using a filespec or partspec. Using a partial filename or extension provides the opposite function of using a wcc. Refer to the following:

Using a partial filename will display all files whose name starts with those characters, regardless of how many other characters follow. The commands:

```
DIR BA:Ø
DIR BACK:Ø
DIR BA/C
DIR BACK/CMD
```

would all display the file BACKUP/CMD, although any other files matching the partspecs would also be displayed.

The wcc mask ($) is used to mask out leading characters in a filename or extension. The commands:

```
DIR $$$$UP:Ø
DIR $$CK:Ø
DIR BACK/$$D:Ø
```

would again all display the file BACKUP/CMD, along with any other files that match the criteria. Using wcc's after a partspec will have no effect on the command. All files that meet the specified leading criteria will be displayed, regardless of the number of other characters in the filename or extension. A wcc may also be used in the middle of a partspec. For example, the commands:

```
DIR B$CK:Ø
DIR B$$$$P:Ø
DIR BA/C$D:Ø
```

would all display the file BACKUP/CMD, along with other matching files.

Using -filespecs and -partspecs

Entering the "not" symbol (the minus sign) in front of a filespec or partspec declares it to be a "not filespec" or "not partspec". The -specs are used to exclude files from a directory display. The same rules concerning filespecs, partspecs and the wcc mask apply exactly the same for -specs as for normal file and part specs. For example, the commands:

```
DIR -BA:Ø
DIR -B$$K:Ø
DIR -/CMD:Ø
DIR -/$$D:Ø
```

would show all files on drive Ø except for BACKUP/CMD, and any other files that match the -spec criteria.

D O

The DO command executes a user created JCL (Job Control Language) file. The syntax is:

```
DO  character  filespec (@LABEL,parm,parm...);

character is an optional DO control character $, =, *

filespec  is a valid filespec - default extension /JCL.

@LABEL    is an optional LABEL indicating a start
          point in the JCL file.

parm      optional parameter(s) to be passed to the
          filespec (JCL program) during execution.

;         is the optional semi-colon. When used, allows
          a DO command line greater than 64 characters.

abbr: NONE
```

NOTE: Please refer to the Job Control Language section of the manual for the creation of a JCL file.

The DO command will compile and/or execute a series of commands that have been created by the user and stored in an ASCII disk file. The default file extension of the filespec is /JCL. No line in a JCL file may exceed 63 characters in length. The DO command will also pass optional parameters and variables to the program being done.

The DO function is normally a two step operation - the compile phase and the execute phase. During the compile, a line is read from the specified file and then written to a file named SYSTEM/JCL. If this file does not exist, it will be placed on the first available drive. Once the line is compiled, it is then executed directly from the SYSTEM/JCL file. There must be at least one available (enabled and not write protected) drive in the system to compile and execute a JCL file. However, an execute only option is available with a DO control character, and will be explained later.

Please note that the occurrence of any error will terminate the DO execution. The <BREAK> key, if not disabled, will allow you to manually abort the DO.

The three control characters ($, =, *) will change the compile and execution phases of the DO command. When using these characters, a space character is mandatory between the word DO and the character. If the space is omitted, the character will be ignored. Note that if no character is specified, both the compile and execution will be done.

The @LABEL parameter will allow you to create JCL files with multiple entry points. Each entry point can indicate a different location at which processing will begin. NOTE: If the @LABEL function is used, the compile phase must be done or the DO will abort with an error message. If the @LABEL parameter is specified, the JCL file will be scanned WITHOUT execution up to the specified LABEL. Once the LABEL is reached, execution will begin and continue until the next LABEL, or until the end of the JCL procedure/file has been reached. The primary reason for the @LABEL parameter is to allow many different functions to be built into one large file. This will greatly conserve disk space, as a series of small JCL files would take up a minimum of 1 granule apiece. For complete definitions of JCL LABELs, refer to the JCL section of the manual.

If the @LABEL and parameters cause the DO command line to exceed 64 characters, the semi-colon character (;) will allow you to continue passing parameters once the DO has started. The proper use of the semi-colon is as follows:

1) Terminate the DO command line by enclosing as many parameters as you can in the parentheses. Close the parentheses, then insert the semi-colon character and press <ENTER>.

2) A question mark will appear on the screen. At this point, you may enter the remaining parameters, making sure they are enclosed in parentheses.

Refer to the following examples and descriptions as a guide to the uses of the DO function.

CHARACTER: $

The $ character will DO the compile phase only, without actually executing the commands. The DO will compile your JCL file to the SYSTEM/JCL file. This will test if the syntax of a new JCL file will compile properly. Use the LIST library command to examine the SYSTEM/JCL file to see the resultant JCL lines that will be executed.

CHARACTER: =

The = character will skip the compile phase and directly execute your JCL file. Be aware that some of the JCL features will be ignored if the compile phase is skipped. Refer to the Job Control Language section of the manual for a complete list of these features and limitations.

CHARACTER: *

The * character will rerun the last DO command that was compiled, by using the existing SYSTEM/JCL file. If this file does not exist, nothing will be done and an error message will be generated.

DO DRIVE/JCL

This command will compile and execute a file named DRIVE/JCL. The system will search the drives for a file named DRIVE/JCL and compile it to a file named SYSTEM/JCL. After it has been compiled, the resultant SYSTEM/JCL file will be executed.

DO = DRIVE/JCL

This command will execute the file DRIVE/JCL without compiling it to the SYSTEM/JCL file.

DO $ DRIVE

This command will compile the file DRIVE/JCL to the SYSTEM/JCL file. The file will not be executed. Note that the filespec DRIVE will use the default extension of /JCL.

DO MY/JCL:Ø (@THIRD)

This command will compile and execute the program MY/JCL. All instructions in the program will be ignored up to the LABEL (@THIRD). Compilation will begin at the line following the label and will continue until the next LABEL or End of File is reached.

DO *

This command will execute the SYSTEM/JCL file. If the file does not exist, an error will be generated.

DO TEST/NEW:2 (D=5,E=6)

This command will compile and execute the file TEST/NEW on drive 2. The file will be compiled to the SYSTEM/JCL file and each line will be executed from this file. The variables D=5 and E=6 will be passed as needed during the compilation.

The following examples show what will happen if the space is omitted in a DO command.

DO=TEST/JCL

The use of the = character normally tells the DO command to skip the compile phase and directly execute each line of the JCL file. If the space between the DO command and the = is omitted, the compile phase WILL BE DONE! This means that the TEST/JCL file will compile to the SYSTEM/JCL (creating the SYSTEM/JCL file if none exists).

DO$TEST/JCL

The $ character normally tells the DO to compile the TEST/JCL file without executing it. If the space between DO and the $ character is omitted, the execution WILL BE DONE!

DO*

This command will ignore the asterisk (*) and generate the error message FILE SPEC REQUIRED!

* D U M P

This command DUMPs a specified block of memory to a disk file. The dump may be in load module or ASCII format. The syntax is:

```
DUMP filespec (START=,END=,TRA=,ASCII,ETX=)

          filespec is any valid filespec

START=  is the starting address of the memory block

END=    is the ending address of the memory block.

TRA=    is the transfer address or execution point.

ASCII   is an optional parameter for an ASCII DUMP.

ETX=    optional End of Text marker for ASCII DUMPS.

abbr: START=S, END=E, TRA=T, ASCII=A
```

The DUMP command writes an exact image of the specified memory locations to a disk file in load module or ASCII format. The default file extensions are /CIM for non-ASCII dumps, and /TXT for ASCII dumps.

The following restrictions are placed on the DUMP command addresses.

START= The memory block must start above address X'55ØØ'.

END= The ending address must be greater than or equal to the starting address.

TRA= The transfer address may be any valid address. If not specified, the transfer address (TRA) will be back to the system.

Addresses may be entered in either decimal or hexadecimal format. Hex addresses must be in the form X'aaaa'.

The ASCII and ETX parameters are used to dump memory to a pure ASCII file. Address loading information is not present in the file and the file cannot be loaded by the system loader. The file is identical to the file structure of most word processor systems such as SCRIPSIT. Following the last dump character, an End of Text (ETX) character is written. This character is normally an X'Ø3', but may be changed with the ETX parameter to a character of your choice. For example, a Scripsit file will normally have an X'ØØ' as the ETX character.

Here are some examples of using the DUMP command.

```
DUMP ROUTINE/CMD (START=X'7ØØØ',END=X'8ØØØ',TRA=X'7ØØØ')
DUMP ROUTINE/CMD (S=X'7ØØØ',E=X'8ØØØ',T=X'7ØØØ')
DUMP ROUTINE/CMD (S=28672,E=32768,T=28672)
```

These three commands will create identical files. The first two use hex notation for the addresses, while the third is in decimal format. The results of these commands will be to dump the area of memory starting at X'7ØØØ' and ending at X'8ØØØ'. This block of memory will be written to a disk file named ROUTINE/CMD. If the file already exists it will be overwritten. If it does not exist, it will be created on the first available drive. The transfer address of the program will be X'7ØØØ'.

```
DUMP TEST:1 (S=X'9ØØØ',E=X'BCØF')
```

This command will DUMP the specified block of memory to a disk file named TEST/CIM on drive 1. Since the file extension was not specified, it defaulted to /CIM. The transfer address was not specified and will be written to the file as a return to the system.

```
DUMP WORD/TXT:Ø (S=X'7ØØØ',E=X'AØØØ',A)
```

This command will dump the specified memory range to a disk file named WORD/TXT. Since the A (ASCII) option was specified, no load module information will be written to the file, and the EXT (End of Text) character will be the normal X'Ø3'.

```
DUMP WORD:Ø (S=X'7ØØØ',E=X'AØØØ',ETX=X'FF',A)
```

This command is identical to the last one except that the End of Text marker will be written as an X'FF'. In addition, the file's extension was not specified and will default to /TXT.

F I L T E R

The FILTER command establishes a program to filter (modify) the I/O path of a specified device. The syntax is:

```
FILTER devspec USING filespec (parm,parm,...)

devspec     any valid LDOS device

filespec    the filespec of a FILTER program, with the
            default extension being /FLT.

parm        optional parameters for the filespec program

abbr: NONE (except as allowed by the filespec program)
```

The FILTER library command is used to filter or modify data as it passes between the specified device and its driver program. LDOS is structured so that any device may be easily filtered to provide modification of standard I/O paths. There are several filter programs provided on your LDOS disk, and are listed in the Table of Contents, DEVICE DRIVER/FILTERS section. These files all have the extension /FLT.

You will find that filter programs are usually written to provide other than "standard" functions for available devices. This ability is provided since the standard ROM device driver programs may not meet your particular needs. LDOS provides many different filter programs for many different devices, and easily allows the user to either write filters or have them written for him. Documentation on the writing of filter programs will be found in the Technical Information of the manual, FILES AND FILTERS section. Any programmer familiar with Z-8Ø assembly language should be able to construct a filter program after examining the documentation in the Technical Information, where several actual filter programs are shown. Hopefully, this will enable most users to write or have written specific filters to meet the needs of their applications programs. If a totally new driver program is needed, you may wish to consult the SET library command section.

A filter program can provide many useful functions during I/O processing. Lines and/or characters could be counted, with certain actions taking place when pre-set limits are reached. Character conversions could be performed, such as simply changing each linefeed to a null, or a complete conversion from ASCII (normal TRS-8Ø character set) to EBCDIC (IBM character set). Keyboard entries may be intercepted and acted upon, as is done in the KSM and MiniDOS filters.

Several filter programs are provided on your LDOS master diskette. This example shows the use of the FILTER command with the LDOS filter program PR/FLT (the Printer filter).

```
FILTER *PR USING PR/FLT (CHARS=8Ø,INDENT=6)
FILTER *PR PR (C=8Ø,I=6)
```

These two filter commands will produce identical results. Note that the use of the word USING is optional. Also, the default extension for the filespec is /FLT. This example will filter I/O directed to the line printer through the PR/FLT program, described in the DEVICE DRIVER/FILTER section of the manual. As a result of this filter routine, printed output will be limited to 8Ø characters per physical line.

Also, any single line which is greater than 8Ø characters in length will wrap around, and be indented 6 spaces on the next line. NOTE: These parameters are determined totally by the PR/FLT program, not the FILTER command.

Another filter routine provided on your LDOS diskette is called KSM/FLT. It provides the KeyStroke Multiply feature of LDOS.

```
FILTER *KI USING KSM/FLT USING MYKEYS/KSM
FILTER *KI KSM MYKEYS
```

The above examples would produce identical results, and are illustrations of how to establish a KSM filter program. The KSM feature will now be enabled, and would use the file MYKEYS/KSM to provide the KSM phrases. From the example, you should see that the filtered device would be *KI (the keyboard), and the filter program used would be called KSM/FLT. For complete information on the LDOS KSM feature, refer to the DEVICE DRIVER/FILTER section of the manual.

```
FILTER *CL REMLF
```

This command would filter the *CL device's I/O using the filter routine found in a user developed filter program called REMLF/FLT. For example, if *CL had been set with an RS-232 driver, this command would filter I/O to and from the RS-232 interface. From the name of the program it may be assumed that the filter program may do something such as removing a linefeed after a carriage return.

* F R E E

This command will show the used and available space and files on each disk in the system, or display a space map of a disk drive. The syntax is:

```
FREE (P)
FREE :d (P)

:d      an optional drivespec, specifying a free space
        map of a specified floppy drive.

P       an optional parameter that directs output to
        the printer as well as the video display.

abbr:  NONE
```

To execute the free command simply type FREE at the LDOS Ready prompt. LDOS will respond with a display similar to this:

```
Drive Ø - LDOS-5.1 11/15/81  Files= 97/128, Space=  87/  18Ø K
Drive 1 - LDOS-5.Ø Ø6/Ø1/81  Files= 39/ 64, Space=  19/   88 K
-------   -------- --------  --------- ---  -----------  ------
aaaaaaa   bbbbbbbb cccccccc  ddddddddd eee  fffffffffff  ggggggg
```

Several fields are displayed in each line, representing the free (unused) space information about one diskette. The information given in each of the fields is:

aa This field shows the drive number that the rest of the information in the line will pertain to.

bb This field shows the name of the disk, established with the FORMAT utility program.

cc This field shows the date of creation or the date of the last Mirror Image backup to the diskette.

dd This shows the number of directory entries that are available for use (number of files that may be added).

ee This shows the total number of directory entries that the disk will support.

ff This shows the amount of free space in "K" (1Ø24 byte blocks) that remains available for use on the disk.

gg This shows the total amount of space the disk will support in "K".

The FREE library command without drivespec is global in its nature. It will search all active drives, and may not be confined to a single drive. The free space available on a diskette is also shown in a DIR library command.

Using the FREE command with a drivespec will bring up a free space map as shown below.

FREE :Ø

```
Drive => Ø     Size => 5"     Heads => 1     Density => DOUBLE
-------------------------------------------------------------
 Ø-  5   x..      xxx      xx.      ...      xxx      xxx
 6- 11   xxx      ...      xx.      ...      ***      ***
12- 17   ...      ...      ...      ...      ...      ...
18- 23   x..      ...      DDD      xxx      xxx      ...
24- 29   ...      ...      ...      ...      ...      ...
3Ø- 35   ...      ...      ...      ...      xx.      ...
36- 39   ...      ...      ...      ...
Free =>  128 K /   85 G / Name => LDOSDISK   Date =>  Ø2/25/82
```

In this example, the disk used was a 4Ø track single sided, double density, 5" floppy diskette. The top line will display information about the diskette size and type. The bottom line will show the amount of free space in both grans and K (1Ø24 bytes), along with the disk name and date of creation.

The inner display area contains the details of the space allocation on the disk. The numbers on the left represent the cylinders. The grans per cylinder will be shown across each line, with 6 cylinders per line. This disk has 3 grans per cylinder, as it is a 5" double density disk. The grans per cylinder will vary according to diskette size, density, and number of sides.

A gran will be represented as one of 4 characters, explained here.

. (period) - will represent an unused gran.

* - will represent a locked out gran (floppy disks only).

X - will represent a used gran.

D - will represent a gran used for the Directory.

If the drive has more than 7Ø cylinders, you must press <ENTER> to return to the LDOS Ready prompt. This will prevent the top line of the display from scrolling off the screen.

This display may also be sent to *PR (your printer) by using the (P) parameter.

K I L L

This is used to delete a specified file or device from the system. The syntax is:

```
KILL filespec
KILL devspec

no parameters are required

abbr: NONE
```

The KILL command serves two purposes. It removes unwanted files from a diskette, thereby freeing up the space previously allocated to that file. It also removes devices that are no longer needed from the device table.

Killing files

To kill a file, type in the following command at the LDOS Ready prompt:

KILL filespec

If the file is password protected, you must supply the proper password or the file will not be killed. Passwords for all files on your LDOS master diskette may be found in Section I, GENERAL INFORMATION. To deal with killing several files it is often easier to use the PURGE library command, which in effect is a "controlled" mass-kill. This may be the case if the files to be killed contain a common filename or extension, as the PURGE command can deal with these files as a group. The PURGE command also ignores all file passwords as long as you know the master password of the disk.

Here are some examples of KILLing files:

KILL ALPHA/DAT:Ø

This command will KILL the file named ALPHA/DAT that is present on drive Ø. After execution of this command the file and the data in it will no longer be accessible to the system, so KILL carefully.

KILL DELTA/DAT

This command will KILL the file DELTA/DAT on the first drive that it is on. Be careful! Without a drivespec you could KILL a file that you did not intend to.

KILL MIDWEST/DAT.SECRET:Ø

This command will KILL the password protected file MIDWEST/DAT.SECRET on drive Ø, as long as SECRET is the proper password. If the file's attributes include a protection level of NAME or higher, SECRET must be the update password to kill the file. If the proper password is not supplied, an error message will be displayed and the file will not be killed.

Killing devices

The LDOS system does not permit the killing of certain devices referred to as system devices. These devices are *JL, *KI, *DO, and *PR. Attempting to kill these devices will produce an error message, and the kill will abort.

However, any other device may be killed, as long as it is pointed NIL. The status of a device may be seen by issuing a DEVICE library command. If a device is not pointed NIL, it must first be reset with the RESET library command before it can be killed. The command 'KILL devspec' will result in the devspec being completely removed from system device space.

Following are some examples of killing a device:

```
KILL *CL
```

This command will in effect make *CL (the Comm Line) disappear from the system device control table, assuming the *CL was reset or pointed NIL before the kill was done.

```
KILL *SI
KILL *SO
```

These commands will remove *SI and *SO from the device table. These two devices will appear pointed NIL in the device table upon power up. They are currently unused by LDOS, and may be killed if desired.

L I B

This command will display the LDOS command libraries. The syntax is:

```
+-------------------------------------------------------------+
|  LIB                                                        |
|                                                             |
|  no parameters are required                                 |
|                                                             |
|  abbr: NONE                                                 |
|                                                             |
+-------------------------------------------------------------+
```

After execution of this command, the LDOS command libraries will be displayed as shown below.

```
LIBRARY <A>

APPEND      COPY        DEVICE      DIR
DO          FILTER      KILL        LIB
LINK        LIST        LOAD        MEMORY
RENAME      RESET       ROUTE       RUN
SET         SPOOL


LIBRARY <B>

ATTRIB      AUTO        BOOT        BUILD
CLOCK       CREATE      DATE        DEBUG
DUMP        FREE        PURGE       SYSTEM
TIME        TRACE       VERIFY
```

Library <A> is the primary LDOS command library, and is located in the SYS6/SYS system module. Library <B> is the secondary command library, and is located in the SYS7/SYS system module. You may delete either system module containing the libraries if the commands included in it will not be used.

The secondary library commands are indicated throughout this manual by an asterisk preceding the command. This asterisk can be seen in the command name directly above the page numbers, and on the first page of each library command.

L I N K

This command links together multiple logical I/O devices. The syntax is:

```
LINK devspec1 TO devspec2

devspec is any currently enabled logical device

abbr: NONE
```

This command is used to link together two logical devices. Both devices must be currently enabled. Once linked, any output sent to devspec1 will also be sent to devspec2. Any input requested from devspec1 may also be supplied by devspec2.

The user is cautioned about making multiple links to the same device(s), as it is possible to create endless loops and lock up the system.

The order of the devices in the link command line is important, since output to devspec2 will not be sent to devspec1, nor can input requested from devspec2 be supplied by devspec1. Also, using the ROUTE library command on devspec1 will destroy its link to devspec2, but routing devspec2 is perfectly acceptable.

Once linked, devices can be un-linked by the command 'RESET devspec'. A global RESET or a reboot will also un-link devices. See the RESET and BOOT commands for further information.

Following are some examples of the use of LINK.

LINK *DO *PR

This command will link the video display to the line printer. All output sent to the display (devspec1) will also be sent to the line printer (devspec2). Once linked, the line printer must be enabled if it is physically hooked to the system (i.e. the cable is connected to both the printer connector and the printer). If the printer becomes de-selected or faults (out of paper, etc.) the system will hang up. Remember that both linked devices must be enabled. Note that any output sent individually to the printer, such as an LPRINT from Basic, will not be shown on the video display.

LINK *PR *DO

This command will link the line printer to the video display. All output sent to the printer will also be sent to the video display. The line printer must be on line and enabled if any printing is to be done. This link will not send any characters from the video to the line printer.

Although files may not be directly linked to a device, it is still possible to accomplish the same results through the use of "phantom" devices. This example will show how to accomplish a devspec to filespec link.

Suppose you wish to link your line printer to a disk file named PRINT/TXT on your drive Ø diskette. Follow the steps below.

STEP 1) A "phantom" device must be created. For this example we will create a device named *DU. To do this, use the ROUTE command in the following manner:

```
ROUTE *DU TO PRINT/TXT:Ø
```

This will create a device named *DU and ROUTE it to a disk file named PRINT/TXT on drive Ø. If the file does not exist, it will be created and dynamically expanded as needed. If the file already exists, any data sent to the file will be appended onto the end of the file.

STEP 2) The printer can now be linked to the file with the following command:

```
LINK *PR *DU
```

The printer is now linked to the device *DU, which in turn is routed to the disk file PRINT/TXT. All output sent to the line printer will also be sent to the device *DU (in effect, written to the disk file PRINT/TXT).

Please note that the file PRINT/TXT will remain open until a RESET *DU is done. If you wish to break the link between the printer and the file without closing the file, then use the command RESET *PR. For further information, please refer to the ROUTE and RESET commands.

Do not use the SYSTEM (SYSGEN) command if you linked a device to a file. The file will be shown as being open every time you power up or boot the system. You could very easily overwrite other files if you happened to switch disks with the file open.

L I S T

The LIST command will send a listing of a file to the video display or line printer. The syntax is:

```
LIST filespec (parm,parm,...,parm)

ASCII8 Will allow full 8-bit output

NUM    Sets line numbering mode for ASCII text.

HEX    Sets hexadecimal output format.

TAB    Sets tab expansion for ASCII text.

P      Directs output to the line printer.

LINE=  LINE in text file where ASCII list is to begin.

REC=   Record number where hex list is to begin.

LRL=   The Logical Record Length to be used to display
       a file when in the hex mode.

abbr: ASCII8=A8, NUM=N, HEX=H, TAB=T, REC=R, LRL=L
```

All parameters shown after the filespec are optional and need not be used. If no parameters are specified, the LIST command will list the file in ASCII format, and the logical record length (LRL) of the file will be read from the directory. Normal ASCII format will strip the high bit from each character, in effect displaying only those characters in the range X'ØØ' to X'7F'. The ASCII8 parameter may be used to see all characters, including graphics characters.

The parameters shown may be entered in the same command line, such as

LIST TESTFILE:Ø (HEX,REC=5,LRL=8Ø,P).

If an extension is not used in the filespec, a default of /TXT will be used. If no file with the /TXT extension is found, LIST will search for a file with an extension of all blanks.

Here are some examples of how LIST handles the "default" file extension of /TXT.

LIST TESTFILE:Ø

The system will first search drive Ø for a file named TESTFILE/TXT. If not found, it will then search for a file named TESTFILE.

LIST TESTFILE

The system will search all active drives for a file called TESTFILE/TXT, and list the first file named TESTFILE/TXT it encounters. If this file is not found, it will search all active drives for a file named TESTFILE, again listing the first TESTFILE it encounters.

LIST TESTFILE/SCR

The system will search all drives for a file called TESTFILE/SCR and list the first file named TESTFILE/SCR encountered. If the file is not found, the LIST command will not search for TESTFILE/TXT.

The parameters of the LIST command will determine the output format of the information in the listed file. Refer to the following section for a complete explanation and the proper use of these parameters. Note that the NUM and LINE parameters are for ASCII listings only and will be ignored if the HEX parameter is specified.

PARAMETER: ASCII8

This parameter will allow all 8 bits of each character to be displayed during an ASCII list. Normally, any character above X'7F' (decimal 127) will have the high bit reset. The ASCII8 parameter will be useful if you wish to see graphics characters in a file.

PARAMETER: NUM

This parameter will number the lines of the file as they are sent to the video display or printer during an ASCII list. Line numbers will start with one (1) and be in the format ØØØØ1. Lines are determined by the occurrence of a carriage return. Linefeed characters will not generate a new line number.

PARAMETER: LINE

(This parameter may NOT be abbreviated). The LINE parameter is used with ASCII files. It will start the listing with the specified line of the file. Lines are determined by the occurrence of a carriage return character in the file. An example of the proper syntax would be LIST TESTFILE/TXT (LINE=14). This would list the ASCII file TESTFILE/TXT, starting with the line of the file after the 13th carriage return.

IMPORTANT: The NUM and LINE parameters will always be ignored if the HEX parameter is specified.

PARAMETER: HEX

The HEX parameter will cause the file to be listed in the following format.

```
aaaa:bb = cc cc cc cc cc cc cc cc  cc cc cc cc cc cc cc cc
          d  d  d  d  d  d  d  d   d  d  d  d  d  d  d  d
```

aaaa represents the current logical record of the file in hex notation, starting with record Ø.

bb represents the offset from the first byte of the current logical record (bb will be in hexadecimal notation).

cc will be the hexadecimal representation of the byte listed.

d will be the ASCII representation of the byte. A period (.) will be used for all non-displayable bytes.

For example, the command LIST LBASIC/CMD.BASIC (H) would produce a display as shown here:

```
ØØØØ:ØØ = Ø5 Ø6 4C 42 41 53 49 43  1F 32 43 6F 7Ø 79 72 69
             .  .  L  B  A  S  I  C   .  2  C  o  p  y  r  i
ØØØØ:1Ø = 67 68 74 2Ø 28 43 29 2Ø  31 39 38 31 2Ø 62 79 2Ø
           g  h  t     (  C  )     1  9  8  1     b  y
```

This is a listing of the file LBASIC/CMD.BASIC in hex format. The logical record length was not specified in the command, and was found from the directory to be 256.

PARAMETER: REC

This parameter is used for listing hex files. It is entered as a decimal value, and tells the LIST command to start with the specified logical record number of the file. The first record in a file is record Ø. When specifying a record number, REC=1 would list the second record of the file. The command LIST MONITOR/CMD (H,R=5) would start the listing with the sixth record. If this parameter is not specified, the listing will start with record Ø.

PARAMETER: LRL

This parameter tells the LIST command to format the output using the specified LRL for each record. If the LRL parameter is not specified, the LIST command will use the record length in the file's directory entry. The LRL parameter is valid only when used with the HEX parameter.

PARAMETER: P

This parameter directs the output to the line printer rather than the video display. It may be used in conjunction with any of the other LIST parameters. Be sure the printer is enabled before using this command or the system may lock up.

PARAMETER: TAB

This parameter will cause the expansion of any TAB characters (X'Ø9') encountered during ASCII listings to the video display or line printer. The tab locations are at columns 8, 16, 24, 32, 4Ø, 48, and 56.

The following examples will show some different LIST commands.

```
LIST MONITOR/CMD (HEX,LRL=8,REC=Ø)
LIST MONITOR/CMD (H,L=8)
```

These two commands will produce identical results - listing a file called MONITOR/CMD to the video display, using a LRL of 8, and starting with the first record of the file. The second example has merely substituted the abbreviations for the HEX and LRL parameters, and let the REC parameter default to Ø. This listing display will be only 8 bytes wide, as the LRL is also the display width (for LRL's = 1 to 16). Maximum display width is 16 bytes per line. The same line width applies to listings sent to the printer with the (P) option.

```
LIST REPLY/PCL (NUM,TAB,P)
LIST REPLY/PCL (N,T,P)
```

These two commands produce identical results. The second example merely substitutes abbreviations for the parameters. The result of this command would be to send a listing of the file REPLY/PCL to the printer, using ASCII format, expanding all tab characters encountered and numbering each new line that is printed.

```
LIST TESTFILE/OBJ (NUM,HEX,REC=5)
LIST TESTFILE/OBJ (HEX,REC=5)
```

These commands produce identical listings of the file TESTFILE/OBJ. Remember that the NUM and LINE parameters are always superceded by the HEX parameter.

L O A D

The LOAD command will load a load module format file (such as a /CMD or a /CIM) into memory without execution. The syntax is:

```
LOAD (X) filespec

filespec is any valid LDOS filespec that is in load
module format.

(X) is an optional parameter for a LOAD from a
non-system diskette.

abbr: NONE
```

The LOAD command allows you to load into memory a disk file that is in the proper format. The default file extension for the LOAD command is /CMD.

The following address restrictions exist when loading programs:

LOAD Program must reside at or above X'52ØØ'.

LOAD (X) Program must reside at or above X'53ØØ'.

After a program is loaded, control is returned to the system without execution of the loaded program.

The (X) parameter allows the loading of files that reside on a system or non-system disk. This is primarily useful for single drive owners, as a file may be loaded from a disk other than an LDOS system disk. The system will prompt you to insert the diskette with the desired file on it with the message:

INSERT SOURCE DISK <ENTER>

After the load is complete, you will be prompted to place the system diskette back in drive Ø with the message:

INSERT SYSTEM DISK <ENTER>

At this point, the load is complete.

M E M O R Y

The MEMORY command allows you to reserve a portion of memory, see the current HIGH$ (highest byte of unused memory), modify a memory address, or jump to a specified memory location. The syntax is:

```
MEMORY (HIGH=addr,ADD=addr,WORD=dddd,BYTE=dd,GO=addr)
MEMORY (CLEAR)

CLEAR   Will fill memory from X'52ØØ' to HIGH$ with
        X'ØØ'.

HIGH=   Will set the specified address as HIGH$.
        addr must be less than the current HIGH$.

ADD=    Displays the word at the specified address.
        Also specifies the address for WORD and BYTE.

WORD=   Changes the contents of ADD and ADD+1.

BYTE=   Changes the contents of ADD.

GO=     Transfers control to the specified address.
        This parameter is always executed last.


addr    Any address in hex or decimal notation.

dddd    Any hex "word" other than X'ØØØØ'.

dd      Any byte in hex notation, other than X'FF'.


abbr: HIGH=H, ADD=A, WORD=W, BYTE=B, CLEAR=C
```

In all MEMORY commands, the GO parameter, if used, will be the last parameter to be executed, regardless of its physical position in the command line. All other parameters will be acted upon before the actual GO is done.

The following restrictions are placed on the WORD and BYTE parameters:

WORD: Cannot = X'ØØØØ' or decimal value Ø.
BYTE: Cannot = X'FF' or decimal value 255.

Refer to the following examples and descriptions.

MEMORY with no parameters will display HIGH$ (highest unused memory location) in the format X'nnnn'.

MEMORY (HIGH=X'EØØØ')

This command would set HIGH$ to memory address X'EØØØ', as long as the existing HIGH$ was above this location. The MEMORY command will only move HIGH$ lower in memory. The RESET command will allow you to RESET HIGH$ to the top of memory.

MEMORY (ADD=X'4Ø49')

This command will display the contents of memory locations X'4Ø49' and X'4Ø4A'. The display will be in the following format:

```
X'4Ø49' = 16457 (X'ØØEØ')  HIGH = X'EØØØ'
  ----    -----    ----          ----
  aaaa    bbbbb    cccc          dddd
```

aaaa...is the address specified in hex notation.
bbbb...is the decimal equivalent of ADD.
cccc...is the contents of address and address +1, in LO-HI format.
dddd...is the current HIGH$ address.

MEMORY (ADD=X'E1ØØ',WORD=X'3EØA')

This command will modify memory locations ADD (X'E1ØØ') and ADD+1 (X'E1Ø1'), changing them to the value of WORD. The following would be displayed after this command:

```
X'E1ØØ' = 576ØØ (X'ØØØØ' => X'3EØA')  High = X'EØØØ'
  ----    -----    ----       ----          ----
  aaaa    bbbbb    cccc       XXXX          dddd
```

All of the display is identical to the last example, except the contents of the WORD changed will be shown, represented in the display as XXXX.

MEMORY (ADD=X'E1ØØ',BYTE=X'ØD')

This command will change the BYTE of memory at the specified address (X'E1ØØ') to X'ØD'. The display after executing this command would be:

```
X'E1ØØ' = 576ØØ (X'ØØØØ' => X'ØDØØ')  High = X'EØØØ'
  ----    -----    ----       --            ----
  aaaa    bbbbb    cccc       XX            dddd
```

All of the display is identical to the last example, except for the modified BYTE change shown here with the XX.

MEMORY (GO=X'EØØØ')

This command would transfer control to memory address X'EØØØ'.
Note that the GO parameter may not be abbreviated.

* P U R G E

The PURGE command allows for controlled multiple kills of disk files. The syntax is:

```
PURGE :d
PURGE :d (parm,parm,parm)
PURGE partspec w/wcc:d (parm,parm,parm)
PURGE -partspec w/wcc:d (parm,parm,parm)

:d          is the mandatory drivespec.

   partspec  and -partspec are as described in the
   LDOS glossary and under general information.

wcc         Wild-Card Character <$> used as necessary
            for masking characters.

    the allowable parameters are as follows:

QUERY=      ON or OFF. The default is ON.

MPW=        The disk master password.

INV         Specifies Invisible files.

SYS         Specifies System files.


DATE=       represents a date entry as follows:

      "M1/D1/Y1-M2/D2/Y2" indicates all files with mod
      dates between the two dates specified, inclusive.

      "M1/D1/Y1" will indicate all files with a mod
      date equal to the date specified.

      "-M1/D1/Y1" will indicate all files with mod dates
      less than or equal to the date specified.

      "M1/D1/Y1-" indicates all files with mod dates
      greater than or equal to the date specified.

abbr: DATE=D, INV=I, ON=Y, OFF=N, QUERY=Q, SYS=S
```

The PURGE command allows the user to kill multiple disk files without the need to specify the individual filespecs. The user will be prompted for the disk's master password if it is a password other than "PASSWORD" and not passed with the MPW parameter. The PURGE command allows several parameters to be set, providing for a specific group or groups of files to be purged.

If the Q (Query) parameter is not specified, or if Q is specified without a switch, Q=Y is automatically assumed, and you will be asked before each file is purged. The responses are <Y> to purge the file and <N> to skip it. Pressing <ENTER> will also skip the file. The mod flag and date will be shown for each file.

PURGE defaults to visible files only. To include invisible and system files, the I and S switches must be specified in the command line.

The D switch allows you to choose a range of mod dates to be use as criteria for the purge. Only those files meeting the date range will be used. Files without dates will never be shown if the D switch was specified.

NOTE: The files BOOT/SYS and DIR/SYS are not able to be purged and will never appear during execution of any PURGE command.

Following are some examples and explanations of the PURGE command.

PURGE :Ø (MPW="SECRET")

This command will purge all visible files on drive Ø, assuming that the master password of the disk is SECRET. The purge will show each file and wait until a Y (Yes, purge it) or a N (No, don't purge it) is entered. To abort the purge, press the <BREAK> key at this prompt. If the master password does not match the password of the disk, the purge will abort with an error message.

PURGE :1 (Q=N,I,S)

This is a very POWERFUL and DANGEROUS command. It will purge all files, including system files, from drive 1. If the disk's master password is other than PASSWORD, you will be prompted for it. Once the purge starts, it will remove all files from the disk, except BOOT/SYS and DIR/SYS. YOU WILL NOT BE ASKED BEFORE EACH FILE IS PURGED - IT WILL BE AUTOMATIC!! In other words, you will end up with a blank, formatted disk! This is a very convenient way to clean a disk, but be sure you don't actually need any file on the disk.

PURGE /BAS:1 (Q=N)

This command will first ask for the master password of the disk in drive 1 (if it is not PASSWORD). Once entered, it will purge all visible files with the extension /BAS from the disk. You will not be asked (Y/N) before each file is killed as the QUERY was specified as N, for No QUERY desired. You will, however, be able to stop the purge activity by pressing the <BREAK> key.

PURGE $$EX1:Ø (I)

This command will purge all non-system files whose filename has the characters EX1 as the third, fourth, and fifth characters of the filename. The wildcard character ($) masks the first two characters of the filename (the filename may be more than 5 characters in length, as trailing characters in the field are ignored). The file extension will have no effect on this PURGE command. You will be asked before any file is actually purged, as Q was not specified and defaulted to Q=Y. Invisible and visible files will both be shown, as the I switch was used.

PURGE /$$S:2

This command will purge all visible files on drive 2 whose file extension contains 3 characters and ends in the letter S. This would purge files with the extension of /BAS, for example. However, it would not purge the system files, as the S switch was not specified. You will be prompted before each file is purged.

PURGE -/CMD:Ø (I)

This command will purge all non-system files EXCEPT those whose extension is /CMD. You will be asked before each file is purged.

PURGE :1 (D="Ø2/Ø1/81-Ø2/Ø4/81")

This command will purge all visible files on drive 1, as long as their mod date is between Ø2/Ø1/81 and Ø2/Ø4/81, inclusive. You will be asked before each file is purged.

PURGE /SCR:2 (Q=N,D="-Ø6/Ø2/81")

This command will purge all visible files with a /SCR extension, provided their mod date is Ø6/Ø2/81 or earlier. You will not be asked before each file is purged.

R E N A M E

This command will rename a file. The syntax is:

```
RENAME filespec1 TO filespec2
RENAME filespec TO partspec

no parameters are required

abbr: NONE
```

The RENAME command allows you to change the filename and extension of a given file. The RENAME command will use dynamic defaults for the filename, extension, and drivespec of filespec2. This means that any part of filespec2 that is not specified will default to that of filespec1. The drivespec of filespec2, if specified, MUST be the same as that of filespec1 or the rename will abort and an error message will be generated.

RENAME will not allow the changing or deleting of a file's password. To change or alter a password, refer to the ATTRIB library command.

RENAME TEST/DAT:Ø TO OLD/DAT

This command will rename the file TEST/DAT on drive Ø to OLD/DAT.

RENAME TEST/DAT:Ø TO REAL

This command would rename the file TEST/DAT on drive Ø to REAL/DAT. The extension /DAT was not specified for filespec2, and defaulted to that of filespec1.

RENAME TEST/DAT:Ø TO REAL/

This command will rename the file TEST/DAT on drive Ø to a file named REAL. The use of the / with no characters after it in filespec2 kept the extension from defaulting to /DAT.

RENAME TEST/DAT TO REAL/DAT

This command will search the active drives for the file TEST/DAT and rename it REAL/DAT.

RENAME TEST/DAT TO /OLD

This command will search the active drives for a file TEST/DAT and rename it TEST/OLD. The filename was not specified in filespec2, and defaulted to that of filespec1.

RENAME DATA/NEW.SECRET:1 TO /OLD

This command will rename the password protected file DATA/NEW.SECRET on drive 1 to DATA/OLD.SECRET. The filename and password for filespec2 defaulted to those of filespec1.

RENAME TEST/DAT TO TEST

This command is not a valid command, and will produce the error message DUPLICATE FILE NAME. This is because the extension of filespec2 will default to /DAT, thereby creating the same filename for filespec2 and filespec1, which are the same file.

R E S E T

This command will reset logical devices and provide a way to restore HIGH$ (highest unused memory location) to the top of memory.

```
RESET
RESET devspec

abbr: NONE
```

There are two uses for RESET The first is a global reset, the second is the reset of a single device. The global reset will reset all active devices, while the reset of a single device will affect only that device. A device's "default driver routine" referred to in the following explanations can be seen by doing a DEVICE command before any configuration is done. If the device has been filtered, linked, routed, or set to another driver, the new device address will be shown by the DEVICE command.

Single device RESET

A single device reset will accomplish the following. Any filtering, linking, routing, or setting done to the device will be removed. Any open disk file connected to the device will be closed. Doing a DEVICE library command will show the device pointed to it's normal default driver routine. If the device has been created by the user, it will be pointed NIL when reset, and the KILL library command can remove it from the device display at this time.

If high memory was used when this device was altered, it will not always be reclaimed by the system. However, some routines can re-use the same memory allocation if they are enabled again after being disabled or reset.

The following will re-use their original memory allocation if re-activated after being disabled. See the individual sections for exact command syntax and specifications.

SPOOL library command KI/DVR, including the TYPE and JKL Options

MiniDOS/FLT PR/FLT KSM/FLT

Here are some examples of the RESET *devspec command.

RESET *PR

If you had your printer (*PR) filtered with the PR/FLT routine, the RESET *PR command would restore the normal I/O path between the printer DCB and its default driver.

RESET *DU

Suppose you had a dummy device *DU routed to a disk file TEST/TXT, and had your printer (*PR) linked to *DU. This configuration would cause all output to the *PR to also go to *DU, and into the disk file TEST/TXT. If you RESET *DU, the device table would show *DU = Nil, and the file TEST/TXT would be closed. However, *PR would still be linked to *DU. Since *DU = Nil, any output sent to the *PR would be ignored by *DU. The printer (*PR) would function normally. To clear the LINK, issue a RESET *PR command. *DU would continue to be shown in the device table until the system is rebooted, *DU is killed, or a global RESET is performed.

Global RESET

The RESET command with no devspec will do a global reset. All system logical devices will be returned to their default driver routines. All user logical devices will be removed from the device control table. Any filtering, linking, routing, or setting will be cancelled. All open files will be closed.

The Drive Code Table will be returned to its default state, with the drive configuration coming from the system information sectors of the current system disk. Any software write protection will be cancelled. If your system drive has been set to some drive other than physical drive Ø, be sure to insert a system disk into physical drive Ø before performing a global reset. Any high memory disk drivers such as RDUBL, PDUBL, or hard disk drivers will be removed from memory, and access to any drives requiring one of these drivers will not be possible.

The system will also attempt to set HIGH$ to the top of the available memory. If certain system functions that use the task processor are active (such as the SPOOL library command or the blinking cursor on the Model I), the reset cannot restore HIGH$ to the top of memory. If this is the case, the following message will appear.

CAN'T RESET MEMORY, BACKGROUND TASK(S) EXIST

To reset HIGH$, you must turn off the particular function or reset the individual device before doing the global reset.

R O U T E

The ROUTE command re-routes input/output for a specified logical device or creates a device. The syntax is:

```
ROUTE devspec1 TO filespec/devspec2
ROUTE devspec (NIL)

(NIL) is a bit-bucket.

abbr: NONE
```

ROUTE will re-route all I/O for a specified logical device to another logical device, to a disk file, or (NIL). The (NIL), or bit-bucket, means that the device is routed to nothing. Any input sent to a device routed (NIL) will simply be ignored. A device routed (NIL) has no output.

NOTE: No more than 4 devices may be routed at any one time on the Model III.

Anytime a device is routed to a filespec, a File Control Block (FCB) and a blocking buffer will be dynamically allocated in high memory. The system will determine the current HIGH$ (highest unused memory location) and use the space directly below this location for its buffer. HIGH$ will then be decremented to protect this area.

If the designated filespec already exists, the data routed to that file will be appended to the end of the existing file. If you wish the data to be written from the beginning of the file, the file must be killed before the route is established. In some cases, it will be advisable to use the CREATE library command to allocate file space before doing the route.

A new logical device may be created with the ROUTE command. To create a device, simply route the desired devspec to another devspec, to a filespec, or to (NIL). The new device will then appear in the device table.

To examine any currently existing routing, use the DEVICE library command. The device notations shown directly below the disk drive configurations will indicate all currently recognized devspecs and any routing, among other things, that has been done.

Once a device has been routed, it may be returned to its normal power-up state or removed completely from the device table with the RESET or KILL library commands.

ROUTE *PR *DO

> This command will route any data sent to the line printer (*PR) to the video display (*DO). None of the characters will be printed by the line printer, but instead will be shown on the video display. This command is very similar to LINK *PR *DO, the exception being that the characters are not printed by the line printer with the route but are with the link. The line printer need not be hooked to the system if *PR is routed to *DO. To remove the routing, use the command RESET *PR.

ROUTE *DU TO TEST/TXT:Ø

This command will route a user device (*DU) to a disk file TEST/TXT on drive Ø. A File Control Block and a blocking buffer will be established in high memory. The device table will show the routing with an entry of:

*DU => TEST/TXT:Ø

The file TEST/TXT will remain open as long as the device *DU is not RESET. The file must be closed with the RESET *DU command prior to removing the diskette from the drive.

ROUTE *PR TO PRINTER/DAT

This command routes all data normally sent to the line printer (*PR) to a disk file PRINTER/DAT. The system will search all active drives and use the first file PRINTER/DAT it finds. Any data sent to the *PR will then be appended to the end of the PRINTER/DAT file. If the file does not exist, it will be created on the first available drive. An FCB and blocking buffer will be allocated in high memory and the file PRINTER/DAT will remain open until the *PR is reset.

Before routing any device to a disk file, it is advisable to determine the amount of free space available on the diskette. Make sure the space available on the disk is adequate to hold the amount of data you wish to route to it! A "Disk space full" error may hang up the system if encountered when writing to a file via the ROUTE command.

The constant "EOF maintenance" file mode may be useful to invoke when routing to disk files (see use of the "trailing ! character" with filespecs, in the Glossary). This will cause the EOF (End Of File) to be updated after each buffer is written to the file. If EOF maintenance is not invoked, then the EOF will not be written to the file until the routing is reset, which will properly close the file. If a file is not properly closed, the data written to it may not be recoverable. If a "Disk space full" error is encountered when the EOF maintenance has been invoked, all data up to the last "full" buffer written to the file will be intact, and the file will be readable by the system.

R U N

The RUN command will load a program into memory and then execute it. The program must be in load module format. The syntax is:

```
RUN (X) filespec (parm,parm,...)

filespec    is any valid LDOS filespec of a file in
            load module format.

(X)         is optional to execute the program from a
            non-system disk for the single drive user.

parm        optional parameters to be passed to the
            filespec program.

abbr: NONE
```

The RUN command will load in a load module format program that resides above X'51FF', and then execute the program. The default extension for the filespec is /CMD. If the program resides on a non-system diskette, the (X) parameter may be specified to load the program from that diskette and begin execution only after a system disk has been reinserted in drive Ø.

If the (X) parameter is used the program must load above X'52FF'.

The RUN command is identical to the LOAD command except for the fact that control is transferred to the program module transfer address rather than returning to the system. Load module format programs may also be directly loaded and executed from the LDOS Ready prompt by simply typing in the name of the program.

Following are some examples of the RUN command.

```
RUN SCRIPSIT/LC
SCRIPSIT/LC
```

Both of these commands will produce the same results. The program SCRIPSIT/LC will be loaded into memory and executed.

```
RUN LBASIC
LBASIC
```

Both of these commands will produce the same results. They will load a program named LBASIC/CMD and execute it. Note that the file extension defaulted to /CMD when not specified by the RUN command.

```
RUN (X) INVADERS/CMD
```

This command is for the single drive user. It will load the program INVADERS/CMD from any disk, whether or not it is an LDOS system disk. After the command has been entered, you will be prompted with the message:

INSERT SOURCE DISK <ENTER>

At this point, you should insert the diskette containing the program in drive Ø and press <ENTER>. After the program is loaded, you will be prompted:

INSERT SYSTEM DISK <ENTER>

You should now insert your LDOS system disk back into drive Ø and press <ENTER>. Program execution will begin at this point.

S E T

This command sets a logical device to a driver routine. The syntax is:

```
SET devspec TO filespec (parm,parm,...)

devspec     any currently enabled logical device.

filespec    any valid "driver type" program.

parms       optional parameters required by the driver
            program specified with filespec.

abbr: NONE (except as allowed by the driver program).
```

The SET command will set a logical device to a driver program. It does this by loading the specified driver program, which will relocate itself into high memory just below HIGH$, lowering HIGH$ to protect itself. Once a device is set, any I/O to or from the device will be controlled by the new driver routine. LDOS will allow the passing of parameters to the driver program. These parameters are totally independent of the SET command, and are determined only by the needs of the driver program.

The filespec parameter is the filespec of the driver program. The default extension for this file is /DVR.

When a device is set, any previous filter, route, link, or set of that device will be destroyed. Once a device has been set, it will remain set until it is either routed or reset. The driver program will remain in high memory even if the device is reset.

The KI/DVR program will re-use its original high memory allocation if reset and then set again. See the KI/DVR section for exact details.

All other setting of devices will produce the following results:

If a device is reset, and then set again, the driver routine will load in below the current HIGH$. As a result, the setting, resetting, and then setting again of devices will cause the available memory to continue shrinking. Once a driver program is loaded, it will not be removed from memory or overwritten, even if the same device is reset and then set to the same driver. A global reset (if allowable) will remove this driver program and free up the memory by resetting HIGH$.

For complete information on the Device Driver programs supplied with the LDOS system, refer to section 5, DEVICE DRIVERS.

```
SET *CL TO RS232T/DVR (BAUD=3ØØ,WORD=7)
SET *CL RS232T (BAUD=3ØØ,WORD=7)
```

These two commands produce identical results. The TO is optional and may be replaced by a single space. Specifying the filespec RS232T/DVR produces the same results as specifying the filespec RS232T, as the default extension is /DVR. The RS232T program is for the Model III computer. The Model I version could be used instead with identical results.

These commands set the Comm Line (*CL) to a driver routine called RS232T/DVR. This is an actual LDOS driver program, and is described in the DEVICE DRIVER section. The parameters BAUD and WORD are valid parameters of the RS232T/DVR program. All I/O to/from the Comm Line will be sent through this driver routine, and be properly dealt with to be sent out the RS-232 interface.

SET *KI INKEY (F=64)

This command sets the keyboard (*KI) to a user driver program named INKEY/DVR, and passes the parameter F=64 to the driver program.

SET *PR RS232x

This command would set *PR (the line printer) to one of the supplied RS232 driver programs. This would be the normal way to use a serial printer with LDOS.

S P O O L

The SPOOL command establishes a FIFO (First In, First Out) buffer for a specified device (usually a line printer). The syntax is:

```
SPOOL devspec TO filespec (MEM=aa,DISK=bb)
SPOOL devspec (OFF)

devspec is any valid LDOS device.

filespec is an optional LDOS filespec.

OFF    turns off the spooler and resets devspec.

MEM=   Memory to be used by the spooler.

DISK=  Disk space to be used by the spooler (Ø to bb)

aa     Amount of memory to be used, in blocks of
       1K (1Ø24 bytes). 1K is automatically used.

bb     Amount of disk space to be used in blocks of
       1K (1Ø24 bytes).

abbr: DISK=D, MEM=M, OFF=N
```

*** NOTE ****

For proper SPOOL operation, the KI/DVR program must have been set before turning on the spooler.

When using the SPOOL command with a serial printer, you must use the following steps for proper operation. First, SET *PR to the RS232 driver, FILTER *PR with the PR/FLT filter, then turn on the spooler.

The SPOOL command will establish a FIFO buffer for a specified device. All output sent to the device will be placed in an output buffer consisting of memory and/or disk buffers, and will be sent to the device whenever that device is available to accept this data.

The minimum amount of memory required by the SPOOL command is 1K (1Ø24 bytes) for the memory buffer. The filespec is optional, and if no disk buffers are required, it is possible to spool strictly to memory. If disk space is requested, additional memory will be used to map the spool file area. The more disk space used, the larger the required memory block will become.

Using parameters that would cause the memory used to go below X'8ØØØ' will not be allowed (Note that X'8ØØØ' is an approximate value, and may vary +/- 256 bytes).

When the spooler is active, output to the specified device is treated in the following manner. Any output data which cannot immediately be accepted by the device is sent to the memory buffer. When the memory buffer is full, the data is sent to the disk buffer (if one has been specified). The stored information is sent to the device in a FIFO manner. Output of the stored data to the device is carried on as a background task even when the system is performing other functions.

The following rules govern the memory and disk space allocation:

PARAMETER: MEM

As stated earlier, the SPOOL command will always require a minimum of 1K (1Ø24) bytes for a memory buffer. If more memory is required, it may be allocated with this parameter. For example:

MEM=1Ø

This command will allocate 1ØK (1Ø,24Ø bytes) of memory to be used as a spool buffer. This memory will be dynamically allocated by the fully integrated spool system processor to provide the most efficient operating environment, depending on the particular configuration you have established for your LDOS system.

If this parameter is not specified, 1K of memory will automatically be allocated for the spool buffer.

PARAMETER: DISK

This parameter sets the maximum amount of file space to be allocated for the spooling. Disk space is allocated in blocks of 1K (1Ø24 bytes), the same as memory. When this parameter is set, the system will create a file of the size specified. If this parameter is not specified, the SPOOL command will automatically allocate approximately 5K of disk space, depending on the particular disk type. The file name of this file will be determined by the filespec parameter.

To prevent the SPOOL command from using any disk space, specify this parameter:

(DISK=)
(DISK=Ø)

By specifying the DISK= parameter with no size, the system will not allocate any disk space to the SPOOL command and will not create any file.

PARAMETER: filespec

NOTE: For the filespec parameter to be valid, the DISK= parameter must not have set the disk file allocation to zero.

The filespec is the name of the file the SPOOL command will write to any time its memory buffer is full. The default extension for this file is /SPL. The default filename will be the two letters of the devspec which is being spooled. Refer to the following examples.

SPOOL *PR TEXTFILE:Ø...The filespec will be TEXTFILE/SPL:Ø, as the file extension was not specified and defaulted to /SPL.

SPOOL *PR PR/TXT:Ø...The filespec will be PR/TXT:Ø. Specifying the /TXT extension will override the default /TXT.

SPOOL *PR :1...This command will look on drive 1 for a file named PR/SPL. If the file PR/SPL:1 is not found, it will be created on drive 1 with a length determined by the DISK= parameter.

SPOOL *PR...This command will search all active drives for a file named PR/SPL. If this file is not found, the file PR/SPL will be created on the first available drive (with the file size determined from the DISK= parameter).

The following examples will show some possible combinations of the spool parameters.

SPOOL *PR TEXTFILE:Ø (MEM=5,DISK=15)

This command will allocate 5K of memory and 15K of disk space in a file named TEXTFILE/SPL on drive Ø. Any output sent to the printer will be buffered and sent to the line printer (*PR) as fast as the printer can accept the characters. Even if current program printing functions exist, other functions will be carried out and the line printer will continue to receive data from the spooled buffers as fast as it can accept the data. The other program function processing will be carried on with little noticeable interruption. If the 5K memory buffer is filled, the data will then be written to the disk file TEXTFILE/SPL on drive Ø.

SPOOL *PR (MEM=1Ø,DISK=)

This command will create a 1ØK memory buffer for any data that is to be sent to the line printer (*PR). If a printer command is received, the data will be immediately sent to the 1ØK memory buffer, and then spooled to the line printer whenever the printer can accept it (i.e whenever the printer is not printing or otherwise in a BUSY or FAULT state). Since the data is sent to the printer as a background task, normal program execution will continue to take place. Note that none of the spooled data will be sent to a disk file, as the parameter DISK= was specified without any size. If the memory buffer is filled, processing of the current program functions will halt until the line printer has printed enough data to bring the outstanding character count below 1Øk (the size of the memory buffer).

If you are running an applications program that involves output to the line printer, it is possible that the overall efficiency of the program may be improved by activating the LDOS spooler. The size of the program and the available free memory and disk space will determine the amount of spooling available for your needs.

ONE NOTE OF CAUTION: the spool file on disk will remain open as long as the spooler is active. Do not kill this file or remove the diskette without first closing the file! You will not be allowed to do a SYSTEM (SYSGEN) if the spooler is active.

The file may be closed by turning the spooled device OFF. The proper syntax is:

SPOOL devspec (OFF)
SPOOL devspec (N)

Either of these two commands will turn off the spooler and close the associated disk file. Additionally, any other filtering, linking, setting, or routing done to *PR will be reset. Please note that the disk file will not be closed by resetting or killing the spooled device. It must be turned off to close the file.

Once the spooler is turned off, it may be turned on again. Doing so will re-use the same memory locations allocated when it was originally turned on. The following restrictions will apply:

The original parameters will be stored. If turned off and then back on, any parameters specified may not exceed the memory or disk parameters originally given, or an error will occur. However, memory or disk space parameters may be diminished.

The original stored parameters will not be affected by turning the spooled device off and then back on.

* S Y S T E M

This command is used to configure the user definable areas of your LDOS system. The syntax is:

```
SYSTEM (parm,parm,...)

Allowable SYSTEM parameters are:

        ALIVE       BASIC2      BLINK
        BREAK       BSTEP       DATE
        DRIVE       FAST        GRAPHIC
        SLOW        SVC         SYSGEN
        SYSRES      SYSTEM      TIME
        TYPE        UPDATE

Parameter arguments will be detailed in this section.

abbr: ON=Y, OFF=N
```

The existing configuration of your LDOS system can be seen by doing the DEVICE and MEMORY commands. The SYSTEM command can set or change the disk drive configuration as well as turn on or off different keyboard, video, and hardware drivers. Each valid SYSTEM command will be discussed in this section.

Once your LDOS system has been configured, you may store the configuration on the drive Ø disk with the SYSTEM (SYSGEN) parameter. Please read this section thoroughly to determine the different SYSTEM command uses, and to discover exactly how other LDOS commands will affect the (SYSGEN) parameter.

Certain of the SYSTEM commands must load driver routines into high memory to accomplish their functions. When they do this, they determine the highest unprotected memory location (referred to as HIGH$) and load directly below this location. After loading, the LDOS system moves HIGH$ down to protect these routines. If you have executed any SYSTEM commands that require the use of this high memory, be aware that your overall free memory will be decreased accordingly.

Most of the following parameters for the SYSTEM command may be used together in the same command line. To do this observe the syntax: SYSTEM (parm,parm,....,parm). Each parameter must be accompanied by its switch or setting as required.

Following are complete descriptions of the SYSTEM parameters.

SYSTEM (FAST)
SYSTEM (SLOW)

These commands are used only if a suitable clock speed-up modification has been installed in the TRS-8Ø computer unit. (Speed up kits are neither endorsed or condemned by LSI, although some effort has been made to support this type of modification). They will modify certain timing loops in the LDOS system to accommodate the current processor clock speed, as well as switching the software controlled clock. These two commands have precedence over any other SYSTEM command, and will always be executed before any of the other commands are carried out. No memory will be used by these parameters.

The clock speed is controlled in the following manner:

FAST issues an OUT Port 254,1 command.

SLOW issues an OUT Port 254,Ø command.

SYSTEM (ALIVE=switch)

The SYSTEM (ALIVE) parameter displays an "alive" character in the upper right corner of the screen. It is primarily used to determine the current state of the task processor. If the alive bug is moving, the task processor is running. Note that the alive bug may continue moving (indicating an alive system) even when the TRACE library command display has stopped.

The switch is either ON or OFF. If not specified, ON is assumed. The ALIVE parameter uses some RAM in high memory.

SYSTEM (BASIC2)

This command will direct you to the ROM Basic in the TRS-8Ø computer. Typing in SYSTEM (BASIC2) while in the LDOS READY mode will function identically to pressing the reset button with the <BREAK> key held down. The screen will clear, and "Cass ?" (Model III) or "Memory Size?" (Model I) will be displayed in the upper left corner of the display. Any routing, linking, or driver routines set under LDOS will be reset to the normal ROM Basic drivers. While in ROM BASIC, none of the disk functions are available for use and you cannot return directly to LDOS or LBASIC. You must press the reset button or turn off the computer and go through power up to get back to the operating system.

SYSTEM (BLINK=aaaa)

This command controls the LDOS cursor character. The parameter aaaa can be represented as ON/OFF or as a decimal value. The cursor character numbers in the following examples are the ASCII values (in decimal) of the TRS-8Ø character set. This command will use high memory on the Model I.

ON.....Turns the blinking cursor on, with the cursor character being a graphics character (character 176).

OFF....Turns off the blinking cursor. On the Model III, the cursor character will be a non-blinking graphics character. The Model I will have the normal power-up cursor.

aaaa can also be represented as any displayable ASCII character value. For example, if the command SYSTEM (BLINK=42) were given, the blinking cursor character would be an asterisk (character 42).

SYSTEM (BLINK,LARGE)

This command turns on a large (character 143) blinking cursor.

SYSTEM (BLINK,SMALL)

This command turns on a small (character 136) blinking cursor.

SYSTEM (BREAK=switch)

This command will enable or disable the <BREAK> key. The allowable switches are ON or OFF. If switch is not specified, the default will be ON. Once the <BREAK> key is disabled by doing a SYSTEM (BREAK=OFF) command, pressing it will have no effect, and the system break bit will not be set. It may be re-enabled at any time by doing a SYSTEM (BREAK=ON) command. The (BREAK=ON) will also enable the <BREAK> key if it was disabled by the AUTO library command. No memory will be used with this parameter.

NOTE: Specifying (BREAK=OFF) will prevent routines such as the BUILD Library command from exiting when the <BREAK> key is pressed!.

SYSTEM (BSTEP=n)

This command will establish the default bootstrap step rate used with the FORMAT utility. The BSTEP value can be Ø, 1, 2, or 3. These values correspond to the step rates as described in the SYSTEM (DRIVE=,STEP=) command. This value will be stored in the system information sector on the current drive Ø. If you switch drive Ø disks or change drive Ø's with the SYSTEM (SYSTEM) command, be aware that this default value will be taken off the new drive Ø disk.

SYSTEM (DATE=switch)

This command is used to enable or disable the initial prompting for the date on power up. The diskette may not be write protected when using this command. The switch may be ON or OFF as follows:

ON....Enables the date prompt if it has been disable with the OFF parameter. If the switch is not specified, on is assumed.

OFF...Disables the date prompt on power up or reset.

Since the date is used extensively throughout the LDOS system, it is recommended that you never disable the initial date prompt with this command. The date will remain set even if you press the computer's reset button, so you will not have to re-enter it.

SYSTEM (DRIVE=d,parm,parm,...)

This command sets certain parameters for the disk drives in your system. Refer to the following for explanation of the allowable parameters. This command uses no extra memory.

DRIVE=d represents any valid drive number in your system. Only one DRIVE=d parameter can be used in any system command line.

CYL=nn.....This command will set the default number of cylinders (in the range 35 to 96) to be used with the FORMAT utility for the specified floppy drive. This value will be written to the system information sector on drive Ø. If you switch drive Ø disks, be aware that this value will be taken from the new drive Ø disk when formatting.

DELAY=OFF....This command is valid only for 5 1/4" drives. The DELAY is the time allowed between drive motor start up and the first attempted read of the diskette in that drive. The OFF parameter sets this delay to .5 seconds.

DELAY=ON.....This command is valid only for 5 1/4" drives. The ON parameter sets the delay between drive motor start up and the first attempted read to 1 second. This is the normal DELAY time for all 5 1/4" drives.

DISABLE.....This command will remove the specified drive number from the Drive Code table. Once disabled, any attempt to access that drive will cause the message "Illegal Drive Number" to appear. The drive can be re-enabled with the ENABLE parameter.

ENABLE.....This command will enable the specified drive number and place its configuration information in the Drive Code table. If you enable a drive that has not been previously enabled or set up with the SYSTEM (DRIVE=,DRIVER) command, totally unpredictable results may follow.

STEP=n.....This parameter will set the stepping rate for the specified drive number, where n is a number Ø to 3. The following table lists the different stepping rates in ms (milliseconds) for 8" and 5 1/4" drives. Do not select a step rate faster than your drive can handle. If in doubt, contact the drive manufacturer.

n=Ø	8" step rate = 3 ms	5 1/4" step rate = 6 ms
n=1	8" step rate = 6 ms	5 1/4" step rate = 12 ms
n=2	8" step rate = 1Ø ms	5 1/4" step rate = 2Ø ms
n=3	8" step rate = 15/2Ø ms	5 1/4" step rate = 3Ø/4Ø ms

The 15/2Ø and 3Ø/4Ø are dependent on whether you are using a double/single density disk controller chip. Model III owners will have the 15 and 3Ø ms step rate, while Model I interface owners will have the 2Ø and 4Ø ms step rate. The fastest step rate for the Model I will be 12 ms when using single density disks. Using a double density board in the Model I will provide the 6 and 3Ø ms step rates.

SYSTEM (DRIVE=d,DRIVER="filespec")

To access the disk drives, LDOS will use information stored in memory in the Drive Code Table (DCT). No special configuration should have to be done unless drives other than 5 1/4" floppy drives are used. To configure the system for other drive types, it will be necessary to use this SYSTEM command.

The MODx/DCT program supplied will allow you to change the logical drive numbers for your 5 1/4" floppies. This may be desirable when running hard disk systems.

SYSTEM (DRIVE=d,WP=sw)

This command will allow you to software write protect any or all drives currently enabled. Only one DRIVE=d parameter may be entered on the command line.

The parameters for this command are as follows:

d = the drive number affected.

sw = the switch ON or OFF. ON will set the write protect status, and OFF remove it and allow the drive to be written to.

The command with no drivespec specified will act globally. That is, SYSTEM (WP=ON) will write protect all drives in the system, and SYSTEM (WP=OFF) will remove any software write protection that has been done on any drive. The WP=OFF parameter will have no effect on a disk physically protected with a write protect tab. Note that if the flag ON or OFF is not specified, ON is assumed.

SYSTEM (GRAPHIC)

This command informs the LDOS system that your line printer has the capability to directly reproduce the TRS-8Ø graphics characters during a screen print (screen print is enabled as a parameter of the KI/DVR program, DEVICE DRIVER section). If this parameter is used, any graphics characters on the screen will be sent to the line printer during a screen print command, either from the DOS level or with LBASIC's CMD"*". Do not use this parameter unless your printer is capable of directly reproducing the TRS-8Ø graphics characters.

SYSTEM (SVC)

This command will load a Supervisory Call (SVC) table into high memory. A complete description of the SVC table can be found in the Technical Information section. You must have set *KI to the KI/DVR program if you wish to use the SVC table.

SYSTEM (SYSGEN=switch)

This command creates or deletes a configuration file on the drive Ø diskette, where switch represents the following parameters:

ON.....creates a configuration file.

OFF....deletes the configuration file.

If switch is not specified, ON is assumed. That is, SYSTEM (SYSGEN) is the same as SYSTEM (SYSGEN=ON). After a SYSTEM (SYSGEN=OFF) command has been given, the current configuration of the system will not change until the system is booted again.

When the SYSGEN parameter is used, all current device and driver configurations will be stored on the diskette in drive Ø. An invisible file named CONFIG/SYS will be created to hold the configuration. Each time the system is booted, the configuration stored in this file will be loaded and set. To prevent this automatic configuration, hold down the <CLEAR> key while the boot is in progress. Note that the system configuration will take place before any AUTO'ed command is executed. In addition to the SYSTEM commands and parameters listed in this section, the following will be stored in the configuration file by a SYSTEM (SYSGEN) command:

1) All filtering, linking, routing, or device setting that has been done. This includes the RS232 and KI drivers.

2) Any active background tasks (such as CLOCK, DEBUG, TRACE, etc).

3) Any special utility routines or user assembly language programs loaded into high memory and protected with the MEMORY command. All memory from HIGH$ to the physical top of memory will be written to the CONFIG/SYS file.

4) The present state of VERIFY (either ON or OFF).

5) All Device Control Blocks. This will include the current lines per page and line counter stored in the printer DCB.

6) The state of the CAPS lock for the keyboard.

Certain LDOS features should never be SYSGENed if a disk file is involved. They are any ROUTE or SET involving a disk file. SYSGENing open files can cause loss of data if the disks are switched in the drives without the files being closed. Disk switches with open files can also cause existing data to be overwritten.

SYSTEM (SYSRES=n)

This command will allow you to reside certain LDOS system overlays in high memory. SYS files 1-5, and 8-12 may be loaded using this command. Note that each overlay will require 1K (1Ø24 bytes) of memory. This command does NOT allow multiple entries on the command line. For example, SYSTEM (SYSRES=1,2,3) will result in only SYS3 being made resident.

Having certain of these SYS overlays resident in memory will speed up most disk I/O operations, as these modules will not have to be loaded from disk. It will also allow you to purge these overlays from your system disk, providing more room for data and programs. A description of the SYS overlay functions may be found in Section I. Overlays 2, 3, 8, and 1Ø must be resident for certain types of backups (see the BACKUP utility section). SYS2 and SYS3 must remain on any booting disk if a configuration file created with the SYSGEN parameter is to be loaded.

The DEVICE library command will show any overlays that are currently resident in high memory.

SYSTEM (SYSTEM=n)

This command will allow you to assign a drive other than drive Ø as your system drive. It will do this by swapping the DCT (Drive Code Table) information of the drive specified with the current system drive. Note that there must be a diskette containing the necessary system files in the drive specified!

Once this command has executed, LDOS will look for any needed system files on the new system drive. Also, the defaults for number of cylinders and the bootstrap step rate use by the FORMAT utility will now be taken from the new system drive. If necessary, use the SYSTEM parameters (BSTEP) and (DRIVE=,CYL=) to establish these defaults on the new system disk. The logical drive numbers will also be changed - addressing drive Ø will now access the newly specified system drive, and vice versa.

This procedure may be repeated, and a swap of the current system drive with the drive specified will occur. The logical drive numbers will also change again. Be careful when repeating this command, or you may lose track of which drive is currently assigned to what logical drive number.

Note that doing a global RESET library command will reset all drive DCTs to their default configurations. Be sure to have a system disk in physical drive Ø before performing a global RESET command.

SYSTEM (TIME=switch)

This command will enable or disable the prompt for the time on power up or reset. You must not have a write protected disk in drive Ø if using this command. The switch is either ON or OFF as follows:

ON....Enables the time prompt on power up or reset. If the switch is not specified, ON is assumed.

OFF...Disables the prompt for the time on power up or reset.

SYSTEM (TYPE=switch)

This command will turn on or off the task processing of the KI/DVR type ahead feature. If you have set *KI to the KI/DVR program specifying the (TYPE) parameter and wish to temporarily suspend the type ahead feature, use the SYSTEM (TYPE=OFF) command. This will turn off the type ahead processing without disturbing any other filters you may have applied to the keyboard. The type ahead task processing may be restarted with the SYSTEM (TYPE=ON) command.

SYSTEM (UPDATE=switch) - *** Model I only ***

The UPDATE parameter will allow the system date to be updated if the real time clock passes midnight (23:59:59). The date will advance one day and the day of the week and day of the year will also change. This routine will use some high memory. Due to hardware differences, this routine will not work on the Model III.

The switches are ON or OFF, to enable or disable the update function. Doing a global RESET library command will disable the update function.

* T I M E

This command is used to set the time for the "real time" clock.

```
TIME hh:mm:ss
TIME

hh=    hours ØØ-23

mm=    minutes ØØ-59

ss=    seconds ØØ-59

abbr: NONE
```

The TIME library command is used to adjust the time kept by the system's real time clock. You can also be prompted on power up or reset for the time. This prompt may be enabled and disabled with the SYSTEM (TIME=) library command. The clock function is normally controlled by hardware circuits in the expansion interface (Model I) or by a signal developed from the AC power line (Model III). This time is not an actual "real time" clock, as the clock referred to is used by many different software and hardware devices. Certain system operations require that the clock be turned off altogether. You are advised not to depend on the clock for constantly accurate time and date information.

The time setting of the clock can be examined or set as follows:

Issuing a TIME command with no parameters will display the current setting of the clock. The clock will be reset to ØØ:ØØ:ØØ every time you power up, press the reset button, or issue a BOOT library command.

To set the clock, use the command TIME hh:mm:ss, specifying the hours, minutes and seconds desired. The latest time acceptable is 23:59:59, as the clock will always run in the 24 hour mode.

The time lag between pressing the <ENTER> key and the time that will actually be set on the clock will be approximately 2 seconds (the time needed to execute the TIME command). It is usually best to type in the command TIME and then the time plus several seconds after the correct time. Wait for (seconds -2) to come up on your watch and press <ENTER>. This will give you the correct time on the clock.

There are several ways for application programs to retrieve the current time setting of the clock. At an assembly language level, a call to the @TIME vector will return the time. When using LBASIC, the time can be returned through the TIME$ variable.

The time may be constantly displayed on the video screen by issuing a CLOCK library command or with the <C> key function of the MiniDOS keyboard filter program. Either of these commands will enable or disable the clock display in the upper right hand corner of the screen.

On the Model III, the LBASIC commands CMD"R" and CMD"T" will also turn on and off the clock display.

* T R A C E

This command displays the user's Program Counter address (PC register of the Z-8Ø processor) in the upper right corner of the video display. The syntax is:

```
TRACE
TRACE (ON)
TRACE (OFF)

abbr: ON=Y, OFF=N
```

This command will display the contents of the Program Counter on the video display. The display will be a hexadecimal address. Any information normally displayed on the top line (print locations 45 - 48) will be overwritten by the trace display. The display is constantly updated as a high priority background task. The TRACE command is primarily useful during debugging of assembly language programs.

The trace display will halt if an assembly language program disables the interrupts, or if an LBASIC program (Model I only) does a CMD"T". Doing a CMD"R" will restart the trace display.

The allowable commands are:

TRACE (ON) Turns the TRACE on.

TRACE (OFF) Turns the TRACE off.

TRACE Turns the TRACE on, the default parameter being ON.

NOTE: TRACE, along with some other operations, may not function properly on the Model III when the display is in the 32 character mode.

* V E R I F Y

The VERIFY command forces all disk writes to be verified with a read-after-write operation. The syntax is:

```
VERIFY (switch)

switch is the parameter ON or OFF, ON is the default

abbr: ON=Y, OFF=N
```

The VERIFY command will determine whether or not writes to a disk file are verified with a read-after-write operation. The state of the VERIFY command may be saved in the configuration file with the SYSTEM (SYSGEN) library command. The normal power up condition is VERIFY (OFF). To cause a read after write verify of every write operation, you must specify the command VERIFY or VERIFY (ON).

The VERIFY command works by reading the checksum of the last written sector and comparing it against the checksum recorded when the sector was written. It does not do a byte for byte verify on the information in the disk sector. Anytime that an error is detected, the appropriate error message will be displayed.

Although having the VERIFY function turned on will provide the greatest reliability during disk I/O, it will also increase the overall processing time whenever a disk file is written to. The user must determine if the increase in reliability warrants the increase in processing time. It is recommended that verify be turned on anytime critical data or program files are being written.

The command VERIFY (OFF) will disable the read-after-write verification.

All disk writes will automatically be verified during any BACKUP utility function, whether the VERIFY command has been issued or not. Also, certain critical writes to system tables and any write to the directory will always be verified.

B A C K U P

The BACKUP utility is provided to duplicate data from a source disk to a destination disk. The syntax is:

```
BACKUP :s TO :d (parm,parm)
BACKUP partspec w/wcc:s TO :d (parm,parm)
BACKUP -partspec w/wcc:s TO :d (parm,parm)

:s          the SOURCE drivespec.
:d          the DESTINATION drivespec.

       Allowable parameters are as follows:

MPW="aa" passes the source disk's Master Password.

VIS      indicates Visible files.

SYS      indicates System files.

INV      indicates Invisible files.

MOD      indicates files Modified since last backup.

QUERY    parameter indicating Query each file before
         moving. The switch ON or OFF may be specified.

OLD      will backup only those files already existing
         on the destination disk.

NEW      will backup only those files not already on
         the destination disk.

X        allows backups with no system disk in drive Ø.

DATE=    "M1/D1/Y1-M2/D2/Y2" will backup only those
         files whose mod dates fall between the two
         dates specified, inclusive.

         "M1/D1/Y1" will backup all files with mod
         dates equal to the specified date.

         "-M1/D1/Y1" will backup all files with mod
         dates less than or equal to the specified date.

         "M1/D1/Y1-" will backup all files with mod
         dates greater or equal to the specified date.

abbr: QUERY=Q, INV=I, MOD=M, SYS=S, VIS=V, DATE=D
```

The BACKUP command will move all or part of the data from a specified source disk to a specified destination disk. The parameters of the BACKUP command may be used to determine which data will be moved. All of the parameters are optional, with only the source and destination drivespecs being prompted for if not entered. If the source disk contains a password other than "PASSWORD", it will be prompted for if not passed with the MPW= parameter.

*** NOTE ***

Due to the complexities involved with handling many different disk drive configurations, the LDOS BACKUP utility demands that destination disks must be formatted before the backup begins. This format before backup requirement is found in most large operating systems that allow many different diskette types. Having the destination disk formatted will allow the BACKUP utility to determine if a Mirror Image (exact cylinder for cylinder copy) backup is possible, or if it will be necessary for the backup utility to do a file by file duplication.

There are three types of backups available with LDOS. They are MIRROR IMAGE, BACKUP BY CLASS, and BACKUP RECONSTRUCT. Certain rules determine which type of backup will be done.

A mirror image backup will be attempted if the size (5" or 8" floppy), the density, and the number of sides are identical on the source and destination disks. The number of cylinders need not be identical as long as the destination disk has a cylinder count greater than or equal to the source disk.

A backup by class will be done if the user specifies a partspec or any parameter except "X" or "MPW" in the command line.

A backup reconstruct will be done if the size (5", 8" or hard), the density, or the number of sides differs between the source and destination disks.

A backup by class and a backup reconstruct function identically, doing a file by file copy. The only difference is that a backup by class is initiated by the user and a backup reconstruct is initiated by the system.

It is necessary for backup to turn off the system real time clock during certain operations. For this reason, the message:

NOTE: REAL TIME CLOCK NO LONGER ACCURATE

will appear after the completion of the backup. This is merely an informative message reminding you the clock is no longer accurate.

If the backup is being done from a JCL file, the following rules will apply:

If the backup is mirror image, the Pack IDs (disk name and master password) must be the same or the backup will abort.

Backup with the (X) parameter, single drive backups, and backups with the (Q) parameter cannot be done from a JCL file.

Mirror image backups

A mirror image backup is basically a cylinder for cylinder copy from the source to the destination disk, with only those cylinders that actually contain data being moved. The date on the destination disk will be changed to the current system date. However, the boot sector containing the bootstrap step rate will remain untouched on the destination disk.

A mirror image backup will always compare the disk Pack ID's (disk name and master password) to make sure they are identical. If they are not, you will see the following message:

DIFFERENT PACK ID'S! ABORT BACKUP?

Answer this question <Y> to abort the backup or <N> to continue the backup. If you use informational disk names when formatting diskettes, this checking of Pack ID's should help prevent you from backing up the wrong disks.

If the source and destination disks have different cylinder counts, the following message will appear:

CYLINDER COUNTS DIFFER - ATTEMPT MIRROR IMAGE BACKUP?

Answer this prompt <Y> to attempt a mirror image backup or <N> to force a backup reconstruct. The destination disk will have its directory on the same track as the source disk, even though this may not have been the case before the backup began. The information on the destination disk will be updated to reflect the true cylinder count and available free space.

You may also see the following message appear at times:

BACKUP ABORTED! DESTINATION NOT MIRROR-IMAGE.

This will occur if the destination disk is missing a cylinder that contains information on the source disk. This may be the case if the destination disk was formatted with fewer cylinders than the source disk, or if cylinders were locked out on the destination disk during formatting. You can use the FREE library command to check the destination disk for locked out cylinders.

After all cylinders are moved to the destination disk, the backup utility will attempt to remove the mod flags from the source disk. If the disk is write protected, you will see the message:

CAN'T REMOVE MOD FLAGS - SOURCE DISK IS WRITE PROTECTED

After the backup has completed, the destination disk will have the same Pack ID as the source disk. The destination disk date shown with the DIR or FREE library command will be changed to the current system date.

Backup by class and Backup reconstruct

These two backup types function identically, doing a file for file copy from the source to the destination disk. Unlike a mirror image backup, files that exist on the destination disk but are not on the source disk will remain untouched by the backup. When the backup is complete, the destination disk will contain all files moved from the source disk plus any other files that existed on the destination disk before the backup began. The destination disk Pack ID and date will not be changed by the backup. These types of backups may NOT be done on a single drive.

There are some things done when the file SYSØ/SYS is included in this type of backup that are not readily apparent. Certain information about the default drive types and the state of the SYSTEM (SYSGEN) configuration parameter are moved from the source to the destination disk. The destination will have the following set equal to the source disk, regardless of how the destination disk was previously configured.

The state of the SYSGEN (on or off) of the destination disk will be changed to match that of the source disk.

The initial date and time prompts (on/off) on power up will be set to match those of the source disk.

The default drive configurations will match those of the source disk.

It is possible to backup from a disk with a capacity greater than that of the destination disk, such as from a hard drive to a 5" floppy. To do this, format as many destination disks as will be needed to hold all of the information to be moved. As the backup progresses and the first destination disk is filled, you will be prompted with the flashing message:

DISK IS FULL. ENTER NEW FORMATTED DESTINATION DISK <ENTER>

At this point, remove the full destination disk and insert a new formatted disk in the drive. Pressing <ENTER> will cause the backup to continue. You may perform this disk swap as many times as necessary to complete the backup.

Backup will not allow a single file to be split across destination disks. If you have a file that is larger than the capacity of the destination disk, you will not be able to copy it with the backup command.

Both backup by class and backup reconstruct will attempt to remove the mod flags from the source disk. If the source disk is write protected, you will see the following message appear after the first file has been copied:

CAN'T REMOVE MOD FLAGS - SOURCE DISK IS WRITE PROTECTED

To provide a more readable display, this message will not be displayed after every file, although the mod flags will not be removed from any source files.

Backups with the (X) parameter

The X parameter will allow you to perform backups without the need for the system files to be on the drive Ø disk. This will allow backing up data disks of different sizes or capacities on a 2 drive system. Single drive owners will be limited to mirror image type backups.

If the backup will be by class or a reconstruct, two drives must be used. Also, the system modules 2, 3, 8, and 1Ø must be resident in memory (see the SYSTEM (SYSRES=) library command).

When doing a backup with the X parameter, you will be prompted to insert the proper disk in drive Ø. You may be prompted to switch drive Ø diskettes, depending the type of backup you are doing and the system modules you have resident.

Using the backup parameters

Many of the backup parameters are identical to those in the DIR and PURGE library commands. These parameters will allow you to choose the groups of files you wish to backup to your destination disk. All parameters may be used singly or in combination with any other parameters.

If no parameters are specified and a backup reconstruct is initiated by the computer, all files will be moved from the source to the destination. You may restrict this to visible, invisible, or system files with the VIS, INV, or SYS parameters.

The MOD and DATE parameters will allow you to choose only those files that have been modified since their last backup, or fall within a specified range of dates. This will be very useful on drives with large capacities, as it will not be necessary to backup the whole disk to obtain new copies of files that have changed.

The OLD and NEW parameters provide an easy method to update disks without placing unwanted information on the destination disk. For example, using the OLD parameter will allow you to update your working disks if changes are made to the system, copying over only those files which are already on your working disks.

The QUERY parameter will show you each file before it is backed up, including the file's date and mod flag status. You may tell backup to copy the file by pressing the <Y> key. Pressing <N> or <ENTER> will bypass the file and show you the next. Pressing the <C> key will copy the current file, and shut off the Query function. All files from that point on will automatically be copied.

The use of partspecs, -partspecs (not partspecs), and the wcc (wildcard character) will let you choose files based on their filename and extension. You may use these in combination with any of the previously mentioned parameters.

Examples of backup commands

Following are some examples and descriptions of the backup command. Please note that in all examples, the source disk's master password will be asked for if it is other than PASSWORD and is not specified with the MPW parameter. If the Q parameter is specified, the file's mod date and mod flag will be shown along with the filespec.

BACKUP :Ø :1

This command will attempt a mirror image backup, using drive Ø as the source drive and drive 1 as the destination drive. If the drives are differently configured, a backup reconstruct will be invoked. All files will be moved from drive Ø to drive 1, with the exception of DIR/SYS and BOOT/SYS if a reconstruct is invoked.

BACKUP $:Ø :1

The wcc ($) in this command will cause a backup by class. All files will be examined, and all files (except BOOT and DIR) will be copied because they will "match" the single wcc. This command is the way to force a backup by class in situations where a mirror image would normally have been done. This might be to remove unwanted "extents" from files on the source disk by copying them onto a cleanly formatted destination drive.

BACKUP :Ø :1 (Q)

This command will function identically to the previous example, except that you will be asked before each file is moved. You will also see the mod date and mod flag for each file.

BACKUP :1 :2 (VIS)

This command will copy all visible files in drive 1's directory to drive 2. A backup by class will automatically be invoked.

BACKUP :2 :1 (INV)

This command will copy all files that are invisible in drive 2's directory to drive 1, invoking a backup by class. Note that the system files will not be copied, although they are invisible in a normal directory display.

BACKUP :Ø :1 (SYS)

This command will backup all system files from drive Ø to drive 1, invoking a backup by class.

BACKUP :Ø :1 (VIS,INV)

This command will backup every visible and invisible user file from drive Ø to drive 1, invoking a backup by class. In other words, this command will copy all files except the system files.

BACKUP :Ø :1 (MOD,Q,MPW="SECRET")

This command will copy all files that have been modified (written to) from drive Ø to drive 1. It will query each file before it is copied, also showing the file's mod date and flag. The master password was passed with the (MPW=) parameter and will not be asked for.

BACKUP /CMD:Ø :1
BACKUP $/CMD:Ø :1

This command will force a backup by class, with the file class specified as /CMD. All files with the extension /CMD will be copied from drive Ø to drive 1. Note that the wcc ($) has no actual effect on the backup. Specifying the /CMD will look at all /CMD files, just as the $/CMD will. If the file exists on drive 1 it will be overwritten, otherwise it will be created at this time. No files on drive 1 will be touched except for the /CMD files copied from drive Ø.

BACKUP $$$$$$AT:2 :3 (MOD)

This command will backup all files whose filename is 8 characters long and contains AT as the last two letters. Only those files that meet this criteria and have been modified will be copied. A backup by class will be invoked.

BACKUP /$$S:1 :2

This command will backup all files whose extensions are 3 characters long, ending with the letter S. The wcc ($) masks the first two characters of the extension, so the extensions /BAS, /TSS, /SYS, etc. would all match. A backup by class will be invoked.

BACKUP -/CMD:Ø :1
This command will backup all files from drive Ø to drive 1, EXCEPT those that have the extension /CMD.

BACKUP :1 :1

This command will backup between two disks in drive 1. You will be prompted to switch between the source disk and destination disk at the appropriate times. The disks involved in this type of backup must allow a mirror image backup, or the backup will abort. This command could be used to backup a data disk. See the next example with the (X) parameter for another example of data disk backups.

BACKUP :Ø :1 (X)

This command will backup a disk in drive Ø to a disk in drive 1. Its primary use is to backup non-system disks, such as data disks, in a two drive system. When using this backup parameter, you will be prompted to insert the proper disk in drive Ø. You may be prompted to re-insert a system disk into drive Ø during certain backups.

When the backup is complete, you will be prompted to insert a system disk back in drive Ø. If the backup will be by class or a reconstruct, SYS overlays 2, 3, 8, and 1Ø must be resident in memory (see the SYSTEM library command).

BACKUP :1 :2 (OLD)

This command will backup files from drive 1 to drive 2, only if they already existed on drive 2.

BACKUP :1 :2 (NEW,Q)

This command will backup files from drive 1 to drive 2, only if they do not already exist on drive 2. You will be prompted before each file is moved, as the Q parameter was specified.

BACKUP /ASM:3 :2 (D="Ø5/Ø6/81-Ø5/1Ø/81")

This command will backup all files with the extension /ASM, as long as their mod dates fell between the two dates specified, inclusive.

Many more examples of the power of BACKUP could be given, but the best method for the user to understand the scope of BACKUP is through its use. Experiment until you are comfortable with this utility. In most cases, you can see exactly what files will be moved by a particular BACKUP command by doing a "DIR" command of the source disk using the same partspec and/or parameters you intend to use with the BACKUP.

As a final note, it is not allowable to specify passwords in any BACKUP command line. The BACKUP utility will ignore any password protection on a file, whether doing a backup by class or a mirror image backup.

C O M M A N D F I L E (CMDFILE)

The LDOS COMMAND FILE Utility is a general purpose disk-to-disk, tape-to-disk and disk-to-tape, machine language program that has been designed to provide the capability of appending two or more command (CMD, CIM, OBJ) files (machine language load modules) or SYSTEM tape files (machine language) files that can be loaded with the BASIC "SYSTEM" command. Inherent in its capability of performing I/O to disk or tape are the following functions:

1. Append two or more 'COMMAND' disk files or 'SYSTEM' cassette tape files into one file. This is useful to concatenate two or more separately assembled OBJECT code files, concatenate two or more non-contiguous blocks of code, or also couple two or more programs together so they load together.

2. Offset a tape or disk file so that it loads into a region other than originally programmed. A driver routine is optionally appended that moves it back to its original load region. User options provide for disabling the clock interrupts and keyboard debounce routines in the event that the SYSTEM program would have overlayed the debounce routine of LDOS.

3. Machine language programs (tape or disk files) can be appended with patched code to correct errors in a manner similar to the PATCH/CMD. This operation requires use of an Editor Assembler and, of course, knowledge of the corrective patch code.

4. Command files can be copied from one SYSTEM diskette to another SYSTEM diskette on a single drive system provided both diskettes use the same operating system.

5. SYSTEM cassette tape files can be created from non-contiguous blocks of memory; heretofore only possible via direct assembly from the Editor Assembler.

6. For the disk user, during input of 'COMMAND' files, the load address range of each block of code is displayed to the CRT and optionally to a line printer. The file's transfer address or entry point is also displayed.

TO ENTER THE COMMAND FILE UTILITY

At LDOS READY simply type: CMDFILE <ENTER>

COMMAND FILE will load and execute.

COMMAND STRUCTURE

All functions and procedures are specified by responding to a series of queries. Some queries request yes/no responses (abbreviated Y/N), some request disk/tape responses (abbreviated D/T), while others request specific information (i.e. file names, new addresses, etc.). Most yes/no and disk/tape responses can also be answered with a "C" to cancel the request and return to the main prompt as noted above. If you want to return to LDOS, responding with "E" for EXIT will return you to the respective system. Each query displays the valid responses acceptable to it. All queries accept lower case responses as well as upper case.

Query (1):

ADDRESS LOAD LOG TO PRINTER (Y,N,E)? >

The address load log will be displayed only for files read in from disk. The timing on tape reads is too critical to perform the extra processing necessary to detect the load limits and display them during a tape read. If you are a disk user, have a line printer, and want this log displayed on your printer, respond with a "Y", otherwise respond with an "N". If you want to exit CMDFILE, enter "E". This query is referred to as the main query. Whenever it is displayed, the memory buffer, used to store input files, will be reset to its beginning position to initialize for a series of input requests. This effectively "clears" the input buffer.

Query (2):

INPUT FROM DISK OR TAPE (D,T,E,C,Q) OR <ENTER> TO END READS? >

This query cycles anytime CMDFILE is ready to read in another file. Any file read in will be appended to any file previously input since the main query prompt. If you want to read in a disk file, respond with a "D". If the file is to be input from tape, respond with a "T". You may quit and return to LDOS by entering an "E". A response of "C" will cancel the input and return you to the main query, thus reinitializing the memory buffer. The <Q> response permits display of a diskette directory. Its syntax is:

Qd

Where d is the drive spec. If you omit the drive spec, the zero drive will be assumed. If you enter an erroneous drive spec, your entry will be ignored. If you enter a drive which is not in your system, the command will time out after about 10 seconds and you will receive another query (2).

If you have read in file(s) and want to begin a writing operation, respond with <ENTER> (i.e. just depress the <ENTER> key without entering any other character).

In order to read in a disk file (response to query (2) with "D"), you will be prompted for the filespec via the query:

ENTER INPUT FILE FILESPEC >

Enter the filespec to begin the read operation. This utility will default the filespec to an extension of /CMD if you leave the file extension blank. If any disk I/O error results, or any disk problem that results in the file not being read to completion, you will be returned to query (2) and no fragment of the file will be added to the memory buffer. A disk file that is reread will properly append any file previously read in.

In order to read in a cassette file, you will be prompted to ready the tape with:

Model I => READY CASSETTE AND DEPRESS <ENTER>
Model III => READY CASSETTE AND DEPRESS <H,L>

Model III users should depress the <H> (1500 baud) or <L> (500 baud) key after preparing the tape for input. If the 1500 baud rate is used, the HITAPE utility must have been previously executed. There is no need to enter a file name. The next program found on the tape will be read. The upper right screen will show flashing asterisks during the load. The letters C, D, or BK may appear if an error is detected. A checksum error during the load will display the message:

TAPE CHECKSUM ERROR DETECTED - REREAD TAPE!

Any previously read in file will not be destroyed. The partial tape load will be ignored and subsequent reads will properly append any previously read in file.

If the file being loaded uses up your machine's memory, the message "OUT OF MEMORY" will appear. Again, no file or files previously read into the memory buffer will be disturbed. You can proceed to save the buffer contents prior to the file that exhausted your machine's memory.

If you attempt to read in a file that is not a 'COMMAND' or 'SYSTEM' file, you will most likely receive the message:

REQUESTED FILE IS NOT A COMMAND OR SYSTEM FILE!

The read operation will cease. You will be returned to query (2).

As a disk file is read, each block detected will produce the message:

BLOCK LOADS FROM XXXX TO XXXX

At the conclusion of the file read, whether from disk or tape, the transfer address (the program address that is jumped to after loading to begin its execution) is displayed as in:

TRANSFER ADDRESS (ENTRY POINT) IS XXXX

At this point, you will recycle to the INPUT FROM DISK OR TAPE query (2).

Query (3):

PROGRAM LOADS FROM BASE ADDRESS XXXX TO XXXX

ENTER NEW BASE ADDRESS OR <ENTER> >

Query (3) will be output if one or more files were input from disk or tape. If you do not need to offset the output file, just depress the <ENTER> key and proceed to query (7). In general, if you are transferring a SYSTEM tape file to disk, and the tape file would ordinarily overlay the operating system's resident program (42ØØH-51FFH), you cannot load the disk file into memory from disk unless it is offset from the resident system. Once in memory, a block move routine can restore it to its original load point.

The Command File Utility will not offset any part of a load module that loads below 42ØØH. This is to permit programs that purposely affect system variables or display messages to the memory mapped video (3CØØH-3FFFH) to load properly.

If you have input a program file that loads below 42ØØH and you are requesting to OFFSET the program, the following message will be displayed:

Program loads below 42ØØH
Enter Address to restrict offset or <ENTER> >

This gives you the option of restricting the offsetting operation below a specified address. For instance, if the program loaded a message directly to the screen, it would have a load block within the range 3CØØH-3FFFH. You can maintain that load block in its original location (to the screen) by entering the lowest address above the screen area as identified in the ADDRESS LOAD LOG in response to the above query. This would provide the offset to any portion of the program originally loading at an address greater than the screen end (3FFFH) and maintain the original load addresses for any block loading into an area below the address entered.

For example, the ADDRESS LOAD LOG begins with:

```
Block loads from 3CØØ to 3C7F
Block loads from 52ØØ to 5282
Block loads from 5283 to 53Ø4
...
```

The entire program module can be offset starting at 52ØØ by entering "52ØØ" in response to the "Enter Address to restrict offset or <ENTER> >" query. In this manner, the load address of the load block addressed to the screen memory will be retained as 3CØØ to 3C7F.

The Command File Utility has been further improved to read the SYS6/SYS and SYS7/SYS ISAM module of LDOS. If CMDFILE interprets the module being loaded as one conforming to the load format of LDOS's ISAM files, then the query:

```
File has ISAM overlays - enter # >
```

will be displayed. If you enter the 2-character overlay number, CMDFILE will read only the desired overlay into its memory buffer. If you respond with "FF", then the entire module will be loaded. There is no attempt in the CMDFILE documentation to explain the LDOS ISAM file structure.

If you want to change the load addresses of the output file (offset it), enter the new base load address. For example, if the existing load is from 43ØØH to 5ØØØH and you want it to load starting at 53ØØH, enter the base address 53ØØH. After entering the new base address, you will receive the query:

Query (4):

```
DO YOU WANT TO ADD THE OFFSET DRIVER ROUTINE (Y,N,E,C)? >
```

A response of "E" will EXIT the program, while "C" will cancel the request and return you to the main query. If you do not want the restoring driver routine appended, respond with "N" and proceed to query (7), otherwise respond "Y". It may not be immediately obvious why you would want to offset a file but not add the appendage. One use would be to change the base address of a relocatable driver routine. Another would be to change the load address of "Tiny PASCAL" files.

Query (5):

```
DO YOU WANT TO DISABLE THE INTERRUPTS (Y,N,E,C)? >
```

A response of "E" will EXIT the program, while "C" will cancel the request and return you to the main query. If you want to disable the interrupts (which should be done if the program does any tape operation or will overlay the disk operating system's interrupt processing routine), then respond "Y", else "N". The next query is:

Query (6):

```
DO YOU WANT TO DISABLE THE KEYBOARD DEBOUNCE (Y,N,E,C)? >
```

A response of "E" will EXIT the program, while "C" will cancel the request and return you to the main query. If you want to disable the keyboard debounce routine (which should be done if the output file will overlay the disk system's debounce routine between approximately 43ØØH and 44ØØH), respond with a "Y", else respond with "N". Query (7) will now be bypassed, as the driver routine appendage dictates the transfer address. Proceed to query (8).

Query (7):

```
ENTER NEW TRANSFER ADDRESS OR <ENTER> TO USE XXXX >
```

If you want to change the transfer address (entry point), you can enter the new address. This is useful when appending two or more files since the transfer address used would default to the transfer address of the last file read in unless otherwise specified. If you had requested the driver appendage, you wouldn't be able to change the transfer address (entry point).

Query (8):

OUTPUT TO DISK OR TAPE (D,T,E,C) OR <ENTER> TO RESTART? >

Again, a response of "E" will EXIT the program, while "C" will cancel the request and return you to the main query. Just depressing <ENTER> will also return you to the main query. Cancellation is available if you do not want to create an output file but rather just want to determine disk files' load addresses.

If you want to create an output disk file, respond with a "D". You will be prompted for the filespec with:

ENTER FILESPEC TO WRITE OUTPUT >

After entering the filespec (remember /CMD will be used as a default extension), the output file will be written to disk (using VERIFY).

If you want to create an output tape file, respond with a "T". You will be prompted to enter the filename with:

ENTER TAPE FILE NAME >

After entering the filename (up to six characters), you will be prompted to ready the cassette. The tape will then be written.

At the conclusion of the disk or tape writing operation, you will receive the query:

Query (9)

MODULE WRITE IS COMPLETE - WRITE ANOTHER (Y,N,E,C)? >

The "E" and "C" responses are as before. A response of "N" will also return you to query (2). If you want to generate an additional output copy, respond with "Y". If you had selected TAPE output, you would be prompted to ready the cassette and another copy would be written using the same file name as was entered, followed by query (9). If you had selected DISK output, you would be returned to query (8) so that additional output files could be written to tape or other filespecs.

TECHNICAL SPECIFICATIONS

Appendage Driver Routine

If you requested an offset to the output file's original base load address, the following routine would be appended to the end of the program provided the new base address exceeds the old base address:

```
ORG     NEWHI+1          ;DRIVER ORIGIN
DI  (OR NOP)             ;INTERRUPTS OFF?
LD      HL,NEWLOW        ;PT TO OFFSET START
LD      DE,OLDLOW        ;PT TO WHERE IT GOES
LD      BC,ENDLOD-BGNLOD+1 ;LENGTH OF MOVE
LDIR                     ;MOVE IT IN PLACE
JP      OLDTRA           ;GO TO ORIG ENTRY PT
```

Where: NEWHI => the highest load address after offset,
NEWLOW => the lowest load address after offset,
OLDLOW => the original lowest load address,
greater than 41FFH
ENDLOD => the original highest load address
BGNLOD => same as OLDLOW
OLDTRA => the original transfer address

If the new base address is less than the old base address (offset towards lower memory), then the driver routine appended would look like this:

```
ORG     NEWHI+1          ;DRIVER ORIGIN
DI  (OR NOP)             ;INTERRUPTS OFF?
LD      HL,NEWHI         ;PT TO OFFSET END
LD      DE,OLDHI         ;PT TO WHERE IT GOES
LD      BC,ENDLOD-BGNLOD+1  ;LENGTH OF MOVE
LDDR                     ;MOVE IT IN PLACE
JP      OLDTRA           ;GO TO ORIG ENTRY PT
```

Where: NEWHI => the highest load address after offset
OLDHI => the original highest load address

Appending two or more files

In order to append (concatenate) two or more files into one contiguous file, keep responding to query (2) with the "D" or "T" indicator depending on where each file resides, in order to read all the desired files into the memory buffer. When the last file has been read in, respond to query (2) by depressing <ENTER> to initiate the output cycle. Note that the transfer address jumped to on initial loading of the concatenated file would be the transfer address detected from the last file input. Thus, if you want to provide control to another address, use query (7) to modify the transfer address to one of your own choosing.

If you also have to offset the concatenated file, it cannot be done during this output writing. Complete the above procedure, thereby creating the appended file. Now reinput the appended file and offset it. This second operation will provide for your transfer address as the control point after the driver routine restores the loaded module to its original load point.

Patching programs

Programs can be patched in a manner similar to LDOS's "PATCH" command. PATCH applies program corrections at the end of a load module so that the corrected bytes will overlay the incorrect bytes during the load process. Once you are made aware of the patch code, assemble it using the Editor Assembler. You may need to employ a series of ORGs and data assembly statements (DEFBs, DEFWs, etc.). The assembled object file can now be appended to the end of the original program. During the load operation of the "patched" program, the original code is first loaded but is then overlayed by your appended "patch" code.

Transferring a disk file to tape

Any load module, written using the format shown in the Technical Information section, File Formats, can be transferred to tape as a SYSTEM file. This feature is especially useful to assembly language programmers developing machine language programs. By using a disk editor/assembler, your assembly language development can proceed using disk I/O. Even programs whose source is too large to load into the text buffer can be assembled in segments and later concatenated into one contiguous file. The file can then be transferred to a tape cassette to create a "master" for duplication. Even if you are not an assembly language programmer, this disk-to-tape feature is very useful for making SYSTEM tape backups of your disk load modules.

Transferring a SYSTEM tape to disk

If you want to employ the tape-to-disk facility, all that is needed is to perform the input from tape and not input a second file. Just use the single file read in to output it to disk. Appending "TWO" files together is not required.

C O N V (CONV/CMD)

The CONV utility will allow you to move files from a Model III TRSDOS diskette onto an LDOS formatted diskette. Two drives are required. The syntax is:

```
CONV :s :d (parm,parm,...,parm)
CONV partspec w/wcc:s :d (parm,parm,...,parm)

:s  is the source drive. It cannot be drive Ø.
:d  is the destination drive.

partspec and wcc are as defined in the Glossary.

The allowable parameters are as follows:

VIS     Convert visible files.

INV     Convert invisible files.

SYS     Convert system files.

NEW     Convert files only if they do not exist on
        the destination disk.

OLD     Convert files only if they already exist on
        the destination disk.

QUERY   Query each file before it is converted.

abbr:   All parameters may be abbreviated to their
        first character.
```

The CONV utility will allow you to move all or groups of files from a Model III TRSDOS disk onto your LDOS disks. Model I owners must have double density hardware to use this utility.

PARAMETERS - (VIS,INV,SYS)

If none of these parameters are specified, all file groups will be considered. Specifying only one parameter will automatically exclude the other two. Thus to move all files except the system files, you would specify both VIS and INV.

PARAMETERS - (NEW,OLD,QUERY)

The NEW parameter is used to move files onto the destination disk only if they do not already exist. The OLD parameter will move only those files that already exist on the destination disk.

Unless you specify QUERY=NO, CONV will ask you before each file is moved onto the destination disk. You should answer the prompt <Y> to move the file, or press <N> or <ENTER> to bypass it.

You may specify a filespec/partspec to be used to determine which files to move. Wildcard characters are also acceptable. Refer to the following examples.

CONV :2 :1

This example will allow you to move all files from drive 2 onto drive 1. You will be asked before each file is moved. If the file already exists on drive 1, you will be asked again before it is copied over.

CONV :1̂ :Ø (VIS,Q=N)

This example will move all visible files from drive 1 onto drive Ø. You will not be asked before each file is moved.

CONV /BAS:2 :Ø (NEW)

This example will move only those files with the extension /BAS from drive 2 to drive Ø. Because the NEW parameter was specified, only those files that do not already exist on drive Ø will be moved.

CONV $$$DATA:1 :2 (OLD)

This example will move those files that have the characters "DATA" as the fourth through seventh letters in their file name. You will be asked before each file is moved, and only those files that already exist on drive 2 will be considered.

F O R M A T

This is the command that allows a diskette to be formatted with cylinders (tracks), sectors, and a directory, so that it may be used by the system. The syntax is:

```
FORMAT :d (parm,parm,parm)

The following optional parameters may be used:


 NAME="name"  The name that will be given to the disk.

 MPW="mpw"    The Master PassWord assigned to the disk.


 SDEN    The density that will be used to FORMAT the
 DDEN    disk, DDEN (double) or SDEN (single).

 SIDES=  The number of sides to be formatted, either
         1 or 2.

 CYL=    The number of cylinders (tracks) that are
         to be placed on the disk, up to 96.

 STEP=   The boot track step rate that will be put
         on track Ø, either Ø,1,2,3.  These values
         represent the step rates in milliseconds:

         5" Ø=6ms, 1=12ms, 2=2Øms, 3=3Ø/4Øms
         8" Ø=3ms, 1=6 ms, 2=1Øms, 3=15/2Øms

QUERY    will prompt you for density, sides, step,
         and number of cylinders.

SYSTEM   will add system information to a previously
         formatted hard disk.

ABS      If specified, will format the disk even if the
         disk is already formatted and contains data.

abbr: QUERY=Q
```

The FORMAT utility is the program that will create the proper information on a diskette so the LDOS system can read and write to that diskette. A disk to be formatted may be blank, or it may have already been formatted. Note that if the FORMAT command is to be used in a JCL file, the disk to be formatted must be blank unless the ABS parameter is specified.

The SYSTEM parameter will be used only with hard disks. Complete documentation on its use will be found in the Hard Disk addendum or in the Start Up Manual.

Typing in the format command with no parameters will prompt you for them in the following order. If the drivespec, disk name or master password were specified on the command line, their prompts will not appear. Additionally, entering any one of the remaining DEN, SIDES, CYL, or STEP parameters will cause format to use the defaults for the other parameters, and you will not be prompted for them.

```
WHICH DRIVE IS TO BE USED ?
DISKETTE NAME ?
MASTER PASSWORD ?
SINGLE OR DOUBLE DENSITY <S,D> ?
ENTER NUMBER OF SIDES <1,2> ?
NUMBER OF CYLINDERS ?
BOOT STRAP STEP RATE <6, 12, 2Ø, 3Ø/4Ø> ?
```

If you are formatting in drive Ø, the following prompt will appear after you have answered the step rate question:

```
LOAD DESTINATION DISK AND HIT <ENTER>
```

The first prompt will ask for the drive number to use. If you are going to format a disk in drive Ø, do not remove the system disk and insert the disk to be formatted until prompted to do so.

The next prompts after the disk number will be for the disk name and master password. These two pieces of information are used by several of the LDOS library commands and utilities. They will be referred to as the Pack ID throughout the manual. You will be allowed up to 8 characters for either entry. Characters used for the password must be either alphabetic or numeric. Using any other characters will cause an error, and the format will abort. Pressing only <ENTER> will use the default values.

The density prompt will always appear on the Model III. It will not appear when using a Model I unless you are using a double density board and one of the driver programs. Pressing <ENTER> in response to this prompt will use the default density value.

The side prompt must be answered <1> or <2> for single or double sided. If you are using double sided, you must have the proper hardware setup as explained in Section I, SYSTEM DEVICES AND DISK DRIVES. Pressing <ENTER> will default to 1 side.

The cylinder prompt will not appear for 8" drives, as they always have 77 cylinders. For 5" drives, any number up to 96 may be entered. Pressing <ENTER> will use the default cylinder value.

The bootstrap step rate is important only if you will be using the disk in drive Ø to boot up the system. Refer to the Section I, SYSTEM DEVICES AND DISK DRIVES for an explanation of the term "step rate". Be aware that too low a step rate may keep the disk from booting.

Before the actual formatting begins, the target disk will be checked to see if has been previously formatted. If it has, the following message will appear:

```
DISK CONTAINS DATA -- NAME=diskname  DATE=mm/dd/yy
ARE YOU SURE YOU WANT TO FORMAT IT ?
```

If the disk contains an incomplete or non-standard format, one of the following messages may appear in place of the NAME=diskname.

```
UNREADABLE DIRECTORY
NON-STANDARD FORMAT
NON-INITIALIZED DIRECTORY
```

You will see the disk's name and date, and can abort the format at this point. Press <N> to abort the format, or <Y> to continue. If you have specified the ABS prompt, you will see this message but will not be prompted to abort the format.

FORMAT parameter default values

The NAME and MPW parameters may be specified in the command line followed by the desired string enclosed in parentheses. If either parameter is specified without being followed by a string, you will be prompted for it before the formatting begins.

Parameters not passed in the format command line will default as follows:

NAME will default to LDOSDISK.

MPW will default to PASSWORD.

DENSITY will use different default values depending on the hardware. Model III will default to double density. Model I Radio Shack interface will default to double density if there is a doubler board and driver program in use. Otherwise, it will default to single density.

SIDES will default to 1 side.

CYLinders will default to the value set with the SYSTEM (DRIVE=,CYL=) command and stored in the system information sectors on drive Ø. If no value has been set, the default will be 4Ø cylinders for the Model III, 35 cylinders for Model I 5" drives, and 77 cylinders for Model I 8" drives.

STEP rate will default to the value set with the SYSTEM (BSTEP=) command, also stored in the system information sectors. If no value has been set, the defaults will be 3Ø/4Ø ms on the Model I, and 6 ms on the Model III.

The QUERY parameter defaults to YES, and you are normally prompted to enter all parameters. If you are sure that the default values will produce the exact format you desire, you may specify the parameter Q=N to bypass all parameter prompts.

The ABS parameter is primarily useful when the FORMAT utility is executed from a JCL file. As explained in the JCL section, an unexpected prompt from an executing program can cause the JCL processing to abort. Using the ABS parameter assures that there will be no prompt from the FORMAT utility to abort the formatting if the target disk already is formatted.

FORMAT cylinder verification

When the formatting begins, you will see the cylinder numbers appear as the necessary information is written to them. After all cylinders are written, format will verify that the proper information is actually on each cylinder. If the verify procedure detects an error, an asterisk and the cylinder number will be shown on the video display. This space will be locked out, so that no files will be written to the defective area. The FREE library command provides a method to see the locked out tracks on a diskette.

The WAIT parameter

The WAIT parameter was not listed in the parameter table because it is not normally used when formatting. It was put in the format command to compensate for hardware incompatibilities when using certain types of disk drives. The only time it should be used is when ALL tracks above a certain point are locked out when verifying.

To use this parameter, specify:

WAIT=nnnn

The value for nnnn will normally be a number between 5ØØØ and 5ØØØØ. The exact value can vary depending on the particular disk drive. It is recommended that a value around 25ØØØ be used at first. This value can be adjusted higher if tracks are still locked out, or lower until the bottom limit is determined.

H I T A P E

The HITAPE utility is for Model III only, and will permit the use of high speed (15ØØ baud) cassette I/O in the LBASIC and CMDFILE programs. The syntax is:

```
============================================================
|  HITAPE                                                  |
|                                                          |
|  No parameters are required                              |
|                                                          |
|  abbr: NONE                                              |
|                                                          |
============================================================
```

*** NOTE ***

This program is for the Model III only!

Due to space constraints and our desire to provide a high level of sophistication through the proper use of interrupt tasks, it was necessary to disable the use of 15ØØ baud cassette loading in the resident LDOS system. We still wanted to have the 15ØØ baud tape capability in the system, so a small utility was added. The utility is called HITAPE/CMD and is invoked by simply typing HITAPE <ENTER> at the LDOS Ready prompt. You may then use 5ØØ or 15ØØ baud tapes in the normal manner. If HITAPE is in when a SYSTEM (SYSGEN) is performed, it will be saved with the configuration file. Both CMDFILE and LBASIC allow the use of high speed cassette only if HITAPE has been executed. If HITAPE has NOT been executed and a 15ØØ baud tape load is attempted, the tape will not load. It may be necessary to depress the <BREAK> key to regain control of the system.

L C O M M

The LCOMM utility is a sophisticated program that provides communications capabilities between two TRS-8Ø systems, between a TRS-8Ø and a Bulletin Board System such as Forum-8Ø, PSBBS, or other similar systems, or between a TRS-8Ø and a large timesharing system such as MicroNET (tm), The Source (tm), or other main-frames. LCOMM provides the capabilities of keyboard send/receive, automatic spooling to a printer through a dynamic memory buffer, and the transfer of files from system to system, without the need for handshaking when operating at 3ØØ baud. For those users interfacing to systems supporting XON/XOFF protocol, LCOMM provides for optional XON/XOFF support using Device Control 1 (DC1) and Device Control 3 (DC3) ASCII controls. The syntax of the LCOMM command is:

```
LCOMM devspec (parm,parm,parm)

  devspec is a valid LDOS active device, usually *CL,
  SET to an RS232 driver.

  Allowable parameters are as follows:

  XLATES=X'fftt'  Translates a send character.

  XLATER=X'fftt'  Translates a receive character.

           ff     Character to translate from.

           tt     Character to translate to.

  NULL=           ON or OFF, the default is ON. If OFF
                  is specified, it will prevent any
                  nulls (ØØ's) from being received.

abbr: XLATES=XS, XLATER=XR
```

Note: You must have established the LDOS keyboard driver (KI/DVR) via the command:

SET *KI to KI/DVR (parms...)

before attempting to enter LCOMM as LCOMM makes extensive use of control and function keys only available with the KI/DVR program.

LCOMM does not "talk" directly to the RS-232 hardware, but rather to a device that has been previously coupled to the RS-232 hardware through an appropriate software driver. This is accomplished with the SET library command and one of the supplied RS232 driver programs. The device LCOMM will interface with is passed as a device specification in the command line.

The device name normally utilized for this purpose is "*CL", an acronym for "Communications Line", although any other device name could be used. However, throughout this section, the *CL device will be used for reference purposes.

It is imperative that the SPOOLer not be in use while using LCOMM since the SPOOLer shares the same task slot as LCOMM (LCOMM has its own spool buffer). It is also useful when receiving large files to pre-allocate the file space with the CREATE library command. This reduces the system overhead while running LCOMM.

LCOMM provides many user options. It interfaces with the user by utilizing the top row of keys as Programmed Function (PF) keys used in concert with the <CLEAR> key - just as the KeyStroke Multiplier (KSM) Filter uses the <CLEAR> key to provide special functions with the A-Z keys. Since most of the top row of keys are used in both their shifted and unshifted form, a brief user menu is provided to aid in jogging your mind until you become familiar with the programmed functions. This menu can be displayed by simultaneously depressing the <CLEAR><8> keys.

In describing the functions of LCOMM, the following conventions will be used:

<CLR> - - WILL REPRESENT THE <CLEAR> KEY.

<CONTROL> WILL REPRESENT THE <LEFT SHIFT><DOWN ARROW> KEYS.

<SH> - - WILL REPRESENT THE <SHIFT> KEY.

< > - - WILL REPRESENT THE ACTUAL KEY THAT IS TO BE USED.
SUCH AS <!>, <#>, <%>, <6>, <9> etc.

Some of the PF keys are used to select logical devices so as to be able to turn them on or off - indicating whether the device should be acceptable for I/O. Other PF keys control the selection of parameters associated with filespecs. Still others control additional functions which aid in the interface between two communicating users.

You may find the need for characters not normally visible on the TRS-80 keyboard. LDOS provides all ASCII characters in the range 00-127. Accessing these characters is described completely in the KI/DVR section.

LCOMM will generate a modem break (extended null) if you press the <BREAK> key. To produce a normal TRS-80 "break" code (X'01'), press <CTL><A>. A local CLEAR SCREEN function is also available, and can be requested by pressing the <CLR><SH><(> keys.

LCOMM uses all of available memory below (HIGH$) for dynamic buffering of certain device I/O. The amount of buffer space devoted to each device dynamically expands and shrinks according to how fast characters are sent to the device and how fast the device can accept them. Each buffer is essentially a variable length First-In-First-Out (FIFO) storage compartment. The amount of free space available for the buffers is noted in the bottom line of the menu display. When this free space shrinks to less than 2K (<2048 characters), a warning message is displayed and an X-OFF is automatically sent to the Communications Line. This is quite useful when receiving a file from a system that supports handshaking. A more detailed discussion on the use of handshaking will be presented in the "Communicating With Other Computers" section.

Using the PF keys

*KI .... <CLR><1>

This designates the keyboard device. When LCOMM is first entered, the *KI is in an "on" state. If you desire to turn it off to avoid accidentally brushing the keyboard while you are transmitting a file, you can turn off the keyboard by <CLR><1> followed by a <CLR><->, which indicates the "off" function. While the *KI is off, all PF keys are still active.

*DO ... <CLR><2>

This designates the video display device. When LCOMM is first entered, the *DO is in an "on" state. If you desire to turn it off when, for instance, the printer has been activated, a simple <CLR><2> followed by a <CLR><-> will perform the requested function. The video display will be re-activated by a <CLR><2> followed by a <CLR><:>.

*PR ... <CLR><3>

This PF key references the printer device. When LCOMM first initializes, this device is off. If you want to direct the communications transactions to your printer, do a <CLR><3> followed by a <CLR><:>. Output being routed to the printer is fully buffered through dynamic memory buffers. Therefore, it is not necessary for your printer to be capable of operating at the communications line transmission rate. Even after transactions cease, you may discover the printer still typing away.

*CL ... <CLR><4>

This PF key references the communications line device. LCOMM initializes with *CL in an ON state. You may wish to temporarily block output from being sent to the *CL so as to be able to review a file prior to transmission. Depending on your PF switch setup, if you go to a half-duplex mode (DUPLEX-ON) after turning off the *CL, you could perform a File Send (FS) which would display the file to your screen without actually sending it to the communications line. Of course, if the distant end attempted to send to you while you had the *CL off, you would not receive their transmission.

*FS ... <CLR><5>

This designates the "File Send" device. With it, you can cause a file to automatically be transmitted to the distant end. This PF key works in concert with a number of other keys. Other PF keys exist to open a designated file, rewind a designated file, position to the end of a designated file, and close a designated file. As with the other devices discussed, the functions available to this File Send device are activated by the two-step process of first depressing <CLR><5> followed by some other PF key appropriate to the intended function. Specific details will be presented as the other PF keys are discussed.

*FR ... <CLR><6>

This is the device to be used for either receiving a file being transferred to you, or for making a file copy of the communications line dialogue. This device will also be used to download from a bulletin board system. All of the PF keys available to the FS device are also available to the FR device. These will be discussed later. This device may be turned on or off by control characters received from the communications line if the HANDSHAKE switch is on. The characters received will be put in a memory buffer, and may be written to disk with the DTD function.

DTD ... <CLR><7>

The (DTD) Dump To Disk is used to write the memory buffer used with FR to the disk. DTD may be turned on before or after a file has been received. If turned on before, the file will be written to disk as it is being received. This will be necessary if the file will exceed the size of the FR memory buffer. When LCOMM first initializes, DTD is set to ON. When an FR RESET is performed, DTD is set to its OFF

mode. Model I users may want to set DTD to OFF until an entire receive file (FR mode) has been received to guard against occasional dropping of a character during disk I/O. On the Model III, it will be necessary to wait until the entire file is received before turning on DTD if you are using floppy disks and any baud rate above 3ØØ.

MENU ... <CLR><8>

This PF command will display a menu to the screen. It looks like this:

```
  *    *                                      *         * ØØ
DUPLX ECHO  ECOLF ACCLF REWND PEOF  DCC CLS 8-B       HNDSH EXIT
==1== ==2== ==3== ==4== ==5== ==6== =7= =8= =9= ==Ø== ==:== ==-==
 *KI   *DO   *PR   *CL   *FS   *FR  DTD ??? ID   RES   ON    OFF
  *     *          * *          *

FR File: DOWNLOAD/TXT:3  MEMORY: 36K
```

Notice that the display will be left to right and in the relative positions of the keys which are used for the functions. This is not intended to be a complete menu, it is like having a built in "quick reference card". The <CLR><8> may be executed at anytime. The screen display will be altered to display the menu (scrolled 5 lines), but no data will be lost as LCOMM will still be buffering the incoming characters. The buffered characters will be displayed after the menu is placed on the screen.

The menu display will show which devices and functions are active, as well the amount of available memory. The asterisks above and below the PF labels will indicate whether the function is active or not. Those above the labels are for the shifted PF functions; those below for the unshifted functions. *CL is shown with two asterisks, denoting that it is capable of both input and output. A single asterisk under a device indicates single direction I/O. If handshake is active, the auto X-OFF character selected will be shown in hex. Also, any FS or FR file that has been ID'd will have its filespec displayed.

ID ... <CLR><9> use with <CLR><5> (FS) and <CLR><6> (FR)

The ID function is used with either FS or FR to designate and open the desired file. If you are going to receive a file, you will perform an FR-ID by depressing <CLR><6> followed by a <CLR><9>. You will receive the prompt:

FILE NAME:

You should enter the file specification of your choice. The system will then open the file for receiving and await your next command. At this point the file is open and ready but is NOT turned ON (see ON, <CLR><:>).

If a receive file is already open, the system will ignore your ID request and issue the warning message:

FILE ALREADY OPEN

This is to guard against inadvertantly issuing another FR-ID before you have closed the last file received. The same protection is available to FS. Only one FS file can be open at a time. You should close your send file after transmitting it.

RESET ... <CLR><Ø>

The RESET PF key performs the function of closing either a receive file or a send file. A receive file must be closed so its directory is updated. Don't forget to turn "off" a receive file before closing it. You also must close a receive file to be able to receive a subsequent file. If a device is reset, its buffer is cleared.

ON ... <CLR><:>

This PF key is used with one of the six previously mentioned device PF keys to turn "on" the device. For instance, once you have defined a receive file to the system with the FR-ID functions, you must do a FR-ON before any data will be written to it. Before a send file will start transmitting after the FS-ID, the FS-ON must be done.

OFF ... <CLR><->

This PF key performs the opposite function of the ON key. It is used in conjuction with any of the device keys to turn off the keyboard, video screen, printer, communications line, file send, or file receive. Just remember that like the ON function, the OFF function is performed in two steps. If you want to stop the receive file from further receiving, you FR-OFF by <CLR><6> followed by a <CLR><->.

The program function keys also have second functions programmed for them. These additional functions are accessible by depressing the <CLR><SH> keys along with the specified PF key. The following explains the capabilities of these second functions.

DUPLEX ... <CLR><SH><!>

This PF key is the full-duplex/half-duplex switch. In the LCOMM ON/OFF arrangement, DUPLEX-ON designates half-duplex operation. In this mode, your video display screen will display your key entries or file transmission as it is taking place. The DUPLEX-OFF mode is a full-duplex operation. Your video display will display what you send only if the distant end echoes back to you what it receives from you. Although it may be convenient to operate half-duplex (DUPLEX-ON) when communicating with another TRS-8Ø, it may be more useful for one of the TRS-8Øs to play HOST and operate in half-duplex with echo to the distant end while the distant end is full-duplex (DUPLEX-OFF). This will become more evident under the discussion of file transmission.

LCOMM initializes in the full-duplex or DUPLEX-OFF state. The PF key is also one that operates in concert with the ON and OFF keys. If you want to go to half-duplex after LCOMM initializes, you must depress the <CLR><SH><!> keys followed by the <CLR><:> keys.

ECHO ... <CLR><SH><">

This will provide the function normally undertaken by a host system. If ECHO-ON is specified, everything received from the communications line will be re-transmitted. This mode is desirable if the distant end must operate full-duplex and has no "local" copy. A caution to be observed is that if both ends are set for ECHO-ON, then the first character sent will be echoed back and forth indefinitely - at least until one end turns ECHO-OFF.

ECHO-LINEFEED ... <CLR><SH><#>

The echoing of a line feed is the desired mode if the distant end is a dumb hard copy terminal that has its own local copy but expects the line feed to be sent by

the host computer. With ECHO-LF-ON, anytime a carriage return is received from the communications line, a line feed character will be sent back, and a line feed will be added to any carriage return transmitted.

ACCEPT-LINEFEED ... <CLR><SH><$>

LCOMM normally ignores the first line feed received after a carriage return. If this function is turned on, all line feeds will be accepted. This may be desirable if the distant end is another TRS-8Ø.

REWIND ... <CLR><SH><%>

The REWIND function works only with the *FR and *FS devices (FILES). It is used to rewind either file back to its beginning. For instance, say during the transmission of a file, the transmission is aborted prior to its completion. In order to resend it, it should be rewound to its beginning so the NRN pointer is pointing to the first record. REWIND performs that function.

PEOF ... <CLR><SH><&>

This function is used to position a file to its end. A common use would be to append to an existing receive file. If you open a file for receiving by means of the FR-ID and then do a FR-PEOF, the existing receive file would NOT be overwritten with the new data, but rather the new data received will be concatenated to the old data.

DCC ... <CLR><SH><'>

The DCC (Display Control Characters) function will force a display of any character received that has a value less than an X'2Ø' as a two digit hexadecimal number surrounded by braces. This will be useful if you suspect that unwanted control characters are being received.

CLS ... <CLR><SH><(>

The CLS (Clear Local Screen) function will erase the contents of the screen without transmitting any character to the communications line. The cursor will be positioned at the upper left hand corner of the screen.

8-BIT ... <CLR><SH><)>

The 8-BIT switch is used to indicate that all 8 bits of each character received from the communications line are valid. If it is not turned on, bit 7 is stripped from each character received. Do not specify this switch unless the RS-232 word length was set to 8.

HANDSHAKE ... <CLR><SH><*>

If the handshake switch is turned on, LCOMM will respond to the following four control codes received from the communications line:

X'11' DC1 - Resume transmission (standard X-ON character)
X'12' DC2 - *FR ON
X'13' DC3 - Pause transmission (standard X-OFF character)
X'14' DC4 - *FR OFF

The DC2 and DC4 characters function identically to the *FR ON and *FR OFF programmed function keys. DC3 causes transmission through the *CL device to be halted until a DC1 is received. Reception is not affected. You can override a DC3 with the *CL ON keyboard command.

HANDSHAKE may also be turned on with the sequence <CLR><SH><*>, followed by any non-PF key (rather than the ON key). If this is the case, any time LCOMM sends the specified character it will pause transmission until a DC1 is received or a *CL ON is issued directly from the keyboard. Typically, the <ENTER> key would be specified so that line-at-a-time transmission could occur with automatic stopping at the end of each line.

EXIT ... <CLR><SH><=>

This PF key is used to return to the LDOS command level. It does not require any ON or OFF. It is a stand-alone key sequence. Prior to exiting LCOMM, the *FR device is checked to see if an open file exists. In the event that one does, it will be closed before the exit to LDOS is made. This little feature will protect against your inadvertant exit without overtly saving an open receive file.

USAGE TIPS

Some TRS-8Ø Bulletin board systems permit the reception of graphics characters. In order to be able to accept these graphics, the RS-232 driver will have had to be initialized at 8-Bit word length and the 8-Bit mode in Lcomm will have to be used (<CLR><SH><)> followed by <CLR><:>).

The beginning LCOMM user may find it useful to make up a tape containing each key's function and place the tape directly above the keys. Avery or Pres-A-Ply self-adhesive removable labels may be used for this purpose. Any other pressure sensitive label will suffice. The labels can even be typed to provide a neater appearance. Your keys should be labeled as follows:

KEY	UNSHIFTED	SHIFTED
1	*KI	DUPLEX
2	*DO	ECHO
3	*PR	ECHO-LF
4	*CL	ACCEPT-LF
5	*FS	REWIND
6	*FR	PEOF
7	DTD	DCC
8	???	CLS
9	ID	8-BIT
Ø	RESET	not used
:	ON	HANDSHAKE
-	OFF	EXIT

Communicating With Other Computers

Two examples will be given to show how LCOMM can be used to communicate with other computers. The first example will describe operations when communicating with a main frame. The second example will describe how LCOMM can be used to communicate between two TRS-8Ø's.

When communicating with a main frame computer, it will not generally be necessary to change the default device or function settings when entering LCOMM. Most main frames operate in the host mode, and will provide echo functions for you. You must be sure, however, to have specified the RS-232 parameters to match those expected by the main frame. To download a file, use the following procedure:

Type in the command to cause the main frame to list the file, but do not press the <ENTER>.

ID your receive file by pressing <CLR><6> followed by <CLR><9>. Type in the filename in response to the prompt.

Type in <CLR><6> followed by <CLR><:> to open the receive buffer. If the file you wish to receive will be larger than your available memory buffer, you should press <CLR><7> followed by <CLR><:> at this time. This will cause the file to be written to the disk as it is being received.

Press <ENTER> to start the file listing.

When the listing is complete, type in <CLR><6> followed by <CLR><-> and if you have not already done so, <CLR><7> followed by <CLR><:> to write the file to disk. When the write is complete, type <CLR><6> followed by <CLR><Ø> to turn off the FR and DTD and close the receive file.

Most main frame computers and some bulletin board systems support XON/XOFF handshaking. This mode is used for transmitting files to the host as a series of single lines. Each line is transmitted to the host while your machine pauses until the host acknowledges receipt by transmitting an XON to you thus resuming your transmission with the next line. To accomplish this, your file must have each line terminated with some line terminating character (usually an ENTER). As hosts generally have a maximum line length that they accept, you should be sure that your file does not have any lines exceeding the host's limit. The upload can follow this series of steps:

Designate the file that you want to send by entering <CLR><5> followed by <CLR><9> and entering its filespec in response to the ID query.

Turn on the handshake mode by entering <CLR><SH><*> followed by <ENTER> (assuming that the line terminating character in your file is <ENTER>). Open the file at the host end and ready it for receiving by whatever command process your host requires. Then turn on your file send by <CLR><5> followed by <CLR><:>. You will note that one line of your file will be transmitted and then your machine will pause. Once the host sends you the XON, the next line of the file will be automatically transmitted. If you are operating in half-duplex, you may see the entire file displayed without any pauses as the file is read from your disk and is buffered awaiting transmission.

When the transmission is complete, turn off the handshake mode by <CLR><SH><*> followed by <CLR><->. Then close up the file at the host end by whatever command process the host accepts. You may then close your file send by entering <CLR><5> followed by <CLR><Ø> which will turn off the FS and close the file.

If at any time you want to force the transmission to resume after a line is ended, you may turn the *CL back on by entering <CLR><4> followed by <CLR><:>. This is also explained under handshake.

When using LCOMM to communicate between two TRS-8Ø's, it will be necessary for one end to turn on half duplex <CLR><SH><!> followed by <CLR><SH><:> and echo <CLR><SH><"> followed by <CLR><:>. If files are to be sent and received, this should be done at the

RECEIVING end. To transfer files, use one of the following two methods. If the receiving end's system is subject to character loss during disk I/O (some TRS-8Ø Model I machines) or you are operating above 3ØØ baud on a Model III, then Method A should be used. If your system can dynamically write to disk during transmission, Method B should be chosen.

METHOD A

The sending end should do a <CLR><5> followed by a <CLR><9> and enter in the filespec to be sent.

The receiving end should do a <CLR><6> followed by a <CLR><9> and enter in the filespec to be received. The dump-to-disk (DTD) must be turned off by entering a <CLR><7> followed by a <CLR><->. This will buffer the file in memory as it is being received. If the sending end supports XON/XOFF handshaking (is it another LCOMM user?), then you should engage handshake by entering <CLR><SH><*> followed by <CLR><:>

When both ends are ready, the receiving end should do a <CLR><6> followed by <CLR><:>, and the sending end should then do a <CLR><5> followed by <CLR><:>.

If your free buffer space decreases to less than 2K during receipt of the file, a warning message will be issued and an X-OFF will automatically be sent to the sending end. Transmission from the sender should cease. Once it does, dump the receive buffer to disk by turning on DTD with <CLR><7> followed by <CLR><:>. You can observe the increase in available free buffer space by displaying a menu as the buffer is written to disk. Once ample free space is available, turn off the DTD with <CLR><7> followed by <CLR><->. You then can manually restart the sender's file by transmitting an X-ON from your keyboard with <CONTROL><Q> (note the CONTROL is obtained by simultaneous depression of LEFT SHIFT & DOWN ARROW).

After the file is totally received, the receiving end should do a <CLR><6> followed by a <CLR><->. The last receive buffer should be dumped to disk by turning on DTD with <CLR><7> followed by <CLR><:>. The sending end should do a <CLR><5> followed by a <CLR><-> then a <CLR><5> followed by a <CLR><Ø>.

When the receiving end file is finished writing to the disk, close the file by resetting the FR with a <CLR><6> followed by <CLR><Ø>. This will do a FR-OFF, a DTD-OFF followed by a close of the file just received.

Method B

The sending end should do a <CLR><5> followed by a <CLR><9> and enter in the filespec to be sent.

The receiving end should do a <CLR><6> followed by a <CLR><9> and enter in the filespec to be received. The dump-to-disk (DTD) must be turned on by entering a <CLR><7> followed by a <CLR><:>. It may already be ON. Its state can be ascertained by obtaining a menu and noting if an asterisk is positioned beneath its key in the menu display.

When both ends are ready, the receiving end should do a <CLR><6> followed by <CLR><:>, and the sending end should then do a <CLR><5> followed by <CLR><:>. This will turn on the receive and send files.

After the file is totally received, and the file is finished writing to the disk, close the file by resetting the FR with a <CLR><6> followed by <CLR><Ø>. This will do an FR-OFF, a DTD-OFF followed by a close of the file just received.

L O G (LOG/CMD)

LOG is a program that will log in the directory track and number of sides on a diskette. The syntax is:

```
LOG :d

:d is any currently enabled drive
```

The LOG utility will provide a way to log in diskette information and update the drive's DCT. It will perform the same log in function as the DEVICE library command, except for a single drive rather than all drives. It will also provide a way to swap the drive Ø diskette for a double sided diskette.

LOG :Ø will prompt you and allow you to switch the drive Ø diskette. The "LOG :Ø" command must be used when switching between double and single sided diskettes in drive Ø. Otherwise, it is not needed.

NOTE: Double sided disks cannot be used to boot the Model I system.

P A T C H

The LDOS PATCH utility is used to make minor changes or repairs to existing program or data files. The syntax is:

```
PATCH filespec1 USING filespec2 (YANK)
PATCH filespec USING (information in patch format)

 filespec1  Any valid filespec. The default extension
            will be /CMD.

 filespec2  Any valid filespec for a "PATCH format"
            file. The default extension will be /FIX.

 YANK       Will remove the PATCH specified by
            filespec2 from filespec1. The PATCH
            to remove must have been in the
            X'nnnn' type format.

abbr: NONE
```

The PATCH utility will allow you to change information in a file in one of two ways. If the file is in load module format (/CMD type files), it may be patched by memory load location. Any type of file may be patched by the direct disk modify method.

A patch is applied to a file either by typing in the patch code directly from the command line or by creating an ASCII file consisting of the patch information.

Either the BUILD library command or a word processing program such as SCRIPSIT may be used to create PATCH files, as long as the file is a pure ASCII file (with SCRIPSIT use the S,A type of save). Also, SCRIPSIT sometimes leaves extra spaces after the last carriage return in a file. To prevent this, position the cursor just after the last carriage return and do a delete to end of text to remove any extra spaces.

It is desirable to use some logical method of naming patch files. The filename of the patch code file could be followed by a letter or a number that would be advanced as different patches become available for the same program. For example, BASIC1/FIX, BASIC2/FIX, SYS7A/FIX and SYS7B/FIX. Although not required, it is strongly suggested that all patch code files use the extension /FIX. This will make it easier to use these files as that is the default file extension that the PATCH utility will use.

PATCH LINE SYNTAX:

For PATCH to work properly, a definite structure and syntax must be observed when creating the file. All lines in a patch file must start with either a period or one of the three patch code type identifiers. Refer to the following:

A period indicates that the line is a comment line, and should be ignored by the patch utility. Comments in patch files are very useful for documenting the changes you are making.

The actual patch code lines will start in one of three ways:

X'nnnn'=
Drr,bb=
Lnn

The Lnn line is used to identify a particular library command module, and is not normally needed by the user.

The X'nnnn'= and Drr,bb= are use to identify the patch line as either a patch by memory load location or a direct disk modify patch, respectively. Information following the = sign will be the actual patch code. It must be entered in one of two ways:

It may be entered as a series of hexadecimal bytes separated by a single space.

It may be entered as a string of ASCII characters enclosed in quotes.

No matter which method is used, there is never a space left between the = sign and the start of the patch code.

LDOS PATCH MODES

X'nnnn'=nn nn nn nn nn nn .....
X'nnnn'="String"

This type of patch will patch a file by memory load location. The patch code will be written into a load module added to the end of the file being patched. This ending module will then load with the program and overlay or extend the code at X'nnnn', where nnnn is the memory load address for the patch code. The patch code can be entered either as hexadecimal bytes, or may be represented as an ASCII string. It must be noted that this patch mode will extend the disk file, even if all of the patching is to the "inside" of the program. Because this type of patch will merely be added to the end of the file to be patched, it may later be removed with the YANK parameter.

Drr,bb=nn nn nn nn nn nn .....
Drr,bb="String"

This is the direct disk modify patch mode. The rr represents the record number in the file to be patched, and the bb is the byte in that record where the patch is to begin. Again, the actual patch code can be either hexadecimal bytes or an ASCII string. This type of patch line does not extend the file and is applied directly to the record of the file. Because no identification of the existence of this patch will be placed in the file, this type of patch cannot be removed by the YANK parameter.

The LIST library command with the (HEX) parameter can be used to display a file, showing the record number and the offset byte. This is an easy way to find the location in the file you wish to patch. Be aware that the first record in a file will be record Ø, not record 1.

Lnn

This format is the indicator that the patch code that follows will be to either the SYS6/SYS or SYS7/SYS library command module. The "Lnn" represents the binary coded location of the desired overlay in the SYS module. The patch code that follows will be in either the X'nnnn' or Drr,bb format.

NOTE: This type of PATCH should not normally be created by the user. Any necessary patches to library commands will be issued by Customer Service.

(YANK)

The patch (YANK) parameter will allow you to remove patches applied with the X'nnnn' format. The following rules will be in effect:

1) The filespec of the patch to YANK must be identical to the filespec used when the patch was applied.

2) If YANK is used without a filespec, no patch will be removed.

3) DO NOT PATCH A FILE MORE THAN ONCE USING THE SAME FILESPEC FOR THE PATCH FILE! It will be impossible to YANK the second patch from the file.

Here are some examples that will show the different patch formats.

```
PATCH BACKUP/CMD:Ø USING SPECIAL/FIX:1
PATCH BACKUP SPECIAL
```

These commands would produce identical results. The default file extensions are /CMD for the file to be patched, and /FIX for the file containing the PATCH information. The patch information in SPECIAL/FIX might look like this:

```
.SPECIAL PATCH FOR MY BACKUP SYSTEM ONLY!
X'6178'=23 3E 87
X'61AØ'=FF ØØ ØØ
```

This is an example of a patch using the X'nnnn' load location format. Note the comment line in the patch code file. This line will have no effect on the patch.

```
PATCH SYS2/SYS.PASSWORD USING TEST/FIX
PATCH SYS2/SYS.PASSWORD TEST
```

Note the abbreviated syntax of the second example. The USING and default /FIX extension are not necessary. The information in the patch file TEST/FIX might look like this:

```
.This will modify the SYS2 Module
DØB,49=EF CD 44 65
DØB,52=C3 ØØ ØØ
.EOP
```

This is an example of the direct patch mode. It will patch the specified record and byte in the file SYS2/SYS. There are 2 comment lines in this patch file. Neither will have any effect on the patch.

```
PATCH SYS6/SYS LIB1
```

This command will patch the SYS6 Library module. The patch file LIB1/FIX might contain the following information:

```
L54
X'52Ø8'=32 2Ø DE AF ØØ C3 66 ØØ
```

This patch is in the memory load location mode. Library patches may also be done with the direct disk modify mode.

Patching with the command line format

Applying a patch from the command line uses the same formats for memory load location and direct disk modify already discussed. A library mode patch may not be done from the command line. It is also possible to specify more than one line of patch code from the command line. This is done by placing a colon (:) between the lines of patch code. Refer to the following examples.

```
PATCH MONITOR/CMD:Ø (X'E1ØØ'=C3 66 ØØ CD Ø3 4Ø)
```

This command would patch the file MONITOR/CMD, creating a load module to replace the 6 bytes starting at X'E1ØØ' with the patch code specified in the command line. Since there is no filespec used for the patch code, the name CLP (Command Line Patch) will be assigned to the patch code. You may use this name if you wish to YANK the patch at a later date. However, if more than one command line patch is applied, only the first one can be yanked.

```
PATCH MONITOR/CMD:Ø (DØ1,13=4C:DØ2,3E=66)
```

This command would patch the file MONITOR/CMD in two places. It uses the direct mode to apply the patches to the file's disk sector 1, relative byte 13, and disk sector 2, relative byte 3E. Note the colon (:) separating the two patch lines.

P D U B L / C M D

PDUBL is a disk driver program for use with the Model I, 5" drives, and a double density modification board other than the Radio Shack board. The syntax is:

```
PDUBL

No parameters are required.
```

This command loads a special disk driver program which allows you to use a double density hardware modification to read, write, and format double or single density 5" disks with the Model I. Before buying a double density board, please check with the manufacturer or LDOS Support to assure compatibility with the LDOS PDUBL driver.

If you have a doubler installed, after you give this command, you can use either single or double density disks in any of your 5" disk drives. LDOS will automatically recognize whether you have a single or double density diskette in a drive, and react accordingly. Once you have installed the PDUBL driver, you will see the prompt "Single or Double density <S,D> ?" appear after you enter the disk name and master password during the disk FORMAT utility. Answer this prompt by pressing the <D> key to create a double density diskette or <S> to create a single density diskette. Pressing <ENTER> for this prompt will default to double density.

PDUBL also includes support for double-sided 5" drives. Both sides of the diskette are treated as a single volume. The drives and cable must be set up correctly for this feature to work.

The PDUBL driver is loaded into high memory and protects itself by lowering the value stored in the HIGH$ memory pointer. Logical drives Ø-7 are set up to use this driver in place of the normal LDOS single density driver. You can use the SYSTEM (SYSGEN) command to save the driver in your configuration file, to be loaded automatically every time you boot. Be sure that any application programs you are using respect the HIGH$ pointer.

Please note that you CANNOT boot up on a double density LDOS diskette when using a doubler. You may, however, boot up on a single density diskette and exchange it for a double density diskette as soon as the bootstrap operation has finished.

R D U B L / C M D

RDUBL is a disk driver program for use with the Model I, 5" drives, and the Radio Shack double density board. The syntax is:

```
RDUBL

No parameters are required.
```

This command loads a special disk driver program which allows you to use the Radio Shack double density hardware modification to read, write, and format double or single density 5" disks with the Model I.

If you have a doubler installed, after you give this command, you can use either single or double density disks in any of your 5" disk drives. LDOS will automatically recognize whether you have a single or double density diskette in a drive, and react accordingly. Once you have installed the RDUBL driver, you will see the prompt "Single or Double density <S,D> ?" appear after you enter the disk name and master password during the disk FORMAT utility. Answer this prompt by pressing the <D> key to create a double density diskette or <S> to create a single density diskette. Pressing <ENTER> for this prompt will default to double density.

RDUBL also includes support for double-sided 5" drives. Both sides of the diskette are treated as a single volume. The drives and cable must be set up correctly for this feature to work.

The RDUBL driver is loaded into high memory and protects itself by lowering the value stored in the HIGH$ memory pointer. Logical drives Ø-7 are set up to use this driver in place of the normal LDOS single density driver. You can use the SYSTEM (SYSGEN) command to save the driver in your configuration file, to be loaded automatically every time you boot. Be sure that any application programs you are using respect the HIGH$ pointer.

Please note that you CANNOT boot up on a double density LDOS diskette when using a doubler. You may, however, boot up on a single density diskette and exchange it for a double density diskette as soon as the bootstrap operation has finished.

R E P A I R (REPAIR/CMD)

REPAIR is a utility program to update and correct information on certain types of diskettes to make them usable by LDOS. The syntax is:

```
============================================================
|  REPAIR :d (ALIEN)                                       |
|                                                          |
|  :d  is any currently enabled drive.                     |
|                                                          |
|  abbr: NONE                                              |
|                                                          |
============================================================
```

REPAIR is a program that will perform the following functions.

1) Update the DAM (Data Address Mark) for the directory track to an X'F8'.
2) Read enable DIR/SYS.
3) Check and correct the excess cylinder byte.
4) Set the grans/cylinder byte (GAT + X'CC').
5) Strip the high bit from disk track Ø, sector Ø, byte 3 (Directory track byte).
6) Write LDOS system information sectors onto the disk.

Before explaining what each of the above functions means to the user, it should be noted when the REPAIR command must be used.

On the Model III, it must be used to read any non-LDOS disk created on the Model I, or any Model I LDOS earlier than 5.Ø.2. Disks created under other Model III operating systems may need to be repaired before being read. Note that TRSDOS 1.2 and 1.3 disks should NEVER be repaired. Use the CONV utility to copy programs from them. The different operating systems and formats that can be read by Model III LDOS are listed in Section I, GENERAL INFORMATION.

On the Model I, it will not generally be necessary to REPAIR a disk to read it. However, other operating systems may mark the location of the disk's directory track in different manners. If you are having trouble reading a disk, the REPAIR command may be necessary. Again, the GENERAL INFORMATION section will list all of the different operating systems and formats currently readable by LDOS.

REPAIR functions

There are two types of DAMs (Data Address Marks) on every diskette - one to mark a data track, and another to mark the directory track. The DAM used to mark the directory track varies between operating systems, and may also be dependent on the computer hardware you are using. With LDOS, this DAM has been standardized to be the same on any LDOS diskette. The REPAIR command will change the directory DAM of the target disk to match the LDOS standard. More on the subject of DAMs may be found in the technical section.

Your disk directory contains information on the space allocated on the disk, as well as the names of your disk files. With LDOS, the directory can be opened as a file called DIR/SYS. The REPAIR command will correct the protection level of the DIR/SYS file on non-LDOS diskettes to allow them to be used in the same manner.

LDOS keeps certain information in the directory about the number of cylinders on a diskette, as well as how much space is available on each cylinder. The REPAIR command will update this information on non-LDOS diskettes. This will be necessary if you will be attempting a mirror image from a non-LDOS to an LDOS disk.

The location of the directory cylinder is stored on cylinder Ø. Certain operating systems store this byte in a non-standard manner. The REPAIR command will correct this so the disk may be read by LDOS.

The LDOS system information sectors on cylinder Ø contain a great deal of information. The REPAIR command will place these sectors on a non-LDOS diskette.

After the repair is complete, you should be able to copy any files off of the repaired disk.

I M P O R T A N T

Once a disk is repaired by LDOS, it may not be readable by the operating system that created it. This is due to the directory DAM change. It is recommended that the REPAIR be used on a backup copy of the disk, if at all possible.

T W O S I D E / C M D

This driver program will allow the use of single density, double sided disks with the Model I. The syntax is:

```
TWOSIDE

No parameters are required.
```

TWOSIDE/CMD is a driver program that will allow the use of double sided disk drives with a single density Radio Shack interface. For double density modification owners, the RDUBL and PDUBL programs include the necessary double sided support for both double and single density diskettes.

To execute the program, simply type in TWOSIDE. The driver will relocate itself into high memory. It may be stored in a configuration file and be loaded during the booting sequence with the SYSTEM (SYSGEN) library command.

Briefly, the restrictions on the use of TWOSIDE are as follows:

1) A single sided disk must be use to boot the system. It may be exchanged after booting is complete. To switch to a double sided disk in drive Ø, use the LOG/CMD utility program to log in the double sided disk.

2) No more than 3 drives, double or single sided, will be allowed (logical drives Ø - 2). Logical drive 3 will automatically be disabled.

3) The disk drives must be internally set to be treated as a single drive. The drive cable must be the type without pins missing in the drive connectors - specifically pin 32, the side select lead. A standard Radio Shack disk drive cable will NOT work. The drive vendor or manufacturer should provide information on the internal drive configuration.

4) Double sided diskettes used must have been formatted with LDOS.

When using double sided disks, LDOS treats both sides of the diskette as a unit - that is, there is only one directory, and files may span sides. The same numbered track on the front and back sides of the disk is treated as a single cylinder, starting on the front and continuing around to the back. Thus an 8Ø track double sided diskette has 8Ø cylinders (numbered Ø - 79), not 16Ø. In effect, this means there are two of every track number - one on the front and one on the back of the diskette.

C O P Y 2 3 B / B A S

This utility is provided to copy files from Model I TRSDOS 2.3B to an LDOS disk. The syntax is:

```
LBASIC RUN"COPY23B"
```

Since this is a BASIC program, LBASIC/CMD must be on a disk in the system. Also, Model III users will have to use the REPAIR utility on the 2.3B disk before using COPY23B.

When the program starts, you will be prompted to enter the source and destination file names. The source name is the name of the file on the 2.3B disk. The destination file is the name you wish to give the file on the LDOS disk. These two names can be identical except for the drive number, and usually will be. If the file is password protected on the 2.3B disk, you must use the proper password in the source file name.

After both names are entered, the information will be read from the 2.3B disk and written to a new file on the LDOS disk. When the copy is complete, the BASIC "Ready" prompt will appear. At this time, you may do a CMD"S" to return to LDOS Ready, or type in RUN to copy another file.

J O B L O G (JL/DVR)

This driver program will establish the LDOS Joblog device. The syntax is:

```
SET *JL TO JL/DVR USING filespec/devspec

filespec/devspec is the file or device to be sent the
Joblog information.

abbr: NONE
```

The JL/DVR program will establish the LDOS Joblog device (*JL). Once set, a log of all commands entered or received will be sent to the specified file or device, along with a time stamp. Note that the time stamp will be determined from the setting of the system's real time clock (see the TIME library command). If a filespec is used, the default extension will be /JBL.

Setting *JL will use high memory. The RESET *JL command will terminate the JobLog, and close any associated disk file. However, if *JL is set again, a new high memory allocation will be made.

To view the contents of a JobLog disk file, you must first RESET *JL, so the file will be closed. You may wish to add a trailing exclamation point "!" to the end of the filespec, so that constant EOF maintenance will be invoked (see the filespec definition in the GLOSSARY). The LIST library command will allow you to list the contents of the file to the screen or to the printer.

Note that if an existing filespec is used when setting *JL, any information sent to the JobLog file will be appended to the end of the file.

You may wish to send the information to a device such as *PR, rather than a file. In this case, a devspec rather than a filespec would be used in the command line when setting *JL to its driver.

K E Y B O A R D D R I V E R (KI/DVR)

The KI/DVR program will enable certain keyboard features. The syntax is:

```
SET *KI TO KI/DVR (parm,parm,...)

The allowable parameters are as follows:

TYPE    activates the type ahead feature.

JKL     activates the screen print option.

DELAY=  sets the delay until the first repeat

RATE=   sets the repeat rate

abbr: TYPE=T, JKL=J, DELAY=D, RATE=R
```

Among other things, the KI/DVR program establishes the <CLEAR> key as a special control key for many LDOS functions. This driver must be set if SPOOL, SYSTEM (SVC), KSM, MiniDOS, LCOMM, or any other program that utilizes the <CLEAR> key as a control key is to be used!

On the Model III, the keyboard repeat and debounce features are part of the ROM keyboard driver, and will be available even if this driver is not used. However, using the KI/DVR program will provide an increased key repeat rate.

On the Model I, the keyboard will use the ROM driver on power up. You will not have key repeat or debounce unless KI/DVR has been set.

As this driver is established with the SET Library command, it must be applied before any other *KI filters. When KI/DVR is set, the driver will reside in high memory. Once KI/DVR is set, you will not be allowed to set it again with additional parameters without first doing a RESET *KI library command. For example, if you had initially SET *KI TO KI/DVR, and later wish to initialize the type ahead feature, you must first RESET *KI, and then SET *KI TO KI/DVR (TYPE). The only additional memory used will be the amount needed for the type ahead option. The original memory used for the KI/DVR will be reused for the original KI/DVR functions.

Specifying the TYPE parameter enables the Type Ahead feature. This will provide a 128 character buffer, and will allow typing ahead even when the system is performing other functions such as disk I/O. If you make a mistake while typing ahead, pressing the <SHIFT><BACK ARROW> will erase the current line. Pressing <CLEAR><@> will empty the entire type ahead buffer. To temporarily disable the type ahead function, use the command SYSTEM (TYPE=OFF). It may be re-enabled with a SYSTEM (TYPE=ON) command.

The screen print option will send the contents of the video screen to *PR (usually a line printer) whenever the <LEFT SHIFT><DOWN ARROW><*> keys are pressed. Characters outside the ASCII range are normally translated to periods. However, the GRAPHIC parameter of the SYSTEM library command will allow graphics characters to be sent to the line printer during a screen print.

The DELAY and RATE parameters deal with the keyboard repeat function. DELAY sets the initial delay between the time a key is first pressed and the first repeat of that key. It can be any value 10 or greater. The default is 30, and provides a delay of about 3/4 of a second. The RATE parameter sets the rate of key repeat, and can be any value 1 or greater. The default is 3, and provides a repeat rate of about 10 per second.

Keyboard equivalents

When KI/DVR is set, the TRS-80 keyboard will be able to produce the entire ASCII character set from X'00' to X'7F' (0 to 127). Keys not normally accessible can be entered as described in the following tables.

Extended Cursor Mode (ECM)

Many applications programs use the four arrow keys to control cursor motion. However, this precludes entering an X'5B' character, as this is the value returned by the <UP ARROW>. To allow the full ASCII character set to be used by an application, the ECM will change the values returned by the four arrow keys. When in the ECM, it will be necessary to use control keys to perform the original arrow functions. <CTRL><H> will perform a backspace, <CTRL><I> a tab, et cetera. Pressing an arrow key will display the corresponding graphics character. The ECM will primarily be useful for applications programs that do their own cursor control and also require that the full ASCII character set be available. The KFLAG$ description in the Technical Information section describes how to toggle the ECM on and off from within an assembly language program.

Forcing CAPS lock or unlock

An application program can force the keyboard input to be in either the CAPS lock mode or the normal upper/lower case mode without having the operator press the <SHIFT><0>. Assembly language programs can set bit 5 of KFLAG$ (Mod 1=X'4423', Mod 3=X'429F') to force CAPS lock, or reset that bit for normal upper/lower case entry. From a BASIC program, the POKE statement can be use. For the Model I, the statement:

```
POKE(&H4423),PEEK(&H4423) OR 32
```

will force CAPS lock, and the statement:

```
POKE(&H4423),PEEK(&H4423) AND 223
```

will force normal upper/lower case entry. For a Model III, the address to use would be &H429F. The logical AND and OR assure that only the CAPS lock bit of the memory location will be changed.

ASCII Character Set Generation

DEC	HEX	TAG	ENTERED BY	DEC	HEX	TAG	ENTERED BY	DEC	HEX	TAG	ENTERED BY	DEC	HEX	TAG	ENTERED BY
0	00	NUL	CTL <@>	32	20	SPA	<SPACE>	64	40	@	<@>	96	60		SH <@>
1	01	SOH	CTL <A>	33	21	!	SH <1>	65	41	A	SH <A>	97	61	a	<A>
2	02	STX	CTL <B>	34	22	"	SH <2>	66	42	B	SH <B>	98	62	b	<B>
3	03	ETX	CTL <C>	35	23	#	SH <3>	67	43	C	SH <C>	99	63	c	<C>
4	04	EOT	CTL <D>	36	24	$	SH <4>	68	44	D	SH <D>	100	64	d	<D>
5	05	ENQ	CTL <E>	37	25	%	SH <5>	69	45	E	SH <E>	101	65	e	<E>
6	06	ACK	CTL <F>	38	26	&	SH <6>	70	46	F	SH <F>	102	66	f	<F>
7	07	BEL	CTL <G>	39	27	'	SH <7>	71	47	G	SH <G>	103	67	g	<G>
8	08	BS	CTL <H> (1)	40	28	(	SH <8>	72	48	H	SH <H>	104	68	h	<H>
9	09	HT	CTL <I> (2)	41	29	)	SH <9>	73	49	I	SH <I>	105	69	i	<I>
10	0A	LF	CTL <J> (3)	42	2A	*	SH <:>	74	4A	J	SH <J>	106	6A	j	<J>
11	0B	VT	CTL <K>	43	2B	+	SH <;>	75	4B	K	SH <K>	107	6B	k	<K>
12	0C	FF	CTL <L>	44	2C	,	<,>	76	4C	L	SH <L>	108	6C	l	<L>
13	0D	CR	CTL <M> (4)	45	2D	-	<->	77	4D	M	SH <M>	109	6D	m	<M>
14	0E	SO	CTL <N>	46	2E	.	<.>	78	4E	N	SH <N>	110	6E	n	<N>
15	0F	SI	CTL <O>	47	2F	/	</>	79	4F	O	SH <O>	111	6F	o	<O>
16	10	DLE	CTL <P>	48	30	0	<0>	80	50	P	SH <P>	112	70	p	<P>
17	11	DC1	CTL <Q>	49	31	1	<1>	81	51	Q	SH <Q>	113	71	q	<Q>
18	12	DC2	CTL <R>	50	32	2	<2>	82	52	R	SH <R>	114	72	r	<R>
19	13	DC3	CTL <S>	51	33	3	<3>	83	53	S	SH <S>	115	73	s	<S>
20	14	DC4	CTL <T>	52	34	4	<4>	84	54	T	SH <T>	116	74	t	<T>
21	15	NAK	CTL <U>	53	35	5	<5>	85	55	U	SH <U>	117	75	u	<U>
22	16	SYN	CTL <V>	54	36	6	<6>	86	56	V	SH <V>	118	76	v	<V>
23	17	ETB	CTL <W>	55	37	7	<7>	87	57	W	SH <W>	119	77	w	<W>
24	18	CAN	CTL <X> (5)	56	38	8	<8>	88	58	X	SH <X>	120	78	x	<X>
25	19	EM	CTL <Y> (6)	57	39	9	<9>	89	59	Y	SH <Y>	121	79	y	<Y>
26	1A	SUB	CTL <Z>	58	3A	:	<:>	90	5A	Z	SH <Z>	122	7A	z	<Z>
27	1B	ESC	CTL <,> (7)	59	3B	;	<;>	91	5B	[	CLR <,> (8)	123	7B	{	CLR SH <,>
28	1C	FS	CTL </>	60	3C	<	SH <,>	92	5C	\	CLR </>	124	7C	|	CLR SH </>
29	1D	GS	CTL <.> (9)	61	3D	=	SH <->	93	5D	]	CLR <.>	125	7D	}	CLR SH <.>
30	1E	RS	CTL <;>	62	3E	>	SH <.>	94	5E	^	CLR <;>	126	7E	~	CLR SH <;>
31	1F	VS	SH <CLR>	63	3F	?	SH </>	95	5F	_	CLR <ENTER>	127	7F	DEL	CLR SH ENTER

Extended (non-ASCII) Character Set Generation

DEC	HEX	TAG	ENTERED BY
128	80		
129	81	ECM	<←>
130	82	ECM	<↑>
131	83		
132	84	ECM	<→>
133	85		
134	86		
135	87		
136	88	ECM	<↓> (10)
137	89	SCM CLR	<→>
138	8A	SCM CLR	<↓>
139	8B		
140	8C		
141	8D		
142	8E		
143	8F		
144	90		
145	91	ECM SH	<←>
146	92	ECM SH	<↑>
147	93		
148	94	ECM SH	<→>
149	95		
150	96		
151	97		
152	98	SCM CLR SH	<←>
153	99	SCM CLR SH	<→>
154	9A		
155	9B	SCM CLR SH	<↑>
156	9C		
157	9D		
158	9E		
159	9F		

DEC	HEX	TAG	ENTERED BY
160	A0	CLR	<SPACE>
161	A1	CLR SH	<1>
162	A2	CLR SH	<2>
163	A3	CLR SH	<3>
164	A4	CLR SH	<4>
165	A5	CLR SH	<5>
166	A6	CLR SH	<6>
167	A7	CLR SH	<7>
168	A8	CLR SH	<8>
169	A9	CLR SH	<9>
170	AA	CLR SH	<:>
171	AB		
172	AC		
173	AD	CLR	<->
174	AE		
175	AF		
176	B0	CLR	<0>
177	B1	CLR	<1>
178	B2	CLR	<2>
179	B3	CLR	<3>
180	B4	CLR	<4>
181	B5	CLR	<5>
182	B6	CLR	<6>
183	B7	CLR	<7>
184	B8	CLR	<8>
185	B9	CLR	<9>
186	BA	CLR	<:>
187	BB		
188	BC		
189	BD	CLR SH	<->
190	BE		
191	BF		

DEC	HEX	TAG	ENTERED BY
192	C0	CLR	<@> (11)
193	C1	CLR	<A> (12)
194	C2	CLR	<B> (12)
195	C3	CLR	<C> (12)
196	C4	CLR	<D> (12)
197	C5	CLR	<E> (12)
198	C6	CLR	<F> (12)
199	C7	CLR	<G> (12)
200	C8	CLR	<H> (12)
201	C9	CLR	<I> (12)
202	CA	CLR	<J> (12)
203	CB	CLR	<K> (12)
204	CC	CLR	<L> (12)
205	CD	CLR	<M> (12)
206	CE	CLR	<N> (12)
207	CF	CLR	<O> (12)
208	D0	CLR	<P> (12)
209	D1	CLR	<Q> (12)
210	D2	CLR	<R> (12)
211	D3	CLR	<S> (12)
212	D4	CLR	<T> (12)
213	D5	CLR	<U> (12)
214	D6	CLR	<V> (12)
215	D7	CLR	<W> (12)
216	D8	CLR	<X> (12)
217	D9	CLR	<Y> (12)
218	DA	CLR	<Z> (12)
219	DB		
220	DC		
221	DD		
222	DE		
223	DF		

DEC	HEX	TAG	ENTERED BY
224	E0		
225	E1	CLR SH	<A>
226	E2	CLR SH	<B>
227	E3 (13)	CLR SH	<C>
228	E4 (13)	CLR SH	<D>
229	E5	CLR SH	<E>
230	E6 (13)	CLR SH	<F>
231	E7	CLR SH	<G>
232	E8	CLR SH	<H>
233	E9	CLR SH	<I>
234	EA	CLR SH	<J>
235	EB (13)	CLR SH	<K>
236	EC	CLR SH	<L>
237	ED	CLR SH	<M>
238	EE	CLR SH	<N>
239	EF	CLR SH	<O>
240	F0 (13)	CLR SH	<P>
241	F1 (13)	CLR SH	<Q>
242	F2 (13)	CLR SH	<R>
243	F3	CLR SH	<S>
244	F4 (13)	CLR SH	<T>
245	F5	CLR SH	<U>
246	F6	CLR SH	<V>
247	F7	CLR SH	<W>
248	F8	CLR SH	<X>
249	F9	CLR SH	<Y>
250	FA	CLR SH	<Z>
251	FB		
252	FC		
253	FD		
254	FE		
255	FF		

Abbreviations:

ECM => Extended Cursor Mode toggled by <CLEAR><SHIFT><SPACE>.
CLR => Notation for the <CLEAR> key.
CLRSH => Depression, in order, of the <CLEAR> and <SHIFT> keys.
CTL => "Control" key operated by <SHIFT><DOWN ARROW>.
SCM => Standard Cursor Mode toggled by <CLEAR><SHIFT><SPACE>.
SH => Notation for the <SHIFT> key.
SHCLR => Depression, in order, of the <SHIFT> and <CLEAR> keys.

NOTES:

1. Can also be generated with LEFT ARROW if not in ECM.
2. Can also be generated with RIGHT ARROW if not in ECM.
3. Can also be generated with DOWN ARROW if not in ECM.
4. Can also be generated with ENTER.
5. Can also be generated with SH-LEFT ARROW if not in ECM.
6. Can also be generated with SH-RIGHT ARROW if not in ECM.
7. Can also be generated with SH-UP ARROW if not in ECM.
8. Can also be generated with UP ARROW if not in ECM.
9. Can also be generated with CTL-ENTER.
10. Also generated in SCM with CLEAR-LEFT ARROW.
11. Used to empty the type-ahead buffer.
12. Used by KeyStroke Multiply, if KSM is active.
13. Used by the MiniDos filter if active.

R S - 2 3 2 R MODEL I, RADIO SHACK INTERFACE ONLY

This Driver program will accept and configure the RS-232 hardware in the Model I Radio Shack interface, using the SET library command as follows:

```
SET devspec TO RS232R/DVR (parm,parm,...)

devspec   is the device to be used with the RS-232,
          normally *CL, or the Comm Line.

parm      parameters used to configure the RS-232 port,
          and establish line conditions:

BAUD=     sets the BAUD rate to any supportable rate.

WORD=     sets the word length, 5 to 8 bits.

STOP=     sets the stop bits, either 1 or 2.

PARITY=   sets the PARITY switch, ON or OFF. If ON is
          specified, EVEN or ODD may also be used.

BREAK     determines whether RS232R can set the
          system BREAK, PAUSE, or ENTER bits

                RS-232 LINE CONDITIONS
  ------------------------------------------------

Output Parameters

DTR=      ON/OFF      (Data Terminal Ready)
RTS=      ON/OFF      (Request To Send)

Input Parameters

DSR=      ON/OFF      (Data Set Ready)
CD=       ON/OFF      (Carrier Detect)
CTS=      ON/OFF      (Clear To Send)
RI=       ON/OFF      (Ring Indicator)


abbr: ON=Y, OFF=N, BAUD=B, WORD=W, STOP=S, PARITY=P
```

This program is a driver for the optional RS-232 board in the Model I Radio Shack interface. It allows you to set your RS-232 parameters to values that match other RS-232 devices.

The defaults for the configuration parameters will default to the switch settings on the RS-232 board. If a parameter is specified when setting the driver, it will override the switch setting.

The Line Condition parameters have been provided so that you may set up the conventions required by most communicating devices. As specified by standard RS232 conventions, a TRUE condition means a logic Ø, or positive voltage. A FALSE condition means a logic 1, or negative voltage. DTR and RTS may be set to a constant TRUE by specifying the ON switch. If DSR, CD, CTS, or RI are specified ON, the driver will

observe the lead and wait for a TRUE condition before sending each character. If specified OFF, the driver will wait for a FALSE condition before sending a character. If not specified, the lead will be ignored.

When using direct connect modems, it may be necessary to specify the DTR parameter to allow the modem to remain off hook.

The BREAK parameter is provided to allow LDOS to recognize BREAK, PAUSE (Shift-@), and ENTER characters received from the communications line. This would be useful in "host" type applications. The BREAK parameter will cause RS232R to set the system break bit whenever a modem break (extended null) or an ASCII X'Ø1' is received. The system pause bit will be set whenever the ASCII code X'6Ø' is received, and the system enter bit will be set whenever a carriage return (X'ØD') is received. If the parameter is not specified, RS232R will never set the break, pause, or enter bits.

Examples of using the RS-232R driver can be found on page 4-13.

R S - 2 3 2 T MODEL III ONLY

This Driver program will accept and configure the RS-232 hardware in the Model III, using the SET library command as follows:

```
SET devspec TO RS232T/DVR (parm,parm,...)

devspec   is the device to be used with the RS-232,
          normally *CL, or the Comm Line.

parm      parameters used to configure the RS-232 port,
          and establish line conditions:

BAUD=     sets the BAUD rate to any supportable rate.

WORD=     sets the word length, 5 to 8 bits.

STOP=     sets the stop bits, either 1 or 2.

PARITY=   sets the PARITY switch, ON or OFF. If ON is
          specified, EVEN or ODD may also be used.

BREAK     determines whether RS232T can set the
          system BREAK, PAUSE, or ENTER bits

                RS-232 LINE CONDITIONS
  ---------------------------------------------

Output Parameters

DTR=      ON/OFF      (Data Terminal Ready)
RTS=      ON/OFF      (Request To Send)

Input Parameters

DSR=      ON/OFF      (Data Set Ready)
CD=       ON/OFF      (Carrier Detect)
CTS=      ON/OFF      (Clear To Send)
RI=       ON/OFF      (Ring Indicator)


abbr: ON=Y, OFF=N, BAUD=B, WORD=W, STOP=S, PARITY=P
```

This program is a driver for the optional RS-232 board in the Model III. It allows you to set the parameters to values that match any other RS-232 devices. The receiving side of the driver is interrupt driven and contains an internal 128 character buffer to prevent loss of characters during disk I/O and other lengthy operations.

The defaults for the configuration parameters are as follows:

```
BAUD = 3ØØ
WORD = 7
STOP = 1
PARITY = ON,EVEN
```

The Line Condition parameters have been provided so that you may set up the conventions required by most communicating devices. As specified by standard RS232 conventions, a TRUE condition means a logic Ø, or positive voltage. A FALSE condition means a logic 1, or negative voltage. DTR and RTS may be set to a constant TRUE by specifying the ON switch. If DSR, CD, CTS, or RI are specified ON, the driver will observe the lead and wait for a TRUE condition before sending each character. If specified OFF, the driver will wait for a FALSE condition before sending a character. If not specified, the lead will be ignored.

The BREAK parameter is provided to allow LDOS to recognize BREAK, PAUSE (Shift-@), and ENTER characters received from the communications line. This would be useful in "host" type applications. The BREAK parameter will cause RS232T to set the system break bit whenever a modem break (extended null) or an ASCII X'Ø1' is received. The system pause bit will be set whenever the ASCII code X'6Ø' is received, and the system enter bit will be set whenever a carriage return (X'ØD') is received. If the parameter is not specified, RS232T will never set the break, pause, or enter bits.

Examples of using the RS232T driver can be found on page 4-13.

RS232 EXAMPLES, ALL MODELS

The following examples show how these driver programs might be used.

```
SET *CL TO RS232x/DVR (BAUD=3ØØ,WORD=8,STOP=1,CTS)
SET *CL RS232x (BAUD=3ØØ,WORD=8,STOP=1,CTS)
```

This example will configure the RS-232 using the values specified. Notice that PARITY was not specified, and will use the default value. The use of TO in the command line is optional. Also, the default file extension for the SET command is /DVR. CTS was specified, so the driver will look at the CTS line for a TRUE condition before sending a character. This would be useful, for instance, when using a serial line printer. If the serial printer had its BUSY line hooked to the CTS line on the computer, characters would be sent to the printer only when the printer was ready to accept them.

The device *CL will usually be the device the system will use to communicate with the RS-232 hardware. However, when using a serial printer, the *PR device would normally be used when setting the RS-232 driver.

```
SET *CL TO RS232x/DVR (BREAK)
SET *CL RS232x (BREAK)
```

This example will configure the RS-232 hardware to the default values. Because the BREAK parameter was specified, certain system functions will recognize break, pause, or enter characters from the RS-232 as if they came from the keyboard.

```
SET *CL RS232x (DTR,CTS,BREAK)
```

This example is identical to the previous, except that the DTR line will be held in a constant TRUE state, and the driver will not transmit any characters unless the external device raises the CTS line.

```
SET *CL RS232x (W=7,P=ON,EVEN)
```

This example will set the word length to 7, and set parity ON and EVEN.

K E Y S T R O K E M U L T I P L Y (KSM/FLT)

KSM/FLT allows the use of files containing phrases associated with the unshifted alphabetic keyboard keys to be used as direct keyboard inputs. The syntax is:

```
============================================================
|  FILTER *KI TO KSM/FLT USING filespec (ENTER=nn)          |
|                                                           |
|  filespec is an existing KSM type file                    |
|                                                           |
|  ENTER= is an optional parameter specifying the character |
|         to be used as an embedded enter.                  |
|                                                           |
|  abbr: ENTER=E                                            |
============================================================
```

Because the KSM filter uses the <CLEAR> key as a special control key, the KI/DVR program must be set before the KSM filter is applied.

As shown in the syntax block, the FILTER library command is used establish the KSM filter. Please refer to the library command section if an explanation of the filter command is needed.

The KSM program will load up to 26 phrases from the specified file (filespec) into memory. These phrases will be taken as though they were typed in from the keyboard when the <CLEAR> key and the specified unshifted alphabetic key are held down together. The default file extension for the filespec is /KSM. To create a KSM file, use the BUILD command in the following manner:

BUILD filename/KSM

This BUILD command will display the alphabetic keys one at a time and allow you to input your desired phrase or command. The extension of the filespec must be specified as /KSM to see the individual key prompts. The actual display will be:

A=>

and will continue up to Z=>. Once all 26 characters have been assigned, the file will be closed and the BUILD will be terminated. The BUILD may also be terminated any time before reaching Z=> by pressing the <BREAK> key in response to any character prompt.

The following rules will govern the entry of phrases during the BUILD.

Each phrase should be terminated by pressing <ENTER>. This does not place an <ENTER> character at the end of the phrase, but merely signifies the end of the phrase.

The ENTER parameter is used to determine what character KSM will see as an embedded <ENTER> key. If not specified, it defaults to a semicolon <;>. The value for enter may be entered as a decimal value between Ø and 255, or as a character enclosed in quotes, such as ":". Whenever this character is encountered in a KSM phrase, it will be translated into an <ENTER>.

Length of phrases should be limited to 63 characters for LDOS command lines and 255 characters for LBASIC lines.

The BUILD (HEX) parameter may be used to create characters or strings that are not directly available from the keyboard. The KI/DVR program does allow the full ASCII character set to be generated. However, if you wish to change key assignments, or to generate characters above X'7F', the BUILD (HEX) command will accomplish this.

It is not absolutely necessary to use the BUILD command with the /KSM file extension to create the KSM files. Any file in ASCII format can be used by the KSM/FLT program. If you wish to use the BUILD command without the /KSM extension, a BASIC program, or a word processor to create the KSM file, observe the following format.

When the file is read in by the KSM/FLT program, it is stored in memory according to lines. This means that all characters up to the first carriage return (X'ØD') will be assigned to the letter <A>, all characters up to the next carriage return to the letter <B>, etc. If a key is to be skipped or left undefined, a carriage return must be inserted for that character. Remember that the character specified with the ENTER parameter (default is the semicolon) will be translated into an <ENTER> by the KSM/FLT program.

Following are some examples of the KSM function in the LDOS command mode.

```
A=> DIR :Ø
```

This string would appear when the <CLEAR> and <A> keys were pressed together. The command DIR :Ø would be shown but would not be acted upon until the <ENTER> key was pressed.

```
A=> DIR :Ø;
```

This is the same as the last example, except the DIR :Ø command would be executed immediately as an <ENTER> was the last character of the phrase (represented by the semi-colon).

```
F=> FREE;DEVICE;
```

This phrase would be read in when the <CLEAR> and <F> keys were pressed together. The LDOS command FREE would be executed, and the command DEVICE would execute afterward.

Following are some examples of the KSM function in LBASIC.

```
F=> FOR
N=> NEXT
C=> CLEAR5ØØØ:DEFINT A-Z:DEFSTR S,U,V:DEFDBL D:DIM S(1ØØ);
```

The keys <F> and <N> could be assigned the phrases FOR and NEXT. Whenever these LBASIC commands were needed, they could be entered in by pressing the <CLEAR> and alphabetic key. The <C> key would insert the entire line associated with it into the program. It is possible to assign the most common LBASIC keywords and commands to a KSM file so they may be instantly inserted while programming in LBASIC.

Once *KI is filtered with a KSM file, a different KSM may be utilized as follows:

If the length of the new KSM file is less than or equal to the original KSM file, merely issue another filter command. The new KSM file will be loaded over the existing one, and will not require any additional memory.

If the new KSM file is larger than the original, you will not be allowed to change it. The message "REQUEST EXCEEDS AVAILABLE MEMORY" will appear, and the FILTER operation will abort. You will have to do a global RESET to remove all configuration, and then reconfigure using the larger KSM file. Be sure to set the KI/DVR program again before applying the new KSM file.

M i n i D O S (MINIDOS/FLT)

The MiniDOS filter program provides a means to access certain LDOS functions without having to be at the LDOS Ready prompt. The syntax is:

```
FILTER *KI USING MINIDOS/FLT

no parameters are required.

abbr: NONE
```

*** NOTE ***

Because the MiniDOS filter uses the <CLEAR> key as a special control key, the KI/DVR program must be set before the MiniDOS filter is applied.

The MiniDOS filter allows the keyboard driver to intercept certain keyboard inputs and immediately act on them. To allow the MiniDOS filter to properly intercept all keys, it must be the last filter applied to the keyboard (*KI).

Note that the MiniDOS filter will reside in high memory. If *KI is reset, the MiniDOS filter will re-use its initial memory allocation if activated again.

Once the MiniDOS filter is applied, pressing the <CLEAR><SHIFT> and the specified alphabetic key will cause the following:

<C> - Toggle the CLOCK display on or off.

<D> - Enter the system DEBUGger (if activated).

<F> - Display FREE space for all active drives.

<K> - Kill a file.

<P> - Send a character to a line printer.

<Q> - Display a disk's directory.

<R> - Repeat the last DOS command.

<T> - Issue a Top Of Form to the line printer.

When the MiniDOS filter intercepts one of these keys, it will immediately execute the associated function. These keys are active inside any program that uses the LDOS keyboard driver, including LBASIC. When the function has been completed, control will be returned to the calling program as though no key had been pressed. For this reason, if some of these functions are executed from the DOS level, the LDOS Ready prompt will not appear on the screen when the operation is complete. However, the system is still positioned as if the prompt were on the screen, and is ready to take another input.

The full descriptions and parameters for each command will be listed here.

<C> - The <C> command will toggle the clock display on or off. This is identical to issuing a CLOCK (ON) or CLOCK (OFF) library command.

<D> - The <D> command will enter the system DEBUGger or extended DEBUGger, providing it has been previously activated with the DEBUG or DEBUG (EXT) command.

<F> - The <F> command will allow you see the free space available on a drive, along with the disk's name and date of creation. After selecting this command, you will see the letter F enclosed within braces. At this point, enter the drive number of the target drive and press <ENTER>. The display will be in the following format:

NNNNNNNNDDDDDDDD FFFF K Free

N = the disk name.

D = the disk date of creation.

F = the free K (1Ø24 bytes) in Hex notation.

<K> - The <K> command will allow you to kill a specified file. You will see the letter K appear with in braces. At this point, type in the filespec you wish to kill. If no drivespec is included, all drives will be searched and the first matching filespec will be killed.

<P> - The <P> command will allow you to send a character to a line printer. The character must be in the form of two hexadecimal digits. This feature will allow you to send control characters to the line printer to switch printing modes, etc. If an invalid character is entered, an asterisk will appear. You will be allowed to re-enter the character at this point.

<Q> - The <Q> command will show the visible files on a specified drive. You will see the letter Q appear within braces. Type in the drivespec of the target drive, and the visible files will be displayed in 4 across format. You may also specify a particular file extension. The syntax would be:

d/EXT

where "d" is the drivespec, and /EXT is the desired extension. The wildcard character ($) may be used, but all three characters must be specified. For example, to find all file extensions starting with B, /B$$ must be specified.

<R> - The <R> command will repeat the last issued DOS command.

<T> - The <T> command will issue a Top Of Form to the line printer. This will also clear the line counter.

Entering an invalid parameter for any of the above commands will display the associated LDOS error message.

P R I N T E R F I L T E R P R O G R A M (PR/FLT)

The FILTER program PR/FLT is provided to format the data sent to the line printer. The syntax is:

```
FILTER *PR PR/FLT (parm,parm,...)

parms are the parameters described below.

ADDLF    Will add a linefeed after a carriage return.

CHARS    The number of characters per printed line.

FFHARD   Will issue an X'ØC' for a form feed, rather
         than a series of linefeeds.

INDENT   Number of characters to indent from left
         margin on lines longer than CHARS parm.

LINES    The number of lines printed on each page.

MARGIN   Sets the left margin.

PAGE     Sets the physical page length in lines.

PORT     Sends output to a specified port (Model I only)

TAB      Causes expansion of X'Ø9' tab characters.

SLINE=   Adjusts the printer line counter to Model I
         or Model III conventions.

XLATE=X'aabb'  specifies a one-character translation.

         aa = the character to be translated.
         bb = what "aa" will be translated to.

abbr:  SLINE=SL  All other parameters except PORT can
       be abbreviated to their first character.
```

This filter program adds certain enhancements to the normal ROM printer driver routine. If you have entered a command that uses the printer, you will no longer experience "lock up" if the printer is not connected to the system. Realize that if the printer is merely in a deselected or alert state, the system will wait until printer capabilities have been re-established.

Once PR/FLT has been applied, *PR may be returned to its power up driver with the RESET *PR library command. If you wish to re-apply PR/FLT, it will occupy the same high memory initially allocated.

This filter program also adds two features to operation under LBASIC. The command LPRINT CHR$(6) will reset the system line counter to top of page. This may be used when manually positioning to top of form. Also, the command LPRINT with no arguments will now cause a blank line to be generated.

The PR/FLT filter will allow you to determine the format of the data sent to your line printer. There are several configurable parameters used to set the format of the PR/FLT output. They are:

ADDLF If this parameter is specified, a linefeed will be issued after every carriage return.

CHARS= This parameter sets the number of characters that will be printed on each line. It may be any integer between 1 and 255.

FFHARD If this parameter is specified, any form feed determined by the PAGE and LINES parameters will be sent as an X'ØC' character rather than a series of linefeeds. If you use this parameter, be sure your printer will recognize the X'ØC' character.

INDENT= This parameter sets the number of spaces a line is to be indented if the line length exceeds (CHARS=) characters. The default value for this parameter is zero (Ø).

LINES= This parameter sets the number of lines that will be printed on each page. It may not exceed the PAGE parameter, and if not specified, it will default to the PAGE parameter of 66.

MARGIN= This parameter sets the width of the left margin. It is especially useful for printers with fixed position tractors.

PAGE= This parameter sets the physical page size in lines. It should be set to the particular form size you are printing on (66 for normal printer paper, 6 for mailing labels, etc.). The default value is 66 lines per page.

PORT= On the Model I, this parameter changes printer output from the normal memory mapped location to a user specified port. Any port between 1 and 255 may be specified.

SLINE=n The allowable values are Ø or 1. A zero will set the page length to 66 lines per page and the initial line count to Ø, matching Model I conventions. A one will set the lines per page to 67 and the initial line count to 1, matching Model III conventions. If this parameter is not specified, the normal convention will be used (Model I = 66,Ø and Model III = 67,1).

TAB If this parameter is specified, any X'Ø9' character will be expanded to a standard 8 column tab.

XLATE This parameter will translate a specified character to another character. The format is X'aabb', where aa is the character to be translated, and bb is desired character result. Both aa and bb must be hexadecimal values. This parameter may be useful to translate printer control characters when using more than one type of printer on the same system.

```
FILTER *PR USING PR/FLT (CHARS=8Ø,INDENT=6,PAGE=51,LINES=45,FFHARD)
```

This command will establish the PR/FLT program in high memory and filter the following parameters for the *PR (line printer) output.

(CHARS=8Ø) will allow a maximum of 8Ø characters per printed line. If a line contains more than 8Ø characters, the excess will be printed on the next line(s).

(INDENT=6) will indent 6 spaces the remainder(s) of any line that exceeds 8Ø characters (determined by the CHARS=8Ø parameter).

(PAGE=51) sets the physical page size to 51 lines.

(LINES=45) will allow for 45 lines to be printed on a page. Since the page length is 51 lines (determined by the PAGE=51 parameter), the PR/FLT program will normally send 6 linefeeds after the 45th line has been printed. These linefeeds are determined by the formula (PAGE minus LINES). If no linefeeds are required, do not specify either PAGE or LINES.

(FFHARD) will cause an X'ØC' to be sent rather than 6 linefeeds when the line count reaches 45.

```
FILTER *PR USING PR/FLT (MARGIN=1Ø,CHARS=8Ø,INDENT=6)
FILTER *PR PR (M=1Ø,C=8Ø,I=6)
```

This example will cause all lines to start 1Ø spaces in from the normal left-hand starting position (MARGIN=1Ø). Any line longer than 8Ø characters will be indented 6 spaces when wrapped around, and will be printed starting at position 16.

```
FILTER *PR PR (TAB,ADDLF)
FILTER *PR PR (T,A)
```

This example will cause expansion of all X'Ø9' characters to their normal 8 column tab position. Also, a linefeed will be sent every time a carriage return is sent.

```
FILTER *PR PR (XLATE=X'2A2E')
FILTER *PR PR (X=X'2A2E')
```

This example will translate all X'2A' characters (asterisks) to an X'2E' characters (periods). This may be useful to change the appearance of a report format.

```
FILTER *PR PR (FFHARD,SLINE=Ø)
FILTER *PR PR (F,SL=Ø)
```

This example will respond to a Top of Form command by issuing an X'ØC' Top of Form character rather than a series of linefeeds. Also, the printer line counter will start from Ø rather than 1, and use a page length of 66 lines per page as the SLINE=Ø parameter was specified. The SLINE parameter will match Model I conventions, and may be necessary when running Model I software on the Model III.

M O D x / D C T D R I V E S E T U P

The MOD1 and MOD3 DCT programs are used to change the logical drive numbers of 5" floppy drives. The SYSTEM Library command is used to execute the driver program.

```
SYSTEM (DRIVE=d,DRIVER="MODx")

d  is the new logical drive number, 1 to 7.
x  is 1 or 3, depending on the computer model.
```

This program is provided to allow you to change the logical numbers of your 5" floppy drives. It will primarily be used when running a hard drive.

Upon execution, the following prompt will be displayed:

ENTER DRIVE I/O ADDRESS <1-4>

The drive I/O address requested will be a number between 1 and 4, and will correspond to the drive's physical location on the drive cable. On the Model I, the first physical drive on the cable will be 1, the second will be 2, etc. On the Model III, the lower built in floppy will be 1, the upper built in floppy will be 2, and the two external drives will be 3 and 4.

This program, used in conjunction with the SYSTEM (SYSTEM=) Library command, will allow you to set up your hard and floppy drives to any desired logical number sequence.

J O B C O N T R O L L A N G U A G E (JCL)

The LDOS Job Control Language (JCL) is one of the most powerful features of the LDOS operating system. It allows the user to construct a sequence of commands and statements to control the actions of the operating system or applications programs. There are many different features to JCL, providing for user prompts and alerts, allowing the input of specified variables at runtime, providing for logical branching of program control based on user inputs, and allowing for variable substitution.

How JCL works

To use JCL, it is first necessary to understand how it works. In the most basic sense, this is the procedure:

1) The user creates an ASCII file consisting of commands and statements he wishes to be executed.

2) The user starts the JCL processing with the DO Library command.

3) The JCL processor takes over control of the keyboard.

4) A line is read in from the file and passed to the system EXACTLY AS IF IT CAME FROM THE KEYBOARD.

5) When the end of the file is reached, keyboard control returns to the user and the JCL processing stops.

From this description, the purpose of JCL could be summarized as a method to execute a series of commands to control the computer with no input from the computer operator other than to start up the procedure.

While the above description is correct, it is by no means a complete description of JCL's capabilities. The following sections of the JCL documentation will describe how to use many different features. The layout of the sections will start with the basics of creating a JCL file, and then show how to incorporate the more advanced features. It is recommended that you read these sections in order, as later sections will refer to material presented in the earlier ones.

Creating a JCL file

As noted in the above description, a JCL file is an ASCII file. For the purposes of JCL, this means a file containing those characters normally available from the keyboard. There are many different ways to create a JCL file. The BUILD Library command will let you create or extend a JCL file, but does not provide a means to edit an existing file. You could also create a JCL file with an LBASIC program, creating the lines as strings and writing them to a sequential file. Any word processor or text editor can also be used to create or edit a JCL file, as long as it can save a file in ASCII format without line numbers.

**** NOTE ****

No single line in a JCL file may be more than 63 characters in length. Depending on the JCL method used (Execute only or Compiled), JCL will either ignore all characters after the 63rd, or abort the processing entirely.

Restrictions of JCL

Certain LDOS library commands and utilities cannot be executed from a JCL file. As the main concept of JCL is to use a pre-determined set of commands, any program with unpredictable prompts will not function properly when run from a JCL file. Also, any program which requires removing the system disk will certainly cause the JCL to abort. Among the commands NOT valid from a JCL file are certain BACKUP commands, BUILD, COPY (X), certain CONV commands, DEBUG, certain PURGE commands, RESET, RESET *KI, SYSTEM (SYSGEN), and SYSTEM (SYSTEM=). As a general rule, you should not use any library command or utility program when specifying a QUERY parameter (although the global RESET and RESET *KI commands cannot be used in a JCL file, a RESET *device can be used with any device other than *KI).

However, if the order of prompts or inputs in a program is known, it is allowable to pre-arrange the proper responses in a JCL file, being careful that they remain in sync with the prompts. In this manner, you can have a JCL file totally run a program or other procedure with no operator input. This will depend on the method used by the program to normally take keyboard input. The section on INTERFACING WITH APPLICATIONS PROGRAMS will describe the type of input statements that can be answered with a line from a JCL file.

S I M P L E J C L E X E C U T I O N

JCL files that contain only executable comments, commands, or execution JCL macros are very common in day to day use of the LDOS system. The easiest JCL file to understand is one containing only commands. For example, let's assume that you have a program that requires the use of the printer filter program PR/FLT to set the printer line length, margin and page length. You could put the following command line in a JCL file:

```
FILTER *PR PR/FLT (CHARS=8Ø,MARGIN=1Ø,LINES=6Ø)
```

If this JCL file were called START/JCL, using the command DO =START at the LDOS Ready prompt would execute the line and apply the printer filter, returning to the LDOS Ready prompt.

Let us further assume that you now wish to go into LBASIC and run a program. The JCL file could be expanded as follows:

```
FILTER *PR PR/FLT (CHARS=8Ø,MARGIN=1Ø,LINES=6Ø)
LBASIC
RUN"PROGRAM/BAS"
```

Now using the command DO =START will establish the printer filter, enter LBASIC, and pass the command RUN"PROGRAM/BAS" to LBASIC. The program would be loaded in from disk and executed. However, like the first example, you will return to the LDOS Ready prompt as soon as the first keyboard input is requested by the program! To solve this problem, we must add one of the special JCL execution commands, called a MACRO, to the end of the file.

JCL Execution Macros and Comments

JCL execution macros perform many different functions. They are always entered in the JCL file as two slashes followed by the name of the macro. An execution comment is any line that starts with a period. These comments will be displayed to the screen during execution. Following is a list of all JCL execution macros:

```
JCL EXECUTION COMMENT  - . COMMENT

JCL TERMINATION MACROS - //ABORT, //EXIT, //STOP
JCL PAUSE/DELAY MACROS - //DELAY, //PAUSE, //WAIT
JCL ALERT MACROS       - //ALERT, //FLASH
JCL KEYBOARD MACROS    - //KEYIN, //INPUT
```

CAUTION : AN EXECUTION MACRO CANNOT BE THE FIRST LINE IN A JCL FILE!

The execution comments provide a means to display informative messages as the JCL file executes. You could label your JCL file and show other useful information as follows:

```
. Program start up JCL, last modified Ø1/Ø1/82
FILTER *PR PR/FLT (CHARS=8Ø,MARGIN=1Ø,LINES=6Ø)
LBASIC
RUN"PROGRAM/BAS"
```

This comment would be displayed when the JCL executes, and show the file's purpose, and the last date you made modifications to the file.

Remember from our last example that an unwanted return to the LDOS Ready prompt would be made as soon as a keyboard input was requested by the program. To keep this from occurring, you can use the //STOP macro.

JCL "TERMINATION" MACROS

//STOP

The //STOP macro is used to halt execution of the JCL file and return keyboard control to an application requesting keyboard input. Thus, our JCL example could be expanded as follows:

```
. Program start up JCL, last modified Ø1/Ø1/82
FILTER *PR PR/FLT (CHARS=8Ø,MARGIN=1Ø,LINES=6Ø)
LBASIC
RUN"PROGRAM/BAS"
//STOP
```

The JCL file is now complete, and as soon as the program requested a keyboard input, the keyboard would become "alive". The response to the prompt could then be input from the keyboard.

However, perhaps the program is one that requires no keyboard input during its execution. In this case, you might want to return to the LDOS Ready prompt when the program is completed. Using the //STOP macro in this case would not be correct. When the program completed, the //STOP would be executed, and the LBASIC Ready prompt would appear. You would not return to the LDOS Ready level.

As noted before, a return to LDOS Ready will happen automatically if you do not use the //STOP macro at the end of the JCL file. Another way to force an end to the JCL execution and return to LDOS Ready is to use either the //ABORT or //EXIT macros to end the JCL file.

//ABORT

The //ABORT macro is used to exit a JCL procedure and return to the program that initiated the DO command. It is quite similar to the //EXIT macro. A return to calling program will take effect after displaying the message:

```
JOB ABORTED
```

It would be used if your JCL processing logic detected an invalid run-time condition, and wanted to display an informative message. Also, any error that the operating system detects that will result in a jump to the @ABORT DOS vector will disable further JCL processing and display the above message. Basically, this macro should be used to exit JCL execution any time an undesired condition occurred.

//EXIT

The //EXIT macro is used to end the execution of JCL processing and return to the program that initiated the DO command. If no termination macro is entered in a JCL file, the JCL processing will terminate upon reaching the end of the file as though //EXIT was the last line in the JCL file, displaying the message:

```
JOB DONE
```

The JOB DONE message indicates a normal conclusion of the JCL file. This type of JCL exit should be used if the conclusion of the JCL command file also represents the conclusion of the job that is running. Therefore, the following JCL could be used to run a program that did not require any keyboard input, and needed to return to the LDOS Ready prompt after it finished.

```
. Program start up JCL, last modified Ø1/Ø1/82
FILTER *PR PR/FLT (CHARS=8Ø,MARGIN=1Ø,LINES=6Ø)
LBASIC
RUN"PROGRAM/BAS"
//EXIT
```

As you can see from these macros, you have three different ways to end a JCL file.

```
//STOP  - Stop JCL execution, remain in the user's program.
//ABORT - Stop execution, display "Job Aborted".
//EXIT  - Stop execution, display "Job Done".
```

Be sure to use the proper termination macro for the intended job application.

JCL "PAUSE/DELAY" MACROS

The other execution macros can be used to provide special effects if you need them in your JCL files. One of the most often used is the //PAUSE macro, which provides a means to temporarily suspend JCL execution.

//PAUSE optional message string

When this macro is encountered in an executing JCL file, it will be displayed on the screen along with any optional message. The message can inform you why the pause was ordered. Pressing <ENTER> will resume JCL execution, while pressing <BREAK> will abort the JCL. For example:

```
. Program start up JCL, last modified Ø1/Ø1/82
FILTER *PR PR/FLT (CHARS=8Ø,MARGIN=1Ø,LINES=6Ø)
LBASIC
//PAUSE Be sure data disks are mounted in drives 1 and 2!!
RUN"PROGRAM/BAS"
//EXIT
```

This example will suspend the JCL as soon as LBASIC executes and before the program is run and loaded. You can then check that the needed disks are in your other drives, and press <ENTER> to continue the JCL.

The //DELAY and //WAIT macros are similar to the //PAUSE macro, and used to give JCL execution a specific delay period.

//DELAY duration

The //DELAY macro will provide for a definite timed pause. JCL execution will automatically continue at the expiration of the delay period. The actual delay will be approximately Ø.1 seconds per count. The count may range from 1 to 256. Thus, a delay from Ø.1 seconds to a delay of 25.6 seconds is possible. The following example shows the proper syntax of the //DELAY macro.

```
. THIS COULD BE AN INFORMATIVE MESSAGE FOR THE OPERATOR
//DELAY 5Ø
lbasic
run"newprog/bas"
//STOP
```

This example JCL will print an informative message and then delay for approximately 5 seconds. After the delay, it will execute LBASIC and then run the desired program.

//WAIT hh:mm:ss

The //WAIT macro is similar to //DELAY, except that the length of the delay depends on the setting of the system clock. Providing the system clock is functioning, the //WAIT macro will put the entire system in a "sleep" state until such time as the system clock matches the time specified in the macro operand. The system clock can be set with the TIME library command. You can also set the time from a JCL file by using a direct execution of the TIME library command, or with the //INPUT macro, which will be discussed later. Consider the following example:

```
. example JCL for running alarm program
//wait Ø2:15:ØØ
lbasic
run"alarmset/bas"
//exit
```

Assuming that the system clock was set, this example would display the comment and then wait until the clock matched the time of Ø2:15:ØØ specified in the //WAIT macro. It would then execute LBASIC, and run the program ALARMSET/BAS, exiting to LDOS Ready after completion of the program.

JCL "ALERT" MACROS

The //FLASH and //ALERT are provided to give both visual and audio alerts to the operator. The //FLASH will flash a message line on the video screen, making it easy to emphasize an important piece of information. The //ALERT will send an audio tone to the cassette port, allowing an audio alert.

//FLASH duration message

This macro will flash a message on and off on the video screen. The duration can be any number from Ø to 255. This is the number of times the message will flash. If no duration is specified, the message will flash 256 times. The message string can be any comment you wish displayed. For example:

```
. TEST/JCL
//FLASH 1Ø Starting initialization JCL
```

When the TEST/JCL executes, the //FLASH line will be displayed. It will flash on and off 1Ø times, as specified by the duration count. At any time during this period, you may press <ENTER> to stop the flash and proceed to the next line. Pressing <BREAK> at this point will abort the JCL and display the message "Job Aborted".

//ALERT

The //ALERT macro may be used to provide an audible signal to the operator. It will generate up to eight different tones and direct its output to the cassette port on the line that normally goes to the AUX cassette jack. By plugging the AUX lead of the cassette cable into a small amplifier, this macro could prove quite useful. You could use it to signify the end of a large JCL procedure. It could also be used during the execution of a procedure to bring attention to a specific process. The proper syntax is:

```
//ALERT (tone,silence,...)
```

The actual tone selected is controlled by a tone number. The number range is Ø-7, with "7" producing the lowest tone, and "Ø" producing the highest tone. Any value entered will be used in its modulo 8 form. That is, if you enter the number "8", a zero value will be assumed. The value 65 will produce the tone assigned to a "1". The tone is followed by a period of silence by entering a second number. Tone and silence must be entered as number pairs (e.g. "1,Ø"). In fact, this can be repeated for as many number combinations as can fit on one line.

The tone-silence sequence can be made to repeat by enclosing the entire string in parentheses. If parentheses are used, the sequence will keep repeating until the <ENTER> key is pressed, at which time execution will continue. Pressing <BREAK> would abort the JCL. No display is made during the tone generation. Therefore, if your JCL has a repeating tone and you do not have an amplifier connected to the cassette port, the system may appear to hang.

The following example shows several uses of //ALERT:

```
. example of tone generation
//alert (1,Ø,7,Ø)
. another example
//alert Ø,Ø,1,Ø,2,Ø,3,Ø,4,Ø,5,Ø,6,Ø,7,Ø
. still another
//alert (Ø,Ø,1,1,2,2,3,3,4,4,5,5,6,6,7,7)
//exit
```

JCL "KEYBOARD ENTRY" MACROS

The //KEYIN and //INPUT macros provide a means to take keyboard inputs during JCL execution. Two other macros are used along with //KEYIN to establish execution-time conditional blocks.

//KEYIN optional comment string

This macro is used to prompt for a single character entry, with the entire //KEYIN line being displayed, including any comment message. The resultant entry must be a single numeric character in the range Ø-9 and will be used to select one of up to ten different execution phase blocks of JCL. //KEYIN can not be used to enter data at execution time but can only provide for the selection of a predefined block of JCL lines. If it is necessary to provide run-time keyboard interfacing, then the //INPUT macro should be used instead of //KEYIN.

```
. MENU/JCL
. Program 1 is MAIL
. Program 2 is LEDGER
. Program 3 is PAYABLES
//KEYIN Select program, 1 - 3
```

This example shows how you could build a menu using execution comments to display different program choices. By pressing a single key, you could execute the desired program. Refer to the following expansion of the MENU/JCL example:

```
//KEYIN  Select program 1 - 3
//1
LBASIC
RUN"MAIL/BAS"
//STOP
//2
```

```
LBASIC
RUN"LEDGER/BAS"
//STOP
//3
LBASIC
RUN"PAYABLES/BAS"
//STOP
///
```

There are two new macros used in this example. They are:

```
//NUMBER
///
```

The //NUMBER is used to start a block of lines that correspond to a value selected with the //KEYIN macro. This block will extend until the next //NUMBER or to the ///.

The triple slant /// is used to mark the end of all //NUMBER blocks. Regardless if a //NUMBER has been found to match the //KEYIN entry, the JCL processor will stop looking for a match as soon as it encounters a ///. Execution will begin with the following line.

In the above example, pressing 1, 2, or 3 would select the corresponding block of lines, entering LBASIC and running the appropriate program. If a key other than 1, 2, or 3 were pressed, all three //NUMBER blocks would be ignored, and execution would begin with the line after the ///. That line is totally dependent on what options you want to allow. If it is mandatory that one of the three programs be run, then an //ABORT macro could be used to abort the JCL. If other options were available, they could be placed here in the JCL file.

One of these options may be to let the user type in his own command. If this is the case, the //INPUT macro could be used.

//INPUT optional message string

> When using the //INPUT macro in a JCL file, it is recommended that the keyboard type ahead feature of the KI/DVR program either not be active or be disabled with a SYSTEM (TYPE=OFF) command. This can be done as a JCL line, if desired. If not done, using any macro such as //PAUSE that requires pressing the <ENTER> key will cause the JCL to abort if it later encounters a //INPUT.

The //INPUT macro is used to input a line from the keyboard. Our definition of JCL explained that JCL execution worked by taking over the keyboard, and substituting lines from a JCL file in place of keyboard entry. With this macro, control of the keyboard is temporarily returned to the operator. Now, any command may be typed on the keyboard and then passed to the system. The number of characters allowed will depend on where the JCL execution was when the //INPUT was encountered. For instance, if the JCL was executing at the LDOS Ready level, then up to 63 characters could be entered, the same as for a normal LDOS command. If the //INPUT was encountered after going into LBASIC, then up to 255 characters could be entered.

Consider a slight re-write of the MENU/JCL example used with the //KEYIN macro:

```
. MENU/JCL
. Program 1 is MAIL
. Program 2 is LEDGER
. Program 3 is PAYABLES
```

```
LBASIC
//KEYIN Select program 1-3
//1
//RUN"MAIL/BAS"
//STOP
//2
RUN"LEDGER/BAS"
//STOP
//3
RUN"PAYABLES/BAS"
//STOP
///
//INPUT your own choice, as RUN"PROGRAM"
//STOP
```

As you can see, this examples is slightly different than the //KEYIN example. First of all, LBASIC is entered before the //KEYIN command is displayed. Therefore, the command to enter LBASIC has been removed from the three conditional blocks. Now, if a key other than 1, 2, or 3 is pressed for the //KEYIN, the //INPUT line will be displayed. The user can then enter in a RUN"PROGRAM" command to start up his own program choice. In fact, the response to the //INPUT does not even have to be a RUN"PROGRAM" command. Any valid LBASIC statement could be used. As soon as LBASIC acted on the line, the //STOP would halt JCL execution and keyboard control would return to LBASIC.

This type of direct input to the system is equally valid at the LDOS Ready level. When describing the //WAIT macro, it was mentioned that the time for the system clock could be set by the operator in the middle of a JCL file. The next example shows how this is done:

```
. Example JCL for alarm program
//INPUT Enter the time command, use the format TIME HH:MM:SS
//WAIT Ø2:15:ØØ
LBASIC
RUN"ALARMSET/BAS"
//EXIT
```

This example JCL will prompt the operator and allow him to enter a TIME Library command to set the system clock. The //INPUT message also describes the proper format of the TIME command. After the input, the //WAIT would pause the system until the clock matched Ø2:15:ØØ, and then continue execution.

When using the //INPUT macro, some caution should be exercised to assure that the command typed in is valid at the level it will be executed. For example, mistakenly pressing only <ENTER> with no other characters for an //INPUT at the LDOS Ready level will abort JCL execution. The JCL would also abort if, for instance, a program name was entered incorrectly, resulting in an LDOS "Program not found" error.

S I M P L E J C L C O M P I L I N G

The previous section of JCL has shown how to create and use execute only JCL files. While this type of JCL file is useful, it does not allow for logical decisions, substitution capabilities, or combination of JCL files. To do that, you must use the features provided by the compile phase of JCL. As the title of this section implies, the basics of the compile phase will be discussed. You will find the documentation laid out according to the following overview:

1] Compilation description and terms
2] Conditional decisions
3] Substitution fields
4] Combination of files

There are certain features of the JCL compile phase that will not be discussed in this section of the documentation. The purpose of this section is to describe the basic functions of the JCL compiler, and to show some practical examples of JCL files. The ADVANCED JCL COMPILING will contain further examples, including those features not discussed here.

Although JCL is a compiled language, you do NOT have to be a programmer to understand it! In fact, the main purpose of JCL is to let the computer operator create files to control applications programs and to maintain the data generated by these programs.

COMPILATION DESCRIPTION AND TERMS

The purpose of the compilation phase is to read in the JCL file line by line, checking for directly executable lines, keyboard responses, and execution macros, and to evaluate any compilation statements, and to write the resultant lines to a file called SYSTEM/JCL. After the compilation is complete, control would normally then be passed to the second phase - the execution of the compiled SYSTEM/JCL file. The DO library command allows for four different methods to DO a JCL file. Briefly recapped, they are:

```
DO = filespec  (execute only)
DO *           (execute current SYSTEM/JCL file)

DO $ filespec  (compile only to SYSTEM/JCL)
DO filespec    (compile to SYSTEM/JCL, then execute SYSTEM/JCL)
```

As stated earlier, the JCL works by substituting lines in a file for keyboard entries. However, when using the compile phase, a JCL file is not restricted to using a series of executable commands to create these substitution lines. All that is required is that the SYSTEM/JCL file contain only executable lines AFTER THE COMPILE PHASE IS COMPLETED. The user is allowed to create a file consisting of:

Directly executable commands
Pre-arranged keyboard responses
JCL execution macros
JCL conditional macros
JCL labels

It is allowable to compile any JCL file, even if it contains only executable lines. Any of the examples in the previous execution JCL section could be compiled. The compile phase would examine each line, determine that it is an execution line, and write it to the SYSTEM/JCL file. After the compilation is completed, the SYSTEM/JCL file would be executed, producing exactly the same results as if the file were executed without compiling.

There are several new terms that will be used when discussing the JCL compilation phase. Briefly described, they are:

TOKEN

The term "token" is the most important term to understand when using the compile phase of JCL. It is, fortunately, very easy to understand. A token is merely a string of up to 8 characters, and may contain upper and lower case alphabetic characters A-Z and a-z, and the numbers Ø-9. Tokens have two uses - as a true/false switch for logical decisions, and as a character string value for use in substitution fields.

LOGICAL OPERATOR

There are three simple logical operators available. They are:

Logical AND (represented by the ampersand symbol "&")
Logical OR (represented by the plus symbol "+")
Logical NOT (represented by the minus symbol "-")

Although mentioned here, these logical operators will be discussed only in the ADVANCED JCL COMPILING section.

LABEL

A JCL label is the AT sign "@" followed by up to 8 alphanumeric characters. It is used to define the start of a JCL procedure, allowing many small JCL procedures to be combined into one large file.

Like the execution phase, there are several special JCL statements, or MACROS, available with the compile phase. As with the execution macros, they are in the form of two slashes (//) followed by the appropriate word. In order of explanation, they are:

//IF	//ASSIGN
//ELSE	//. COMMENT
//END	//QUIT
//SET	//INCLUDE
//RESET	

JCL CONDITIONAL DECISIONS

Certain JCL macros can be used to establish "blocks" within a JCL file. During the compilation, these blocks will be evaluated for a logical true or false condition. The following shows the basic methods of evaluation:

1) If the evaluation is true...
   Include all the lines until the block end.

2) If the evaluation is false...
   Ignore all the lines until the block end.

An alternate to a false evaluation is also provided.

3) If the evaluation is true...
   Include these lines...
   Or else...
   Include these lines.

There are three compilation macros provided to define a block of lines:

//IF (defines the start of a block)
//END (defines the end of a block)
//ELSE (defines the alternative to a false //IF)

Translating the above three examples for use with these three JCL macros would produce the following:

1) If the evaluation is true.
```
//IF
include these lines
//END
```

2) If the evaluation is false.
```
//IF
ignore these lines
//END
```

3) An alternative to a false statement.
```
//IF
include these lines
//ELSE
include these lines
//END
```

As might be apparent, the //IF macro by itself does not determine the truth or falseness of the block. There must be something that the //IF can test to determine a true or false condition. Since our definition of a token described one of its uses as a true/false switch, that something is a token.

A real example of a conditional block would be:

```
//IF drive
. Display this execution comment
//END
```

The token in this example is "drive". The //IF will test whether or not "drive" is true or false. Assume that this block of lines is contained in a file called TEST/JCL. Refer to the three following DO library command examples:

```
1)  DO TEST (drive)
2)  DO TEST (DRIVE)
3)  DO TEST
```

Examples 1 and 2 would both set the token "drive" to be true. These two examples again emphasize an important point - there is NO difference between upper and lower case for any JCL macro, token, or label. Example 3 would set the token "drive" to be false. From these examples, you can see how easy it is do set a token true or false:

1) To set a token true, simply specify it on the DO command line.

2) To set a token false, do NOT specify it on the DO command line.

According to these rules, using either DO command line 1 or 2 would cause the execution comment in the TEST/JCL example to be written to the SYSTEM/JCL file. Using DO command line 3 would bypass any lines between the //IF and the //END.

This type of logical decision capability allows a single JCL file to be created, and lets the computer operator pick a course of action by merely typing in the same "DO filespec" command with different tokens. For example, consider the following JCL example, which shall be referred to as START/JCL (the first line is an execution comment, as previously explained in the SIMPLE JCL EXECUTION section).

```
. START/JCL for program start-up
//IF PR1
FILTER *PR PR/FLT (CHARS=8Ø)
//END
//IF PR2
FILTER *PR PR/FLT (CHARS=132)
//END
```

Let us assume that these are the first lines in a JCL file that will start some applications program running. The two conditional blocks let the operator apply the PR/FLT printer filter, defining the number of characters per line for printed output. The DO commands to accomplish this would be:

```
DO START (PR1)
DO START (PR2)
```

When the first DO command is issued, the //IF PR1 will test true, and the 8Ø character FILTER command will be written to the SYSTEM/JCL file. Because PR2 was not specified, the //IF PR2 will be false, and the second FILTER command will not be written to the SYSTEM/JCL file. Using the second DO command example would reverse the results. In the case where either one or the other FILTER command is always desired, there is an easier way to accomplish the same results, and requires only the use of the PR1 token as shown in the following example:

```
. START/JCL for program start-up
//IF PR1
FILTER *PR PR/FLT (CHARS=8Ø)
//ELSE
FILTER *PR PR/FLT (CHARS=132)
//END
```

By using the //ELSE macro, an alternative course of action is provided in case the //IF test is false. Thus the command "DO START (PR1)" would use the 8Ø character FILTER line, and ignore everything between the //ELSE and the //END. The command "DO START" would cause the //IF PR1 to test false, and therefore use the 132 character FILTER line.

Although the previous examples have shown a single line in each conditional block, any amount of lines may be included. Refer to the following MENU/JCL example:

```
. MENU/JCL selection start-up
//if KI
SET *KI KI/DVR (TYPE,JKL)
FILTER *KI MINIDOS/FLT
FILTER *PR PR/FLT (FFHARD,CHARS=8Ø)
//end
LBASIC
//if P1
RUN"PROGRAM1/BAS"
//end
//if P2
RUN"PROGRAM2/BAS"
//end
//stop
```

This example references three tokens - KI, P1, and P2. The first conditional block is easy to understand. If the KI token was entered on the DO command line, KI/DVR, MINIDOS/FLT, and PR/FLT would all be applied. If not, all lines up to the first //end would be ignored. Regardless, the command line "LBASIC" would be written to the SYSTEM/JCL file for execution. As the compilation continued, the //if P1, and then the //if P2 would be tested. Again, if either token was specified, the RUN"PROGRAM" line would be written out. In any case, the last line written out would be the //stop execution macro.

Instead of the two separate //if macros to determine which, if any, LBASIC program would be run, we could have used an //else macro in the following manner:

```
//if P1
RUN"PROGRAM1/BAS"
//else
RUN"PROGRAM2/BAS"
//end
```

Although this lets the operator select either program with a single token, it also means that one program or the other will always be selected.

To test a JCL procedure, the compile only "DO $ filespec" command can be used. Once the compiling is complete, the results can be examined by using the LIST library command to list the SYSTEM/JCL file to the video or printer. Refer to the following examples, using the previous MENU/JCL example.

Command 1) DO $ MENU (KI,P2)

The resultant SYSTEM/JCL file would be:

```
. MENU/JCL selection start-up
SET *KI KI/DVR (TYPE,JKL)
FILTER *KI MINIDOS/FLT
FILTER *PR PR/FLT (FFHARD,CHARS=8Ø)
LBASIC
RUN"PROGRAM2/BAS"
//STOP
```

Command 2) DO $ MENU

The resultant SYSTEM/JCL file would be:

```
. MENU/JCL selection start-up
LBASIC
//STOP
```

From the above examples, you should now have a basic understanding of how the //IF macro can be used to create a single JCL file that can be used for different purposes. All that is required is that the proper tokens to be entered on the DO command line. To reduce the number of tokens needed, and to provide for higher conditional logic statements to be handled, JCL provides the //SET, //RESET, and //ASSIGN tokens.

Using //SET, //RESET, //ASSIGN

The //SET macro is used to give a token a logical true value, the same as specifying it on the DO command line. The //RESET token does the opposite, giving a token a false value. One basic use for //SET is to let one token set the value of another. For example:

```
//IF KI
//SET P1
//END
```

If these lines were added to the beginning of the MENU/JCL example file, you can see that specifying only the KI token will also set P1 to a true condition. Again referring to the MENU/JCL file, it is possible that the operator could enter both the P1 and P2 tokens, generally producing undesired results. To keep this from happening, the following lines could be added to the beginning of the file:

```
//IF KI
//SET P1
//RESET P2
//END
```

This conditional block would assure that P2 was reset if P1 was entered on the command line, EVEN IF P2 WERE ALSO ENTERED! Now let's rewrite MENU/JCL slightly, assuming that PROGRAM1 requires the keyboard driver and 8Ø character print lines, and that PROGRAM2 requires no keyboard driver and 132 column print lines. We will also assume that if P1 is not entered, P2 should be the default.

```
. MENU/JCL, revision 1
//IF P1
//RESET P2
SET *KI KI/DVR (TYPE,JKL)
FILTER *PR PR/FLT (CHARS=8Ø)
//ELSE
//SET P2
FILTER *PR PR/FLT (CHARS=132)
//END
LBASIC
//IF P1
RUN"PROGRAM1/BAS"
//END
//IF P2
RUN"PROGRAM2/BAS"
//END
//STOP
```

Evaluating the results of different DO command lines would show the following:

DO $ MENU (P1) or DO $ MENU (P1,P2)

```
. MENU/JCL, revision 1
SET *KI KI/DVR (TYPE,JKL)
FILTER *PR PR/FLT (CHARS=8Ø)
LBASIC
RUN"PROGRAM1/BAS"
//STOP
```

DO $ MENU or DO $ MENU (P2)

```
. MENU/JCL, revision 1
FILTER *PR PR/FLT (CHARS=132)
LBASIC
RUN"PROGRAM2/BAS"
//STOP
```

The first //IF macro tests if P1 is true. If so, P2 is reset false, and the KI/DVR and 8Ø character print mode are applied. If P1 is was not entered on the DO command line, the //ELSE sets P2 to true, and applies the 132 character print mode, even if P2 was not entered. The compiling continues, writing the LBASIC line, the selected PROGRAM line, and the //STOP to the SYSTEM/JCL file.

As previously mentioned, the //SET macro can be used to reduce the number of tokens that have to be entered on the DO command line. Consider the following SYSOPT/JCL example:

```
. Establish LDOS system options
//IF ALL
//SET KITY
//SET PR
//SET MINI
//SET SRES
//END
//IF KIALL
//SET KITY
//SET MINI
//END
//IF KITY
set *ki ki/dvr (type)
//END
//IF PR
filter *pr pr/flt (chars=8Ø)
//END
//IF MINI
filter *ki minidos
//END
//IF SRES
system (sysres=2)
system (sysres=3)
system (sysres=8)
system (sysres=1Ø)
//END
```

This example shows how many different LDOS options can be established with a JCL file. The way it is structured, the operator can choose any or all of the options. Without the use of //SET, it would be necessary to enter four separate tokens to establish all of the options. By using a conditional block, the single token ALL can be made to set all of the necessary tokens true. Also, the token KIALL can be used to set only the two keyboard related options. Notice the use of upper and lower case. As stated previously, the case of a line has no effect on any JCL macro, token or label. This is also true when the line is an LDOS command, as are the lower case lines in this example. You may find that for editing purposes, the readability of a JCL file can be improved by using upper case for macros and lower case for executable lines, or vice versa. In the case of large JCL files, this generally makes itself readily apparent.

//ASSIGN

The //ASSIGN macro has two purposes. Like the //SET macro, it will set a token's logical value true. It can also assign a character string value to a token. The syntax for the //ASSIGN macro is:

```
//ASSIGN TOKEN=CHARACTER STRING
```

The character string value assigned to the token will be useful as described in the next section of the JCL documentation, SUBSTITUTION FIELDS. The character string can consist of up to 32 upper or lower case alphabetic characters, the numbers Ø-9, and the special characters slash (/), period (.), and colon (:). The important point to remember is that the //ASSIGN does set the token's logical value to true, the same as the //SET macro. Note that any time //ASSIGN is used, there must be at least one character assigned as a value. The statement //ASSIGN ALL would be an invalid statement, and would abort the compiling. You must have an equal sign (=) and some character string value following it when using the //ASSIGN macro, such as //ASSIGN ALL=EVERYONE.

In any of the previous examples that used the //SET macro, the //ASSIGN macro could have been substituted. The character string value assigned to the token would have no effect on the JCL logic. The important fact would be that the //ASSIGN also set the token true, as shown in the next example.

```
. TEST/JCL                  . TEST/JCL
//IF A                      //IF A
//SET P1                    //ASSIGN P1=PROGRAM/BAS
//SET KI                    //ASSIGN KI=ALL
//SET PR                    //ASSIGN PR=8Ø
//END                       //END
```

Logically speaking, these two examples are identical. If the token A is true, the tokens P1, KI, and PR will all be set to true. Additionally, the //ASSIGN example will assign character string values to the tokens. However, these character string values will have no effect on the logical value.

//. COMMENT
//QUIT

So far, the only method described for testing and debugging a JCL file has been to use the compile only DO command and then list the resultant SYSTEM/JCL file. The //. COMMENT and the //QUIT are provided to give you run time debugging. Unlike execution comments, the compilation comments are not written to the SYSTEM/JCL file. Rather, they are displayed on the screen immediately as encountered when the compiling is done. Thus, they can act as a visual status log of the compile. The //QUIT macro is used to abort the compiling if an invalid condition is detected. This gives you the ability to make sure all needed tokens are entered before any execution takes place.

```
. START/JCL
filter *pr pr/flt (lines=6Ø)
//IF KI
set *ki ki (type)
//ELSE
//. KI was not entered!
//QUIT
//END
lbasic
run"program/bas"
//STOP
```

If this JCL file was compiled without the token KI being entered on the DO command line, the screen display would show:

```
//. KI was not entered!
//QUIT
```

No actual lines would be executed from the SYSTEM/JCL file, as the compile phase was aborted before completion. The compilation comment tells the operator why the abort took place. The //QUIT macro may seem very similar to the //ABORT execution macro. The main difference is when the actual abort takes place. Substituting //ABORT for //QUIT in the previous example, and then doing the JCL without the token KI would produce the following screen display:

```
//. KI was not entered!
. START/JCL
filter *pr pr/flt (lines=60)
//ABORT
```

As you can see, the comment line will be displayed as the compiling is taking place. However, since //ABORT is an execution macro, the SYSTEM/JCL file will execute until it reaches the //ABORT line! This means that any executable lines up to that point will be executed. In this case, the PR/FLT program would have been applied. Now, if you would do the same JCL file again, specifying the KI token on the command line, problems will occur. Since the PR/FLT program cannot be applied if it is already active, the JCL will abort again when it tries to execute the FILTER line for the second time. As you can see, the //QUIT macro definitely should be used rather than the //ABORT.

JCL SUBSTITUTION FIELDS

Perhaps the most powerful feature of the JCL language is the ability to substitute and concatenate character strings to create executable lines. The character strings can be entered as token values on the DO command line, or can be set with the //ASSIGN macro. A substitution field is created by placing pound signs (#) around a token. For example:

```
. TEST/JCL
filter *pr pr/flt (chars=#C#)
lbasic
run"#P1#"
//STOP
```

This example uses two substitution fields; one in the FILTER command line representing the number of characters, and one in the run"program" line representing the name of the program. If the DO command "DO TEST (C=132,P1=PROGRAM1)" were used, the lines written to the SYSTEM/JCL file would be:

```
. TEST/JCL
filter *pr pr/flt (chars=132)
lbasic
run"PROGRAM1"
//STOP
```

As you can see, the compile phase substituted the character string value of the tokens into the actual command line! In effect, you could set any valid number of characters for the FILTER command and run any program simply by specifying different values for the C and P1 tokens. This example brings out another important point - the number of characters in the replacement string can be less than, equal to, or greater than the length of the token name in the replacement field between the # signs.

To reduce the number of tokens needed on the DO command line, and to increase the program options at the same time, the //ASSIGN macro can be used as follows:

```
. TEST/JCL
//ASSIGN c=8Ø
//ASSIGN p1=program1
//IF num2
//ASSIGN c=132
//ASSIGN p1=program2
//END
filter *pr pr/flt (chars=#C#)
lbasic
run"#P1#"
//STOP
```

In this example, the DO command would not have to specify any tokens if the default of the 8Ø character printer filter and PROGRAM1 were desired. Otherwise, specifying NUM2 would override the defaults. The values of C and P1 would automatically be set with the //ASSIGN tokens inside the //IF conditional block.

Another very practical use of the substitution field feature is for replacing drive specifications. The following example shows how a FORMAT and BACKUP JCL file could be structured:

```
. FB/JCL, FORMAT with BACKUP
//PAUSE insert disk to format in drive #D#
format :#D# (name="data1",q=n,ABS)
backup :#S# :#D# (mod)
//EXIT
```

In this example, the token D could be used to represent the destination drivespec, and the token S the source drivespec. Entering the command DO FB (S=1,D=2) would first pause the JCL and prompt you to insert a disk in drive 2. As you can see, the substitution fields can be used in message lines and comments as well as in executable command lines. After pressing <ENTER>, the JCL would continue, formatting the disk in drive 2, and then executing the backup command with drive 1 as the source drive and drive 2 as the destination drive.

Because the # sign is used to mark the start of a substitution field, some caution is necessary when trying to display a single "#" in a comment or message. Consider the following example.

```
//PAUSE Insert a disk in drive #1
```

If the JCL file was execute only, this line would be properly displayed. However, if the JCL were compiled, an error would occur. For this line to be properly displayed in a compiled JCL, it would have to be written as:

```
//PAUSE Insert a disk into drive ##1
```

The double pound sign is a special case, and lets the JCL compiler know that you wish a single # sign to be displayed, and do not wish to start a substitution field.

Another practical use for substitution fields is copying password protected files from one drive to another. In this example, a group of files will be copied from drive Ø to a drive specified in the DO command. Also, the user will have to supply the proper password for the copies to work.

```
. MOVE/JCL file transfer
copy program1.#P#:Ø :#D#
copy program2.#P#:Ø :#D#
copy program3.#P#:Ø :#D#
copy program4.#P#:Ø :#D#
//EXIT
```

This JCL would be done with a command such as "DO MOVE (D=2,P=SECRET). Now, as long as the password for the files were SECRET, the JCL would move the files from drive Ø to drive 2. If the wrong password were used, the appropriate LDOS error would be displayed and the JCL would abort.

Substitution fields can also be added together, or concatenated, to create new fields. The next example shows how this is done.

```
. ADD/JCL
copy #F##E#:Ø :1
copy #F1##E#:Ø :1
//EXIT
```

This example uses two substitution fields, one for the filename and one for the extension. The results of a DO command such as "DO ADD (F=SORT,E=/CMD,F1=SORT1)" would produce the following SYSTEM/JCL file after compiling:

```
. ADD/JCL
copy SORT/CMD:Ø :1
copy SORT1/CMD:Ø :1
//EXIT
```

As in previous examples, the //IF and //ASSIGN macros could be used to allow a single token to select the F, F1, and E tokens.

COMBINATION OF FILES

Most of the JCL examples in the previous sections have been very short. In a practical operating environment, this is often the case. However, each of these small files is taking up the minimum disk allocation of one gran and using one directory entry. Also, you may sometimes wish to duplicate a JCL file inside of another, without having to retype the lines. To allow this, the //INCLUDE macro and the LABEL feature of JCL can be used.

//INCLUDE

The //INCLUDE macro is used to merge together two or more JCL files during the compile phase. The correct syntax is:

```
//INCLUDE filespec
```

> Before describing the //INCLUDE any further, one point MUST be emphasized - an //INCLUDE macro CANNOT be the last line in a JCL file. If it is, a RECORD NUMBER OUT OF RANGE error will occur, and the JCL will abort.

The filespec would be the name of the JCL file to be included. This command is similar to specifying the filespec in a DO command line. However, it is NOT allowable to enter tokens or other information after the file name, and any information after the filespec will be ignored. If you need to pass tokens to the included program, they will have to be established in the program that is doing the //INCLUDE. This next example will show two JCL files and the results of the compile phase.

```
. TEST1/JCL                 . TEST2/JCL
. comment line 1            . This comment is included
//INCLUDE TEST2
. comment line 2
//EXIT
```

The command "DO TEST1" would produce the following SYSTEM/JCL file:

```
. TEST1/JCL
. comment 1
. TEST2/JCL
. This comment is included
. comment line 2
//EXIT
```

As you can see, the compiling starts with the file named in the DO command line. As soon as the //INCLUDE is reached, all lines in the second JCL file are processed, and then the compiling returns to the rest of the original file. There is no limit to the number of //INCLUDE macros you can use other than having enough disk space for the resulting SYSTEM/JCL file.

For example, let us assume that the TEST2/JCL file contains a procedure that you wish to repeat three times, with pauses in between. You could re-write the TEST1/JCL file as follows:

```
. TEST1/JCL
//PAUSE Initial pass now ready
//INCLUDE TEST2
//PAUSE Get ready for pass 2
//INCLUDE TEST2
//PAUSE Get ready for pass 3
//INCLUDE TEST2
//EXIT
```

As should be evident, this JCL will compile to a series of pauses with the TEST2 procedure done after each pause.

JCL LABELS

The LABEL feature of JCL will allow you to permanently merge together many small JCL procedures into one large file, and then access those procedures individually. This will save disk space and directory entry space for you. The format for a LABEL is:

```
@LABEL
```

The label name can be up to 8 characters long, either upper or lower case letter A-Z or a-z, or the numbers Ø-9. Following is a brief example of a JCL file containing labels:

```
. TEST/JCL label example
@FIRST
. this is the first procedure
//exit
@SECOND
. this is the next procedure
@THIRD
. this is the last procedure
```

This file contains three labels. To select any procedure, specify the label on the DO command line. DO TEST (@FIRST) would start compiling with the first line after the @FIRST label. The following rules determine how much of a labeled JCL file will be included in the compile phase:

1) If no label is specified on the DO command line, all lines from the beginning up to the first label will be compiled.

2) If a label is specified, compiling will include all lines until the next label or the end of the file is encountered.

Doing the TEST/JCL file using the @FIRST label would write the first comment and the //EXIT macro to the SYSTEM/JCL file for execution. Specifying either of the other labels would include only the appropriate single comment line. If the file were done with no label specified, only the initial execution comment ". TEST/JCL label example" would be written out.

There is no limit to the size of a labeled procedure. They may range from one to as many lines as you can fit on your disk. The only requirement is that a JCL file containing labels must be compiled.

When using labels in a JCL file, one word of caution is necessary. It is recommended that the file start with a comment line or some executable line other than a label. Consider the following short example:

```
@FIRST
. Print this comment
```

Now, if a DO command were to do this file without specifying the @FIRST label, the following would result. First the compiling phase would get the first line, see that it is a label, and quit. This is normal, as the compiler will start with the first line and continue to the first label or the end of the file. Since the compile is complete, the SYSTEM/JCL file would be executed! In other words, whatever lines had been compiled to the SYSTEM/JCL file from a previous DO command would now be executed. Needless to say, this is NOT what normally is desired.

ADVANCED JCL COMPILING

The previous section on JCL compiling showed the basic uses of tokens and compilation macros. If you do not understand the SIMPLE JCL COMPILING section, please re-read it. If you actually type in and try the examples, you should have an understanding of how to structure a compiled JCL file. This section will describe additional features, and show different ways to accomplish logical decision branching. It will be laid out as follows:

1] Using the Logical Operators
2] Using nested //IF macros
3] Using nested //INCLUDE macros
4] Use of the special % symbol

1] USING THE LOGICAL OPERATORS

There are three logical operators available for use with the //IF macro. These operators will specify the type of logical testing of the tokens. They are AND, OR, and NOT, represented as follows:

AND is represented by the ampersand (&).
OR is represented by the plus (+).
NOT is represented by the minus (-).

All previous examples of //IF have tested the logical truth or falseness of a token, such as "//IF token". By using the logical operators, more complex and more efficient testing can be done. Consider the following series of examples using the tokens A and B:

```
//IF A                      "if A" - true only if A is true
. include these lines
//END

//IF -A                     "if not A" - true only if A is false
. include these lines
//END
```

By using NOT, it is possible to see not only if a token is true, but to see if it is false. This provides an alternative method to select a block of lines for compiling.

```
//IF A+B                    "if A or B" - true if either is true.
. include these lines
//END

//IF A&B                    "if A and B" - true only if both are true
. include these lines
//END
```

These examples show how multiple tokens may be tested in a single //IF statement. In the OR example, the //IF will test true if either A or B were true. The AND example requires that both A and B be true to include the lines up to the //END. It is allowable to use any combination of logical operators in an //IF statement. When doing so, it is important to know how the statement will be evaluated.

Evaluation of the statement will be from left to right.

Parentheses are not allowed, and will abort the JCL compiling.

All logical operators have the same precedence.

Following are some examples of //IF statements using multiple logical operators:

```
//IF A+B+C          if either A or B or C is true
//IF -A&-B          if A is false and B is false
//IF -A&B+C         If A is false and either B or C is true.
```

As you can see, the logical operators can be combined to test almost any arrangements of tokens you may need. This is especially handy for setting up default conditions and in checking for missing tokens, as the following examples will demonstrate.

```
. CHECK/JCL
//IF -S
//ASSIGN S=Ø
//END
//IF -D
//ASSIGN D=2
//END
```

```
. CHECK1/JCL
//IF -S+-D
//. You MUST enter S and D!
//QUIT
//END
```

Let us assume that the S and D in these two examples will be used as source and destination drivespecs later in the file. The CHECK example tests S and D individually, and assigns them default values if they were not true. The CHECK1 example, on the other hand, is structured so that both S and D must be true, or the JCL will abort. The //IF line in the CHECK1 example reads "if not S or not D". Although the use of logical operators may seem harder to understand than a simple "//IF token" statement, it does provide easier ways to determine if needed tokens have been specified.

2] NESTED //IF MACROS

By definition, a conditional block begins with an //IF and concludes with an //END. When the //IF evaluates true, the lines between the //IF and the //END are compiled. It is also possible to include other //IF-//END blocks within the main conditional block, in effect nesting them. As previously explained, the //ELSE macro provides an alternative course of action in case an //IF evaluates false. It is also allowable to have more //IF-//END statements following the //ELSE. Refer to the following examples:

```
. TEST/JCL
//IF A
. comment 1
//ELSE
//IF B
. comment 2
//END           (ends the //IF B statement)
//END           (ends the //IF A statement)
```

The TEST example is fairly straight forward. If A evaluates true, comment 1 would be written out, and the //ELSE would be ignored. If A was false, B would be tested. The comment 2 would be written out only if B was true. Notice the two //END macros. As stated earlier, there must be one //END for every //IF. What might not be readily apparent is which //END matches which //IF.

In this example, there are comments in parentheses to show the way the //ENDs correspond to the //IFs. It is allowable to use this type of comment identifier in real JCL files. You will find that labeling //END macros greatly increases the readability of the file, especially when editing a file that you have created some weeks (or months) previously.

This next example and the following description again show how nested //IFs are evaluated.

```
First IF        //IF A
                . Comment A
Second IF       //IF B
                . Comment B
Third IF        //IF C
                . Comment C
                //END           (ends Third IF)
                //END           (ends Second IF)
                . Comment D
                //END           (ends First IF)
```

Evaluating this example produces the following results. When the first //IF is false, all lines up to the corresponding //END will be ignored. Since the last //END corresponds to the first //IF, none of the lines in this example would be written out to the SYSTEM/JCL file.

Assuming from this point on that the first //IF evaluates true, two lines will always be written out. These are the Comment A and Comment D lines.

The first nest is //IF B. If B is true, the Comment B line will be written out. If B is false, all lines, including the //IF C block, will be skipped up to the //END corresponding to the //IF B.

The next nest is //IF C. The only time this will be considered is if both A and B have tested true. As normal, if C is true the Comment C will be written out.

Although not shown in the example, it is perfectly allowable to use the logical operators when nesting //IFs. Again, note the use of the comments after the //END macros. Using comments such as these will help you follow the logic flow, especially until you become familiar with using nested //IF macros.

3] NESTED //INCLUDE MACROS

When using the //INCLUDE macro, it is allowable for the included file to also contain another //INCLUDE macro. This is referred to as nesting. Briefly stated, the following rules will apply:

The maximum nest level will be ten active //INCLUDE macros.

An //INCLUDE macro cannot be the last line in a JCL file.

The following example uses three files to show how the lines in nested //INCLUDE files are processed:

```
File #1 =>   //. NESTØ/JCL
           . nested procedure example
           //INCLUDE nest1
           . this is the end of the primary JCL
           //EXIT

File #2 =>   //. NEST1/JCL
           . this is the first nest
           //INCLUDE nest2
           . this is the end of the first nest

File #3 =>   //. NEST2/JCL
             . this is the second nest
```

The above will result in a nest level of two (two pending //INCLUDEs). If these three JCL files are saved as NESTØ/JCL, NEST1/JCL, and NEST2/JCL, and the NESTØ/JCL is compiled and executed, it will result in the following dialogue:

```
//. NESTØ/JCL
//. NEST1/JCL
//. NEST2/JCL
. nested procedure example
. this is the first nest
. this is the second nest
. this is the end of the first nest
. this is the end of the primary JCL
```

The three compilation comments will be shown immediately as the JCL file is compiled. When the compilation phase is complete, the compiled SYSTEM/JCL file will be executed. In this example, the execution phase will merely display a series of execution comments. As you can see from the order of the displayed comments, the files are executed similarly to nested FOR-NEXT loops in BASIC. After all //INCLUDEs are detected, the innermost (last encountered) //INCLUDE file completes execution first, with execution proceeding back towards the original //INCLUDE.

The //INCLUDE macro can very easily be used to compile a large JCL procedure from a series of smaller JCL routines. If the finished SYSTEM/JCL file is a procedure that will be executed many times, it may easily be saved by copying SYSTEM/JCL to a file with another name.

4] USING THE SPECIAL % SYMBOL

The % symbol is used to pass character values to the system as though they came from the keyboard. The proper syntax is the % symbol directly followed the the hexadecimal value of the character, such as %1F. The following values are all valid after the % symbol:

HEX VALUE	RESULT
Ø9	TAB 8 SPACES
ØA	LINEFEED
1F	CLEAR SCREEN

Also, the value of any printable character may used. although this is not normally done.

When using the clear screen character, it should be placed at the start of a line. For example:

```
%1F. This is a comment line
%1F//PAUSE Insert disk in drive 1, press <ENTER>
```

In both examples, the screen will clear and the JCL line will be displayed in the top left corner of the screen. The TAB (Ø9) and linefeed (ØA) characters can be used to position comments or other lines in different positions on the screen. These characters should always be placed AFTER the period in the comment line, or after the macro in an executable line. For example:

```
.%Ø9%Ø9 This comment will be tabbed 16 spaces
//PAUSE %ØA%ØA%ØA This line will appear 3 lines down
```

When this file is compiled and executed, the comment line will be tabbed over 16 places. Notice that the first % is after the period in the comment. If the symbol were before the period, LDOS would not recognize it as a comment line and the JCL would abort. In the //PAUSE line, the //PAUSE would be displayed, and the remaining message line would be displayed 3 lines lower on the screen. Using the tab and linefeed characters in this manner can sometimes help improve the readability of the messages displayed during JCL execution.

Although any other ASCII character may also be sent to the keyboard, the system generally will not respond to any other characters less than a space (X'2Ø'). Characters above this value may be used with the % symbol, but it is easier merely to type them in as a command line in the JCL file.

INTERFACING WITH APPLICATIONS PROGRAMS

This section will describe how to use JCL to start up and even control applications programs. After reading this section, it should be very easy for a user to interface between an application and the JCL processer. Two languages will be discussed - LBASIC and Z-8Ø assembler.

INTERFACING WITH LBASIC

A JCL file is the perfect method to interface between the operating system and the LBASIC language. JCL can be used to create procedures that require only the insertion of a diskette to start up a program. Additionally, you may utilize the features of JCL from within an LBASIC program.

To use a JCL file to initiate an automatic start up of an LBASIC program, it will be necessary to use the AUTO library command to execute a JCL file. Assuming the JCL file is named LBAS/JCL, issuing the command AUTO DO LBAS/JCL will automatically execute the desired LBASIC program every time the computer is booted with the AUTOed system disk.

The actual JCL file should be laid out as this next example shows:

```
ESTABLISH any necessary drivers, filters, or other LDOS options.
LBASIC (any needed parameters)
RUN"PROGRAM/BAS"
//STOP
```

This example shows the normal way to execute an LBASIC program from a JCL file. Any necessary system options are established, LBASIC is entered with any necessary parameters (such as memory size and number of files), and the LBASIC program is loaded and executed. Notice the termination macro //STOP used in the JCL file. As explained in the JCL EXECUTION section, if this macro was not used or if the //EXIT macro was used instead, the JCL file would return to the LDOS Ready prompt as soon as the first keyboard entry was requested. The //STOP macro will terminate the JCL execution and leave keyboard control with LBASIC.

It is not necessary to use the AUTO library command when using a JCL file to execute an LBASIC program. The DO command may be entered directly from the LDOS Ready prompt, such as DO LBAS/JCL.

To execute a JCL file once you have entered LBASIC, the command format is:

```
CMD"DO filename"
```

This command can be typed in directly or may be entered as an LBASIC program line. As with any CMD"dos command" function done from LBASIC, it will be necessary to have approximately 4K of free memory available or an "Out of memory" error will occur. Also, any JCL file that will be called from LBASIC should have the //EXIT termination macro so control will return to LBASIC when the JCL is completed. For example, suppose you wished to use the JCL //ALERT macro to inform you when a lengthy LBASIC procedure had completed. After the lines containing the LBASIC procedure, you could have an LBASIC program line such as:

```
1ØØØ CMD"DO =ALERT/JCL:Ø"
```

which might execute the ALERT/JCL file:

```
. Your procedure is complete , press <ENTER> to resume
//ALERT (1,Ø,7,Ø)
//EXIT
```

When LBASIC reached line 1ØØØ, the JCL file ALERT/JCL would be executed. This would send a series of repeating tones out the cassette port. Assuming you had a suitable amplifier hooked to the cassette cable, you would be notified you that your LBASIC procedure had completed. Pressing <ENTER> would end the JCL alert and return you to LBASIC. There are two important points to be made about this example. First, the comment line in the ALERT/JCL file is absolutely necessary, as a JCL file cannot start with an execution macro. Second, the //EXIT termination macro is mandatory to assure that keyboard control will be returned to LBASIC.

Although the example demonstrated an execute only JCL file, it is perfectly allowable to call compiled JCL procedures from LBASIC. You may even construct a CMD"DO filename (parameters)" command using LBASIC string substitution.

Anytime you wish to use a CMD"DO filename" command to execute a JCL file and NOT return to LBASIC, you will have to change the format of the command. This is especially important if the new JCL file is one that will also enter LBASIC and run a program. To do these types of JCL files from LBASIC, use the format:

```
CMD"I","DO filename"
```

Using this format for the command will assure that a proper exit is made before the new JCL file is started.

Controlling an LBASIC program

In some cases, the prompts in a BASIC program can be answered with a line from a JCL file. This will be true if the program uses the INPUT or LINEINPUT BASIC statement to take the input. If the INKEY$ statement is used, response will have to come from the keyboard rather than from a JCL file. If the program is using the proper input method, creating a JCL for TOTALLY HANDS-OFF OPERATION can be done as follows:

Run through the program normally, making note of every prompt to be answered.

Create a JCL file to enter LBASIC and run the program as explained above in the LBAS/JCL example, leaving off the //STOP macro.

Now, add the responses to the prompts as lines in the JCL file.

Using this method will provide automatic program execution. All that is required is for you to have the proper responses for any program prompts as lines in the JCL file. Terminating the JCL file will depend on what needs to be done when the application program has completed. If you desire to run more programs, you could add the proper RUN"PROGRAM" line to the JCL file, followed by any needed responses to program prompts. If you desired to return to the LDOS Ready mode, you could end the file with the //EXIT macro. Ending the file with a //STOP would leave you at the LBASIC Ready prompt when the program completes.

INTERFACING WITH Z-8Ø ASSEMBLER

It is very simple to interface an assembly language program with the DO processer. All programs that utilize the line input handler (identified as @KEYIN in the System Entry Point technical section) will be able to accept "keyboard" input from the JCL file, just as though you typed it in when the program was run. This gives the capability of pre-arranging the responses to a program's requests for input, inserting the responses into the JCL file, initiating the procedure, then walking away from the machine while it goes about its business of running the entire job.

Keyboard input normally handled by the single-entry keyboard routines (@KBD, @KEY, and LBASIC's INKEY$) will continue to be requested from the keyboard at program run time and will not utilize the JCL file data for input requests. Thus by understanding fully the dynamics of JCL processing, you can write applications that take full advantage of the power inherent in the Job Control Language.

JCL - PRACTICAL EXAMPLES

It is virtually impossible to explore all the possibilities that exist concerning the creation of JCL files. The examples that follow will give you some ideas as to how JCL may be used to make your day to day operating of the LDOS system even more efficient.

Example #1

This example will show you how to SYSRES system modules using a JCL file. The modules that will be resided are 2,3,8 and 1Ø. These modules are required to be resident in order to perform a backup by class between two non-system diskettes in a two drive system. The JCL file used to perform such a function may look something like this:

```
.BURES/JCL - JCL used to SYSRES system modules 2,3,8,1Ø
SYSTEM (SYSRES=2
SYSTEM (SYSRES=3
SYSTEM (SYSRES=8
SYSTEM (SYSRES=1Ø
.end of BURES/JCL
```

When executed, this JCL file will cause the system modules 2,3,8 and 1Ø to be resided in high memory. Because this JCL uses no labels or compilation macros, the compilation phase may be skipped.

Example #2

This example will show you a JCL file that may be used to perform diskette duplication. A minimum of three drives will be required. Drive Ø will contain a system diskette with the JCL file. Drive 1 will be the source diskette of the backup. Assume that the diskette name is MYDISK, and it is a single sided, 35 track, single density diskette, with a master password of PASSWORD. Drive 2 will be used as the destination of the backup. It may contain either a new, unformatted diskette or a diskette which has been previously formatted and contains information. The following JCL may be used to perform such a duplication:

```
.DUPDISK/JCL - Disk duplication JCL
//pause Source in 1, Destination in 2, <ENTER> when ready
format :2 (name="mydisk",sden,cyl=35,q=n,abs)
//pause format ok? <ENTER> if yes, <BREAK> if no
backup :1 :2
.end of backup - will now restart JCL
do *
```

The second line of the JCL will cause the computer to pause until the <ENTER> key is pressed. This will allow you to insert the proper diskettes into drives 1 and 2. Once this has been done, you may press <ENTER>, and the third line of the JCL will be executed.

The format line passes the parameters NAME, SDEN, and CYL to the format utility. Note that the number of cylinders, diskette name and diskette password of the destination diskette must be an exact match of the source disk. If they do not exactly match, the backup command that follows will issue some type of unwanted prompt, which would cause the JCL to abort. Also, note that the parameters Q=N and ABS were specified. Both are necessary. The Q=N parameter causes the computer

to use the default of PASSWORD for the master password, and hence the "Master Password" prompt will be bypassed. The ABS parameter ensures that no prompt will appear if the destination diskette contains data.

The pause after the format statement allows you to check whether or not the format was sucessfull. If the destination diskette was formatted properly, you may press <ENTER> to continue the JCL. If tracks were locked out during the format, you may press <BREAK>. Realize that doing so will cause the JCL to abort, and it will be necessary to restart the JCL activity.

After <ENTER> has been pressed in response to the second pause, the backup will take place. If any error occurs during the backup, the JCL will be aborted.

Once the backup has been completed sucessfully, the comment line will appear, and the DO * command will be executed. This command will cause the SYSTEM/JCL file to be executed. Realize that if this is to be a duplicating JCL, the compilation phase cannot be skipped.

L B A S I C T A B L E O F C O N T E N T S

INTRODUCTION TO LBASIC

Contained on your LDOS Master Diskette is a program named LBASIC/CMD (LBASIC). As was noted in the GETTING STARTED portion of the manual, your computer contains two different types of memory, ROM (Read Only Memory) and RAM (Random Access Memory). Your computer, as received from your dealer, does contain a ROM BASIC. This ROM Basic does allow you some capabilities of programming in the Basic language. However, ROM Basic does not allow you to interface with your disk drives when programming, and hence does not fully utilize your TRS-80 disk system. For this reason, LBASIC has been included with your LDOS system. LBASIC is an extension of ROM Basic and resides in RAM. LBASIC utilizes commands found in ROM Basic, and adds commands to ROM Basic which will allow you to interface your Basic programs with the disk operating system. Because of this, programs and data files created under LBASIC may be stored on your disk drives. In addition, many LDOS functions may be performed when programming in LBASIC, without having to return to the "LDOS Ready" level.

This section will detail all enhancements to ROM Basic which are contained in LBASIC. Commands which are inherent in ROM Basic will not be detailed in this manual. Refer to your Radio Shack owner's manual (Model I Level II Basic Manual or Model III Operation and Basic Language Reference Manual) for a complete description of ROM Basic commands.

One final point concerning the LBASIC section. It is written as a reference guide only. All commands will be explained in terms of the function which they serve. In no way will this section serve as a tutorial on implementation of these commands. There are many such books currently on the market that deal with using a "Microsoft compatible" disk Basic for generalized and specific applications. If you require tutorial aids for implementing LBASIC, contact your computer dealer for a list of such material.

E N T E R I N G L B A S I C

This is the syntax to be observed when entering LBASIC.

```
LBASIC (parm,parm,...,parm) command

LBASIC *  used to re-enter LBASIC with the program and
          the variables intact.


The allowable parameters are as follows:

BLK=    parameter that specifies Blocked file mode,
        either ON or OFF. ON is the default.

FILES=  parameter that specifies the maximum number of
        files LBASIC will be able to access (1 to 15).
        If not specified, 3 is assumed.

MEM=    parameter to set the highest memory address
        to be used by LBASIC. All memory above this
        address will be "protected". If not specified,
        all memory up to HIGH$ will be available.
        This parameter may be specified as either a
        decimal (MEM=nnnnn) or hexadecimal (MEM=X'xxxx')
        value.

EXT=    parameter used as a switch to turn on or off
        the default file extension "/BAS" used with
        the LBASIC commands LOAD, RUN, MERGE and SAVE.
        Either ON or OFF may be specified. If not
        specified, ON is assumed. See LBASIC - GENERAL
        INFORMATION for a detailed description.

HIGH    Model III parameter that sets the cassette baud
or      rate, either HIGH or LOW (HIGH=15ØØ and LOW=5ØØ).
LOW     The default is HIGH. ** If HIGH is used, the
        HITAPE command must be issued prior to entering
        LBASIC. **

command - This may be any valid LBASIC command
          which will execute immediately upon entering
          LBASIC, such as RUN"MYPROG/BAS", AUTO1ØØ, etc.

abbr: BLK=B, FILES=F, MEM=M, ON=Y, OFF=N, HIGH=H, LOW=L
      EXT=E
```

Any or all of the parameters may be specified when entering LBASIC. If no parameters are specified, the default values listed in the above syntax block will be assumed.

To provide compatibility with existing application programs and documentation, a short program called BASIC/CMD is provided. When entering a command, the term "BASIC" may be used in the command line instead of "LBASIC". The program BASIC/CMD will translate the command into the equivalent LBASIC command, and will also issue an EXT=OFF parameter.

The "command" specification is also optional. If not specified, you will enter into LBASIC, and the following lines will appear on the screen:

```
LBASIC - Version 5.x.x - mm/dd/yy
(C) 19xx by Logical Systems Incorporated

Ready
```

The "Ready" prompt will indicate that LBASIC is ready to accept any command that you wish to give it.

If you have rebooted the system, or have performed an exit from LBASIC to the operating system (usually done by issuing a CMD"S" command), and wish to re-enter LBASIC, you may enter the command:

```
LBASIC *
```

at the LDOS Ready level. Doing so will cause LBASIC to be re-entered, and any program that was resident in memory prior to performing the exit to the LDOS Ready level will remain intact. Be aware of the fact that if LBASIC * is used to re-enter LBASIC from the LDOS Ready level, any commands which affect HIGH$, or any commands that utilize memory (such as BACKUP and COPY) may cause your LBASIC program to be overwritten with other information. For this reason, LBASIC * should only be used as a last resort. You may perform certain LDOS Library commands directly from LBASIC (using the CMD command). If a function cannot be performed from LBASIC using the CMD command, it is not advised to re-enter Basic using LBASIC * if you have exitted back to LDOS to perform the command, as the integrity of your program will be suspect.

Example

One of the following commands may be given if you wish to enter LBASIC in the blocked file mode with 2 files open, having memory protected up to location 6144Ø (X'FØØØ'). Also, you wish to have the program MYPROG/BAS loaded upon entering LBASIC.

```
LBASIC (FILES=2,MEM=6144Ø,BLK=ON,EXT=ON) LOAD"MYPROG/BAS"
LBASIC (F=2,M=X'FØØØ') LOAD"MYPROG"
```

Issuing either of the above two commands will produce the same results. The second command above uses the abbreviations F and M for FILES and MEM, and also utilizes the default "ON" for the BLK and EXT parameters. Note that the extension for the program MYPROG/BAS need not be specified if EXT is ON. Also, realize that for either of the above commands, if HIGH$ is lower than 6144Ø (X'FØØØ'), an "Out of Memory" error will occur, and you will be returned to the LDOS Ready prompt without entering LBASIC.

L B A S I C - G E N E R A L I N F O R M A T I O N

ABBREVIATED COMMANDS

Each of the following LBASIC commands may now be represented as single characters. When using a single character command, the effect will be identical to using the entire word. This abbreviated form is only acceptable when typed on a command line, not in a program line or JCL file.

A represents the command AUTO.

D represents the command DELETE.

E represents the command EDIT.

L represents the command LIST.

The following commands are implemented by pressing the indicated key as the first character in the command line. No carriage return is necessary; the indicated action will take place immediately. Note that any of the following single key commands must be the first character entered after the "Ready prompt" appears.

. (period) This will perform the same function as "LIST.<ENTER>", which will instruct LBASIC to list the currently active line.

, (comma) This will perform the same function as "EDIT.<ENTER>", which will instruct LBASIC to enter the "edit mode" for the currently active line.

<UP ARROW> This will cause LBASIC to display the next lower numbered line in the program.

<DOWN ARROW> This will cause LBASIC to display the next higher numbered line in the program.

<LEFT ARROW> This will cause LBASIC to display the first line of the program.

<RIGHT ARROW> This will cause LBASIC to display the last line of the program.

DEFAULT EXTENSIONS

LBASIC allows you to utilize the default extension of /BAS when issuing the LOAD, RUN, MERGE and SAVE commands. If the EXT parameter is set to ON (or not specified) when entering LBASIC, all filespecs used with the above commands that do not have extensions will be assigned the extension /BAS. If EXT is on and an extension is specified, the extension used in the filespec will override the default extension.

If EXT is ON and the file in question has no extension, it must be specified as "filename/" (i.e. the "/" will override the default /BAS). If the EXT parameter is turned OFF when entering LBASIC, all file extensions will have to be specified.

FILE BLOCKING

LBASIC provides a Blocked file mode (which has often been misnamed Variable Length Files). This mode allows files with Logical Record Lengths (LRL) of less than 256 bytes to be created and accessed. Any record length from 1 to 256 bytes will be allowed, even if the record size is not evenly divisible into 256.

All blocking and de-blocking across "sector boundaries" will be performed by LDOS. In this way, user records can span across sectors to provide maximum disk storage capacity. If the LRL is not specified when OPENing a Random file, 256 will be assumed. Note that an LRL of Ø will signify a 256 byte LRL. Enhancements have also been made to the allowable methods of OPENing both Random and Sequential type files (See OPEN).

If the Blocked file mode is ON, each file declared when entering LBASIC will take 544 bytes of memory. If the Blocked mode is OFF, each file will take 288 bytes.

LBASIC OVERLAYS

Three overlays are present on a Master LDOS diskette. They are:

LBASIC/OV1 - This overlay contains the renumbering program used with the LBASIC CMD"N" function. It may be killed if no renumbering will be done.

LBASIC/OV2 - This overlay contains the cross reference program used with the LBASIC CMD"X" function. It may be killed if no cross referencing will be done.

LBASIC/OV3 - This overlay contains the LBASIC error handling and the sort routine used for the CMD"O" function. It MUST be present when using LBASIC.

PROGRAM PROTECTION

LBASIC programs may be protected with an "Execute only" password. This means that the program may be RUN, but not LOADed, LISTed, LLISTed, or otherwise examined. Any attempt to break the program execution and examine the program will cause the program to be erased from memory, and the message "Protection has cleared memory" will be displayed. The DEBUGger will also be disabled during program execution.

SINGLE STEPPING AN LBASIC PROGRAM

This new feature allows the LBASIC programmer to step through each program statement singly, with a "HOLD" after each step. To invoke this feature simply do a normal pause (<SHIFT @>), which will cause LBASIC to go into a wait state. While continuing to hold down the <SHIFT @> press the <SPACE BAR>, and the next LBASIC statement will execute. After execution of that statement the computer will immediately go into its wait state again. Holding down the <SPACE BAR> will execute statements at the normal keyboard repeat rate. If you press any key without holding down the <SHIFT @>, normal program execution will resume. Note that this feature also functions when listing a program.

TAPE ACCESS

Model I users need to disable the interrupts prior to performing tape I/O, and must re-establish them after the input/output has been performed. To disable the interrupts, use the LBASIC command - CMD"T" -. To enable the interrupts, use the command - CMD"R" -. See the LBASIC Commands Section for more information on these two commands.

Model III users need to do one of several things, depending on the type of tape involved. If you are dealing with a 5ØØ baud tape, you will need to specify the LOW parameter when entering LBASIC (Remember, if HIGH or LOW is not specified, the default will be HIGH). If you are dealing with a 15ØØ baud tape, you will need to establish the HITAPE utility. For more information on HITAPE, refer to the Utilities section of the LDOS manual.

L B A S I C C O M M A N D S

This section of the manual will detail commands found in LBASIC which are not included in ROM Basic. These commands will be listed in alphabetical order. For the novice, this type of grouping might be a bit confusing in terms of when and how these commands will be used. However, for the person who is somewhat versed in using a disk oriented Basic, this will be a very convenient way of locating information dealing with any LBASIC command.

&H - Hexadecimal Representation of a number

To represent a number in its hexadecimal format, you may use the characters - &H - as a prefix to the number. This may be useful when you wish to define an address for a user machine language subroutine (see DEFUSR).

One to four hexadecimal digits may follow the &H prefix. Hexadecimal digits consist of the numeric digits Ø-9, as well as the alphabetic letters A-F. The number represented using the &H prefix will always be taken as two's complement notation.

Examples

```
A=&H11    (A would be set equal to the decimal number 17).
A=&HA9    (A would be set equal to the decimal number 169).
A=&HFØØØ  (A would be set equal to the decimal number -4Ø96).
```

&O - Octal Representation of a number

To represent a number in its octal format, you may use the characters - &O - (or just - & -) as a prefix to the number.

One to six octal digits may follow the &O prefix. Octal digits consist of the numeric digits Ø-7. The number represented using the &O prefix will always be taken as two's complement notation. The largest octal number which may be represented is &O177777.

Examples

```
A=&O11    (A would be set equal to the decimal number 9).
A=&17ØØØØ (A would be set equal to the decimal number -4Ø96).
```

CLOSE - Close any or all open disk files

The CLOSE command is used in conjunction with the OPEN command. After a file has been opened, it is capable of being read from and/or written to. To disable this read/write capability of a disk file, a CLOSE of the file must be done. In addition, CLOSE will update the Mod flag, Mod date and end of file marker in the directory record of that file (provided the file has been written to). See OPEN for more information on file access.

The syntax for the CLOSE command may be in one of the following formats:

```
CLOSE
CLOSE #,...,#
```

The CLOSE command issued by itself will close all open files. The CLOSE #,...,# command will close only those files that have been opened with the specified buffer number (where # represents the buffer number used to define a particular file in an OPEN statement).

If you issue any command which will perform a CLEAR (such as EDIT, CLEAR or RUN), a global CLOSE will automatically be performed for you. However, if you issue a CMD"S", CMD"A", or CMD"I" command, closing of any open files will not occur. For this reason, you should always make sure files have been closed prior to exitting back to the LDOS Ready prompt.

CMD - Perform an LDOS or Special Command

The CMD command allows you to perform certain LDOS library and utility commands without having to leave LBASIC. In addition, there are 13 distinct parameters that may be used in conjunction with the CMD command which will allow you to perform various different functions. The syntax used for the CMD command is as follows:

```
CMD"dos command"
CMD"x" (Where 'x' is the letter assigned to the special command).
```

We will first describe how to use the CMD command to issue an LDOS command, after which we will explain the use of the 13 distinct parameters with the CMD command.

CMD"dos command"

LDOS Library commands and Utilities that do not affect HIGH$ may be executed from LBASIC by use of the CMD"dos command". The following examples should illustrate implementation of this feature

```
CMD"DIR :Ø"          - Will display a Directory of the disk in drive Ø.
CMD"DEVICE"          - Will display the device table.
CMD"LIST DAT1/SCR"   - Will list the file DAT1/SCR.
CMD"BACKUP :Ø :1"    - Will perform the designated Backup.
```

After the desired LDOS function has been completed, control will be returned to LBASIC with your program and variables intact. This type of CMD command will function whether it is called from LBASIC's command line or from within an LBASIC program. If performed from within an LBASIC program and an error occurs, or the CMD command is aborted with the break key prior to being completed, the appropriate error message will be displayed, or the message "System Command Aborted" will appear, and execution of the Basic program in question will be terminated. The command may also be contained within a string variable, such as the following format:

```
A$="DIR :Ø":CMD A$
```

Approximately 4K of free memory must be available for these types of CMD commands, or an "Out of Memory" error will occur.

CMD"*"

This command will send the contents of the screen to the printer. This will allow you to perform a screen print from within a program, without having to physically initiate the screen print. CMD"*" may also be issued from the LBASIC Ready prompt. Note that the JKL parameter of the KI/DVR need not be active to utilize this command.

CMD"A"

This command will perform an abnormal return to LDOS. Any active DO command will be cancelled.

CMD"B","switch"

This command will enable or disable the <BREAK> key, with "switch" being either ON or OFF. A string constant or string expression may be used to represent the "switch".

CMD"D"

Turns on and enters the system Debugger.

CMD"D","switch"

This command is similar to the CMD"D" command, with the following exceptions. The switch ON will turn on the system Debugger, but will remain in LBASIC. Pressing the <BREAK> key (or <CLEAR> <SHIFT> <D> keys if Minidos is active) will cause you to enter the Debugger. The switch OFF will turn off the Debugger.

CMD"E"

This command will return the last LDOS error message encountered. If no error has been encountered, the message "No Error" will appear. CMD"E" may be useful when you wish to pinpoint the exact nature of an error. LBASIC's error dictionary is not as extensive as that found in LDOS, hence various LDOS errors can produce the same LBASIC error message. Performing a CMD"E" will give you the exact error seen by LDOS. This may be of use when you get the LBASIC error message "Disk full or write protected" or "Disk I/O error".

CMD"I","dos command"

This command functions much the same as the CMD"dos command", with the exception that control will return to LDOS after the "dos command" has been executed. Dos command can be represented as a string constant or a string expression. If represented as a string constant, it must be contained within quotes.

CMD"L","filespec"

This command will load a Load Module Format file (a machine language program) into memory, much the same as the LOAD Library command does. Filespec may be represented as a string constant or a string expression. If represented as a string constant, it must be contained within quotes.

CMD"N"

This command provides you with a program line renumbering function. For the specific parameters involved with this command, please refer to page 75 at the end of the LBASIC section.

CMD"O",number of elements to sort, first element of array to sort

This command will allow you to sort a single dimensioned string array. The sort will start at the element specified, and will sort the number of elements specified. The number of elements to be sorted must not force the sort past the end of the array. In order to utilize the CMD"O" function, the module LBASIC/OV3 must be present on a disk in the system.

Example

```
CMD"O",15,A$(1Ø)
X=15:Y=1Ø:CMD"O",X,A$(Y)
```

Issuing either of the above commands will cause a sort to be performed on the A$ array. After the sort has been finished, elements 1Ø-24 will be sorted in alphabetical order.

CMD"P",variable

This command will return the printer status in the variable specified. The variable may be any type, including a string. The value will have the bottom 4 bits stripped before being passed back to LBASIC.

CMD"R"

Model I - This command enables the interrupts. It should be performed after a CMD"T" command has been issued. For more information see the CMD"T" command.

Model III - This command will turn on the clock display.

CMD"S"

This command is the normal way to return to LDOS Ready from LBASIC.

CMD"T"

Model I - This command will disable the interrupts. It must be issued prior to performing tape I/O. After the tape I/O has been completed, the interrupts must be enabled with the CMD"R" command.

Model III - This command will turn off the clock display.

CMD"X"

This command provides you with a program cross reference function. For the specific parameters involved with this command, please refer to page 76 at the end of the LBASIC section.

CVD - Convert to Double Precision

This command is used to convert an 8 byte string into a double precision number. The 8 byte string should be a representation of a double precision number stored in compressed format. This command is used primarily to uncompress double precision values which have been retrieved from a disk file (in essence, it performs the opposite function of the MKD$ command). For more information on storing a double precision number in compressed format in a disk file, refer to the MKD$ command.

Example

```
A#=CVD(A$)
```

In the above example, assume that A$ is an 8 byte string which represents a compressed double precision number. After the above command is performed, A# will be set equal to the uncompressed number that A$ represents.

Realize that you are not limited in using CVD to assign a value to a variable. The value generated by a CVD command may be used directly (e.g. PRINT CVD(A$), or IF CVD(A$)<1ØØØØØ THEN GOTO 1ØØØ).

CVI - Convert to Integer

The CVI command functions identically to the CVD command with the following exceptions. The CVI command will convert a two byte string into an integer. This two byte string should be a representation of an integer stored in compressed format. CVI performs the opposite function of the MKI$ command. The value returned from the CVI function will be an integer within the range of -32768 to +32767 inclusive. For more information, refer to the MKI$ command.

Example

```
A%=CVI(A$)
```

In the above example, assume that A$ is a 2 byte string which represents a compressed integer. After the above command is performed, A% will be set equal to the uncompressed number that A$ represents.

CVS - Convert to Single Precision

The CVS command functions identically to the CVD command with the following exceptions. The CVS command will convert a four byte string into a single precision number. This four byte string should be a representation of a single precision number stored in compressed format. CVS performs the opposite function of the MKS$ command. For more information, refer to the MKS$ command.

Example

```
A!=CVS(A$)
```

In the above example, assume that A$ is a 4 byte string which represents a compressed single precision number. After the above command is performed, A! will be set equal to the uncompressed number that A$ represents.

DEF FN - Define Function

There are many intrinsic functions provided for you in ROM Basic and LBASIC (i.e. VAL, STR$, SIN, etc.). The DEF FN command allows you to define your own functions. This may be of use when performing lengthy calculations at different points in your program when you do not use the same variable names to perform these calculations.

The syntax for the DEF FN command is as follows:

```
DEF FNfunction name(parm,...,parm)=expression
```

The "function name" is the name that you will assign to the function, and has the same restrictions as those imposed on a variable name. The function name must be of the same type as the value to be returned from the function.

The "(parm,...,parm)" is a list of variables to be passed to the function. The variable names used are local to the function, and act as dummy variables. They will have no effect on other variables in the program which have the same name. However, they must be of the same variable type as the variable represents in the function (i.e. string, integer, single precision, double precision). Also, if more than one variable is to be passed to the function, they must be passed in the same order as that defined in "(parm,...,parm)" (see example below).

The "expression" represents how the variables passed to the function are to be worked on.

The example below will show how to define and invoke your own functions.

Example

This example will show how to create a function which will build a filespec. This function will be passed three variables; the filename, the file extension, and the drive specification. It will return a filespec in the form - filename/ext:d. A DEF FN statement to create such a function might take on the following format:

```
DEF FNFS$(X$,Y$,Z%)=X$+"/"+Y$+":"+MID$(STR$(Z%),2,1)
```

The function name is FS$, and is of string type, since a string value will be returned from the function.

Three values will be passed to the function. The first two values passed will be strings, while the third value will be an integer.

The function that will be performed is as follows. The first string passed to the function will have a '/' added onto the end of it, after which the extension, a ':', and the drivespec will be added to the string, respectively.

The following example will illustrate how to invoke the function, as well as changes that will occur to the variables involved.

```
X$="HELLO":F$="MYPROG":E$="BAS":G%=2
F1$=FNFS$(F$,E$,G%)
F2$=FNFS$(E$,F$,G%)
```

After execution of the above three lines, the following variables will be assigned the following values:

```
X$="HELLO"     F1$="MYPROG/BAS:2"     F2$="BAS/MYPROG:2"
F$="MYPROG"    E$="BAS"               G%=2
```

Note that the value of X$ does not change from the calling of this function. Also note the difference between F1$ and F2$. The order in which variables appear when invoking the function determines the value that will be returned from the function.

As a final note on DEF FN, the value returned from the function can be used directly, and does not have to be stored in a variable (e.g. PRINT FNFS$(F$,E$,G%)).

DEFUSR - Define the entry point to a user machine language subroutine

This command is used to define the starting address (entry point) of a user created machine language subroutine. A DEFUSR statement must be done prior to utilizing the machine language subroutine via the USR command. The syntax for the DEFUSR statement is:

```
DEFUSRn=xxxx
```

where n is a numeric constant (Ø-9) which is used to identify the machine language subroutine, and xxxx is the address which represents the entry point into the machine language subroutine.

The number assigned to the subroutine (n) must be the same as the number used to reference the subroutine with the USR statement.

The entry address to the subroutine may be a constant (i.e. a hexadecimal or decimal number), or it may be a numeric expression. Note that if the starting address is specified as a decimal number, and this address is greater than 32767, it must be specified as the address minus 65536.

Example

Suppose you have a machine language subroutine that has a starting address of &HFØØØ (61440), and you wish to reference this routine as machine language subroutine number 2. To define this subroutine, one of the following commands may be given:

```
DEFUSR2=&HFØØØ
DEFUSR2=(6144Ø-65536)
DEFUSR2=(-4Ø96)
```

EOF - Determine if "End of File" has been encountered

This command is used to determine if the end of file has been reached when inputting from an open disk file. It is used primarily in conjunction with sequential files, but can also be used with random files. EOF is a function, and will return a Ø (false) if the end of file has not been reached, or a -1 (true) if the end of file has been reached. It can be used with the IF statement, and will determine the outcome of the IF, as it will return either a logical true or a logical false.

The syntax for the EOF command is:

```
EOF(#)
```

where # is the buffer number used to open the file.

Example

Assume that you have created a sequential file named MYDATA, and wish to access the information in it, but you do not know the amount of data in the file. The following program lines will illustrate how to use EOF to determine when the last piece of data has been accessed.

```
1ØØØ OPEN"I",1,"MYDATA"
11ØØ IF EOF(1) THEN PRINT"ALL DATA HAS BEEN ACCESSED":END
   xxxx
   xxxx 'lines used to input and process data
   xxxx
15ØØ GOTO 11ØØ
```

Notice that the EOF command is used prior to inputting any information. This will ensure that you will not try to input from an empty file, or after the end of file has been encountered. Either case would result in an "INPUT PAST END" error.

FIELD - Partition the buffer associated with a random file

The field statement is used to partition the buffer associated with an open random file. This partitioning allows you to break a record up into fields, where each field denotes a particular piece of information in that record. The fielding of a record determines the length of each piece of information in the record, and where this information will physically reside in the record.

The syntax used in the FIELD statement is:

FIELD#,aaa AS variable1,bbb AS variable2,...,nnn AS variableN

is the buffer number used in the associated OPEN statement. It may be a constant, or a numeric expression. The value of this number must be in the range of 1 to the total number of files allocated when entering LBASIC, inclusive, and must correspond to an open file.

aaa, bbb and nnn are numeric constants or expressions denoting the maximum length (in bytes) of the fielded variable. The value of these numeric constants or expressions must be in the range of Ø to 255, inclusive, as the length of a string cannot exceed 255 bytes. If denoted as numeric expressions, these values must be enclosed within parentheses.

variable1, variable2, and variableN are intermediate variables used to retrieve information from and pass information to the buffer. They must be string variables.

When information passes between the computer and the disk, a buffer is used as a temporary storage place for this information. Information is placed in this buffer with the LSET and RSET commands. Where this information is physically placed in the buffer is determined by the FIELD statement.

The field statement will allow you to break up the buffer into various "slots", assigning a variable name to each of these slots. When information is placed into or accessed from the buffer, it is done so by using the variable name which was assigned to each slot in the FIELD statement. The length of each of these slots is also determined by the FIELD statement. The total number of bytes to be fielded in a record must be less than or equal to the number of bytes that a record will contain.

The following example will illustrate how the FIELD statement is used.

Suppose that you wish to deal with a file that will contain records whose lengths will be 1ØØ bytes. In each record, there will be 4 pieces of information (fields). Field 1 will be 2Ø characters long, and will represent the name of a person. Field 2 will be 1Ø characters long, and will represent an account number. Field 3 will be 3Ø characters long, and will represent address information. Field 4 will be 4Ø characters long, and will represent an account description. The following OPEN and FIELD statements will allow you to open such a file and field the buffer accordingly.

```
OPEN"R",1,"MYFILE/DAT",1ØØ
FIELD1,2Ø AS NA$,1Ø AS AC$,3Ø AS AD$,4Ø AS DE$
```

Using the above lines in a program will produce the following results. A file by the name of MYFILE/DAT will be opened, and records in this file will have a length of 1ØØ bytes. A buffer for this file will be set up in memory. The first 2Ø bytes of this buffer will represent name, and will be referenced by the variable NA$. The next 1Ø bytes of this buffer will represent the account number, and will be referenced by the variable AC$. The next 3Ø bytes will represent the address, and will be referenced by the variable AD$. The last 4Ø bytes will represent the description, and will be referenced by the variable DE$.

More than one field statement may correspond to the same buffer. Variable names used in a FIELD statement may only be used to pass information to or retrieve information from the buffer. Using fielded variables for any other purpose will break the link between the variable and the buffer, and the variable will not be connected to the buffer until the original FIELD statement is re-executed. For more

information on passing information to and retrieving information from the disk, see OPEN, GET, PUT, LSET, RSET, MKI$, MKS$, MKD$, CVI, CVS and CVD.

GET - Retrieve a record from a random file

The GET command is used to retrieve information from a random file. The information that is retrieved is stored in the buffer that was used to open the file. The syntax for the GET command is:

```
GET#,r
GET#
```

where # is the buffer number used to open the file, and r is the record number you wish to retrieve. Both # and r may be numeric constants or numeric expressions. If the record number (r) is not specified, the computer will increment the current record number by one, after which it will perform a GET of the current record number. If no current record number has been established, the computer will perform a GET of record number one, and the current record number will be set equal to one.

Example

Suppose you have opened a file and fielded the corresponding buffer. The buffer number used is 3. One of the following GET commands may be used to retrieve the 17th record of the file.

```
GET3,17
N%=2:N1%=16:GETN%+1,N1%+1
```

After executing one of the above statements, record 17 of the file will be contained in the designated buffer, and information dealing with this record may now be accessed by referencing the variables used in the FIELD statement.

INPUT# - Input information from a sequential file.

The INPUT# statement is used to retrieve information from a sequential file. The syntax used with the INPUT# command is:

```
INPUT#n,variable1,...,variableN
```

where n is the buffer number used to open the file, and variable1,...,variableN are the variables used to store the information retrieved.

Sequential files are created by specifying an OPEN"O"/OPEN"E" command, followed by one or more PRINT# commands. After a sequential file has been created, the information in it may be accessed by using the OPEN"I" and INPUT# commands. The INPUT# command can be thought of as performing a function similar to the INPUT command, the exception being that the information is not entered from the keyboard. Rather, it is retrieved from the disk. Like the INPUT command, INPUT# can only be executed from within a program, and cannot be executed from the Basic Ready prompt.

The variable types used in an INPUT# statement must be the same type of variable used when the information was written to the file via the PRINT# command. At least one variable must be specified with the INPUT# command. If multiple variables are specified with the INPUT# command, they must be separated by commas.

After execution of an INPUT# command, the variable(s) specified will be assigned values corresponding to the data retrieved from the disk. If you try to execute an INPUT# command after all of the data has been retrieved from the file, an INPUT PAST END error will be generated.

Example

Suppose a file called MYFILE/SEQ was created using the OPEN"O" and PRINT# commands, and this file contains the following pieces of data:

```
JONES
THOMAS
12
MALE
```

The following commands may be used to access this information:

```
OPEN"I",1,"MYFILE/SEQ"
INPUT#1,LN$,FN$,AG%
INPUT#1,SE$
```

After the execution of the first two commands, the file MYFILE/SEQ would have been opened for sequential input, the variable LN$ would have been assigned the value "JONES", the variable FN$ would have been assigned the value "THOMAS", and the variable AG% would have been assigned the value 12. Note that the last piece of data in the file ("MALE") would not have been accessed by either of the first two commands. However, after the third command (INPUT#1,SE$) has been executed, the variable SE$ would be assigned a value of "MALE".

INPUT# deals with data in a disk file in a special way. For more information on creating sequential files that are accessed by the INPUT# command, refer to OPEN ("O", "E" and "I") and PRINT#.

INSTR - Locate the position of a sub-string within a target string

The INSTR command allows you to search for a specified sub-string within a given target string, and returns the position number in the target string of where the sub-string was found. The syntax for the INSTR command is:

```
INSTR(starting position,target string,sub-string)
```

"starting position" is the point where you wish the search to begin in the target string (e.g. start the search from the third character in the target string). If not specified, starting position will default to 1.

"target string" is the string you wish to search.

"sub-string" is the string you wish to search for within the target string.

The starting position may be either a numeric constant or a numeric expression, and must represent an integer value in the range of 1 to 255, inclusive. The target string and sub-string may be either string constants or string expressions.

INSTR will begin the search of the target string for the sub-string from the starting position specified (if no starting position is specified, INSTR will begin the search from the first character of the target string), and will return a numeric value corresponding to the position in the target string of where the first occurrence of the sub-string is found. If the sub-string is not found in the target string, INSTR will return a Ø. If the sub-string to be searched for is a null string, INSTR will return the starting position of the search, as the null string is a sub-set of any string.

Other occurrences may cause INSTR to return a zero. They are:

If the target string is a null string.

If the starting position is a number greater than the length of the target string.

The following example will illustrate the use of the INSTR command.

Example

Suppose you have the following lines in a program:

```
A$="ROY IS A BOY":B$="OY":C$="ROY":D$="oy":E$="ROYIS"
A%=INSTR(A$,C$)
B%=INSTR(2,A$,B$)
C%=INSTR(3,A$,B$)
D%=INSTR(2,A$,C$)
E%=INSTR(A$,D$)
F%=INSTR(A$,E$)
```

After executing the above lines, the following variables will have been assigned these values:

A%=1 B%=2 C%=11 D%=Ø E%=Ø F%=Ø

Note that the value of E% will be Ø. This is because the sub-string ("oy") is in lower case, and there are no lower case letters in the target string. Also note that the value of F% will be Ø. This is because the string "ROYIS" does not appear in the target string (there is a space between the words ROY and IS in the target string).

KILL - Kill (Remove) a disk file from the directory

The KILL command will allow you to kill a file from a disk directory, making that file inaccessible, and freeing up the space on the diskette that the file consumed. The KILL command functions identically to the LDOS Library command "KILL". The syntax for the KILL command is:

KILL"filespec"

where filespec is any valid LDOS file specification. Filespec may be represented as a string constant or a string expression.

Realize that if the filespec given with the KILL command does not exist, you will get the error message FILE NOT FOUND.

Example

Suppose you wish to remove the file MYFILE/DAT from the diskette currently in drive 1, and free up the space consumed by that file. The following command will perform this function.

```
KILL"MYFILE/DAT:1"
```

Realize that after the kill is performed, you will no longer be able to access any information which was previously stored in the file. Also note that since the filespec is being represented as a string constant, it must be enclosed in quotes.

N O T E

When performing a KILL of a data file, the file in question must NOT be in an OPENed state. The KILLing of an open file may cause certain parts of the diskette in question to be totally inaccessible!

LINEINPUT - Input a line into one variable

The LINEINPUT command is very similar to the INPUT command. It will allow you to input information in from the keyboard to be stored in a variable. The differences between the LINEINPUT command and the INPUT command are as follows:

No question mark will appear when the input is taken.

Only one variable may be assigned a value.

All characters entered before <ENTER> is pressed will be assigned to the variable specified (i.e. commas and quotes may be input from the keyboard, and leading spaces are not ignored).

The syntax for the LINEINPUT command is:

LINEINPUT"prompting message";variable

The prompting message is optional; if used, it must be included within quotes, and must be separated from the variable by a semicolon. If the prompting message is not used, a semicolon cannot be used. As is the case with the INPUT command, LINEINPUT cannot be issued from the Basic Ready prompt.

Example

Suppose that you wish to input a person's name and title into a program, and you wish to separate the name from the title by use of a comma. Using the LINEINPUT command, you may now input the comma from the keyboard to be taken as part of the input. The following LINEINPUT command may be used to accomplish this.

```
LINEINPUT"Enter Name, Title";A$
```

When the computer executes the above command, you will see the prompt "Enter Name, Title" appear, and there will be no question mark after the prompt. The computer will now be awaiting your input. If you answer this prompt by typing in the response "JOHN JONES, PRESIDENT" , A$ will be assigned all characters that you have typed in, prior to pressing the <ENTER> key.

LINEINPUT# - Input a line from a disk file into a variable.

The LINEINPUT# command will allow you to input a line from a disk file into a variable. It functions similarily to the LINEINPUT command, with the exception being that the input is taken from the disk, rather than the keyboard.

The syntax for the LINEINPUT# command is:

LINEINPUT#b,variable

where b is the buffer number used when the file was opened, and variable is a string variable used to stored the retrieved information.

LINEINPUT# differs from INPUT# in several ways. As noted in the PRINT# command,

INPUT# will read information in from the disk until it encounters a comma, a carriage return, the end of file, or the 255th character when dealing with string information. When using LINEINPUT#, commas will not be taken as delimeters of the string, and hence may be included in the input from disk. The LINEINPUT# of a variable will terminate when a carriage return, the end of file, or the 255th character of a string is encountered. As is the case with INPUT#, LINEINPUT# cannot be executed from the Basic Ready prompt.

Example

Assume the following data is stored in a disk file, and the file has been opened using buffer number 1 (<cr> represents a carriage return).

JOHN JONES , PRESIDENT , ABC CORPORATION<cr>

If the command LINEINPUT#1,A$ is used to input the above information, A$ would be assigned the value:

JOHN JONES , PRESIDENT , ABC CORPORATION

Realize that all of the characters (including the commas) would be read in and assigned to A$.

If the command INPUT#1,A$ were used instead of LINEINPUT#, the value of A$ would be "JOHN JONES", as INPUT# will read information until it encounters a comma. For more information on how data is stored on the disk in a sequential file, see PRINT#.

LOAD - Load a BASIC program into memory

The LOAD command allows you to retrieve a BASIC program that has been stored on disk, and place it in the computer's memory so that it may be executed or edited. The syntax for the LOAD command is:

LOAD"filespec",R

filespec may be represented as a string constant or a string expression. If represented as a string constant, filespec must appear within quotes.

The R parameter is optional; if used, the program to be loaded will be executed after it is loaded, and all open files will remain open. Performing a LOAD without the R option will cause any open files to be closed.

Loading a program will always overwrite any program in memory with the program to be loaded. Basic programs cannot be concatenated with the LOAD command (see MERGE for program concatenation). The LOAD command may be given from the BASIC Ready prompt, or can be issued from within a program. If issued from within a program, the program issuing the LOAD command will be overwritten by the program to be loaded, and execution will be terminated.

Example

LOAD"MYPROG/BAS"

After execution of this command, any program which was in memory will be replaced by the program MYPROG/BAS.

LOC - Get current record number

The LOC command is used primarily with random files, and will return a value corresponding to the current record number of the given file. The syntax for the LOC command is:

```
LOC(#)
```

where # represents the buffer number used to open the file in question. # may be either a numeric constant or a numeric expression, and must correspond to an open file.

When a file is in an open state, the computer maintains some control information dealing with that file. One piece of information that is available to the user is the record number currently being dealt with. The LOC command will return the current record number that the computer has accessed. If no record in an open file has been accessed, LOC will return the value Ø.

Example

Suppose you have opened a file using buffer number 2, and have fielded the buffer accordingly. If the following commands are executed:

```
GET2,17
A%=LOC(2)
```

the variable A% will be assigned the value 17.

LOF - Get last record number

The LOF command is used primarily with random files, and will return a value corresponding to the last record number of the given file. The syntax for the LOF command is:

```
LOF(#)
```

where # is the buffer number used to open the file in question. # may be either a numeric constant or a numeric expression.

The LOF command provides a means of determining the number of records that have been written to a random file. Note that if a file has been pre-created using the CREATE library command, LOF will return a number corresponding to the highest record number actually written to, not the number of records that have been pre-created.

Example

Suppose you have a file named MYFILE/DAT, and the highest record number written to is record number 43. If the file has been opened using buffer number 3, and has been fielded accordingly, the following command will result in the variable A% being set equal to 43.

```
A%=LOF(3)
```

LSET - Place data into the buffer assigned to an open file

The LSET command will allow you to place information in the buffer associated with a random file, prior to writing the information in the buffer out to disk. The syntax for the LSET command is:

LSET fielded string variable=value

fielded string variable is the variable used in the FIELD statement that points to the location in the buffer where the data is to be placed.

value is the value that you wish to place in the buffer, and must be a string constant or a string expression.

When dealing with random files, the FIELD statement is used to set up and partition the buffer associated with the file. String variables are used in the FIELD statement to designate various slots for information storage and retrieval in the buffer. The LSET command allows you to place information in these slots in the buffer, prior to writing the information out to disk.

The LSET command will left-justify the information in the buffer. That is to say, if the length of the string to be placed in the buffer is less than the length allocated for the particular slot, trailing spaces will be inserted at the end of the string in the buffer. This will make the string in the buffer the same length as specified in the FIELD statement.

If the length of the string to be LSET into the buffer is greater than the fielded length, the left most part of the string will be placed in the buffer, and any characters to the right of the total allocated space will be truncated. See RSET to right-justify a string into the buffer.

The commands MKI$, MKS$, and MKD$ are also used in conjunction with the LSET statement. Because the buffer is fielded in terms of string variables, only string values may be LSET into the buffer. The MKI$, MKS$, and MKD$ commands are used to change numeric data into compressed string representations of numbers, and will create strings of 2 bytes, 4 bytes, and 8 bytes respectively. When performing an LSET using the MKI$, MKS$ or MKD$ commands, the length of the fielded variable to be LSET must be at least 2 bytes, 4 bytes, or 8 bytes, respectively. For more information on commands that are used with LSET, refer to the commands MKI$, MKD$, MKS$, and FIELD, and the example below.

Example

Suppose you have a file called MYFILE/DAT, and have opened the file to have record lengths of 45 bytes. In addition, assume that the buffer corresponding to the file (buffer number 1) has been fielded with the following statement, and the variables listed below have been assigned the given values:

```
FIELD 1, 31 AS NA$, 2 AS A2$, 4 AS A4$, 8 AS A8$
NM$="JOHN JONES, PRESIDENT":A2%=92:A4!=23.79:A8#=123498.63
```

The LSET statements you may use to place these values into the buffer may look like this:

```
LSET NA$=NM$
LSET A2$=MKI$(A2%)
LSET A4$=MKS$(A4!)
LSET A8$=MKD$(A8#)
```

The values of the variables A2%, A4!, and A8# will be stored in the slots in the buffer pointed to by the variables A2$, A4$, and A8$, respectively. They will be stored as compressed string representations of the values the variables have been assigned.

The value of NM$ will be stored in the slot in the buffer pointed to by the variable NA$. Realize that since the length of NM$ is 21 characters, the last 1Ø characters of the slot in the buffer pointed to by NA$ will be spaces (CHR$(32)). If the length of NM$ would have been longer than 31 characters, the left-most 31 characters would have been placed in the buffer, and the remaining characters would have been truncated (in essence, ignored).

The LSET command will typically be used prior to performing a data write to a random file. For more information on performing a data write to a random file, see OPEN, FIELD and PUT.

MERGE - Merge a program from disk with current program in memory

The MERGE command will allow you to merge a program file stored on disk (in ASCII) with a program resident in memory, with the resultant program being stored in memory. The syntax for the MERGE command is:

```
MERGE"filespec"
```

where filespec represents a BASIC program stored on disk in ASCII (For more information on storing BASIC programs on disk in ASCII, see SAVE). Filespec may be represented as a string constant or a string expression. If represented as a string constant, filespec must be contained within quotes.

The MERGE command will read in (line by line) the program from disk, and merge these lines in with the existing program. Any line number in the program to be merged that does not exist in the program in memory will be added to the program in memory. Any line number in the program to be merged that does exist in the program in memory will overwrite the line in memory.

The MERGE command provides for an easy way to merge subroutines which are common to several different programs into these programs without always having to type in the subroutine. The following example will illustrate how the MERGE command functions.

Example

Suppose you have a program which is resident in memory, and this program consists of the following statements:

```
1Ø FOR L=1TO1ØØ
2Ø PRINT L
3Ø NEXT L
```

Assume also that you have a program named MYPROG/ASC stored in ASCII on disk, and this program consists of the following statements:

```
5 DEFINT A-Z
1Ø FORL=1TO5ØØ
25 'THIS LINE HAS BEEN MERGED IN
4Ø GOTO 1Ø
```

If you wish to merge the program MYPROG/ASC with the program currently in memory, you may do so by issuing the following command:

```
MERGE"MYPROG/ASC"
```

By giving the above command, the program resident in memory will be changed to the following:

```
5 DEFINT A-Z
1Ø FORL=1TO5ØØ
2Ø PRINT L
25 'THIS LINE HAS BEEN MERGED IN
3Ø NEXT L
4Ø GOTO 1Ø
```

Before merging in a program, you should make sure that there is enough free memory for the program to be merged in. Also, note that the MERGE command is usually issued from the BASIC Ready prompt. However, if incorporated within a program, the MERGE will be done, but execution of the program will cease.

MID$= - Replace a portion of a string

The MID$= command will allow you to perform a character for character replacement of any characters within a string. MID$= is the only BASIC function which may be used on the left-hand side of the equal sign. The syntax for the MID$= command is:

MID$(string value,starting position,length)=replacement string

"string value" may be either a string constant or a string expression, and represents the target string for the replacement.

"starting position" is the place in the string value where the replacement is to start. This may be either a numeric constant or a numeric expression.

"length" is the number of characters to be changed. This may be either a numeric constant or a numeric expression. The length parameter is optional; if omitted, the number of characters to be replaced will be determined by the replacement string.

"replacement string" is the string you wish to replace the specified portion of the current string with. This may be either a string constant or a string expression.

The MID$= command will perform a character for character replacement on a given string with the replacement string. It may not be used to lengthen or shorten an existing string. If the length parameter is not specified, the number of characters involved in the replacement will be determined by the length of the replacement string. If the length parameter differs from the length of the replacement string, one of several things may happen.

If the length parameter is less than the length of the replacement string, the length parameter will take precedence, and only the left-most number of characters as specified in the length parameter will be changed.

If the length parameter is greater than the length of the replacement string, the replacement string will take precedence, and only those characters specified in the replacement string will be changed.

If the parameters specified in the MID$= command would cause the original string to become larger, only those characters up to the end of the original string would be changed, and the length of the string would remain unchanged. In essence, the extra characters at the end of the replacement string would be ignored.

The following example should clarify how the MID$= command functions.

Example

Suppose you have a string variable A$ set equal to the value "THIS IS IT". The following MID$= commands would have these affects on A$.

```
MID$(A$,3,2)="AT"       ---> A$ would change to "THAT IS IT"
MID$(A$,6,2)="WAS"      ---> A$ would change to "THIS WA IT"
MID$(A$,3,8)="AT'S IT"  ---> A$ would change to "THAT'S ITT"
MID$(A$,9,3)="ALL"      ---> A$ would change to "THIS IS AL"
```

MKD$ - Change a numeric value into an 8 byte compressed string

The MKD$ command (MaKe Double precision string) will change a numeric value into an 8 byte string which is a compressed representation of the value. This command is used primarily with the LSET and RSET commands to place numeric data into the buffer associated with an open random file. The syntax for the MKD$ command is:

```
MKD$(numeric value)
```

where numeric value may be either a numeric constant or a numeric expression. Numeric value can represent any value which may be assigned to a double precision variable. Up to 16 significant digits will be maintained. To convert an 8 byte compressed string representation of a number back to a numeric value, use the CVD command.

Since only strings may be stored in the buffer associated with an open random file, there exists a need to change numeric data into a string form. MKD$ provides a way to change numeric data into a string. The string formed by MKD$ will always be 8 bytes in length, regardless of the actual value to be converted. The resultant string value obtained when performing an MKD$ command will be the compressed form of a number, contained in an 8 byte string. After a numeric value has been changed into an 8 byte compressed string, it may then be placed into a buffer via the LSET and RSET commands. (Note: This is not the same as the STR$ command, as STR$ produces an ASCII string, not a compressed string representation of a number.)

Example

Suppose you have opened and fielded a random file, and wish to place a double precision value into the buffer. The fielded variable you are dealing with is A8$, and the value you wish to place in the part of the buffer pointed to by A8$ is contained in the variable A8#. The following command will cause an 8 byte compressed string representation of the value stored in A8# to be written to the portion of the buffer pointed to by A8$.

```
LSET A8$=MKD$(A8#)
```

Note that the fielded length of the variable A8$ must be at least 8 bytes, and in most cases will be exactly 8 bytes.

MKI$ - Change a numeric value into a 2 byte compressed string

The MKI$ command (MaKe Integer string) will change a numeric value into a 2 byte string which is a compressed representation of the value. This command is used primarily with the LSET and RSET commands to place numeric data into the buffer associated with an open random file. The syntax for the MKI$ command is:

```
MKI$(numeric value)
```

where numeric value may be either a numeric constant or a numeric expression. Numeric value must be within the range of -32768 to +32767, inclusive. If numeric value is not an integer, any numbers to the right of the decimal point will be truncated. To convert a 2 byte compressed string representation of a number back to a numeric value, use the CVI command.

Since only strings may be stored in the buffer associated with an open random file, there exists a need to change numeric data into a string form. MKI$ provides a way to change numeric data into a string. The string formed by MKI$ will always be 2 bytes in length, regardless of the actual value to be converted.

The resultant string value obtained when performing an MKI$ command will be the compressed form of an integer, contained in a 2 byte string. After a numeric value has been changed into a 2 byte compressed string, it may then be placed into a buffer via the LSET and RSET commands. (Note: This is not the same as the STR$ command, as the STR$ command produces an ASCII string, not a compressed string representation of a number.)

Example

Suppose you have opened and fielded a random file, and wish to place an integer value into the buffer. The fielded variable you are dealing with is A2$, and the value you wish to place in the part of the buffer pointed to by A2$ is contained in the variable A2%. The following command will cause a 2 byte compressed string representation of the value stored in A2% to be written to the portion of the buffer pointed to by A2$.

```
LSET A2$=MKI$(A2%)
```

Note that the fielded length of the variable A2$ must be at least 2 bytes, and in most cases will be exactly 2 bytes.

MKS$ - Change a numeric value into a 4 byte compressed string

The MKS$ command (MaKe Single precision string) will change a numeric value into a 4 byte string which is a compressed representation of the value. This command is used primarily with the LSET and RSET commands to place numeric data into the buffer associated with an open random file. The syntax for the MKS$ command is:

```
MKS$(numeric value)
```

where numeric value may be either a numeric constant or a numeric expression. Numeric value can represent any value which may be assigned to a single precision variable. Up to 6 significant digits will be maintained. To convert a 4 byte compressed representation of a number back to a numeric value, use the CVS command.

Since only strings may be stored in the buffer associated with an open random file, there exists a need to change numeric data into a string form. MKS$ provides a way to change numeric data into a string. The string formed by MKS$ will always be 4 bytes in length, regardless of the actual value to be converted. The resultant string value obtained when performing an MKS$ command will be the compressed form of a number, contained in a 4 byte string. After a numeric value has been changed into a 4 byte compressed string, it may then be placed into a buffer via the LSET and RSET commands. (Note: This is not the same as the STR$ command, as STR$ produces an ASCII string, not a compressed string representation of a number.)

Example

Suppose you have opened and fielded a random file, and wish to place a single precision value into the buffer. The fielded variable you are dealing with is A4$, and the value you wish to place in the part of the buffer pointed to by A4$ is contained in the variable A4!. The following command will cause a 4 byte compressed string representation of the value stored in A4! to be written to the portion of the buffer pointed to by A4$.

```
LSET A4$=MKS$(A4!)
```

Note that the fielded length of the variable A4$ must be at least 4 bytes, and in most cases will be exactly 4 bytes.

OPEN - Open a random/sequential disk file

The OPEN command allows you to open random/sequential data files in order that input/output may occur between the computer and the given file. The general syntax for the OPEN command is:

```
OPEN"file type",buffer number,"filespec",record length
```

"file type" is the type of file you wish to deal with (random or sequential). It may be represented as a string constant enclosed within quotes, or as a string expression.

"buffer number" is the number of the buffer you wish to use to perform the input/output from/to the disk. This may be either a numeric constant or a numeric expression, and must be an integer value within the range of 1 to the total number of active files declared when entering LBASIC, inclusive.

"filespec" is the name, extension, password and drive number of the file to be opened. Filespec must conform to all of the rules governing LDOS filespecs. It may be represented as a string constant or a string expression.

"record length" pertains to random files only, and will determine the record length used when accessing the file. It must be an integer value, and may be represented as either a numeric constant, or a numeric expression whose value must be in the range of Ø to 255, inclusive. This parameter is optional; if not used, record length will default to 256. If record length is specified as Ø, it will be assumed to be 256. If the parameter BLK=OFF is specified when entering LBASIC, record length cannot be specified in an OPEN statement, and will default to 256.

In order to write information to and retrieve information from a disk file, the file must be opened using the OPEN command. The OPEN command establishes the capability of reading from and writing to a disk file by creating a file control block (FCB). This FCB contains information needed by the computer, so that the computer may interact with the disk file. In addition, the OPEN command establishes a buffer which is used by the computer as a temporary storage place for information that will pass between the computer and the disk file.

There are two types of files available to you when storing information in a disk file; sequential files and random files.

Sequential files are file types that allow for accessing data in a specified sequence. That is to say, if you wish to retrieve the 1Øth piece of information in a file, you must read in the nine data items preceding the item in question before it may be accessed.

Random files are file types that allow you to directly access any piece of information in a file, regardless of the physical location of the data within the file.

It is beyond the scope of this manual to discuss the techniques involved in creating and accessing information in random and sequential files. What will be provided for you here is the syntax needed to open all types of random and sequential files. For the novice, it is strongly recommended that supplementary material be obtained for the purpose of learning filing techniques.

IMPORTANT NOTE

It is strongly advised that no data file be in an open state at any given time using more than one buffer. LBASIC will allow you to open the same file at the same time using more than one buffer; however, this practice may lead to the destruction of data files on the diskette in question!!

Opening sequential files.

There are two basic modes available for use when dealing with sequential files; the input mode, and the output mode. The following list shows all of the different OPEN commands that may be issued when dealing with sequential files.

```
OPEN"I"    --> Open an existing sequential file for input

OPEN"O"    --> Open a sequential file for output
OPEN"OO"   --> Open an existing (old) sequential file for output
OPEN"ON"   --> Open a non-existing (new) sequential file for output

OPEN"E"    --> Open for output and extend a sequential file
OPEN"EO"   --> Open for output and extend an existing sequential file
OPEN"EN"   --> Open for output and extend a non-existing sequential file
```

The input mode of sequential files allows you to input information from an existing file. No output to the file may be done if it has been opened for input. The file to be opened for input must exist, or the OPEN"I" command will return a FILE NOT FOUND error. Once the file has been opened, information may be retrieved from it using the INPUT# and LINEINPUT# commands.

The output mode of sequential files allows you to output information to the file. No input from the file may be done if it has been opened for output. Once the file has been opened, information may be written out to it using the PRINT# command. There are six types of output modes available for use with sequential files.

The OPEN"O" output mode functions in the following manner. If the file opened does not exist, it will be created, and information will be written to the file starting at the first byte of the file. If the file opened does exist, any information previously stored in the file will be lost, as the new information to be placed in the file will be written over the existing information, starting at the first byte of the file.

The OPEN"OO" output mode functions in the following manner. If the file opened does not exist, a FILE NOT FOUND error will be generated, and the file will not be created. If the file opened does exist, OPEN"OO" will function identically to OPEN"O" in the case where the file already exists.

The OPEN"ON" output mode functions in the following manner. If the file already exists, you will not be allowed to open the file, and the error FILE ALREADY EXISTS will be generated. The existing file will not be altered in any way. If the file does not exist, it will be created, and information will be written to the file starting with the first byte of the file.

The OPEN"E" output mode functions in the following manner. If the file does not exist, OPEN"E" will function identically to OPEN"O". If the file already exists, the file will be opened, and any information that will be written to the file will be appended to the end of the existing information. The file will be extended to include both the old and the new information.

The OPEN"EO" output mode functions in the following manner. If the file does not exist, a FILE NOT FOUND error will be generated, and no file will be created. If the file already exists, the file will be opened, and any information that will be written to the file will be appended to the end of the existing information. The file will be extended to include both the old and the new information.

The OPEN"EN" mode functions identically to the OPEN"ON" output mode.

Example - Opening sequential files

Suppose that you wished to open a sequential file named MYDATA/SEQ, using buffer number 1. The statement used to open the file for input would be as follows:

```
OPEN"I",1,"MYDATA/SEQ"
```

If you wished to open the same file for output using buffer number two, one of the following commands could be used, depending on whether or not you request that the file be new or old, and whether or not you wish to extend the file:

```
OPEN"O",2,"MYDATA/SEQ"
OPEN"OO",2,"MYDATA/SEQ"
OPEN"ON",2,"MYDATA/SEQ"
OPEN"E",2,"MYDATA/SEQ"
OPEN"EO",2,"MYDATA/SEQ"
OPEN"EN",2,"MYDATA/SEQ"
```

Opening random files

Unlike sequential files, when dealing with a random file, you have the capability of reading from and writing to the file using only one OPEN command. The statements PUT and GET differentiate between writing to the file and reading from the file, respectively. There are three different types of OPEN statements that may be executed when opening a random file. They are:

```
OPEN"R"   --> Open a random file whether or not it exists.
OPEN"RN"  --> Open a random file only if it does not exist.
OPEN"RO"  --> Open a random file only if it already exists.
```

The OPEN"R" mode functions in the following manner. The file specified will be opened whether it exists or not, and will be created if it does not exist. After the file has been opened, the buffer used in the OPEN statement may be fielded using the FIELD statement, and records may then be retrieved from or placed into the file via the PUT and GET statements.

The OPEN"RN" mode functions in the following manner. If the file already exists, you will not be allowed to open it. The file will remain untouched, and the error FILE ALREADY EXISTS will occur. If the file does not exist, it will be created, and the OPEN"RN" command will function in the same manner as the OPEN"R" command.

The OPEN"RO" mode functions in the following manner. If the file does not exist, no file will be created, and the error FILE NOT FOUND will occur. If the file does exist, OPEN"RO" will function in the same manner as OPEN"R".

Example - Opening random files

Suppose you wish to open a random file named MYDATA/RND, using buffer number 3, with record lengths of 52 bytes. One of the following OPEN commands may be used to open the file, depending on the specific requirements needed by the user (i.e. open the file only if it does or does not exist).

```
OPEN"R",3,"MYDATA/RND",52
OPEN"RN",3,"MYDATA/RND",52
OPEN"RO",3,"MYDATA/RND",52
```

For more information on using both random and sequential files, refer to FIELD, GET, PUT, LSET, RSET, INPUT#, LINEINPUT#, and PRINT#.

PRINT# - Output data to a sequential file

The PRINT# command allows you to output data to a sequential file. The syntax is:

PRINT#buffer number,list of constants and/or expressions

buffer number is the buffer used to open the file. It may be expressed as a numeric constant or a numeric expression.

list of constants and/or expressions contains the data that you wish to output to the file. Numeric constants, numeric expressions, string constants and string expressions may all be contained within this list. If more than one value is to be output to the file using a single PRINT# statement, these values must be separated by some type of delimiter. The uses of delimiters in a PRINT# command will be explained throughout this section.

The PRINT# command is used in conjunction with any type of OPEN"O" or OPEN"E" command. After a file has been opened, data may be output to the file via the PRINT# command. Once a file has been created using the OPEN"O"/OPEN"E" and PRINT# commands and then closed, the information in the file may be accessed using the OPEN"I" and INPUT#/LINEINPUT# commands.

In most cases, data written to a sequential file is stored in ASCII format. For numeric data, a sign byte will always precede the numeric information. If the value is positive, the sign byte will be represented by a space. A trailing space will always follow the ASCII representation of the value. Keeping the above in mind, the minimum amount of bytes required to store a numeric value in a sequential file is 3 (the sign byte, a digit, and the trailing space).

For string data, all characters included in the string value will be written to the file, and no preceding or trailing characters will be written to the file. Special considerations do need to be taken into account when writing string values to a sequential file, as there are some peculiarities involved with the INPUT# command when trying to access string information stored in a sequential file. These special cases will be pointed out throughout this section.

The PRINT# command resembles the PRINT command in many ways with respect to how information is physically written to the file. Some of the punctuation used in the PRINT# command will cause data to be written to the file in much the same way that this punctuation causes data to be printed to the screen using the PRINT command.

Punctuation is very important when using the PRINT# command. The following will describe the punctuation which is allowed with the PRINT# command, and the effects of using different punctuation.

Use of punctuation with the PRINT# command.

Different types of punctuation used to separate values to be output in a PRINT# statement will cause the data to be physically written to the file in different ways. The following list shows the punctuation required to separate values contained in a PRINT# statement.

, - comma
; - semicolon
"," - explicit comma

When separating output data contained in a PRINT# statement, you may use either a comma or a semicolon. A semicolon will cause the next piece of information to be written directly after the preceding data. A comma will cause the next piece of information to be written at the next available "tab" position in the file. Tab positions will be denoted by 16 byte blocks, starting from the last occurrence of a carriage return (ØDH) in the file.

In some cases, the explicit comma is used after string information has been written to the disk, to demark the end of the string value from the beginning of the next piece of information to be written out.
The following examples will illustrate the methods used to write data to a sequential file, as well as the occurrences that will result when this data is to be retrieved.

Example 1 - Writing numeric data to a sequential file.

Suppose you wish to write two numeric values out to a sequential file, using one PRINT# command. The file you wish to write these values out to is named DATA1/SEQ, and has been opened using buffer number 2. The variables you wish to write out to the file are A%, which has been assigned a value of 362, and B!, which has been assigned a value of -2618.7. The following PRINT# command may be used to write these values out to the file:

```
PRINT#2,A%;B!
```

The above statement will cause the values 362 and -2618.7 to be written to the file in ASCII format. The image produced on the disk by this PRINT# statement is shown below. (Note that throughout the rest of this section, the image produced by the example PRINT# statements will always follow the PRINT# statement. The image shown will be similar to the LDOS LIST (H) library command; each ASCII character will be displayed with its corresponding hex value shown below the character.)

```
     3  6  2     -  2  6  1  8  .  7
2Ø 33 36 32 2Ø 2D 32 36 31 38 2E 37 2Ø ØD
```

Note the sign byte preceding each value, and the trailing space following each value. Also note that the last byte written to the file is a carriage return (ØDH). A carriage return will always be written to the file after the last item listed in a PRINT# statement.

Realize that a semicolon was used to separate the variables A% and B! in the PRINT# command. A comma could have been used instead; however, the image of the data on the disk would have changed to the following if a comma would have been used instead of a semicolon.

```
     3  6  2
2Ø 33 36 32 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø

 -  2  6  1  8  .  7
2D 32 36 31 38 2E 37 2Ø ØD
```

Notice the series of spaces following the number 362. These will be written to the disk as a result of a comma being used to separate the variables A% and B!. As was noted earlier, when using a comma to separate variables in a PRINT# statement, the value following the comma will be written to the next tab

position (the beginning byte of the next block of 16 bytes). As depicted in the above displays, much disk space will be wasted in writing to sequential files if the values in a PRINT# statement are separated by commas instead of semicolons.

Example 2 - Writing string data to a sequential file.

Suppose you wish to write 3 string values out to a sequential file, using one PRINT# command. The file is named DATA2/SEQ, and has been opened using buffer number 1. The variables you wish to write out to the file are A$ (which has been assigned the value "AMBER"), B$ (which has been assigned the value "BROWN"), and the string constant "GRAY". The following PRINT# command may be used to write these values out to the file:

```
PRINT#1,A$;",";B$;",";"GRAY"
```

The above statement will cause the values "AMBER", "BROWN" and "GRAY" to be written to the file. The image produced on the disk by this PRINT# statement is shown below.

```
 A  M  B  E  R  ,  B  R  O  W  N  ,  G  R  A  Y
41 4D 42 45 52 2C 42 52 4F 57 4E 2C 47 52 41 59 ØD
```

There are many things to be noted in this example. The most prominent of these is the use of the explicit comma (","). You will note from the above display that along with the string values, commas were also written out to the file (since they were enclosed within quotes as part of the list of values to be written out). In most applications dealing with writing strings out to sequential files, you will need to incorporate the explicit comma within the list of values to be printed out by the PRINT#. The reason behind this stems from the way INPUT# deals with retrieving information from a sequential file.

Before continuing with more examples on the use of PRINT#, a brief discussion of using INPUT# with files created by PRINT# is in order.

How INPUT# ties together with PRINT#

As shown throughout this section, the punctuation used in the PRINT# command is very important, and determines the manner in which INPUT# will access this information. INPUT# deals with retrieving numeric data in a different fashion than it does with string data.

When INPUT# requests the input of a numeric variable, it will begin reading from the last accessed byte in the file. Any leading spaces that are encountered will be ignored. Once INPUT# finds a non-space character, it will read until it encounters either a space or a delimiter, and the value assigned to the variable will be determined by performing a VAL function on the characters read in. This is to say that any characters may be input into a numeric variable, and the inputting of string values into a numeric variable will not cause a TYPE MISMATCH error.

When INPUT# requests the input of a string variable, it will begin reading from the last accessed byte in the file, and proceed until it finds a non-space character. Once it finds a non-space character, it will read until it encounters a delimiter, and the value assigned to the variable will be all characters read in from the first non-space character to the delimiter. Note from the above description that any "leading" spaces which are present in the data file for the data element in question will be ignored by INPUT#, and the value assigned to the string will never have leading spaces.

In all cases, when INPUT# requests an input of a variable, the input will be terminated when a delimiter character is read in. For numeric inputs, delimiters can be represented by either a space, a comma, or a carriage return (ØDH). In most cases, a comma should not be used as the delimiter for a numeric input.

For string inputs, a delimiter can be represented by either a comma or a carriage return. Realize that for any input of a variable, if the number of characters read in will exceed 255, the input of the variable will terminate after the 255th character has been accessed.

One point to note is that in most cases, two delimiter characters should not appear together in a sequential file. This occurrence will cause unpredictable results when trying to input information from the file.

From the above paragraphs, it can be seen that in any one physical PRINT# statement, if values are to be written out following a string value, they must be separated from the string value by use of the explicit comma. The general format which is recommended to perform such a data write is as follows:

```
PRINT#b,...;string value;",";next value;...
```

Example 3 - Writing numeric and string data to a file.

Suppose you wish to write several string and numeric values out to a sequential file using the same PRINT# statement. The file you wish to write these values out to is named DATA3/SEQ, and has been opened using buffer number 2. The string values you wish to write out are contained in the variables A$ (which has been assigned the value "ANN"), B$ (which has been assigned the value "BETTY") and C$ (which has been assigned the value "CAROL"). The numeric values you wish to write out are contained in the variables A% (which has been assigned a value of 2Ø), B% (which has been assigned a value of 32), and C% (which has been assigned a value of 23). The following will show a PRINT# statement which may be used to write these values out to the file, and the associated image that will be written to the disk as a result of performing the PRINT#.

```
PRINT#2,A%;A$;",";B%;B$;",";C%;C$
```

```
   2  Ø    A  N  N  ,    3  2    B  E  T  T
2Ø 32 3Ø 2Ø 41 4E 4E 2C 2Ø 33 32 2Ø 42 45 54 54

 Y  ,    2  3    C  A  R  O  L
59 2C 2Ø 32 33 2Ø 43 41 52 4F 4C ØD
```

Please note from the above example that no explicit comma needs to follow numeric data. Also note that since C$ is the last variable to be written out in this PRINT# command, no explicit comma is needed after it, as a carriage return will always be written out to the file after the last variable in a PRINT# command. This carriage return will serve as the delimiter for subsequent PRINT# commands.

This concludes our discussion of the PRINT# command. It is recommended that test files be created by the user in order to explore the results of various PRINT# statements. After sequential files have been created, they may be examined by use the the LDOS LIST (H) command. For further information, see OPEN, INPUT#, and LINEINPUT#.

PRINT# USING - Output data to a sequential file using a specified format

The PRINT# USING command will allow you to output data to a sequential file using a specified format. The syntax for the PRINT# USING command is:

PRINT#buffer number,USING format string;list of values

buffer number is the buffer used to open the file. It may be expressed as a numeric constant or a numeric expression.

format string is the format you wish to use to write the list of values out to the file. It may be represented as either a string constant or a string expression.

list of values is the same as list of constants and/or expressions as defined in the PRINT# command.

The PRINT# USING command will allow you to output data to a sequential file in the format specified by the format string. Any format string which is allowable in the PRINT USING command will also be allowable in the PRINT# USING command, and will function in an identical manner. For more information on allowable format strings, refer to PRINT USING in the ROM Basic manual. (For more information on the specifics involved in writing information out to a sequential file, see PRINT#.)

Example

Suppose you wish to write three numeric values out to a sequential file. The name of the file is DATA/SEQ, and it has been opened using buffer number 1. The values you wish to write out are contained in the variables A% (which has been assigned a value of 25), B! (which has been assigned a value of 13.73), and C% (which has been assigned a value of -17). The format string you wish to use has been assigned to the variable A$, and has the value:

```
###    ####.###  ####
```

The following will show a PRINT# USING command that may be used to write out the above values, and the disk image created by the PRINT# USING command.

```
PRINT#1,USINGA$;A%,B!,C%

   2  5                  1  3  .  7  3  Ø        -  1  7
2Ø 32 35 2Ø 2Ø 2Ø 2Ø 2Ø 2Ø 31 33 2E 37 33 3Ø 2Ø 2Ø 2D 31 37 ØD
```

Note from the above example that the image created on disk conforms to the format string specified. Unlike the PRINT# command, the use of delimiters to separate the values to be printed out is arbitrary. That is to say, there is no difference in using a comma as a delimiter as opposed to a semicolon.

PUT - Write a record out to a random file

The PUT command is used to write information out to a random file. The information that is to be written out to the file must have been placed into the buffer that was used to open the file prior to being written out to the file. The syntax for the PUT command is:

```
PUT#,r
PUT#
```

where # is the buffer number used to open the file, and r is the record number you wish to write. Both # and r may be numeric constants or numeric expressions.

If the record number (r) is not specified, the computer will first increment the current record number by one, after which it will perform a PUT of the current record number. If no current record number has been established, the computer will perform a PUT of record number one, and the current record number will be set equal to one.

Example

Suppose you wish to output data to a random file. The file you wish to perform the output to has the name FILE/RND, and has been fielded using buffer number 2. The record you wish to write out to the file is record number 23. Assume also that all of the values you wish to write out to the file have been placed into the buffer using the proper LSET and RSET commands. One of the following PUT commands may be used to write the information to the 23rd record of the file.

```
PUT2,23
N%=1:N1%=3Ø:PUTN%+1,N1%-7
```

After executing one of the above statements, the information stored in the buffer associated with the file FILE/RND will be written out to the disk, and will be placed in the file as representing the 23rd record in the file. Once this information has been placed into the file, it may be retrieved using the GET command.

For more information on using PUT, see OPEN, FIELD, LSET and RSET.

RESTORE nnnn - Reset data pointer

This command is similar to the regular RESTORE command, except that a line number may be specified. The data pointer will be reset to the specified line, and any subsequent READ statements will start from that line. This command must be the first statement in a program line.

RSET - Place data into the buffer assigned to an open file

The RSET command will allow you to place information into the buffer associated with a disk file, prior to writing this information out to the disk. It is used primarily in conjunction with random files. The syntax for the RSET command is:

```
RSET fielded string variable=value
```

fielded string variable is the variable used in the field statement that points to the location in the buffer where the data is to be placed.

value is the value that you wish to place in the buffer, and must be a string constant or string expression.

The RSET command functions identically to the LSET command, with the following exception. Rather than the information being placed into the buffer left-justified, RSET will place the information into the buffer right justified. If the length of the string to be placed into the buffer is less than the fielded length of the particular slot of the buffer, spaces will be inserted in front of the string in the buffer to make the string in the buffer the same length as specified in the field statement.

If the length of the string to be RSET into the buffer is greater than the fielded length, the right most part of the string will be placed in the buffer, and any characters to the left of the total allocated space will be truncated.

For more information on how to utilize the RSET command and the functions it performs, refer to the LSET command.

RUN - Load a Basic program from disk and execute it

The RUN command will allow you to load an LBASIC program stored on disk into the computer's memory, and immediately begin execution of that program. The syntax for the RUN command is:

RUN"filespec",file/variable parameter,line number

filespec is the name of the program that you wish to be loaded and executed, and may be represented by any valid LDOS filespec. filespec may be either a string constant or a string expression. If filespec is not included, the program currently in memory will be executed.

file/variable parameter is an optional parameter, and is used primarily when LBASIC programs are to be "chained" together. One of two different parameters are available. If the parameter R is used, any files which are currently open will remain open when the new program is loaded and executed. If the parameter V is used, all open files will remain open, and all variable assignments will be maintained. This parameter, if used, must be represented as a letter (R or V), and cannot appear within quote marks, or be represented by a string expression.

line number is an optional parameter, and is used to specify a line number in the program where execution is to start. If not specified, execution will begin with the first line number of the program. It must be represented as a numeric constant.

The RUN command may be issued from the Basic READY prompt to load and execute a program, or may be used from within an LBASIC program to perform a chaining of programs. If the RUN command is given with a filespec, any program which is currently resident in memory will be overwritten, and the program specified in the RUN command will be loaded and executed.

If the RUN command is given with just a filespec (i.e. no additional parameters are specified), no variables will be retained, and any open files will be closed.

If the RUN command is given with the R parameter, all variables will be lost, but any files which were opened will remain open, and will utilize the same buffer number. Realize that if the R parameter is used, any open files must be re-fielded.

If the RUN command is given with the V parameter, any established variables will be maintained, and all open files will remain open. There are several points to be considered when using the V parameter. In addition to all files remaining open, the fielding of the buffer associated with the open file will remain intact. Hence, re-fielding is not required. Any DEFinition statements (such as DEFINT and DEFSTR) must be re-established in the program to be chained. The CLEAR command should not be encountered in the program to be chained, as execution of a CLEAR statement will close all open files and destroy any established variables.

It should be obvious to the user that if the program to be chained is longer than the calling program, or uses more variables than the calling program, an OUT OF MEMORY or OUT OF STRING SPACE error may occur. To utilize this feature to its fullest capabilities, forethought must go into the determination of variable names to be carried over from one program to another.

If the RUN command is given with the line number parameter, the program specified will be loaded, and execution will begin at the line specified. Realize that the line number specified must be an existing line number, or an UNDEFINED LINE NUMBER error will be generated.

The R/V and line number parameters may be specified individually, or they may appear together in the RUN command. If both parameters are specified, the R/V parameter must physically come before the line number parameter.

Example 1

Suppose you have an LBASIC program named MYPROG/BAS, and this program has been saved onto a disk which is currently in drive 1. One of the following commands may be given to load and execute the above program.

```
RUN"MYPROG/BAS:1"
A$="MYPROG/BAS:1":RUN A$
```

After either of the above commands are executed, any program currently in memory will be overwritten, and the program MYPROG/BAS will be loaded and executed. Any open files will be closed, and any established variables will be destroyed.

Example 2

Suppose you wish to load and execute the program MYPROG/BAS as described in the above example, except that you wish execution to begin at line 3ØØØ in the program. The following command will cause the program to be loaded, and execution will begin at line 3ØØØ.

```
RUN"MYPROG/BAS:1",3ØØØ
```

Example 3

This example will illustrate how to use the V parameter of the RUN command to maintain variables between chained programs. Listed below will be two programs that reference each other (PROG1/BAS and PROG2/BAS). The sequence will be started by issuing the command RUN"PROG1/BAS". Both programs must have been saved on disk prior to trying to execute either.

```
5 'PROG1/BAS
1Ø CLEAR 2ØØØ
2Ø DEFINTA-Z:DEFSTRS
3Ø IF A=Ø THEN S="PROG1/BAS"
4Ø CLS
5Ø A=A+5
6Ø PRINT"THIS IS ";S,"A=";A
7Ø IF A>1ØØ THEN END
8Ø S="PROG2/BAS"
9Ø INPUT"PRESS <ENTER> TO RUN PROG2/BAS";S1
1ØØ RUN"PROG2/BAS",V,2Ø
```

```
5 'PROG2/BAS
1Ø CLEAR 2ØØØ
```

```
20 DEFINTA-Z:DEFSTRS
30 CLS
40 A=A+3
50 PRINT"THIS IS ";S,"A=";A
60 S="PROG1/BAS"
70 INPUT"PRESS <ENTER> TO RUN PROG1/BAS";S1
80 RUN"PROG1/BAS",V,20
```

Notice that in each of the RUN commands, the line number 20 was specified. This accomplishes two things. It causes execution to start at line 20 of each program, which will cause the CLEAR command in both programs to be bypassed. Also, line 20 must be executed, as all DEF type statements must be re-established when programs are chained using the V parameter. Although this is a very simplistic example, it should illustrate some of the steps needed to perform program chaining while retaining variable assignments. If both the V parameter and a line number are used, the V parameter must come before the line number.

SAVE - Save the LBASIC program resident in memory to disk

The SAVE command will allow you to save the program currently in memory to a disk file. This will allow you to store programs on disk for future use. The syntax for the SAVE command is:

```
SAVE"filespec",A
```

filespec is the file specification you wish to assign to the program file. It may be represented as either a string constant or a string expression.

the A parameter is an optional parameter. If used, the program will be saved out to the file in pure ASCII format. If not specified, the program will be saved out to the file in "compressed" format.

As LBASIC programs are being written or edited, they are contained in the computer's memory. The SAVE command provides a way to save LBASIC programs which are stored in memory out to a disk file, so that they may be referenced at some later time via the LOAD or RUN command.

When the SAVE command is given, one of two things will happen. If the filespec in the SAVE command represents a non-existing file, SAVE will create a file with the filename, extension, and password specified, and store in this file the Basic program currently in memory. If the filespec in the SAVE command represents an already existing file, SAVE will overwrite the existing file with the program in memory.

When the A parameter is not specified in a SAVE command, the program in memory will be saved to a disk file in its compressed form (i.e. compression codes will be used to represent the LBASIC commands and line numbers). If the A parameter is specified in a SAVE command, the program will be saved to the disk file in pure ASCII (e.g. the command PRINT will take up five bytes of disk storage, one byte for each character).

Note: When using the A parameter to save a program, no line in the program should exceed 240 characters in length. If a program is saved with the A parameter and a line in the program is longer than 240 characters, the program will load up to the line which is longer than 240 characters, and the rest of the program will be inaccessible. A Direct Statement in File error will also be generated.

It should be obvious that saving a program in ASCII will consume more disk space than saving the same program in compressed form, but there are certain situations where a program must be saved in ASCII. One case where you have to save a program in ASCII is if you wish to perform a MERGE of a Basic program stored on disk with a program currently in memory. The program to be merged in from disk must have been saved in ASCII, or the merge will abort with an error.

The SAVE command may be given either from the LBASIC Ready prompt, or may be incorporated as a command within a program. If used within a program, the program will SAVE itself, after which normal execution will continue.

Example

Suppose you have keyed in a Basic program, and wish to save this program out to a disk file. The drive you wish to store this file on is drive 1, and the name you wish to assign to this file is GOODPROG/BAS. One SAVE command that may be used to accomplish this might look like this:

```
SAVE"GOODPROG/BAS:1"
```

If you wish to save this program in ASCII, the following command could be used:

```
FS$="GOODPROG/BAS":SAVE FS$,A
```

Note in the above example that the filespec was represented as a string variable. Also note that the A parameter must appear as a literal constant, and cannot be expressed as a string expression.

SET EOF - Reset end of file marker to "shrink" the size of a random file

The SET EOF command may be used to "shrink" the amount of space taken up by a file, and thus free-up additional disk space. The syntax for the SET EOF command is:

```
SET EOFn
```

n represents the buffer number used to open the file in question, and can be expressed as an integer constant or an integer expression.

The SET EOF command is used primarily in conjunction with random files. In some applications, a random file may contain unwanted records at the end of the file. The SET EOF command will furnish you with a way to eliminate these unwanted records. The function it performs is to reset the end of file marker for the file in question to a value less than the current end of file marker. This will cause all records whose record numbers are greater than the new end of file marker value to be ignored, and thus make these records inaccessible. Also, the space taken up on the disk by these "eliminated" records will be added to the free space available, and thus may be reused.

To use the SET EOF command, you must open the file in question as a random file. It is highly recommended that the record length used to open the file be the same as the record length used for normal access to the file.

After the file has been opened, perform a GET of the record you wish to be the last record in the file. You may then use the SET EOF command to reset the end of file marker to the current record number, and thus eliminate all unwanted records (by doing a GET, the current record number will be changed to the value of the record which was retrieved).

Example

Suppose you have a random file named XTRA/DAT which currently contains 1ØØ records, and you wish to eliminate the last 5Ø records of the file (records 51-1ØØ). Assume also that the file has been opened in the random mode, using buffer number 3. The following commands may be used to accomplish this "file shrinkage".

```
GET3,5Ø:SET EOF3
```

N O T E

Be extremely careful when using the SET EOF command. Once records have been eliminated from a file using this command, they might not be recoverable!! It is beyond the scope of this manual to discuss techniques used to recover lost information in a file. The best prevention for such an occurrence is caution!

TIME$ - Return the date and time as a string

The TIME$ command will retrieve the current date and time (as kept by the real time clock) as a string. The syntax for the TIME$ command is:

```
TIME$
```

The value returned from the TIME$ command can be used in a similar manner as the value returned from the MEM command. It may be used directly (as in the statement PRINT TIME$), or may be assigned to a string variable (as in A$=TIME$). The value returned by the TIME$ command will always be a 17 character string, and will be defined by the following format:

```
mm/dd/yy hh:mm:ss
```

mm, dd, and yy represent the month, day of the month, and year respectively, as kept by the operating system. The hh, mm, and ss represent the hours (ØØ-23), minutes (ØØ-59) and seconds (ØØ-59) respectively, as retrieved from the real time clock when the TIME$ command was actually executed. The slashes ("/") and colons (":") will always be present in the string, and a space will always separate the date information from the time information.

USR - Execute a user written machine language subroutine.

The USR command will allow an LBASIC program to branch to a user written machine language subroutine. The syntax for the USR command is:

```
variable=USRn(integer value)
```

variable must be a numeric variable, and in most cases should be of integer type. If a value is to be returned from the machine language subroutine, it may be contained in this variable when the machine language routine returns to LBASIC.

n is the user routine number (Ø-9) used to identify the routine in question (user routines are defined with the DEFUSR command). The routine number must be represented as a numeric constant.

integer value is a value which will be passed to the user machine language subroutine. It may be represented as a numeric expression or a numeric constant, and must be expressed as an integer value.

The USR command will allow you to jump to a machine language subroutine from within your Basic program. The machine language subroutine will generally be resident in high memory, and the memory used by the module must be protected, either using the MEMORY Library command, or by specifying the MEM parameter when entering LBASIC.

Prior to issuing a USR call, the starting address of the specific machine language subroutine must have been defined using the DEFUSR command.

Once the USR call is performed, execution of your Basic program will be halted, and a jump will be done to the address specified in the corresponding DEFUSR statement. Your machine language subroutine will then take over, until a return to Basic is performed in the machine language module. Once this return to Basic has been encountered, your Basic program will regain control.

Example 1 - Initiating a USR call

Suppose you have loaded and protected a machine language module. In addition, you have defined this machine language module with the following command:

```
DEFUSR5=&HF4ØØ
```

To perform a jump to this machine language module, the following command may be given:

```
XX%=USR5(1Ø24)
```

Upon executing the above command, execution of the Basic program will be halted, and the machine language instruction at address X'F4ØØ' will be executed. The value 1Ø24 will be passed to the machine language routine. The machine language routine will continue to be executed, until a return to Basic is encountered. If any value is to be returned from the subroutine, it will be contained in the integer variable XX% when Basic regains control.

Example 2 - Passing values to and from machine language subroutines

In the above example, the value 1Ø24 was passed to the machine language subroutine. In order to utilize this value in the subroutine, the first statement of the machine language routine should be the following:

```
CALL ØA7FH
```

Executing the above command as the first statement in the machine language subroutine will cause the value 1Ø24 to be placed in the HL register, with H containing the MSB, and L containing the LSB of the value.

To return a value from a machine language subroutine to Basic, you should use the following command as the last statement in your subroutine:

```
JP ØA9AH
```

After your machine language module executes the above command, control will return to Basic (the statement following the USR call), and the variable used in the USR call will be assigned the value that was in the HL register pair prior to the JP command. If no value is to be returned from your machine language module, you may use a RET command to return to Basic.

CMD"N" - LBASIC PROGRAM RENUMBERING

This LBASIC feature will renumber LBASIC program line numbers as well as all line references such as GOSUB and GOTO. The syntax is:

```
CMD"N ! aaaa,bbbb,cccc,dddd"

!     optional parameter to skip the complete scan
      for errors before renumbering begins.

aaaa  line number of the current program to start the
      renumbering from.

bbbb  new line number for line aaaa.

cccc  increment between line numbers.

dddd  the last line number to be renumbered.
```

Both LBASIC/CMD and LBASIC/OV1 must be present on the disk, or a "Program Not Found" error will occur.

You cannot have a line number zero (Ø) if renumbering an LBASIC program.

This renumber feature will allow you to renumber all or parts of the LBASIC program currently in memory. The lines to be renumbered can be anywhere in the program. However, if the parameters you use would result in the renumbered lines being out of sequence, a BAD PARAMETERS error will occur.

If you do not specify the exclamation point (!) character, a full scan for errors will be done before the renumbering starts. If errors do exist, no lines will be changed. It is usually much easier to fix the errors before the lines are renumbered!

If you do specify the !, any error found will still abort the renumbering. However, all internal line number references will have already been changed up to the line that cause the error. Do not use the ! parameter unless you are absolutely sure that no errors exist.

The default values for the line and increment parameters are as follows:

aaaa = 1
bbbb = 2Ø
cccc = 2Ø
dddd = 65529

CMD"X" - LBASIC CROSS REFERENCE

This LBASIC feature will produce a cross reference of variables and line numbers for your LBASIC program currently in memory. The syntax is:

```
CMD"X devspec/filespec  parameter,<title>"

devspec/filespec is the device or file the listing will
be sent to. If not specified, it will go to the screen.

parameter specifies Variables or Lines as follows:

-V          all Variables.

=variable   only the variable specified.

-L          all Line numbers.

=number     only the line number specified.


<title>  an optional title to be printed on the top
         of each page.
```

Both LBASIC/CMD and LBASIC/OV2 must be present on a disk or a "Program Not Found" error will occur.

You cannot have a line number zero (Ø) if you wish to use the cross reference utility.

This cross reference feature will allow you to produce a list of the variable and line number references of an LBASIC program. This list may be sent to any device in the system, such as the *DO (video screen), *PR (line printer), etc. It may also be sent directly to a specified disk file. If sent to a file, CMD"X" will use the default extension of /TXT.

Parameters are allowed to determine which variables or line numbers will be listed. If no parameter is specified, all variables and line numbers will be cross referenced.

If you wish a title to be put on the top of every page in the list, it must be specified between less-than/greater-than symbols in the command line.

L B A S I C E R R O R D I C T I O N A R Y

Incorporated in ROM Basic are various error messages and error codes. These error codes are provided for the user so that certain types of errors may be "trapped" for, and the execution of the Basic program in question will not be interrupted. As pointed out in the ROM Basic manual, the user may determine the exact nature of an error by utilizing the ERR and ERL commands.

Because many new commands are included in LBASIC which are not a part of ROM Basic, LBASIC will have in its error dictionary new error codes (disk error codes), along with the error codes found in ROM Basic. The error dictionary for LBASIC is contained in the file LBASIC/OV3. For this reason, LBASIC/OV3 must always be present on a disk in the system when programming in Basic.

This part of the manual will list the disk error codes and messages, and will include a brief description of each error. The user should realize that the descriptions given for each error are not all inclusive. That is to say, the example circumstances given for a particular error may not encompass all circumstances which could generate the error in question.

Before we begin giving these disk error codes, a few general points should be made. LBASIC's error dictionary is not as large as the error dictionary found in LDOS. For this reason, several different types of disk related errors may produce the same LBASIC error message. To pinpoint the exact nature of a disk related error, it may be beneficial to determine the LDOS interpretation of an error. After a disk related error occurs, you may determine the associated LDOS error message by performing a CMD"E". This may be useful when, for instance, you get the LBASIC error message "Disk I/O Error", as several different occurrences may cause this type of error. For more information, refer to CMD"E".

All error codes given in this manual will be the value returned by the ERR command. In your ROM Basic manual, the error codes given may be derived by the value of ERR/2 or ERR/2+1. If you wish your LBASIC program to conform to these conventions, the error codes listed here must be adjusted accordingly.

Error 1Ø0 ---------- Field Overflow

The Field Overflow error indicates that the number of bytes fielded for a random file exceeds the record length of the file (as specified in the OPEN statement).

Error 1Ø2 ---------- Internal Error

An Internal Error will occur when the error in question cannot be interpretted. One way an Internal Error may be generated is to issue a CMD"L" command, and the file to be loaded is not found.

Error 1Ø4 ---------- Bad File Number

A Bad File Number error will occur when a file is opened using an illegal buffer number (a buffer number greater than the total number of files specified when entering LBASIC), or fielding a buffer which does not correspond to an open random file.

Error 1Ø6 ---------- File Not Found

A File Not Found error indicates that the file being referenced does not exist. This error may occur after an OPEN"I", OPEN"EO", OPEN"OO", OPEN"RO", LOAD or RUN command has been issued.

Error 1Ø8 ---------- Bad File Mode

A Bad File Mode error indicates that a file is being accessed improperly. This may occur when, for instance, you try to access a file opened as a random file in a sequential manner (i.e. issue an INPUT# command after opening a file in the random mode).

Error 11Ø ---------- File Already Open

A File Already Open error will be generated when you try to OPEN a file using a buffer that corresponds to an already open file. Note that no error will be generated if the same file is in an open state using two different buffers at the same time (This practice is NOT advised).

Error 114 ---------- Disk I/O Error

A Disk I/O Error will occur when an input from or an output to a disk file is unsuccessful. A typical LDOS error message which is associated with the Disk I/O Error is a Parity Error.

Error 116 ---------- File Already Exists

A File Already Exists error will be generated when using an OPEN"xN" command if the file already exists.

Error 122 ---------- Disk Full

A Disk Full error will indicate that all of the free space on a disk has been consumed. In some cases, the occurrence of a disk becoming full (i.e. all of the disk space being consumed) may generate a Disk Write Protected error.

Error 124 ---------- Input Past End

The Input Past End error applies only to sequential files opened for input, and will occur when a read of the file is attempted after all data in the file has been input.

Error 126 ---------- Bad Record Number

A Bad Record Number error will be issued when record number Ø (or some other illegal record number) is accessed in a random file.

Error 128 ---------- Bad File Name

A Bad File Name error will be generated when the file specified in an OPEN, SAVE, LOAD, RUN or MERGE command does not conform to the rules governing valid LDOS filespecs.

Error 132 ---------- Direct Statement in File

A Direct Statement in File error will be generated when a LOAD is performed of a file that is not an LBASIC program (usually when a LOAD of a data file is attempted). This type of error will also be generated when an LBASIC program which was saved in ASCII is loaded, and a line in the program exceeds 24Ø characters in length.

Error 134 ---------- Too Many Files

The Too Many Files error will occur when an attempt is made to add another extent to a file when all directory entries have been used. This type of error will be very uncommon.

Error 136 ---------- Disk Write Protected

A Disk Write Protected error usually indicates that a write has been attempted to a write protected disk. Other types of errors may also generate a Disk Write Protected error. If the disk in question is not write protected, use CMD"E" to determine the exact error.

Error 138 ---------- File Access Denied

A File Access Denied error may be generated when a password protected file (either a data file or a program file) is referenced using an incorrect password.

Error 140 ---------- Blocked File Error

A Blocked File Error will occur if you attempt to OPEN a random file with an LRL of other than 256 after entering LBASIC and specifying the parameter BLK=OFF.

Error 142 ---------- System Command Aborted

A System Command Aborted error will occur if an LDOS command called by the CMD"command" function is manually aborted.

Error 144 ---------- Protection Has Cleared Memory

The Protection Has Cleared Memory error will be generated if an attempt is made to illegally access an EXECute only program without using the proper password. The program and variables will be cleared from memory.

TECHNICAL TABLE OF CONTENTS

D E V I C E C O N T R O L B L O C K (DCB)

The Device Control Block (DCB) is an area of memory that contains information used to interface the operating system with various logical devices such as the keyboard (*KI), the video display (*DO), a printer (*PR), a communications line (*CL), or other device you may define. A DCB follows a strict format that defines the utilization of certain of the available bytes. This format must be followed in all Device Control Blocks established by the user. The three bytes of the DCB not specifically defined by the operating system are considered a storage space, and may contain parameters associated with the specific device. For example, LDOS uses the storage space bytes in the video DCB to keep the current address of the video cursor as well as the cursor character. In the line printer DCB, the maximum number of lines per page and the current line number are kept in this storage space. The following information provides specifications for each assigned DCB BYTE.

Byte Ø .... TYPE byte

Bit 7....This bit specifies that the Device Control Block is actually a File Control Block (FCB) with the file in an OPEN condition. Since there is a great deal of similarity between DCBs and FCBs, and devices may be routed to files, tracing a path through device links may reveal a "device" with this bit set, indicating a routing to a file.

Bit 6 & 5..These bits are reserved for future use.

Bit 4....If set, then the device defined by the DCB is routed to another device, and bytes 1 and 2 contain the address of the device in the route chain.

Bit 3....If set, then the device defined by the DCB is a NIL device. Any output directed to the device will be discarded. Any input request will be satisfied with a ZERO return condition. If bit 3 is set, bits Ø, 1, and 2, must be reset.

Bit 2....If set, then the device defined by the DCB is capable of handling requests generated by the @CTL system call. See the System Entry Point Table for additional information.

Bit 1....If set, then the device defined by the DCB is capable of handling output requests which normally come from the @PUT system vector.

Bit Ø....If set, then the device defined by the DCB is capable of handling requests for input which normally come from the @GET system vector.

Byte 1 .... Low-order Vector

This contains the low-order address of the driver routine that supports the hardware assigned to this DCB.

Byte 2 .... High-order Vector

This contains the high-order address of the driver routine that supports the hardware assigned to this DCB.

Bytes 3-5 .... Variable Storage Area

These three bytes are reserved for variable storage of parameters associated with each driver. It is up to the driver software to assign their use.

* MODEL I - Bytes 6-7

These locations normally contain the first and second alphabetic characters of the devspec. The system uses the devspecs as a reference in searching the device control block tables. If the device is routed to a file, DCB+6 will contain the drive number containing the file, and DCB+7 will contain the DEC of the file.

* MODEL III - Bytes 6-7

These bytes are only present in the *KI, *DO, and *PR device control blocks. Their use will vary depending on the individual device.

The system maintains space in low memory for the storage of the Device Control Blocks. This space is assigned as follows (note: address assignments are hexadecimal values):

Address Range		Assignment
Model I	Model III	
<4Ø15-4Ø1C>	[4Ø15-4Ø1C]	*KI - Keyboard
<4Ø1D-4Ø24>	[4Ø1D-4Ø24]	*DO - Video Display
<4Ø25-4Ø2C>	[4Ø25-4Ø2C]	*PR - Printer
<43CØ-43C7>	[42C2-42C7]	*JL - Job Log
<43C8-43CF>	[42C8-42CD]	*SI - Standard Input
<43DØ-43D7>	[42CE-42D3]	*SO - Standard Output
<43D8-43DF>	[42D4-42D9]	- 1st Spare
<43EØ-43E7>	[42DA-42DF]	- 2nd Spare
<43E8-43EF>	[42EØ-42E5]	- 3rd Spare
<43FØ-43F7>	[42E6-42EB]	- 4th Spare
<43F8-43FF>	N/A	- 5th Spare

As can be observed, space for additional devices not currently defined as normal LDOS system devices has been made available. Any device assigned by the user to a spare slot may be removed from the system after the device is RESET by using the "KILL devspec" library command. The LDOS defined devices *DO, *KI, *PR, and *JL are protected and cannot be killed.

F I L T E R S and D R I V E R S

All devices used with LDOS, whether an actual piece of hardware or a "phantom" device created by the user, require some type of driving program (routine). The driver program is used to interface the device with the operating system, and provides the means to deal with special features and requirements of the device hardware. LDOS uses an area of memory called the DCB (Device Control Block) to keep information about a device and its driver program. Some drivers are already implemented within the ROM interpreter to handle standard devices. For instance, the ROM includes drivers for handshaking the keyboard, video, and parallel printer.

LDOS contains library commands that provide easy access to any device so that modifications may be made to the way in which devices are treated by the system. The characteristics of a driver may be modified by introduction of a FILTER. For instance, suppose your printer required a line feed upon receipt of a carriage return to advance the paper. The ROM printer driver does not provide this function. Instead of writing a completely new printer driver, only a filter need be included to add that single function.

LDOS provides two library commands to aid in interfacing drivers and filters. The SET command is used to define a new device or re-define an existing device and set that device to a driver. FILTER is used to interface a filtering routine to an existing device located in the DCB tables.

The SET command takes the device spec (*XY) from the command line "SET *XY to DRIVER" and searches the DCB tables for a matching device. If the requested device is not defined in your configuration (use the DEVICE command to find out), SET establishes a device control block for the new device. What differentiates FILTER from SET is that FILTER will abort and provide an error message "device not available" if the device is not defined - i.e. you can only FILTER an existing device. FILTER also provides a default file extension of /FLT while SET uses /DVR.

In either case, control then passes to the driver or filter program with register pair DE containing the address of the DCB for the device. This points to the TYPE code (see the section on Device Control Blocks for a detailed explanation of the TYPE byte). Register pair HL points to the command line character separating the driver or filter filespec and optional parameters. This provides the program with the opportunity of parsing a parameter string by using a parameter table and the @PARAM system vector.

The SET and FILTER commands are designed such that the driver or filter routine should be ORGed at address X'52ØØ' and automatically relocate itself to high memory. The routine must load between X'52ØØ' and X'59FF' to be non-destructive. HIGH$ must be properly set after your routine relocates. Samples of filters are provided which should demonstrate the technique of writing the "relocating driver" portion of your routine.

To properly place a filter, it must be between the device control block and the existing driver software. This can be accomplished by stuffing the filter entry point into the DCB vector address. But first we save the existing vector to use in our filter so that we can transfer control to the existing driver software after we filter the flow of I/O.

Additional checking can be performed depending on whether the filter is one directional or two directional. The ROM calls @GET, @PUT, and @CTL initialize the CARRY and ZERO flags to indicate the I/O direction before passing control to the routine vectored from the device control table. A driver/filter can thus be aware of the request - input vs output - and act accordingly.

If you examine the TRAP filter assembly language program, you will note that the filter itself takes up little space in high memory. The bulk of the filter is a driver initialization routine. Although it may appear lengthy, its purpose is just to load the filter driver, insert the vectoring between the device control block and the existing device driver, and perform other maintenance functions. This filter also provides an option for specifying the character to filter at the DOS command level.

Ideally, a general purpose filter driver generator can be constructed which could introduce such benefits as code translation, adding of line feed after return, trapping of certain codes, etc.

IMPORTANT - ALL EQUATE LABELS IN THIS EXAMPLE ARE MODEL I REFERENCES, AND MUST BE CHANGED IF NECESSARY FOR MODEL III ASSEMBLY!

Editor Assembler 3.4 12/05/80 00:18:50 SAMPLE FILTER Page 0001

```
00010 ;TRAP/ASM - 11/01/80
00020         TITLE   <SAMPLE FILTER>
00030 ;*****
00040 ;       Sample FILTER routine to demonstrate the
00050 ;       use of the FILTER command in LDOS.
00060 ;       This routine traps certain control codes
00070 ;       from being sent to the device being filtered.
00080 ;       Any output device can utilize this routine.
00090 ;
00100 ;       This routine also demonstrates the use of the
00110 ;       parameter scanner in the operating system.
00120 ;       A single byte to trap can be passed in the
00130 ;       command line as a parameter. If not entered,
00140 ;       it will default to X'0E', the infamous cursor
00150 ;       on character which if sent to a printer, will
00160 ;       cause expanded character mode on a lot of dot
00170 ;       matrix printers if CURSOR ON is sent to *PR.
00180 ;
00190 ;       To filter the printer output, issue:
00200 ;        FILTER *PR using TRAP (BYTE=X'dd')
00210 ;
00220 ;
00230 ;*****
00240 LF      EQU     10
00250 CR      EQU     13      ;<ENTER> key
00260 @EXIT   EQU     402DH   ;DOS return entry
00270 @ABORT  EQU     4030H   ;error abort
00280 HIGH$   EQU     4049H   ;highest usable memory
00290 @DSPLY  EQU     4467H   ;display message
00300 @PARAM  EQU     4476H   ;parameter scanner
00310 @LOGOT  EQU     447BH   ;display & log message
00320 ;*****
00330 ;       LDOS uses a SET & FILTER library command
00340 ;       which loads at X'5A00'. This provides
00350 ;       the opportunity to load all relocatable
00360 ;       drivers initially at X'5200'. The driver loader
00370 ;       should then relocate to high memory honoring
00380 ;       the HIGH$ pointer & resetting it as needed.
00390 ;*****
00400         ORG     5200H
00410 ;*****
00420 ;       After processing the command line, the
00430 ;       SET/FILTER command runs the driver. On
00440 ;       entry to the driver, register pair DE
00450 ;       contains the device code table address
00460 ;       for the device identified in the command
00470 ;       line. The driver loader (initialization)
00480 ;       follows.
00490 ;*****
00500 TRAP    LD      A,(DE)          ;get device type
00510         AND     2               ;make sure its an
00520         JR      Z,NOGOOD        ;output device
00530         PUSH    DE              ;save device DCB
00540         PUSH    HL              ;save command line ptr
00550         LD      HL,MSG          ;point to initialization
```

```
00560          CALL    @DSPLY          ;message and display it
00570          POP     HL              ;rcvr command line ptr
00580          LD      DE,PARMTBL      ;point to parm table
00590          CALL    @PARAM          ;get parms if any
00600          POP     IX              ;recover device DCB
00610          JR      NZ,PARMERR
00620 BPARM    LD      DE,14           ;init to X'0E'
00630          LD      A,E             ;xfer to reg A
00640          LD      (TRAPBYT+1),A   ;stuff in filter
00650          LD      HL,(HIGH$)      ;reduce HIGH$ by the
00660          LD      BC,LAST-START   ;length of this driver
00670          XOR     A               ;clear the carry flag
00680          SBC     HL,BC           ;calculate new HIGH$
00690          LD      (HIGH$),HL      ;driver now protected
00700          INC     HL              ;point HL at new START
00710          LD      A,(IX+1)        ;xfer orig DCB vector
00720          LD      (ACCEPT+1),A    ;to driver CALL
00730          LD      A,(IX+2)
00740          LD      (ACCEPT+2),A
00750          DI                      ;interrupts off for now
00760          LD      (IX+1),L        ;update DCB vector
00770          LD      (IX+2),H        ;to filter entry
00780          EX      DE,HL           ;xfer new START to DE
00790          LD      HL,START        ;load address of driver
00800          LDIR                    ;move driver to top
00810          EI                      ;clock back on
00820          JP      @EXIT           ;return to DOS
00830 ;*****
00840 ;        error abort
00850 ;*****
00860 PARMERR  LD      HL,PRM.MSG
00870          JR      ERREXIT
00880 NOGOOD   LD      HL,ERR.MSG
00890 ERREXIT  CALL    @LOGOT          ;log error message
00900          JP      @ABORT          ;abort the request!
00910 MSG      DM      'Sample filter to trap control codes',CR
00920 ERR.MSG  DM      'This filter is for output only!',CR
00930 PRM.MSG  DB      'Bad parameters',CR
00940 ;*****
00950 ; Actual FILTER routine to shift up to HIGH$
00960 ;*****
00970 START    JR      C,GETBYT        ;jump if *GET request
00980          JR      Z,PUTBYT        ;jump if *PUT request
00990 ;*****
01000 ;        if carry is not set and Z-flag is not set,
01001 ;        then the request came from @CTL. This
01002 ;        routine could just eliminate the "JR Z,PUTBYT"
01003 ;        since it is shown strictly for demonstration
01004 ;        of how to differentiate the three vectors.
01005 ;*****
01010 PUTBYT   LD      A,C             ;grab the output byte
01020 ;*****
01030 ;        any coding for the entrapment of specific
01040 ;        bytes can be done here
01050 ;*****
01060          PUSH    AF              ;save the flag value
01070 TRAPBYT  CP      0               ;space for trap char
01080          JR      NZ,$+4          ;branch if not trapped
01090          POP     AF              ;rcvr flag value
01100          RET                     ;return if trapped!
```

```
01110           POP     AF              ;rcvr flag value
01120 ACCEPT    JP      0               ;output to orig device
01121 GETBYT    EQU     ACCEPT          ;or input w/o filtering
01130 LAST      EQU     $
01140 ;*****
01150 ;         parameter table - format as follows:
01160 ; 1 => 6-character parm word buffered with spaces
01170 ; 2 => address of word to receive the value parsed
01180 ; 3 => repeat 1 & 2 for as many parms desired
01190 ; 4 => end with an X'00' to indicate the table end
01200 ;
01210 ;         The parameter scanner accepts parms in the
01220 ;         following format:
01230 ;          PARM=X'dddd' :=:hexadecimal entry (max 4-digits)
01240 ;          PARM=ddddd   :=:decimal entry (max 65535)
01250 ;          PARM="string":=:string entry, word address
01260 ;                           will contain the address of
01270 ;                           the 1st character of "string"
01280 ;
01290 ;         On return, PARAM will set the Z-flag if parsing
01300 ;         is OK, else the Z-flag will be reset (NZ)
01310 ;*****
01320 PARMTBL   DB      'BYTE  '        ;parameter word
01330           DW      BPARM+1         ;storage address
01340           NOP                     ;table end indicator
01350           END     TRAP
```

IMPORTANT - ALL EQUATE LABELS IN THIS EXAMPLE ARE MODEL I REFERENCES, AND MUST BE CHANGED IF NECESSARY FOR MODEL III ASSEMBLY!

```
Editor Assembler 3.4     12/01/80 11:17:15     LINEFEED FILTER  Page 0001

00010 ;LINEFEED/ASM - 12/08/80
00020         TITLE   <LINEFEED FILTER>
00030 ;*****
00040 ;       FILTER routine to add a line feed after a
00050 ;       carriage return for use with printers that
00060 ;       need a specific line feed to function.
00070 ;
00080 ;       To filter the printer output, issue:
00090 ;               FILTER *PR using LINEFEED
00100 ;
00110 ;
00120 ;*****
00130 LF      EQU     10
00140 CR      EQU     13      ;<ENTER> key
00150 @EXIT   EQU     402DH   ;LDOS return entry
00160 @ABORT  EQU     4030H   ;error abort
00170 HIGH$   EQU     4049H   ;highest usable memory
00180 @DSPLY  EQU     4467H   ;display message
00190 @LOGOT  EQU     447BH   ;display & log message
00200         ORG     5200H
00210 ENTRY   LD      A,(DE)          ;get device type
00220         AND     2               ;make sure its an
00230         JR      Z,NOGOOD        ;output device
00240         PUSH    DE              ;save device DCB
00250         LD      HL,MSG          ;point to initialization
00260         CALL    @DSPLY          ;message and display it
00270         POP     IX              ;recover device DCB
00280         LD      HL,(HIGH$)      ;reduce HIGH$ by the
00290         LD      BC,LAST-START   ;length of this driver
00300         XOR     A               ;clear the carry flag
00310         SBC     HL,BC           ;calculate new HIGH$
00320         LD      (HIGH$),HL      ;driver now protected
00330         INC     HL              ;point HL at new START
00340         LD      A,(IX+1)        ;xfer orig DCB vector
00350         LD      (PUTBYT+1),A    ;to driver CALL
00360         LD      (GETBYT+1),A
00370         LD      A,(IX+2)
00380         LD      (PUTBYT+2),A
00390         LD      (GETBYT+2),A
00400         DI                      ;not during update
00410         LD      (IX+1),L        ;update DCB vector
00420         LD      (IX+2),H        ;to filter entry
00430         EX      DE,HL           ;xfer new START to DE
00440         LD      HL,START        ;load address of driver
00450         LDIR                    ;move driver to top
00460         EI                      ;enable interrupts again
00470         JP      @EXIT           ;return to LDOS Ready
00480 ;*****
00490 ;       error handling
00500 ;*****
00510 NOGOOD  LD      HL,ERR.MSG
00520         CALL    @LOGOT          ;log error message
00530         JP      @ABORT          ;abort the request!
00540 MSG     DM      LF,'This filter will add a '
00550         DM      'line feed to <CR>',CR
```

```
00560 ERR.MSG DM      'This filter is for output only!',CR
00570 ;*****
00580 ; Actual FILTER routine to shift up to HIGH$
00590 ;*****
00600 START   JR      C,GETBYT        ;jump on *GET request
00610 PUTBYT  CALL    0               ;output to orig device
00620         CP      CR              ;was char a <CR>?
00630         RET     NZ              ;go back if not
00640         LD      C,LF            ;else put out the LF
00650         JR      START
00660 GETBYT  JP      0               ;don't filter input
00670 LAST    EQU     $
00680         END     ENTRY
```

D R I V E C O D E T A B L E (DCT)

The Drive Code Table (DCT) is the way in which LDOS interfaces the operating system with specific disk driver routines. This table is one of the examples of the versatility of the system. Ingenuity and oddball hardware will mix well to provide an easy interface. Pay particular attention to the fields indicating the allocation scheme for the drive. This data is an essential ingredient in the allocation and accessibility of file records.

The DCT is located at addresses 47ØØ-474F. It contains eight 1Ø byte positions - one for each logical drive designated Ø-7. The LDOS 5.1 supports a standard configuration of four drives. This will be the default initialization when LDOS is booted.

Here is the table layout:

DCT+Ø :

The 1st byte of a 3-byte vector to the disk I/O driver routines. This would be an X'C3'. If the drive is disabled (see SYSTEM command), this will be an RET instruction (X'C9').

DCT+1 & DCT+2

This will contain the vector transfer address of the disk I/O routines driving the physical hardware.

DCT+3 :

Contains a series of flags for drive specifications. They are encoded as follows:

bit-7....Set to "1" if software write protected, "Ø" if not.

bit-6....Set to "1" for DDEN, set to "Ø" for SDEN

bit-5....Set to "1" if drive is 8" drive. If the drive is a 5-1/4" drive, the bit is "Ø".

bit-4....A "1" will cause the selection of the disk's second side. The first side will be selected if this bit is a "Ø". The bit value will match the side indicator bit in the sector header as written by the FDC.

bit-3....If this bit is set to a "1", it indicates a hard drive (Winchester). A "Ø" in this bit position denotes a floppy device (5-1/4" or 8").

bit-2....Used to indicate the time delay between selection of a 5-1/4" drive and the first poll of the status register. A "1" value indicates Ø.5 seconds while a "Ø" value indicates 1.Ø seconds. See the SYSTEM command for additional details.

If the drive is a hard drive, this bit will instead indicate either a fixed or removable disk, "Ø" = removable, "1" = fixed.

bits-1 & Ø.These contain the step rate specification for the floppy disk controller. Again, see the SYSTEM command.

DCT+4 :

Contains additional drive specifications:

bit-7....This is reserved for future use. It should NOT be used, in order to maintain compatibility with future releases of LDOS.

bit-6....If "1", the controller is capable of double density mode.

Bit-5....A "1" denotes 2-sided operation while a "Ø" indicates single sided operation. Do not confuse this bit with Bit 4 of DCT+3. This bit shows that the disk is 2-sided. The other bit tells the controller what side the current I/O is to be on.

If the hard drive bit is set, a "1" denotes double the cylinder count stored in DCT+6.

bit-4....If "1", indicates an alien (non-standard) disk controller.

bits-Ø-through-3....This contains the physical drive address by bit selection (1, 2, 4, or 8). The system only supports a translation where more than one bit set is not permitted.

If the alien bit is set, these bits may indicate the starting head number.

DCT+5 :

Contains the current cylinder position of the drive. Its normal purpose is to store the track register of the FDC whenever the FDC is selected for access to this drive. It can then be used to reload the track register whenever the FDC is reselected.

If the alien bit is set, this byte may contain the drive select code for the alien controller.

DCT+6 :

This byte contains the highest numbered cylinder on the drive. Since cylinders are numbered from zero, a 35-track drive would be entered as X'22', a 4Ø-track as X'27', and an 8Ø-track as X'4F'. If the hard drive bit is set, the true cylinder count will depend on DCT+4, bit 5. If that bit is a "1", this byte will be only half of the true cylinder count.

DCT+7 :

Contains certain allocation information:

bits-5-through-7.... contain the number of heads for a hard drive.

bits-Ø-through-4....Contain the highest numbered sector relative from zero. A ten-sector per track drive would show a X'Ø9'. If DCT+4, bit 5 indicates 2-sided operation, the sectors per cylinder will be twice this number.

DCT+8 :

Contains additional allocation parameters:

bits-5-through-7....Contain the quantity of granules per track allocated in the formatting process. If DCT+4, bit 5, indicates 2-sided operation, the granules per cylinder will be twice this number. For a hard drive, this is the total granules per cylinder.

bits-Ø-through-4....Contain the quantity of sectors per granule that was used in the formatting operation.

DCT+9 :

Contains the cylinder where the directory is located. For any directory access, the system will first attempt to use this value to read the directory prior to examining the BOOT sector directory storage byte in case the READ operation was unsuccessful.

It is essential that bytes DCT+6, DCT+7, and DCT+8 all relate without conflicts. That is to say, the highest numbered sector (+1) divided by the quantity of sectors per granule (+1), should equal the number of granules per track (+1).

D I S K I / O T A B L E

LDOS interfaces with hardware peripherals by means of software drivers. The drivers are, in general, coupled to the operating system through data parameters stored in the system's many tables. In this manner, hardware not currently supported by LDOS may be easily supported by generating the appropriate driver software and updating the system tables.

Disk drive sub-systems, such as controllers for 5-1/4" drives, 8" drives, and hard disk drives, have many parameters addressed in the Drive Code Table (DCT). In addition to those operating parameters, controllers also require various commands to control the physical devices. These are commands such as SELECT, SECTOR READ, SECTOR WRITE, etc. LDOS has defined a standard linkage to deal with most commands available on standard Floppy Disk Controllers.

The resident system (SYSØ) contains a series of entry points that deal with drivers linking to controllers. However, every function defined by LDOS is not contained in SYSØ since certain disk functions are not normally used in file access. This is not an undue restriction because it is not essential that all controller functions be routed through SYSØ routines. Certain controller function can be just as easily controlled from specialized application software.

The manner in which the driver controller linkage is established is by passing a function value contained in register "B" to the software driver that interfaces to the controller. Sixteen functions have been defined within LDOS. The following table briefly describes these functions:

HEX	DEC	FUNCTION	OPERATION PERFORMED
X'ØØ'	Ø	NO Operation	...tests if drive is assigned in DCT
X'Ø1'	1	SELECT	.....new drive and return status
X'Ø2'	2	INIT	.....set to cylinder Ø, restore, set side Ø
X'Ø3'	3	RESET	.....the Floppy Disk Controller
X'Ø4'	4	RESTOR	.....issue FDC RESTORE command
X'Ø5'	5	STEPIN	.....issue FDC STEP IN command
X'Ø6'	6	SEEK	.....seek a cylinder
X'Ø7'	7	TSTBSY	.....test if requested drive is busy
X'Ø8'	8	RDHDR	.....read sector header information
X'Ø9'	9	RDSEC	.....read sector command
X'ØA'	1Ø	VERSEC	.....verify if sector readable
X'ØB'	11	RDCYL	.....issue an FDC cylinder read command
X'ØC'	12	FORMAT	.....format the device
X'ØD'	13	WRSEC	.....write a sector
X'ØE'	14	WRSYS	.....write a system sector (e.g. directory)
X'ØF'	15	WRCYL	.....issue an FDC cylinder write command

A detailed explanation of interfacing to various controllers is beyond the scope of this reference manual. It is to be understood that complex controller interfacing be undertaken only by those having the necessary assembly language skills.

D I R E C T O R Y R E C O R D S (DIREC)

The directory contains information required to access all files on the disk. The section containing directory records is limited to a maximum of 32 sectors due to physical limitations in the Hash Index Table. Two sectors are used for the Granule Allocation Table and the Hash Index Table. The directory is also contained completely on a single cylinder. Thus, a 1Ø sector per cylinder formatted disk will have, at most, 8 directory sectors. Consult the HIT documentation for the formula calculating the number of directory sectors.

A directory record is 32 bytes in length. Thus, each directory sector contains eight directory records. The first two directory records of the first eight directory sectors are reserved for system overlays. This is true even if the diskette does not contain an operating system (i.e. a data diskette). The total capacity of files is equal to the number of directory sectors times eight (since 256/32 = 8). The quantity available for use will always be reduced by 16 to account for those record slots reserved for the operating system. The following table shows the record capacity (file capacity) of each format type. The dash suffix on the density indicator represents the number of sides formated:

	sectors/ cylinder	directory sectors	files per directory	avail for use
5" SDEN-1	1Ø	8	64	48
5" SDEN-2	2Ø	18	144	128
5" DDEN-1	18	16	128	112
5" DDEN-2	36	32	256	24Ø
8" SDEN-1	16	14	112	96
8" SDEN-2	32	3Ø	24Ø	224
8" DDEN-1	3Ø	28	224	2Ø8
8" DDEN-2	6Ø	32	256	24Ø
5" HARD	128	32	256	24Ø
8" HARD	256	32	256	24Ø

LDOS is upward compatible with other TRSDOS (tm) 2.3 compatible operating systems in its directory format. LDOS has further extended the data contained in the directory to add additional features and needed enhancements. The expert application programmer may find useful information in the directory - especially for those that write catalog programs. Since the directory information is so vital to the friendliness of programs, much information is displayed in the directory command as noted in other sections of this manual. A standard system vector has been included to either display an abbreviated directory or place its data in a user defined buffer area. For detailed information on this facility, see the @DODIR vector in the section on LDOS system vectors.

The following provides detailed information on the contents of each directory field:

DIR+Ø

This byte contains the entire attributes of the designated file. It is encoded as follows:

Bits Ø-2 ... contain the access protection level of the file. The 3-bit binary value is encoded as follows:

Ø - FULL 1 - KILL 2 - RENAME 3 - Reserved for LDOS use
4 - WRITE 5 - READ 6 - EXEC 7 - NO ACCESS

Bit 3 ... Specifies the visibility; if "1", the file is INVisible to a directory display or other library function where visibility is a parameter. If a "Ø", then the file is declared VISible.

Bit 4 ... is used to indicate whether the directory record is in use or not. If set to "1", the record is in use. If set to a "Ø", the directory record is not active although it may appear to contain directory information. In contrast to other operating systems that zero out the directory record when you kill a file, LDOS only resets this bit to zero.

Bit 5 ... This bit is reserved for future use. Do not utilize it for any purpose if you want to maintain compatibility with future releases of LDOS.

Bit 6 ... A SYStem file is noted by setting this bit to a "1". If set to a "Ø", the file is declared a non-system file.

Bit 7 ... This flag is used to indicate whether the directory record is the file's primary directory entry (FPDE) or one of its extended directory entries (FXDE). Since a directory entry can contain information on up to four extents (see later notes on the extent fields), a file that is fractured into more than four extents requires additional directory records. If this bit is a "Ø", it implies it is an FPDE. If set to a "1", it is referencing an FXDE.

DIR+1

This contains various file flags and the month field of the packed date of last modification. It is encoded as follows:

Bit 7 ... This bit will be set to a "1" if the file was "CREATEd" (see CREATE Library Command). It will allocate a file that will never shrink in size. It will remain as large as its largest allocation. Since the CREATE command can reference a file that is currently existing but non-CREATEd, it can turn a non-CREATEd file into a CREATEd one. The same effect could be achieved by changing this bit to a "1".

Bit 6 ... If this flag is set to a "1", it indicates that the file has not been backed up since its last modification. The BACKUP utility is the only LDOS facility that will reset this flag. It is set during the close operation if the File Control Block (FCB+Ø, Bit 2) denotes a modification of file data.

Bit 5 ... This bit is reserved for future use. If you want to maintain compatibility with future releases of LDOS, do not utilize this bit for any purpose.

Bit 4 ... If the file was modified during a session where the system date was not maintained, this bit will be set to a "1" to indicate that the packed date of modification, if any, stored in the next fields is not the actual date when the modification occurred. If a "1", the directory command will display plus signs (+) between the date fields if the (A) option is requested.

Bits Ø through 3 ... contain the binary month of the last modification date. If this field is a zero, DATE was not set when the file was established nor since if it was updated.

DIR+2

This byte contains the remaining date of modification fields. They are encoded as follows:

Bits 3 through 7 ... contain the binary day of last modification.

Bits Ø through 2 ... contain the binary YEAR - 8Ø. That is to say that 198Ø would be coded as ØØØ, 1981 as ØØ1, 1982 as Ø1Ø, and so forth.

DIR+3

Contains the end-of-file offset byte. This byte, and the ending record number (ERN), form a triad pointer to the byte position immediately following the last byte written. This also assumes that programmers, interfacing in machine language, properly maintain the next record number (NRN) offset pointer when the file is closed.

DIR+4

Contains the logical record length (LRL) specified when the file was initially generated or subsequently changed with a CLONE parameter.

DIR+5 through DIR+12

Contain the name field of the filespec. The file name will be left justified buffered with trailing blanks.

DIR+13 through DIR+15

Contain the extension field of the filespec. As in the name field, it is left justified buffered with trailing blanks.

DIR+16 & DIR+17

The UPDATE password hash code is contained in this field.

DIR+18 & DIR+19

The ACCESS password hash code is contained in this field. The protection level in DIR+Ø is associated with this password.

DIR+2Ø & DIR+21

This field contains the ending record number (ERN) which is based on full sectors. If the ERN is zero, it indicates a file where no writing has taken place (or the file was not closed properly). If the LRL is not 256, the ERN value represents the sector where the EOF occurs. Actually, use ERN-1 to account for a value relative to sector Ø of the file.

DIR+22 & DIR+23

This is the first extent field. Its contents tell you what cylinder stores the first granule of the extent, what relative granule it is, and how many contiguous grans are in use in the extent. It is encoded according to the following pattern:

DIR+22 Contains the cylinder value for the starting gran of that extent.

DIR+23, bits Ø through 4, contain the quantity of contiguous granules. The value is relative to Ø. Therefore a "Ø" value implies one gran, "1" implies two, and so forth. Since the field is 5 bits, it contains a maximum of X'1F' or 31, which would represent 32 contiguous grans.

DIR+23, bits 5 through 7, contain the granule of the cylinder which is the first granule of the file for that extent. Again, this value is offset from zero.

DIR+24 & DIR+25

Contain the fields for the second extent. The format is identical to extent 1.

DIR+26 & DIR+27

Contain the fields for the third extent. The format is identical to extent 1.

DIR+28 & DIR+29

Contain the fields for the fourth extent. The format is identical to extent 1.

DIR+3Ø

Is a flag noting whether or not a link exists to an extended directory record. If no further directory records are linked, the byte will contain X'FF'. If the value is X'FE', a link is recorded to an extended directory.

DIR+31

This is the link to the extended directory noted by the previous byte. The link code is the Directory Entry Code (DEC) of the extended directory record. The DEC is actually the position of the Hash Index Table byte mapped to the directory record. For additional information, see the section on the Hash Index Table.

EXTENDED DIRECTORY RECORDS

Extended Directory records (FXDE) have the same format as primary Directory records, except that only bytes Ø, 1 and 21 to 31 are utilized. Within byte Ø, only bits 4 and 7 are significant. Byte 1 contains the DEC of the directory record which this is an extension of. An extended directory record may point to yet another directory record, so a file may contain an "unlimited" number of extents (limited only by the total number of non-system directory records available).

GRANULE ALLOCATION TABLE (GAT)

The Granule Allocation Table (GAT) contains information pertinent to the free and assigned space on the disk. The GAT also contains certain data specific to the formatting used on the diskette.

In order to deal with a wide range of hardware storage devices, an entire disk is partitioned into cylinders (tracks) and sectors. Each cylinder has a specified quantity of sectors. A group of sectors is allocated whenever additional space is needed. This group is termed a granule. The choice of a granule size is a compromise over minimum file lengths and overhead during the dynamic allocation process. The GAT is configured to provide for a maximum of eight granules per cylinder. In the allocation bytes, each bit set indicates a corresponding granule in use (or locked out). A reset bit indicates a granule free to be used.

In the GAT byte, bit Ø corresponds to the first relative granule. Bit 1 corresponds to the second relative granule. Bit 2 the third, and so on. A 5-1/4" single density diskette is formatted at 1Ø sectors per cylinder, 5 sectors per granule, 2 granules per cylinder. Thus, that configuration will use only bits Ø & 1 of the GAT byte. The remaining GAT byte will contain all 1's - thereby denoting unavailable granules. Other formatting conventions are as follows:

	sectors/ cylinder	sectors/ granule	granules/ cylinder	maximum cylinders
5" SDEN	1Ø	5	2	8Ø
5" DDEN	18	6	3	8Ø
8" SDEN	16	8	2	77
8" DDEN	3Ø	1Ø	3	77
5" HARD	128	16	8	153
8" HARD	256	32	8	256

This table assumes single sided media. LDOS supports double-sided operation within the confines of the hardware interfacing the physical drives to the CPU. A two-headed drive will function as a single unit with the second side as a cylinder for cylinder extension of the first side. A bit in the Drive Code Table (DCT) indicates one-sided or two-sided drive configuration.

A Winchester-type hard disk can be partitioned by heads into multiple logical drives. Information will be supplied along with the particular drive.

The Granule Allocation Table is the first relative sector of the directory cylinder. The following describes the layout of the GAT and the information contained in it.

GAT+X'ØØ' through GAT+X'5F'

Contains the free/assigned table information. GAT+Ø corresponds to cylinder Ø, GAT+1 corresponds to cylinder 1, GAT+2 corresponds to cylinder 2, and so forth. As noted above, bit Ø of each byte corresponds to the first granule on the cylinder, bit 1 corresponds to the second granule, etc. A "1" indicates the granule is not available for use.

GAT+X'6Ø' through GAT+X'BF'

Contains the available/locked out table information. It corresponds on a cylinder for cylinder basis as does the free/assigned table. It is used specifically during mirror-image backup functions to determine if the destination has the proper capacity to effect a backup of the source diskette.

GAT+X'CØ' through GAT+X'CA'

Used in hard drive configurations by extending the free/assigned table from X'ØØ' through X'CA'. Hard drives cannot be backed up in a mirror-image manner since their re-mapped cylinder configuration would exceed core limits. Thus, there is no need to reserve space for a lockout table. Hard drive capacity up to 2Ø2 mapped cylinders (4Ø4 standard) is supported.

GAT+X'CB'

Contains the operating system version used in formatting the disk. Disks formatted under LDOS 5.1 will have a value of X'51' contained in this byte. It is used to determine whether or not the diskette contains all of the parameters needed for LDOS 5.1 operation.

GAT+X'CC'

This byte contains the number of cylinders in excess of 35. Its use is to minimize the time required to compute the maximum cylinder formatted on the diskette. It was designed to be excess 35 so as to provide complete compatibility with alien systems not maintaining that byte. If you have a diskette that was formatted on an alien system for other than 35 cylinders, this byte can be automatically configured by using the REPAIR utility. See its reference in another section of this manual.

GAT+X'CD'

This byte contains data specific to the formatting of the diskette. Bit 6 set to "1" implies double density formatting. Bit 5 set to "1" indicates two-sided media. Bits 7, 4, and 3 are reserved for future assignment. Bits 2-Ø contain the number of granules per cylinder -1.

GAT+X'CE' and GAT+X'CF'

Contains the 16-bit hash code of the disk master password. Its storage is in standard low-order high-order format.

GAT+X'DØ' through GAT+X'D7'

Contains the diskette pack name. This is the name displayed at boot up if the diskette is a system diskette used for the boot operation. It is also the name displayed during a FREE or DIR. The name is assigned during the formatting operation or an ATTRIB disk renaming operation.

GAT+X'D8' through GAT+X'DF'

Contains the date that the diskette was formatted or the date that it was used as the destination in a backup operation. If the diskette is used during a BOOT, this date will be displayed adjacent to the pack name.

GAT+X'EØ' through GAT+X'FF'

Contains the AUTO command buffer. This is the command that will be executed during a BOOT operation. If there is no AUTO command in place then GAT+X'EØ' will contain an X'ØD'.

HASH INDEX TABLE (HIT)

The Hash Index Table is the key to addressing any file in the directory. It is designed so as to pinpoint the location of a file's directory with a minimum of disk accesses. A minimum quantity of disk accesses is useful to keep overhead low while providing rapid file access.

The procedure that the system uses to locate a file's directory is to first take the file name and extension and construct an 11-byte field with the file name left justified and padded with blanks. The file extension is then inserted, padded with blanks, and will occupy the three least significant bytes of the 11-byte field. This field is then processed through a hashing algorithm which produces a single byte value in the range X'Ø1' through X'FF' (a hash value of X'ØØ' is reserved to indicate a spare HIT position).

The hash code is then stored in the Hash Index Table (HIT) at a position corresponding to the directory record containing the file's directory. Since more than one 11-byte string can hash to identical codes, the opportunity for "collisions" exists. For this reason, the search algorithm will scan the HIT for a matching code entry, will then read the directory record corresponding to the matching HIT position, and will compare the file name/ext stored in the directory with that provided in the file specification. If both match, the directory has been found. If the two fields do not match, the HIT entry was a collision and the algorithm continues its search.

The position of the HIT entry in the hash table itself is called the Directory Entry Code (DEC) of the file. All files will have at least one DEC. Files that are extended beyond four extents will have DECs for each extended directory entry and use up more than one filename slot. Therefore, to maximize the quantity of file slots available, you should keep your files below five extents wherever possible.

Each HIT entry is mapped to the directory sectors by the DEC's position in the HIT. Conceptualize the HIT as eight rows of 32-byte fields. Each row will be mapped to one of the directory records in a directory sector. The first HIT row to the first directory record, the second HIT row to the second directory record, and so forth. Each column of the HIT field (the Ø-31) is mapped to a directory sector. The first column is mapped to the first directory sector in the directory cylinder (not including the GAT and HIT). Therefore, the first column corresponds to sector number 2, the second column to sector number 3, and so forth. The maximum quantity of HIT columns actually used will be governed by the disk formatting according to the formula: N = number of sectors per cylinder minus two, up to a maximum of 32.

In the 5-1/4" single density configuration, there exist ten sectors per cylinder - two reserved for the GAT and HIT. Since only eight directory sectors are possible, only the first eight positions of each HIT row are used. Other formats will use more columns of the HIT, depending on the quantity of sectors per cylinder in the formatting scheme.

This arrangement works nicely when dealt with in assembly language for interfacing. Consider the DEC value of X'84'. If this value is loaded into the accumulator, a simple:

```
AND     1FH
ADD     A,2
```

will extract the sector number of the directory cylinder containing the file's directory. If that same value of X'84' was operated on by:

```
AND     ØEØH
```

the resultant value will be the low-order starting byte of the directory record assuming the directory sector was read into a buffer starting at a page boundary. This procedure makes for easy access to the directory record.

Note that the first DEC found with a matching hash code may, in fact, be the file's extended directory entry (FXDE). It is therefore important, that if you are going to write system code to deal with this directory scheme, you properly deal with the FPDE/FXDE entries. See the section on directory records for additional information.

The following chart may help to visualize the correlation of the Hash Index Table to the directory records. Each byte value shown represents the position in the HIT. This position value is called the DEC. The actual contents of each byte will be either a X'ØØ' indicating a spare slot, or the 1-byte hash code of the file occupying the corresponding directory record.

	---------- C O L U M N S ----------															
Row 1	ØØ	Ø1	Ø2	Ø3	Ø4	Ø5	Ø6	Ø7	Ø8	Ø9	ØA	ØB	ØC	ØD	ØE	ØF
	1Ø	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
Row 2	2Ø	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F
	3Ø	31	32	33	34	35	36	37	38	39	3A	3B	3C	3D	3E	3F
Row 3	4Ø	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F
	5Ø	51	52	53	54	55	56	57	58	59	5A	5B	5C	5D	5E	5F
Row 4	6Ø	61	62	63	64	65	66	67	68	69	6A	6B	6C	6D	6E	6F
	7Ø	71	72	73	74	75	76	77	78	79	7A	7B	7C	7D	7E	7F
Row 5	8Ø	81	82	83	84	85	86	87	88	89	8A	8B	8C	8D	8E	8F
	9Ø	91	92	93	94	95	96	97	98	99	9A	9B	9C	9D	9E	9F
Row 6	AØ	A1	A2	A3	A4	A5	A6	A7	A8	A9	AA	AB	AC	AD	AE	AF
	BØ	B1	B2	B3	B4	B5	B6	B7	B8	B9	BA	BB	BC	BD	BE	BF
Row 7	CØ	C1	C2	C3	C4	C5	C6	C7	C8	C9	CA	CB	CC	CD	CE	CF
	DØ	D1	D2	D3	D4	D5	D6	D7	D8	D9	DA	DB	DC	DD	DE	DF
Row 8	EØ	E1	E2	E3	E4	E5	E6	E7	E8	E9	EA	EB	EC	ED	EE	EF
	FØ	F1	F2	FF	F4	F5	F6	F7	F8	F9	FA	FB	FC	FD	FE	FF
	---------- C O L U M N S ----------															

The eight directory records for the directory cylinder, sector 2 would correspond to assignments in HIT positions ØØ, 2Ø, 4Ø, 6Ø, 8Ø, AØ, CØ, and EØ. The following positions are reserved for system overlays:

ØØ -> BOOT/SYS	2Ø -> SYS6/SYS
Ø1 -> DIR/SYS	21 -> SYS7/SYS
Ø2 -> SYSØ/SYS	22 -> SYS8/SYS
Ø3 -> SYS1/SYS	23 -> SYS9/SYS
Ø4 -> SYS2/SYS	24 -> SYS1Ø/SYS
Ø5 -> SYS3/SYS	25 -> SYS11/SYS
Ø6 -> SYS4/SYS	26 -> SYS12/SYS
Ø7 -> SYS5/SYS	27 -> SYS13/SYS

These entry positions, of course, correspond to the first two rows of each directory sector for the first eight directory sectors. Since the operating system accesses these overlays by position in the HIT rather than by file name, these positions are always reserved by the system.

The design of the Hash Index Table limits the system to a maximum support of 256 files on any one drive. With the current state of the art in disk drive technology, that limit is not considered to be of major impact - even considering large capacity Winchester drives supported by LDOS. This system will evolve to support the newer types of hardware coming to the market place, as they become available.

F I L E C O N T R O L B L O C K (FCB)

The File Control Block (FCB) is a 32-byte region that is used by the system to interface with a file that has been "opened". Its contents are extremely dynamic. As records are written to or read from the disk file, specific fields in the FCB are modified. It is extremely important that during the time period that a file is open, you avoid changing the contents of the FCB unless you are sure that its alteration will in no way effect the integrity of the file.

During most system access of the FCB, the IX index register is used to reference each field of data. Register pair DE is used primarily for the initial reference to the FCB address. The information contained in each field of the FCB follows:

FCB+Ø

Contains the TYPE code of the control block.

Bit 7 ... If set to "1", will indicate that the file is in an open condition; if set to "Ø", the file is assumed closed. This bit can be tested to determine the "open" or "closed" status of a FCB.

Bits 6-3 ... reserved for future use.

Bit 2 ... will be set to "1" if any WRITE operation was performed by the system on this file. It is used specifically to update the MOD flag in the directory record when the file is closed.

Bits 1 and Ø ... reserved for future use.

FCB+1

Contains status flag bits used in read/write operations by the system.

Bit 7 ... If set to a "1", it indicates that I/O operations will be either full sector operations or byte operations of logical record length (LRL) less than 256. If set to a "Ø", only sector operations will be performed. If you are going to utilize only full sector I/O, system overhead is reduced by specifying the LRL at open time to be Ø (indicating 256). An LRL of other than 256 will set bit 7 to a "1" on open.

Bit 6 ... If set to a "1", it indicates that the end-of-file (EOF) is to be set to ending-record-number (NRN) only if NRN exceeds the current value of EOF. This is the case if random access is to be utilized. During random access, the EOF will not be disturbed unless you are extending the file beyond the last record slot. Any time the position vector (@POSN) is called, it automatically sets bit 6. If bit 6 is set to a "Ø", then EOF will be updated on every WRITE operation.

Bit 5 ... If set to a "Ø", then the disk I/O buffer contains the current sector denoted by NRN. If set to a "1", then the buffer does not contain the current sector. During byte I/O, bit 5 is set when the last byte of the sector is read. A sector read will reset the bit & show the buffer to be current.

Bit 4 ... If set to a "1", it indicates that the buffer contents have been changed since the buffer was read from the file. It is used by the system for determining whether the buffer must be written back to the file before reading another record. If set to a "Ø", the indication is that no buffer modification was performed.

Bit 3 ... Is used to specify that the directory record is to be updated everytime that the NRN exceeds the EOF. Normal operation is to update the directory only when a FCB is closed. Some unattended operations may utilize this extra measure of file protection. It is specified by appending an exclamation mark "!" to the end of a filespec when the filespec is requested at open time.

Bits 2-Ø ... Contain the access protection level as retrieved from the directory of the file. For specific bit patterns, see the directory record explanation.

FCB+2

Is reserved by the system for future use.

FCB+3/4

Contain the buffer address in lo-order - hi-order format. This is the buffer address specified in register pair HL at open time.

FCB+5

Contains the relative byte offset within the current buffer for the next I/O operation. If this offset is a zero value, then FCB+1, Bit 5 must be examined to determine if the 1st byte in the current buffer is the target position or the 1st byte of the next record. If you are performing sector I/O of byte data (i.e. maintaining your own buffering), then it is important to maintain this byte when you close the file if the true end-of-file is not at a sector boundary.

FCB+6

Contains the logical drive number in binary of the drive containing the file. It is absolutely essential that this byte be left undisturbed. It, and FCB+7 are the only links to the directory information for the file.

FCB+7

Contains the directory entry code (DEC) for the file. This code is the relative position in the hash index table where the hash code for the file appears. Do not tamper with this byte. It, and FCB+6, are needed to properly close the file.

FCB+8

Contains the end-of-file byte offset. This byte is similar to FCB+5 except it pertains to the end-of-file rather than the next-record-number.

FCB+9

Contains the logical record length in effect when the file was opened. This may not be the same LRL that exists in the directory. The directory LRL is generated at the file creation and will never change unless another file is cloned to it.

FCB+1Ø/11

Contain the next-record-number (NRN), which is a pointer to the next I/O operation. When a file is opened, NRN is zero indicating a pointer to the beginning. Each sequential sector I/O advances NRN by one.

FCB+12/13

ERN of the file. This is a pointer to the sector that contains the end-of-file indicator. In a null file (one with no records), ERN will be equal to Ø. If one sector had been written, ERN would be equal to 1.

FCB+14/15

Contains the same information as the first extent of the directory. This represents the starting cylinder of the file (FCB+14) and the starting relative granule within the starting cylinder (FCB+15). FCB+15 also contains the number of contiguous granules allocated in the extent. This will always be used as a pointer to the beginning of the file referenced by the FCB.

FCB+16-FCB+19

This is a 4-byte quad that contains cumulative granule allocation information for an extent of the file. Relative bytes Ø & 1 contain the cumulative number of granules allocated to the file up to but not including the extent referenced by this field. Relative byte 2 contains the starting cylinder of this extent. Relative byte 3 contains the starting relative granule for the extent and the number of contiguous granules.

FCB+2Ø-FCB+23

Contain information similar to the above but for another extent of the file.

FCB+24-FCB+27

Contain information similar to the above but for a third extent of the file.

FCB+28-FCB+31

Contain information similar to the above but for a fourth extent of the file.

The file control block contains information only on four extents at any one time. If the file has more than four extents, additional directory accessing will be done to shift the 4-byte quads to make space for the new extent information. Although the system can handle a file of any number of extents, it is wise to keep the total number of extents small. The most efficient file is one with a single extent. The number of extents can be reduced by copying the file to a diskette containing a great deal of free space.

F I L E F O R M A T S

Disk Command File Format

The disk command file (load module) format consists of the following structure:

a. Byte X'Ø5' indicates the FILENAME field, followed by a byte indicating the FILENAME length (typically a six-byte field).

b. Byte X'Ø1' indicates a start-of-block, followed by a one-byte length of block, where the count includes the two byte load address. Following this is the two-byte block load address, followed by the block itself. A block length of X'ØØ' indicates a 254 byte block plus two bytes for the load address. A block length of X'Ø1' indicates a 255 byte block plus two bytes for the load address (ØFFH + Ø2H = Ø1H). Similarly, a block length of X'Ø2' indicates a 256 byte block plus two bytes for the load address (ØØH + Ø2H = Ø2H). Thus, the actual code block length can always be obtained by subtracting two from the value specified. This is repeated for as many blocks as are in the file.

c. Byte X'Ø7' indicates that the following will be a PATCH NAME field. It is followed by the length byte for the actual field containing the PATCH NAME (in ASCII). This PATCH HEADER will be followed by the actual load blocks of the PATCH. When a patch is YANKed or removed the load blocks following the PATCH HEADER will begin with a X'1Ø', which will cause the loader to ignore the blocks.

d. An X'Ø2' is written to indicate the end of the code and the beginning of the transfer address. It is followed by an X'Ø2' block length, then the two-byte transfer address.

e. The standard byte used to indicate a comment is X'1F'. A comment may appear between any two blocks in a file.

NOTE: Those header bytes that have not been defined above are reserved for system use. Some are used in LDOS 5.1, while others are planned for future use. Any byte larger than X'1F' encountered where a header is expected by the system loader will cause a "Load file format error" message.

Tape File Object Code Format

A SYSTEM tape has a similar format; however, the control bytes are different. The SYSTEM file structure is as follows:

a. X'55' indicates the start of the FILENAME, followed by the six character file name (buffered with blanks, X'20' to fill out to six characters).

b. X'3C' indicates the start of a block, followed by a one-byte block length. This, in turn, is followed by the two-byte block load address. The tape block length does not include the load address. The block of code follows. Immediately following the code block is a one-byte checksum determined by the modulo 256 sum of each byte in the block plus each byte of the load address (modulo 256 is achieved by performing 8-bit register addition ignoring all carries out of the high-order bit).

c. (b) is repeated for as many blocks as are needed. An X'78' is written to indicate the end-of-file, followed by the two-byte transfer address.

MEMORY MAP - MODEL I ONLY

This memory map of the LDOS Disk Operating System is not necessarily a complete map of the entire system. Rather, it represents what is felt to be all of the system vectors that could reasonably be used by accomplished assembly language programmers. A few words of caution are in order. All RAM Storage Assignments are for VERSION 5.1.X, MODEL I only. More detailed information concerning these assignments will be found in the SYSTEM ENTRY POINTS and RAM STORAGE Assignments.

ADDRESS	LABEL	DESCRIPTION OF LOCATION
X'ØØØB'	@WHERE	Vector to resolve relocation address
X'ØØ13'	@GET	Input a byte from a logical device or a file
X'ØØ1B'	@PUT	Output a byte to a logical device or a file
X'ØØ23'	@CTL	Output a control byte to a device or a file
X'ØØ2B'	@KBD	Scan the keyboard, and return the character
X'ØØ33'	@DSP	Output a byte to the video display
X'ØØ3B'	@PRT	Output a byte to the printer
X'ØØ4Ø'	@KEYIN	Accept a line of input
X'ØØ49'	@KEY	Input a byte from the keyboard
X'ØØ6Ø'	@PAUSE	Suspend program execution
X'4Ø15'-X'4Ø1C'	KIDCB$	Keyboard DCB
X'4Ø1D'-X'4Ø24'	DODCB$	Video DCB
X'4Ø25'-X'4Ø2C'	PRDCB$	Printer DCB
X'4Ø2D'	@EXIT	Normal program exit and return to LDOS
X'4Ø3Ø'	@ABORT	Abnormal program exit and return to LDOS
X'4Ø33'	@DVRHK	Device Driver hook from ROM for byte I/O
X'4Ø3E'	OSVER$	Contains the operating system version Number
X'4Ø4Ø'	TIMER$	This is the 25 millisecond heartbeat
X'4Ø41'-X'4Ø43'	TIME$	Contains the time-of-day
X'4Ø44'-X'4Ø46	DATE$	Contains the current month and year
X'4Ø47'-X'4Ø48'	DAY$	Contains the current day
X'4Ø49'-X'4Ø4A'	HIGH$	Contains the highest unused RAM address
X'4Ø4B'	INTIM$	Contains an image of the interrupt latch X'37EØ'
X'4Ø4D'-X'4Ø5C'	INTVC$	This area contains eight vectors - one for each bit of the interrupt latch
X'4Ø5D'-X'4Ø7C'	DBGSV$	DEBUG and SYSTEM storage area, DO NOT USE
X'42ØØ'-X'42FF'	SBUFF$	A 256-byte buffer for system disk I/O
X'43ØØ'	@KITSK	Task process during KBD scan
X'43Ø3'	@ICNFG	Initialize configuration
X'43Ø6'	SVDAT1$	Contains year and day in packed format
X'43Ø7'	SVDAT2$	Contains the month in packed format
X'43Ø8'	LDRV$	Currently accessed drive - logical number (Ø-7)
X'43Ø9'	PDRV$	Currently accessed drive - physical drive address (1, 2, 4, or 8)
X'43ØA'-X'43ØB'	JDCB$	Storage area for DCB Address during JCL execution
X'43ØC'-X'43ØD'	JRET$	Storage area for RET Address during JCL execution

ADDRESS	LABEL	DESCRIPTION OF LOCATION
X'43ØE'	OVRLY$	Current system overlay resident
X'43ØF'	SFLAG$	System bit flag
X'431Ø'-X'4311'		Vector to extended DEBUG
X'4312'-X'4314'		Reserved
X'4315'-X'4317'		Used with DEBUG (do not use)
X'4318'-X'4357'	INBUF$	Buffer area of 64 bytes for user command input
X'4358'-X'4377'	JFCB$	JCL FCB during DO processing
X'4396'	@RAMDIR	Get directory record(s) or free space
X'43B8'-X'43B9'	KISV$	KI save vector
X'43BA'-X'43BB'	DOSV$	DO save vector
X'43BC'-X'43BD'	PRSV$	PR save vector
X'43BE'-X'43BF'	KIJCL$	KIJCL save vector
X'43CØ'-X'43C7'	JLDCB$	Joblog DCB
X'43C8'-X'43CF'	SIDCB$	Standard Input DCB
X'43DØ'-X'43D7'	SODCB$	Standard Output DCB
X'43D8'-X'43DF'	S1DCB$	Spare DCB
X'43EØ'-X'43E7'	S2DCB$	Spare DCB
X'43E8'-X'43EF'	S3DCB$	Spare DCB
X'43FØ'-X'43F7'	S4DCB$	Spare DCB
X'43F8'-X'43FF'	S5DCB$	Spare DCB
X'44ØØ'	@CMD	Accept a new command
X'44Ø5'	@CMNDI	Entry to command interpreter
X'44Ø9'	@ERROR	Entry to post an error message
X'44ØD'	@DEBUG	Enter the debugging package
X'441Ø'	@ADTSK	Add an interrupt level task
X'4413'	@RMTSK	Remove an interrupt level task
X'4416'	@RPTSK	Replace the currently executing task vector
X'4419'	@KLTSK	Remove the currently executing task
X'441C'	@FSPEC	Fetch a file or device specification
X'441F'	DFLAG$	System device flag
X'442Ø'	@INIT	Open or initialize a file or device
X'4423'	KFLAG$	System keyboard flag
X'4424'	@OPEN	Open an existing file or device
X'4428'	@CLOSE	Close a file or device
X'442C'	@KILL	Kill a file or device
X'442F'	MFLAG$	Model I machine flag
X'443Ø'	@LOAD	Load a program file
X'4433'	@RUN	Load and execute a program file
X'4436'	@READ	Read a record from a file
X'4439'	@WRITE	Write a record to a file
X'443C'	@VER	Write then verify a record to a file
X'443F'	@REW	Rewind a file to its beginning
X'4442'	@POSN	Position a file to a logical record
X'4445'	@BKSP	Backspace one logical record
X'4448'	@PEOF	Position to the end-of-file
X'444B'	@CKEOF	Check for the end-of-file
X'444E'	@WEOF	Write an end-of-file
X'4451'	XXXXXX	Reserved for future use
X'4454'	@RREAD	Reread the current sector
X'4457'	@RWRIT	Rewrite the current sector
X'445A'	@LOC	Calculate the current logical record number
X'445D'	@LOF	Calculate the EOF logical record number
X'446Ø'	@SKIP	Skip the next record
X'4463'	@DODIR	Do a directory display/buffer

ADDRESS	LABEL	DESCRIPTION OF LOCATION
X'4467'	@DSPLY	Display a message line
X'446A'	@PRINT	Print a message line
X'446D'	@TIME	Get time of day - format (XX:XX:XX)
X'447Ø'	@DATE	Get today's date - format (xx/xx/xx)
X'4473'	@FEXT	Set up a default file extension
X'4476'	@PARAM	Parse an optional parameter string
X'4479'	@MSG	Message line handler
X'447B'	@LOGOT	Display and log a message
X'447E'	@LOGER	Issue a log message
X'448Ø'-X'449F'	CFCB$	File control block for commands
X'44AØ'-X'44B3'	SFCB$	FCB for loading system overlays
X'44B8'	@CKDRV	Check drive for availability
X'44BB'	@FNAME	Fetch file name/ext from the directory
X'44BE'	@DOKEY	Vector for keyboard JCL in DO execution
X'44C1'	@MULT	Multiply HL by A
X'44C4'	@DIV	Divide HL by A
X'45ØØ'-X'4517'	TCB$	Interrupt Task Table
X'47ØØ'-X'474F'	DCT$	Area reserved for the Drive Code Table
X'4754'	SELECT	SELECT new drive
X'4759'	RSELCT	Test if requested drive is busy
X'475E'	SEEK	Seek a cylinder
X'4763'	WRSECT	Write sector
X'4768'	WRPROT	Write system sector
X'476D'	WRTRK	Write a cylinder
X'4772'	VERSEC	Verify a sector
X'4777'	RDSECT	Read a sector
X'478F'	GETDCT	Get Drive Code Table address
X'479C'	DCTBYT	Get a DCT field
X'4B1Ø'	DIRRD	Directory record read
X'4B1F'	DIRWR	Directory record write
X'4B45'	RDSSEC	Read a SYSTEM sector
X'4B65'	DIRCYL	Get the directory cylinder number
X'4B6C'	MULTEA	Multiply E by A
X'4B7B'	DIVEA	Divide E by A
X'4DFE'-X'4DFF'	USTOR$	Pointer to 8 byte user storage area
X'4EØØ'-X'51FF'		LDOS Overlay Area

MEMORY MAP - MODEL III ONLY

This memory map of the LDOS Disk Operating System is not necessarily a complete map of the entire system. Rather, it represents what is felt to be all of the system vectors that could reasonably be used by accomplished assembly language programmers. A few words of caution are in order. All RAM Storage Assignments are for VERSION 5.1.X, MODEL III only. More detailed information concerning these assignments will be found in the SYSTEM ENTRY POINTS and RAM STORAGE Assignments. Most locations that differ from the Model I version of LDOS were moved for compatibility with Model III TRSDOS.

ADDRESS	LABEL	DESCRIPTION OF LOCATION
X'ØØØB'	@WHERE	Vector to resolve relocation address
X'ØØ13'	@GET	Input a byte from a logical device or a file
X'ØØ1B'	@PUT	Output a byte to a logical device or a file
X'ØØ23'	@CTL	Output a control byte to a device or a file
X'ØØ2B'	@KBD	Scan the keyboard, and return the character
X'ØØ33'	@DSP	Output a byte to the video display
X'ØØ3B'	@PRT	Output a byte to the printer
X'ØØ4Ø'	@KEYIN	Accept a line of input
X'ØØ49'	@KEY	Input a byte from the keyboard
X'ØØ6Ø'	@PAUSE	Suspend program execution
X'3Ø33'	@DATE	Get today's date - format (xx/xx/xx)
X'3Ø36'	@TIME	Get time of day - format (xx:xx:xx)
X'4Ø15'-X'4Ø1C'	KIDCB$	Keyboard DCB
X'4Ø1D'-X'4Ø24'	DODCB$	Video DCB
X'4Ø25'-X'4Ø2C'	PRDCB$	Printer DCB
X'4Ø2D'	@EXIT	Normal program exit and return to LDOS
X'4Ø3Ø'	@ABORT	Abnormal program exit and return to LDOS
X'4Ø33'	@DVRHK	Device Driver hook from ROM for byte I/O
X'4Ø3D'	@ADTSK	Add an interrupt level task.
X'4Ø4Ø'	@RMTSK	Remove an interrupt level task
X'4Ø43'	@RPTSK	Replace the currently executing task vector
X'4Ø46'	@KLTSK	Remove the currently executing task
X'4Ø5D'-X'4Ø7C'	DBGSV$	DEBUG and SYSTEM storage area - DO NOT USE
X'42Ø9'	@CKDRV	Check for drive availability
X'4217'	TIME$	Contains time of day
X'421A'	DATE$	Contains the current date
X'421D'	@ICNFG	Initialize configuration
X'422Ø'	JDCB$	Storage area for DCB address during JCL execution
X'4222'	JRET$	Storage area for RET address during JCL execution
X'4225'	INBUF$	Buffer area of 64 bytes for user command input
X'4265'	JFCB$	JCL FCB during DO processing
X'4285'	@KITSK	Task processing during KBD scan
X'4288'	TIMER$	This is the 33.333 ms heartbeat
X'4289'	DFLAG$	The system device flag

ADDRESS	LABEL	DESCRIPTION OF LOCATION
X'428A'	@LOGOT	Display and log a message
X'428D'	@LOGER	Issue a log message
X'429Ø'	@RAMDIR	Get directory record(s) or free space
X'4293'	@FNAME	Fetch file name/ext from the directory
X'4296'	@CMD	Accept a new command
X'4299'	@CMNDI	Entry to command interpreter
X'42A1'	SFCB$	FCB for loading system overlays
X'42B8'-X'42B9	KISV$	Save KI DCB vector
X'42BA'-X'42BB	DOSV$	Save DO DCB vector
X'42BC'-X'42BD'	PRSV$	Save PR DCB vector
X'42BE'-X'42BF'	KIJCL$	Save KIJCL DCB vector
X'42C2'-X'42C7'	JLDCB$	Joblog DCB
X'42C8'-X'42CD'	SIDCB$	Standard Input DCB
X'42CE'-X'42D3'	SODCB$	Standard Output DCB
X'42D4'-X'42D9'	S1DCB$	Spare DCB
X'42DA'-X'42DF'	S2DCB$	Spare DCB
X'42EØ'-X'42E5'	S3DCB$	Spare DCB
X'42E6'-X'42EB'	S4DCB$	Spare DCB
X'42EC'-X'42FF'		Storage for 2 character device names
X'43ØØ'-X'43FF'	SBUFF$	A 256 byte buffer for system disk I/O
X'44ØØ'	EXTDBG$	Vector to extended DEBUG
X'44Ø2'	@MSG	Message line handler
X'44Ø5'	@DBGHK	Used with DEBUG (do not use)
X'44Ø9'	@ERROR	Entry to post an error message
X'44ØD'	@DEBUG	Enter the debugging package
X'4411'-X'4412'	HIGH$	Contains the highest unused RAM address
X'4414'	OVRLY$	Current system overlay resident
X'4417'-X'4418'	DAY$	Contains the day of the year
X'4419'	@DODIR	Do a directory display/buffer
X'441C'	@FSPEC	Fetch a file or device specification
X'441F'	OSVER$	Contains the operating system version number
X'442Ø'	@INIT	Open or initialize a file or device
X'4423'	PDRV$	Currently accessed drive - physical address (1, 2, 4, or 8)
X'4424'	@OPEN	Open an existing file or device
X'4427'	LDRV$	Currently accessed drive - logical number (Ø-7)
X'4428'	@CLOSE	Close a file or device
X'442B'	SFLAG$	System bit flag
X'442C'	@KILL	Kill a file or device
X'443Ø'	@LOAD	Load a program file
X'4433'	@RUN	Load and execute a program file
X'4436'	@READ	Read a record from a file
X'4439'	@WRITE	Write a record to a file
X'443C'	@VER	Write then verify a record to a file
X'443F'	@REW	Rewind a file to its beginning

ADDRESS	LABEL	DESCRIPTION OF LOCATION
X'4442'	@POSN	Position a file to a logical record
X'4445'	@BKSP	Backspace one logical record
X'4448'	@PEOF	Position to the end-of-file
X'444B'	@FEXT	Set up a default file extension
X'444E'	@MULT	Multiply HL by A
X'4451'	@DIV	Divide HL by A
X'4454'	@PARAM	Parse an optional parameter string
X'4458'	@CKEOF	Check for end of file
X'445B'	@WEOF	Write end-of-file
X'445E'	@RREAD	Reread the current sector
X'4461'	@RWRIT	Rewrite the current sector
X'4464'	@SKIP	Skip the next record
X'4467'	@DSPLY	Display a message line
X'446A'	@PRINT	Print a message line
X'446D'	@LOC	Calculate the current logical record number
X'447Ø'	@LOF	Calculate the EOF logical record number
X'4473'	INTIM$	Contains an image of the interrupt latch
X'4475'	INTVC$	This area contains eight vectors - one for each bit of the interrupt latch
X'4485'-X'44A4'	CFCB$	File control block for commands
X'45ØØ'-X'4517'	TCB$	Interrupt Task Table
X'47ØØ'-X'474F'	DCT$	Area reserved for the Drive Code Table
X'4754'	SELECT	Select new drive
X'4759'	RSELCT	Test if requested drive is busy
X'475E'	SEEK	Seek a cylinder
X'4763'	WRSECT	Write sector
X'4768'	WRPROT	Write system sector
X'476D'	WRTRK	Write a cylinder
X'4772'	VERSEC	Verify a sector
X'4777'	RDSECT	Read a sector
X'478F'	GETDCT	Get Drive Code Table address
X'479C'	DCTBYT	Get a DCT field
X'4B1Ø'	DIRRD	Directory record read
X'4B1F'	DIRWR	Directory record write
X'4B45'	RDSSEC	Read a SYSTEM sector
X'4B64'	DIRCYL	Get the directory cylinder number
X'4B6B'	MULTEA	Multiply E by A
X'4B7A'	DIVEA	Divide E by A
X'4DFE'-X'4DFF'	USTOR$	Points to an 8 byte user storage area
X'4EØØ'-X'51FF'		LDOS Overlay Area

MEMORY MAP - ALPHABETIC LISTING

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

```
@ABORT----<4Ø3Ø>,[4Ø3Ø]     @PAUSE----<ØØ6Ø>,[ØØ6Ø]     JFCB$-----<4358>,[4265]*
@ADTSK----<441Ø>,[4Ø3D]*    @PEOF-----<4448>,[4448]     JLDCB$----<43CØ>,[42C2]*
@BKSP-----<4445>,[4445]     @POSN-----<4442>,[4442]     JRET$-----<43ØC>,[4222]*
@CKDRV----<44B8>,[42Ø9]*    @PRINT----<446A>,[446A]     KFLAG$----<4423>,[429F]*
@CKEOF----<444B>,[4458]*    @PRT------<ØØ3B>,[ØØ3B]     KIDCB$----<4Ø15>,[4Ø15]
@CLOSE----<4428>,[4428]     @PUT------<ØØ1B>,[ØØ1B]     KIJCL$----<43BE>,[42BE]*
@CMD------<44ØØ>,[4296]*    @RAMDIR---<4396>,[429Ø]*    KISV$-----<43B8>,[42B8]*
@CMNDI----<44Ø5>,[4299]*    @READ-----<4436>,[4436]     LDRV$-----<43Ø8>,[4427]*
@CTL------<ØØ23>,[ØØ23]     @REW------<443F>,[443F]     MFLAG$----<442F>,[ N/A]*
@DATE-----<447Ø>,[3Ø33]*    @RMTSK----<4413>,[4Ø4Ø]*    MULTEA----<4B6C>,[4B6B]*
@DEBUG----<44ØD>,[44ØD]     @RPTSK----<4416>,[4Ø43]*    OSVER$----<4Ø3E>,[441F]*
@DIV------<44C4>,[4451]*    @RREAD----<4454>,[445E]*    OVRLY$----<43ØE>,[4414]*
@DODIR----<4463>,[4419]*    @RUN------<4433>,[4433]     PDRV$-----<43Ø9>,[4423]*
@DOKEY----<44BE>,[42B5]*    @RWRIT----<4457>,[4461]*    PRDCB$----<4Ø25>,[4Ø25]
@DSP------<ØØ33>,[ØØ33]     @SKIP-----<446Ø>,[4464]*    PRSV$-----<43BC>,[42BC]*
@DSPLY----<4467>,[4467]     @TIME-----<446D>,[3Ø36]*    RDSECT----<4777>,[4777]
@DVRHK----<4Ø33>,[4Ø33]     @VER------<443C>,[443C]     RDSSEC----<4B45>,[4B45]
@ERROR----<44Ø9>,[44Ø9]     @WEOF-----<444E>,[445B]*    RSELCT----<4759>,[4759]
@EXIT-----<4Ø2D>,[4Ø2D]     @WHERE----<ØØØB>,[ØØØB]     S1DCB$----<43D8>,[42D4]*
@FEXT-----<4473>,[444B]*    @WRITE----<4439>,[4439]     S2DCB$----<43EØ>,[42DA]*
@FNAME----<44BB>,[4293]*    CFCB$-----<448Ø>,[4485]*    S3DCB$----<43E8>,[42EØ]*
@FSPEC----<441C>,[441C]     DATE$-----<4Ø44>,[421A]*    S4DCB$----<43FØ>,[42E6]*
@GET------<ØØ13>,[ØØ13]     DAY$------<4Ø47>,[4417]*    S5DCB$----<43F8>,[ N/A]*
@ICNFG----<43Ø3>,[421D]*    DBGSV$----<4Ø5D>,[4Ø5D]     SBUFF$----<42ØØ>,[43ØØ]*
@INIT-----<442Ø>,[442Ø]     DCT$------<47ØØ>,[47ØØ]     SEEK------<475E>,[475E]
@KBD------<ØØ2B>,[ØØ2B]     DCTBYT----<479C>,[479C]     SELECT----<4754>,[4754]
@KEY------<ØØ49>,[ØØ49]     DFLAG$----<441F>,[4289]*    SFCB$-----<44AØ>,[42A1]*
@KEYIN----<ØØ4Ø>,[ØØ4Ø]     DIRCYL----<4B65>,[4B64]*    SFLAG$----<43ØF>,[442B]*
@KILL-----<442C>,[442C]     DIRRD-----<4B1Ø>,[4B1Ø]     SIDCB$----<43C8>,[42C8]*
@KITSK----<43ØØ>,[4285]*    DIRWR-----<4B1F>,[4B1F]     SODCB$----<43DØ>,[42CE]*
@KLTSK----<4419>,[4Ø46]*    DIVEA-----<4B7B>,[4B7A]*    SVDAT1$---<43Ø6>,[442F]*
@LOAD-----<443Ø>,[443Ø]     DODCB$----<4Ø1D>,[4Ø1D]     SVDAT2$---<43Ø7>,[4457]*
@LOC------<445A>,[446D]*    DOSV$-----<43BA>,[42BA]*    TCB$------<45ØØ>,[45ØØ]
@LOF------<445D>,[447Ø]*    GETDCT----<478F>,[478F]     TIME$-----<4Ø41>,[4217]*
@LOGER----<447E>,[428D]*    HIGH$-----<4Ø49>,[4411]*    TIMER$----<4Ø4Ø>,[4288]*
@LOGOT----<447B>,[428A]*    INBUF$----<4318>,[4225]*    USTOR$----<4DFE>,[4DFE]
@MSG------<4479>,[44Ø2]*    INTIM$----<4Ø4B>,[4473]*    VERSEC----<4772>,[4772]
@MULT-----<44C1>,[444E]*    INTMSK$---<4Ø4C>,[4474]*    WRPROT----<4768>,[4768]
@OPEN-----<4424>,[4424]     INTVC$----<4Ø4D>,[4475]*    WRSECT----<4763>,[4763]
@PARAM----<4476>,[4454]*    JDCB$-----<43ØA>,[422Ø]*    WRTRK-----<476D>,[476D]
```

You will find a file called EQUATE1/EQU (Model I) or EQUATE3/EQU (Model III) on your LDOS Master disk. This is a file consisting of the above labels as EQU statements for use with an editor/assembler program. The file is in the format used by the EDAS editor/assembler used to create LDOS.

RAM STORAGE ASSIGNMENTS

All addresses are in hexadecimal, and are indicated <MOD1> and [MOD3].

I/O Control Blocks

DCT$ <47ØØ-474F>,[47ØØ-474F]

Area reserved for the drive code table. Each drive occupies ten table bytes. Specific data on each 1Ø-byte area is discussed in the Technical section entitled Drive Code Table.

DODCB$ <4Ø1D-4Ø24>,[4Ø1D-4Ø24]

Video Display Device Control Block

DCB+Ø .... Device type
DCB+1 .... Driver address, low-order
DCB+2 .... Driver address, high-order
DCB+3 .... Cursor position, low-order
DCB+4 .... Cursor position, high-order
DCB+5 .... Character at cursor position, if any
*DCB+6/7 . Device name DO <Model I>
*DCB+6 ... Cursor character [Model III]
*DCB+7 ... System use [Model III]

KIDCB$ <4Ø15-4Ø1C>,[4Ø15-4Ø1C]

Keyboard Device Control Block

DCB+Ø .... Device type
DCB+1 .... Driver address, low-order
DCB+2 .... Driver address, high-order
DCB+3 .... System use
DCB+4 .... System use
DCB+5 .... System use
*DCB+6/7 . Device name KI <Model I>
*DCB+6/7 . Cursor blink switch, Ø=solid, 1=blink [Model III]

PRDCB$ <4Ø25-4Ø2C>,[4Ø25-4Ø2C]

Printer Device Control Block

DCB+Ø .... Device type
DCB+1 .... Driver address, low-order
DCB+2 .... Driver address, high-order
DCB+3 .... Physical maximum of lines per page
DCB+4 .... Counter of lines printed on current page
DCB+5 .... May contain lines to print
*DCB+6/7 . Device name PR <Model I>
*DCB+6/7 . System use [Model III]

JLDCB$ <43CØ-42C7>,[42C2-42C7]

Joblog Device Control Block

DCB+Ø .... Device type
DCB+1 .... Driver address, low-order
DCB+2 .... Driver address, high-order
DCB+3 .... Unused
DCB+4 .... Unused
DCB+5 .... Unused
DCB+6/7 .. Device name JL <Model I>

SIDCB$ <43C8-43CF>,[42C8-42CD]

Standard Input Device Control Block

DCB+Ø .... Device type
DCB+1 .... Driver address, low-order
DCB+2 .... Driver address, high-order
DCB+3 .... Unused
DCB+4 .... Unused
DCB+5 .... Unused
DCB+6/7 .. Device name SI <Model I>

SODCB$ <43DØ-43D7>,[42CE-42D3]

Standard Output Device Control Block

DCB+Ø .... Device type
DCB+1 .... Driver address, low-order
DCB+2 .... Driver address, high-order
DCB+3 .... Unused
DCB+4 .... Unused
DCB+5 .... Unused
DCB+6/7 .. Device name SO <Model I>

S1DCB$ <43D8-43DF>,[42D4-42D9]

First spare Device Control Block

S2DCB$ <43EØ-43E7>,[42DA-42DF]

Second spare Device Control Block

S3DCB$ <43E8-43EF>,[42EØ-42E5]

Third spare Device Control Block

S4DCB$ <43FØ-43F7>,[42E6-42EB]

Fourth spare Device Control Block

S5DCB$ <43F8-43FF>,[N/A]

Fifth spare Device Control Block

System Control Information

DATE$ <4Ø44-4Ø46>,[421A-421E]

Contains the current date

DATE$+Ø .... Contains the two-digit year
DATE$+1 .... Contains the day of the month
DATE$+2 .... Contains the month

DAY$ <4Ø47-4Ø48>,[4417-4418]

DAY$+Ø ..... Contains bits Ø-7 of the day of the year
DAY$+1
- Bit Ø ..... Contains bit 8 of the day of the year
- Bits 1-3 .... Contain the day of the week
- Bits 4-6 .... Reserved
- Bit 7 ..... Set to "1" if leap year

DOSV$ <43BA-43BB>,[42BA-42BB]

Save area for *DO DCB jump vector address.

EXDBG$ <431Ø-4311>,[44ØØ-44Ø1]

Pointer to location in high memory of the extended DEBUGger.

HIGH$ <4Ø49-4Ø4A>,[4411-4412]

Contains the highest unused RAM address

JDCB$ <43ØA-43ØB>,[422Ø-4221]

Storage area for DCB Address during JCL execution

JRET$ <43ØC-43ØD>,[4222-4223]

Storage area for RET Address during JCL execution

KISV$ <43B8-43B9>,[42B8-42B9]

Save area for *KI DCB jump vector address

KIJCL$ <43BE-43BF>,[42BE-42BF]

Save area for KIJCL DCB jump vector address

LDRV$ <43Ø8>,[4427]

Currently accessed drive - logical number (Ø-7)

OSVER$ <4Ø3E>,[441F]

Contains the operating system version Number

OVRLY$ <43ØE>,[4414]

Contains the LDOS overlay currently resident in the overlay region.

PDRV$ <43Ø9>,[4423]

Currently accessed drive - physical drive address (1, 2, 4, or 8)

PRSV$ <43BC-43BD>,[42BC-42BD]

Save area for *PR DCB jump vector address

TIMER$ <4Ø4Ø>,[4288]

This is the heartbeat counter, <25ms Model 1>, [33.33ms Model III]

TIME$ <4Ø41-4Ø43>,[4217-4219]

Contains the time-of-day

TIME$+Ø .... Contains the seconds
TIME$+1 .... Contains the minutes
TIME$+2 .... Contains the hours

Interrupt Processor Task Vector Storage

INTIM$ <4Ø4B>,[4473]

Contains an image of the interrupt latch

INTVC$ <4Ø4D-4Ø5C>,[4475-4484]

This area contains eight vectors - one for each bit of the interrupt latch.

INTVC$+Ø .... Vector for latch bit Ø
INTVC$+2 .... Vector for latch bit 1
INTVC$+4 .... Vector for latch bit 2
INTVC$+6 .... Vector for latch bit 3
INTVC$+8 ... Vector for latch bit 4
INTVC$+1Ø ... Vector for latch bit 5
INTVC$+12 ... Vector for latch bit 6
INTVC$+14 ... Vector for latch bit 7

TCB$ <45ØØ-4517>,[45ØØ-4517]

This area contains the vector addresses for each of the twelve possible interrupt processor tasks executed by the real-time-clock assigned to INTVC$, Bit 7. Task slots zero through seven are executed at <2ØØ> [267.67] millisecond intervals and are considered "low-priority" tasks. Task slots eight through eleven are executed at <25> [33.33] millisecond intervals and are considered "high-priority" tasks.

TCB$+Ø .... Task slot Ø, currently unassigned.

TCB$+2 .... Task slot 1, currently unassigned.

TCB$+4 .... Task slot 2, currently unassigned.

TCB$+6 .... Task slot 3, assigned to the ALIVE function.

TCB$+8 .... Task slot 4, assigned to the screen print function <Model I>.
[unassigned on Model III]

TCB$+1Ø .... Task slot 5, assigned to the BLINK function <Model I>.
[unassigned on Model III]

TCB$+12 .... Task slot 6, assigned to the CLOCK function <Model I>.
[unassigned on Model III]

TCB$+14 .... Task slot 7, assigned to the TIMER function <Model I>.
[unassigned on Model III]

TCB$+16 .... Task slot 8, assigned to the LCOMM Communications Line scanning function.

TCB$+18 .... Task slot 9, assigned to the SPOOLer function and to the LCOMM printer despooling function.

TCB$+2Ø .... Task slot 1Ø, assigned to the TYPE ahead function.

TCB$+22 .... Task slot 11, assigned to the TRACE function.

System Buffers

CFCB$ <448Ø-449F>,[4485-44A4]

File control block buffer area used during @CMD interpreting.

DBGSV$ <4Ø5D-4Ø7C>,[4Ø5D-4Ø7C]

Area used during DEBUG operation as a register save area and pointer save area. During non-DEBUG operation, this area is used by the SYSTEM and must not be disturbed.

INBUF$ <4318-4357>,[4225-4264]

Buffer area of 64 bytes for user command input. It contains the last command input by the user.

JFCB$ <4358-4377>,[4265-4284]

Buffer area of 32 bytes for SYSTEM/JCL file control block during DO processing

SBUFF$ <42ØØ-42FF>,[43ØØ-43FF]

A 256-byte buffer for system disk I/O.

SFCB$ <44AØ-44B3>,<42A1-42B4]

A 2Ø-byte file control block used for loading system overlays

System Flags

DFLAG$ <441F>,[4289]

Bit Ø .... Set to "1" if SPOOL is active.
Bit 1 .... Set to "1" if TYPE ahead is active.
Bit 2 .... Set to "1" if JKL (screen print) is active.
Bit 3 .... Set to "1" if PR/FLT is active.
Bit 4 .... Set to "1" if KI/DVR is active.
Bit 5 .... Set to "1" if MiniDOS/FLT is active.
Bit 6 .... Set to "1" if KSM/FLT is active.
Bit 7 .... Set to "1" if GRAPHIC is on.

KFLAG$ <4423>,[429F]

Bit Ø .... Set to "1" if <BREAK> pressed.
Bit 1 .... Set to "1" if <PAUSE> pressed.
Bit 2 .... Set to "1" if <ENTER> pressed.
Bit 3 .... currently unused.
Bit 4 .... currently unused.
Bit 5 .... Set to "1" if in CAPS LOCK mode
Bit 6 .... Set to "1" if in ECM (Extended Cursor Mode).
Bit 7 .... Set to "1" if a character is in the type ahead buffer.

MFLAG$ <442F>,[N/A]

Bit Ø .... Set to "1" if LX-8Ø interface, reset if not.
Bit 1 .... currently unused.
Bit 2 .... currently unused.
Bit 3 .... currently unused.
Bit 4 .... currently unused.
Bit 5 .... currently unused.
Bit 6 .... currently unused.
Bit 7 .... currently unused.

SFLAG$ <43ØF>,[442B]

Bit Ø .... Set to a "1" if SVC table active
Bit 1 .... Set to "1" if running an EXEC only file
Bit 2 .... Set to a "1" if LOAD called from RUN
Bit 3 .... "1" if SYSTEM (FAST), "Ø" if SYSTEM (SLOW)
Bit 4 .... "1" if <BREAK> disabled, "Ø" if <BREAK> enabled
Bit 5 .... "1" if DO is in effect, "Ø" if DO is not in effect
Bit 6 .... If set to "1", will force extended error messages
Bit 7 .... "1" if DEBUG is to be turned on, "Ø" if it is to remain off

SYSTEM ENTRY POINTS

NOTE: All addresses in this section are in hexadecimal. The notation <nnnn> will indicate Model I, and [nnnn] Model III. The associated SVC number, IN DECIMAL, will be indicated after the addresses as {dd} wherever applicable.

THIS SECTION IS NOT A TUTORIAL ON OPERATING SYSTEM STRUCTURE OR ASSEMBLY LANGUAGE PROGRAMMING. The system entry points identified in this section are provided for use by skilled assembly language programmers. There is absolutely no attempt to provide an in-depth tutorial on how to write assembly language programs and routines using these entry points. Although much technical information is being provided in this reference manual, it does not replace other reference books covering the subject matter.

The information provided in this section will probably not be useful to the novice programmer. You will cause considerable trouble for yourself if you try to use it without FULLY understanding it. The experienced application programmer will find this an invaluable reference section. Since this information deals with specific functions of LDOS, every effort to verify the accuracy of the data has been undertaken. However, it is strongly suggested that you fully test your coding to be sure that the correct results are being produced prior to working with sensitive data.

This may very well be the first time that all of these system entry points have been identified by an "official source". You will also find the documentation of a number of routines located in the ROM of the TRS-8Ø. We hope you find them informative.

A word of caution. Other operating systems designed to run on the TRS-8Ø may not support all of the system entry points identified herein. If you want to assure yourself that your application is compatible, you may have to contact the other suppliers of operating systems for their technical data. However, you may want to make extensive use of this operating system so as to create sophisticated applications by utilizing the enhancements in LDOS.

Program Entry Conditions

Upon entering a program from the LDOS Ready prompt, certain conditions will be established. The address X'4Ø2D' will be on the top of the stack. Register pair HL will point into the command buffer INBUF$, and will be used in one of two ways to indicate the character that terminated the parsing of the program name. The termination of the program name parsing will occur on any character that is not a vaild part of a filespec, as defined in the Glossary in Section 7 of this User's manual. If the character was a carriage return (X'ØD') or an open parenthesis (X'28'), then HL will point to that character. Otherwise, HL will point 1 byte past the character.

DISK I/O PRIMITIVES

The system vectors contained in this section interface to the disk I/O driver. The address of the driver is contained in the Drive Code Table and can vary from drive to drive depending on your particular installation. Anyone having need to utilize these primitives must be thoroughly familiar with the function performed by each primitive. A discussion on this topic is beyond the scope of this manual. It is recommended that you obtain appropriate reference manuals relating to the particular controller in use for your disk system. If your hardware is capable of double density operation, you most likely are using an FDC in the 179X series. These primitives are identified herein for such time as you have the knowledge and experience to utilize them.

RDSECT .... Vector = <4777>,[4777] {49}

This entry will transfer a sector of data from the disk to your buffer.

HL => pointer to the buffer to receive the sector
D => cylinder to read
E => sector to read
C => the logical drive number
A <= error return code
Z <= set if no error

RDSSEC .... Vector = <4B45>,[4B45] {85}

This entry will read the system (directory) sector identified by the calling linkage. If the cylinder number in D is not the directory cylinder, the value in D will be changed to reflect the real directory cylinder, and then the sector will be read.

HL => pointer to the buffer to receive the sector.
D => cylinder to read
E => sector to read
C => logical drive number
A <= error return code
Z <= set if no error

RSELCT .... Vector = <4759>,[4759] {47}

This entry will perform a test of the last selected drive to see if it is in a busy state. If busy, it will be re-selected until it is no longer busy.

C => should contain the logical drive number

SEEK .... Vector = <475E>,[475E] {46}

This entry is used to seek a specified cylinder. Seek will not return an error if you specified a non-existent or disabled drive. Also, Seek performs no action if the specified drive is a hard disk.

C => logical drive number
D => cylinder requested

SELECT .... Vector = <4754>,[4754] {41}

This vector will select a drive. LDRV$ and PDRV$ are updated. The appropriate time delay specified in your configuration (SYSTEM (DELAY=Y/N)) will be undertaken if the drive selection requires it.

C => logical drive number (Ø-7)
A <= error return code
Z <= set if no error

VERSEC .... Vector = <4772>,[4772] {5Ø}

This entry will verify a sector without transferring any data from disk to the buffer.

D => cylinder to verify
E => sector to verify
C => the logical drive number
A <= error return code
Z <= set if no error

WRPROT .... Vector = <4768>,[4768] {54}

This entry will write a system sector (used in directory cylinder).

HL => buffer containing the sector of data
D => cylinder to write
E => sector to write
C => the logical drive number
A <= error return code
Z <= set if no error

WRSECT .... Vector = <4763>,[4763] {53}

This entry will write a sector to the disk

HL => buffer containing the sector of data
D => cylinder to write
E => sector to write
C => the logical drive number
A <= error return code
Z <= Set if no error

WRTRK .... Vector = <476D>,[476D] {55}

This entry is used to write an entire cylinder of properly formatted data. The data format must conform to that identified in your controller's reference manual.

HL => pointer to format data
D => cylinder to write
C => the logical drive number
A <= error return code
Z <= set if no error

Disk file handler routines

In general, only the flag register and the accumulator are disturbed unless a register is used to return data requested by the system call. All file access routines are accessed by a CALL instruction.

@BKSP .... Vector = <4445>,[4445] {61}

This routine will perform a backspace of one logical record. It should NOT be used when already at the start of a file.

```
DE => FCB of the file to backspace
 A <= Error return code
 Z <= Set if the operation was successful
```

@CKEOF .... Vector = <444B>,[4458] {62}

Check for the end-of-file at the current logical record number.

```
DE => FCB for the file to check
 A <= Error return code
 Z <= Set if not at the end of file
```

@LOC .... Vector = <445A>,[446D] {63}

Calculate the current logical record number.

```
DE => FCB of the file
BC <= the logical record number
 A <= Error return code
 Z <= Set if the operation was successful
```

@LOF .... Vector = <445D>,[447Ø] {64}

Calculate the EOF logical record number.

```
DE => FCB for the file to check
BC <= the logical record number
 A <= Error return code
 Z <= Set if the operation was successful
```

@PEOF .... Vector = <4448>,[4448] {65}

This routine will position an open file to the ERN. An end-of-file-encountered error (X'1C') will be returned. Your program may ignore this error.

```
DE => FCB of the file to position
 A <= Error return X'1C' if sucessful
```

@POSN .... Vector = <4442>,[4442] {66}

Position a file to a logical record. This will be useful for positioning to records of a random access file. When the @POSN routine is used, Bit 6 of FCB+1 is automatically set to ensure that the EOF will be updated when the file is closed only if the NRN exceeds the current ERN.

```
DE => FCB for the file to position
BC => the logical record number
 A <= Error return code
 Z <= Set if the operation was successful
```

@READ .... Vector = <4436>,[4436] {67}

Read a logical record from a file. If the LRL defined at open time was 256 (Ø), then the NRN sector will be transferred to the buffer established at open time. For LRL between 1 and 255, the next logical record will be placed into the user record buffer, UREC. The 3-byte NRN is updated after the read operation.

```
DE => FCB for the file to read
HL => UREC (needed if LRL <> Ø) - (unused if LRL=256)
 A <= Error return code
 Z <= Set if the operation was successful
```

@REW .... Vector = <443F>,[443F] {68}

This routine will rewind a file to its beginning and reset the 3-byte NRN to Ø. The next record to I/O will be the first record of the file.

```
DE => FCB for the file to rewind
 A <= Error return code
 Z <= Set if the operation was successful
```

@RREAD .... Vector = <4454>,[445E] {69}

This routine will force a reread of the current sector to occur before the next I/O request is serviced. Its most probable use would be in applications that reuse the disk I/O buffer for multiple files to make sure that the buffer contains the proper file sector. This routine is only valid for byte I/O or blocked files. It should NOT be used when positioned at the start of a file.

```
DE => FCB for the file to reread
 A <= Error return code
 Z <= Set if the operation was successful
```

@RWRIT .... Vector = <4457>,[4461] {7Ø}

This routine will rewrite the current sector following a write operation. The @WRITE function advances NRN after the sector is written. @RWRIT will decrement the NRN and write the disk buffer again. It should NOT be used when positioned to the start of a file.

```
DE => FCB for the file to rewrite
 A <= Error return code
 Z <= Set if the operation was successful
```

@SKIP .... Vector = <446Ø>,[4464] {72}

This routine will cause a skip past the next logical record. Only the record number contained in the FCB will be changed. No physical I/O will take place.

```
DE => FCB for the file to skip
 A <= Error return code
 Z <= Set if the operation was successful
BC is used
```

@VER .... Vector = <443C>,[443C] {73}

This routine performs a @WRITE operation followed by a test read of the sector (assuming that the WRITE required physical I/O) to verify that it will be readable.

```
DE => FCB for the file to verify
HL <= user buffer containing the logical record
 A <= Error return code
 Z <= Set if the operation was successful
```

@WEOF .... Vector = <444E>,[445B] {74}

This routine will force the system to update the directory entry with the current end-of-file information.

```
DE => FCB for the file to WEOF
 A <= Error return code
 Z <= Set if the operation was successful
```

@WRITE .... Vector = <4439>,[4439] {75}

This routine will cause a write to the next record identified in the FCB. If LRL is less than 256, then the logical record in the user buffer will be transferred to the file. If LRL is equal to 256, a full sector write will be made using the disk I/O buffer identified at file open time.

```
HL => UREC - the user record buffer - (unused if LRL=256)
DE => FCB for the file to write
 A <= Error return code
 Z <= Set if the operation was successful
```

File Control Routines

@CLOSE .... Vector = <4428>,[4428] {6Ø}

This routine will close a file or device. When a file is closed, the directory is updated which is essential. All files that have been written to must be closed.

```
DE => File or Device Control Block
 A <= Error return code
 Z <= Set if no error existed
```

@FEXT .... Vector = <4473>,[444B] {79}

This routine will insert a default file extension into the FCB if the file specification entered contains no extension. This must be done before the file is opened.

```
DE => File Control Block
HL => Pointer to default extension (3 characters)
```

@FNAME .... Vector = <44BB>,[4293] {8Ø}

Recover the file name & extension from the directory.

```
DE => Buffer to receive file name/ext
 B => DEC of file desired
 C => drive number of drive containing the file
 Z <= set if no error
```

@FSPEC .... Vector = <441C>,[441C] {78}

This routine will move a file or device specification from an input buffer into the FCB. Conversion of lower case to upper case will be made.

```
HL => buffer containing file specification
DE => 32-byte File Control Block
HL <= points to the terminating character found
DE <= points to the beginning of FCB
 A <= the terminating character
 Z <= set if valid file specification found
```

@INIT .... Vector = <442Ø>,[442Ø] {58}

INIT will open a file, creating it according to the file specification if it is not found.

```
HL => 256-byte disk I/O buffer
DE => File Control Block containing the file specification
 B => Logical Record Length to be used while the file is open.
HL <= Returns unchanged
DE <= Returns unchanged
 B <= Returns unchanged
 A <= Error return code
CF <= Set if a new file was created
 Z <= Set if no error detected
```

@KILL .... Vector = <442C>,[442C] {57}

This routine will kill a file or device. If a file is to be killed, the FCB must be in an open condition. The file's directory will be updated and the space occupied by the file will be deallocated. The file name, extension, and drive number will be placed back into the FCB. If a device, it will be removed from the device control block tables if it is not a system device. The device should first be RESET.

DE => File Control Block
A <= error code returned
Z <= set if no error detected

@OPEN .... Vector = <4424>,[4424] {59}

This routine will open an existing file or device.

HL => Buffer for disk I/O (256 bytes)
DE => File control block containing the filespec
B => Logical record length for the open file
HL <= Returns unchanged
DE <= Returns unchanged
B <= Returns unchanged
A <= Error return code
Z <= Set if open was successful

Special Purpose Disk Routines

DCTBYT .... Vector = <479C>,[479C]

This entry will recover a byte field from the drive code table.

```
C => logical drive number (Ø-7)
A => relative byte desired in the table (3-9)
A <= content of the DCT field requested
```

DIRCYL .... Vector = <4B65>,[4B64] {83}

This entry will recover the cylinder number containing the directory for the requested drive. It is identical to calling the DCTBYT entry with an argument of 9 (A=9). To assure picking up the proper directory cylinder, @CKDRV should be called first.

```
C => logical drive number (Ø-7)
D <= returns the cylinder number of the directory
```

DIRRD .... Vector = <4B1Ø>,[4B1Ø] {87}

This entry will read a directory sector containing the directory entry for a specified Directory Entry Code (DEC). The sector will be written to the system buffer, SBUF$, and the register pair HL will point to the first byte of the directory entry specified by the DEC.

```
B => Directory Entry Code of the file
C => Logical drive number (Ø-7)
HL <= points to the DEC's directory entry
A <= error return code
Z <= set if no error
```

DIRWR .... Vector = <4B1F>,[4B1F] {88}

This entry will write the system buffer, SBUF$, back to the disk directory sector that contains the directory entry of the DEC specified in the calling linkage.

```
B => Directory Entry Code of the file
C => Logical drive number (Ø-7)
A <= error return code
Z <= set if no error
```

GETDCT .... Vector = <478F>,[478F] {81}

This routine will obtain the address of the drive code table for the requested drive.

```
C => logical drive number (Ø-7)
IY <= the DCT+Ø address of the requested drive
```

System Control Routines

@ABORT .... Vector = <4Ø3Ø>,[4Ø3Ø] {21}

This jump entry will cause an abnormal program exit and return to LDOS. Any JCL execution in progress will cease.

@CMD .... Vector = <44ØØ>,[4296] {23}

This jump vector will initiate a normal return to LDOS and accept a new command. It is identical to @EXIT.

@CMNDI .... Vector = <44Ø5>,[4299] {24}

This jump vector performs an entry to the command interpreter. Your "command" line will be interpreted and executed just as if it was entered in response to an "LDOS Ready".

HL => points to the start of a line buffer containing your command string terminated with an <ENTER> (X'ØD').

If the command to be executed is an LDOS library command, utility, or any other routine that normally exits to the @EXIT or @ABORT vector, you may force a return to your program in the following manner. Create an 8 byte storage area in your program to save the 2 byte Stack Pointer value, and the 3 byte jump vectors at 4Ø2D (@EXIT) and 4Ø3Ø (@ABORT). Place the 3 byte return vectors to your program at 4Ø2D and 4Ø3Ø. Now point HL at your command string and jump to @CMNDI. Upon returning to your program, restore the original vectors and the Stack Pointer.

@EXIT .... Vector = <4Ø2D>,[4Ø2D] {22}

This is the normal "jump" vector to perform a program exit and return to LDOS.

@LOAD .... Vector = <443Ø>,[443Ø] {76}

This routine will load a program file (a file in load module format).

DE => FCB containing the filespec of the file to load.
A <= error return code
Z <= set if load was successful
HL <= transfer address retrieved from file

@RUN .... Vector = <4433>,[4433] {77}

This routine will load and execute a program file. If any errors are encountered during the load, the system will jump to @ERROR, print the appropriate message, and terminate by jumping to @ABORT. If @RUN is entered by a CALL instruction, the stack will contain the return address upon entry to the program. If this is not needed, a JP instruction may be used to enter @RUN.

DE => FCB containing the filespec of the file to RUN.

Special overlay routines

@CKDRV .... Vector = <44B8>,[42Ø9] {33}

This routine will check a drive reference to ensure that the drive is in the system and a formatted diskette is in place.

```
C => Logical drive number
Z <= If drive is ready.
NZ <= If drive is not ready
CF <= Set if disk is write protected.
```

@DEBUG .... Vector = <44ØD>,[44ØD] {27}

This call vector will force the system to enter the DEBUGging package. A "G" command from the DEBUGger will continue program execution with the next instruction.

@DODIR .... Vector = <4463>,[4419] {34}

This routine will read visible files from a disk directory, or find the free space on a disk. The display to the screen will be in a 4 across format. The directory information buffer will consist of 18 bytes per active, visible file - the first 16 bytes of the directory record, plus the ERN. An X'FF' will mark the buffer end.

```
C => Logical drive number. (THIS REGISTER USED IN ALL EXAMPLES).

B => If Ø, will display the directory to *DO, if 1 will send the directory
     to a buffer.
HL => Buffer to receive directory.

B => If 2, will display the directory to *DO. If 3, will send the directory
     to a buffer.
HL => 3 character extension. Any of the extension characters not specified
      must be replaced with a $. If B = 3, then HL also points to the buffer.

B => If 4, will get free space on the disk.
HL => 2Ø character buffer. The information will be:

      bytes  1-16 = disk name and date
      bytes 17-18 = Total K originally available
      bytes 19-2Ø = Free K available
```

@ERROR .... Vector = <44Ø9>,[44Ø9] {26}

This vector will provide an entry to post an error message. Two options exist. @ERROR will normally terminate to the @ABORT function. If bit 7 of the accumulator is SET, the error message will be displayed and return will be made to the calling program. The second option will provide extended or abbreviated error messages. If bit 6 is not set, the complete error display is:

```
*** Errcod=xx, Error Message String ***
      <filespec or devicespec>
   Referenced at X'dddd'
```

whereas if bit 6 is set, then only the "Error message string" is displayed.

```
A => Error number with bits 6 & 7 optionally set.
```

@PARAM .... Vector = <4476>,[4454] {17}

This routine can be used to parse an optional parameter string. Its primary function is to parse command parameters contained in a command line totally enclosed within parentheses. Acceptable parameter format is:

```
PARM=X'dddd' .... hexadecimal entry
PARM=ddddd   .... decimal entry
PARM="string" ... alphanumeric entry
PARM= flag   .... ON, OFF, Y, N, YES or NO
```

The 2-byte memory area pointed to by the address field of your table receives the value of PARM if PARM is non-string. If a string is entered, the 2-byte memory area receives the address of the 1st byte of "string". The entries ON, YES and Y return a value of X'FFFF' while OFF, NO and N will return a X'ØØØØ'. If a parameter name is specified on the command line followed by an equal sign and no value then a X'ØØØØ' or NO will be returned. If a parameter name is used on the command line without the equal then a value of X'FFFF' or ON will be assumed. For any allowed parameter that is completely omitted on the command line, the 2 byte area will remain unchanged.

The valid parameters are contained in a user table which must be of the following format:

A 6-character "word" left justified and buffered by blanks followed by a 2-byte address vector to receive the parsed values. Word and vector may be repeated for as many parameters as are necessary. A byte of X'ØØ' must be placed at the end of the table to indicate its ending point.

```
DE => beginning of your parameter table.
HL => command line to parse
 Z <= Set if either no parameters found or valid parameters.
NZ <= If a bad parameter was found.
```

@RAMDIR .... Vector = <4396>,[429Ø]

This routine will read the directory information of visible files from a disk directory, or get the amount of free space on a disk.

```
HL => Ram buffer to receive information
 B => Drive number

 C => Ø - Gets directory records of all visible files
 C => 255 - Gets free space information
 C => 1-254 - Gets a single directory record (see below)

 A <= Error return code
 Z <= Set if no error
```

Each directory record will require 22 bytes of space in the buffer. If using option Ø (C=Ø), one additional byte will be needed to mark the end of the buffer. For single directory records, the number in the C register should be 1 less than the desired directory record. For example, if C=1, directory record 2 would be fetched and put in the buffer. If a single record request is for an inactive record or an invisible file, the A register will return an error code 25 (File Access Denied).

The directory information will be placed in the buffer as follows:

Byte	Contents
ØØ-14	filename/ext:d (left justified buffered w/spaces)
15	protection level, Ø to 6
16	EOF offset byte
17	Logical record length Ø to 255
18-19	ERN of file
2Ø-21	File size in K (1Ø24 byte blocks)
22	LAST RECORD ONLY. Contains "+" to mark buffer end.

If C=255, HL should point to a 4 byte buffer. Upon return, the buffer will contain:

Bytes ØØ-Ø1	Space in use in K, stored LSB, MSB
Bytes Ø2-Ø3	Space available in K, stored LSB, MSB

LDOS Task Control Vectors

@ADTSK .... Vector = <441Ø>,[4Ø3D] {29}

This routine will add an interrupt level task to the real time clock task table. The task slot can be Ø-11; however, some slots are already assigned to certain functions in LDOS. Slot assignments Ø-7 are low priority tasks executing every <2ØØ milliseconds Model I> or [267.67 milliseconds Model III], while slots 8-11 are high priority tasks executing every <25 milliseconds Model I> or [33.33 milliseconds Model III]. See the RAM STORAGE section for the slot assignments used by LDOS.

DE => Task Control Block (TCB)
A => Task slot assignment
HL is used

Note: The DE register is a pointer not to the location of your task driver, but to a two byte block of RAM called the TCB, which contains the address of the task driver entry point. Upon entry to your task routine, the register IX will contain the TCB address.

@KLTSK .... Vector = <4419>,[4Ø46] {32}

This routine, when called by an executing task driver, will remove the task assignment from the task table and return to the foreground application that was interrupted.

@RMTSK .... Vector = <4413>,[4Ø4Ø] {3Ø}

This routine will remove an interrupt level task from the task control block table.

A => task assignment slot (Ø-11) to remove
HL,DE are used

@RPTSK .... Vector = <4416>,[4Ø43] {31}

This routine will exit the task process executing and replace the currently executing task vector address in the task control block table with the address following the CALL instruction. Return is made to the foreground application that was interrupted.

Byte I/O primitives

@CTL .... Vector = <ØØ23>,[ØØ23] {5}

This routine will output a control byte to a logical device or a file. If a device control block is referenced, the TYPE byte must permit CTL operation.

DE => DCB or FCB to control output
A => Byte to output

@GET .... Vector = <ØØ13>,[ØØ13] {3}

This routine will fetch a byte from a logical device or a file.

DE => FCB
A <= Byte fetched or error return code if disk error
Z <= set if byte was fetched from disk without error

DE => DCB
A <= Byte fetched or Ø if no byte available.
Z <= Set if no byte available.

@PUT .... Vector = <ØØ1B>,[ØØ1B] {4}

This routine will output a byte to a logical device or a file.

DE => Device or File Control Block of the output device
A => the byte to output
A <= error return code if disk output
Z <= if output to disk without error

Keyboard I/O routines

@KBD .... Vector = <ØØ2B>,[ØØ2B] {8}

Scan the keyboard and return the keyboard character, if a key is pressed. If no key was pressed, a zero value will be returned.

A <= Contains the value of the key depressed
Z <= Set if no key depressed
DE is used

If KI/DVR is set, the following will hold true:

CF <= will be set if the control key (SHIFT-DOWN ARROW) was pressed.
MF <= high bit of returned byte will be set if (CLEAR) is pressed.

@KEY .... Vector = <ØØ49>,[ØØ49] {1}

This routine will scan the keyboard and return with the key depressed. It will not return until a key is depressed.

A <= the character entered
DE is used

@KEYIN .... Vector = <ØØ4Ø>,[ØØ4Ø] {9}

This routine will accept a line of input until terminated by either an <ENTER> or <BREAK>. During the input, the routine will display the entries. Backspace, tab, line delete, and 32 cpl mode are supported.

HL => user line buffer of length = B+1
B => maximum number of characters to input
HL <= points to buffer start
B <= the actual number of characters input
CF <= set if <BREAK> terminated the input
DE is used

Video and Printer I/O routines

@DSP .... Vector = <ØØ33>,[ØØ33] {2}

This routine will output a byte to the video display.

A => Byte to display
DE is used

@DSPLY .... Vector = <4467>,[4467] {1Ø}

This routine will display a message line. The line must be terminated with either an <ENTER> (X'ØD') or an ETX (X'Ø3'). If an ETX terminates the line, the cursor will be positioned immediately after the last character displayed.

HL => points to the 1st byte of your message.
HL is returned unchanged.
DE is used

@LOGER .... Vector = <447E>,[428D] {11}

Issue a log message to the Job Log. Message is any character string terminating with an <ENTER> (X'ØD').

HL => points to the first character of the message line.
HL is returned unchanged.

@LOGOT .... Vector = <447B>,[428A] {12}

Display and log a message. This will perform the same function as @DSPLY followed by @LOGER.

HL => points to the first character of the message line.
HL is returned unchanged.
DE is used

@MSG .... Vector = <4479>,[44Ø2] {13}

Message line handler to send a message to any device.

DE => Device or File Control Block to receive output
HL => pointer to the message line
HL is returned unchanged.

@PRT .... Vector = <ØØ3B>,[ØØ3B] {6}

This routine will output a byte to the printer. If a zero value is passed, then the printer status will be returned. This method is recommended over checking the port directly.

A => the character to print
A <= the printer status if a zero was output
DE is used

@PRINT .... Vector = <446A>,[446A] {14}

This routine will output a message line to the printer. The message must conform to the syntax specified under @DSPLY.

HL => Pointer to the message to be output
HL is returned unchanged
DE is used

Miscellaneous routines

ROM control routines

@PAUSE .... Vector = <ØØ6Ø>,[ØØ6Ø] {16}

This routine will suspend program execution and go into a "wait" state. The delay is approximately <14.67 microseconds> [14.796 microseconds] per count.

```
BC => delay count
 A register is used.
```

@WHERE .... Vector = <ØØØB>,[ØØØB] {7}

Vector used to resolve relocation address of calling routine.

```
HL <= the address following the CALL instruction
```

Time and Date routines

@DATE .... Vector = <447Ø>,[3Ø33] {18}

Get today's date in display format (XX/XX/XX)

```
HL => Buffer area to receive date string.
HL <= Points to end of buffer +1
DE, BC are used
```

@TIME .... Vector = <446D>,[3Ø36] {19}

Get the time of day in display format (XX:XX:XX)

```
HL => Buffer to receive the time string.
HL <= Points to end of buffer +1
DE,BC are used
```

Math routines

DIVEA .... Vector = <4B7B>,[4B7A] {93}

This routine performs an 8-bit unsigned integer divide.

```
E => dividend value
A => divisor value
A <= resultant value
E <= remainder
```

MULTEA .... Vector = <4B6C>,[4B6B] {9Ø}

This entry will perform an 8-bit by 8-bit unsigned integer multiplication. It is assumed that the result will not overflow a 8-bit field since the routine is only used on small integer values.

```
A => multiplicand value
E => multiplier value
A <= resultant value
D is used
```

@DIV ..... Vector = <44C4>,[4451] {94}

This routine will perform a division of a 16-bit unsigned integer by a 8-bit unsigned integer.

HL => dividend value
A => divisor value
HL <= resultant value
A <= remainder value

@MULT .... Vector = <44C1>,[444E] {91}

This routine will perform an unsigned integer multiplication of a 16-bit multiplicand by an 8-bit multiplier. The resultant value is stored in a 3-byte register field.

HL => multiplicand value
A => multiplier value
HL <= two high order bytes of resultant value
A <= low-order byte of the resultant value
DE is used

SUPERVISORY CALLS

The LDOS supervisory call table (SVC table) provides an alternative method for accessing system routines and obtaining information about system status within assembly language programs. Most of the services which LDOS provides to user programs can be requested by means of an SVC instead of a call to a fixed RAM address. This is intended to allow programs written for LDOS 5.1 to run on all future versions of LDOS without change, even if hardware or ROM differences between different computer models require system entry points and storage areas to be moved. An SVC is requested by loading the A register with the SVC number (in the range X'ØØ'-X'7F') and performing a RST 28H instruction. Depending on the function requested, the other registers may be used to pass parameters or return results. Generally speaking, any routine that expects a value to be passed in the A register should instead pass the value in the C register.

The following table lists the currently defined SVC numbers. More may be defined in future LDOS releases. Register usage is listed for those functions which differ from the corresponding direct call. Refer to the "Entry Points" section for more information on each call.

Please note that the SVC table is an optional feature in LDOS 5.1 (refer to the SYSTEM library command for SVC table installation instructions.) LDOS 5.1 library commands and utilities do not use SVCs. A program which is written to use SVCs will not run unless the SVC table is in place, and the first SVC call will cause an abort back to the "LDOS Ready" level with a SYS ERROR message.

Using the SVC table requires that KI/DVR be used. The system will not function properly if using the SVC table and the ROM keyboard driver.

DEC	HEX	LABEL	FUNCTION
Ø			Reserved for future use
1	1	@KEY	Scan keyboard, wait for character
2	2	@DSP	Display character at cursor, advance cursor Register <C> must contain character to display
3	3	@GET	Get one byte from a logical device
4	4	@PUT	Write one byte to a logical device Register <C> must contain the character to PUT
5	5	@CTL	Make a control request to a logical device Register <C> must contain the request
6	6	@PRT	Send character to the line printer Register <C> must contain the character to print
7	7	@WHERE	Locate origin of CALL
8	8	@KBD	Scan keyboard and return
9	9	@KEYIN	Accept a line of input
1Ø	ØA	@DSPLY	Display a message line
11	ØB	@LOGER	Issue a log message
12	ØC	@LOGOT	Display and log a message
13	ØD	@MSG	Message line handler
14	ØE	@PRINT	Print a message line
15			Reserved for future use
16	1Ø	@PAUSE	Suspend program execution
17	11	@PARAM	Parse an optional parameter string
18	12	@DATE	Get system date in the format MM/DD/YY
19	13	@TIME	Get system time in the format HH:MM:SS
2Ø			Reserved for future use
21	15	@ABORT	Abnormal program exit and return to LDOS
22	16	@EXIT	Normal program exit and return to LDOS
23	17	@CMD	Accept a new command
24	18	@CMNDI	Entry to command interpreter
25			Reserved for future use
26	1A	@ERROR	Entry to post an error message

27	1B	@DEBUG	Enter the debugging package
28			Reserved for future use
29	1D	@ADTSK	Add an interrupt level task <DE> contains the TCB address, <C> contains the task number.
30	1E	@RMTSK	Remove an interrupt level task <C> contains the task number
31	1F	@RPTSK	Replace the currently executing task vector
32	20	@KLTSK	Remove the currently executing task
33	21	@CKDRV	Check for drive availability
34	22	@DODIR	Do a directory display/buffer
35 - 40			Reserved for future use
41	29	SELECT	Select a new drive
42 - 45			Reserved for future use
46	2E	SEEK	Seek a cylinder
47	2F	RSELCT	Test if requested drive is busy
48			Reserved for future use
49	31	RDSECT	Read a sector
50	32	VERSEC	Verify a sector
51 - 52			Reserved for future use
53	35	WRSECT	Write a sector
54	36	WRPROT	Write a system sector
55	37	WRTRK	Write a cylinder
56			Reserved for future use
57	39	@KILL	Kill a file or device
58	3A	@INIT	Open or initialize a file or device
59	3B	@OPEN	Open an existing file or device
60	3C	@CLOSE	Close a file or device
61	3D	@BKSP	Backspace one logical record
62	3E	@CKEOF	Check for end of file
63	3F	@LOC	Calculate the current logical record number
64	40	@LOF	Calculate the EOF logical record number

65	41	@PEOF	Position to the end of file
66	42	@POSN	Position a file to a logical record
67	43	@READ	Read a record from a file
68	44	@REW	Rewind a file to its beginning
69	45	@RREAD	Reread the current sector
70	46	@RWRIT	Rewrite the current sector
71			Reserved for future use
72	48	@SKIP	Skip the next record
73	49	@VER	Write then verify a record to a file
74	4A	@WEOF	Write end of file
75	4B	@WRITE	Write a record to a file
76	4C	@LOAD	Load a program file
77	4D	@RUN	Load and execute a program file
78	4E	@FSPEC	Fetch a file or device specification
79	4F	@FEXT	Set up a default file extension
80	50	@FNAME	Fetch filename/ext from directory
81	51	GETDCT	Get Drive Code Table address
82			Reserved for future use
83	53	DIRCYL	Get the directory cylinder number
84			Reserved for future use
85	55	RDSSEC	Read a SYSTEM sector
86			Reserved for future use
87	57	DIRRD	Directory record read
88	58	DIRWR	Directory record write
89			Reserved for future use
90	5A	MULTEA	8-bit by 8-bit unsigned integer multiplication <C> contains multiplicand, <E> contains multiplier
91	5B	@MULT	16-bit by 8-bit unsigned integer multiplication <C> contains multiplier, <HL> contains multiplicand
92			Reserved for future use

93	5D	DIVEA	8-bit unsigned integer divide <C> contains divisor, <E> contains dividend
94	5E	@DIV	16-bit by 8-bit unsigned integer division <C> contains divisor, <HL> contains dividend
95 - 99			Reserved for future use
1ØØ	64	HIGH$	Contains the highest unused RAM address If HL = Ø, then HL is loaded with current HIGH$ If HL <> Ø, then HIGH$ is changed to (HL)
1Ø1 - 127			Reserved for future use

Following is an alphabetic listing of the SVC labels and numbers:

LABEL	DEC	HEX	LABEL	DEC	HEX	LABEL	DEC	HEX
@ABORT	21	15	@KILL	57	39	@SKIP	72	48
@ADTSK	29	1D	@KLTSK	32	2Ø	@VER	73	49
@BKSP	61	3D	@LOC	63	3F	@WEOF	74	4A
@CKEOF	62	3E	@LOF	64	4Ø	@WHERE	7	7
@CLOSE	6Ø	3C	@LOGER	11	ØB	@WRITE	75	4B
@CMD	23	17	@LOGOT	12	ØC	DIRCYL	83	53
@CMNDI	24	18	@MSG	13	ØD	DIRRD	87	57
@CTL	5	5	@MULT	91	5B	DIRWR	88	58
@DEBUG	27	1B	@OPEN	59	3B	DIVEA	93	5D
@DIV	94	5E	@PARAM	17	11	GETDCT	81	51
@DODIR	34	22	@PAUSE	16	1Ø	HIGH$	1ØØ	64
@DSP	2	2	@PEOF	65	41	MULTEA	9Ø	5A
@DSPLY	1Ø	ØA	@POSN	66	42	RDSECT	49	31
@ERROR	26	1A	@PRINT	14	ØE	RDSSEC	85	55
@EXIT	22	16	@PRT	6	6	RSELCT	47	3F
@FEXT	79	4F	@PUT	4	4	SEEK	46	3E
@FNAME	8Ø	5Ø	@READ	67	43	SELECT	41	29
@FSPEC	78	4E	@REW	68	44	VERSEC	5Ø	32
@GET	3	3	@RMTSK	3Ø	1E	WRPROT	54	36
@INIT	58	3A	@RPTSK	31	1F	WRSECT	53	35
@KEY	1	1	@RREAD	69	45	WRTRK	55	37
@KEYIN	9	9	@RWRIT	7Ø	46			

ERROR DICTIONARY

The Operating System Error Dictionary is contained in the SYS4 system overlay. System errors will normally result in the display of messages contained in the following list. Information on the accessibility of the dictionary messages to the assembly language programmer is contained in the section on system vector entry points.

Ø No Error

This indicates that the @ERROR routine was called without any error condition being detected. A return code of zero indicates no error.

1 Parity error during header read X'Ø1'

During a sector I/O request, the system was unable to satisfactorily read the sector header. Repeated failures would most likely indicate media or hardware failure.

2 Seek error during read X'Ø2'

During a read sector disk I/O request, the requested cylinder that should contain the sector was not located within the time period allotted by the step rate specified in the drive code table. Either the cylinder is not formatted on the diskette, or the step rate designated is too low a value for the hardware to respond.

3 Lost data during read X'Ø3'

During a read sector request, the CPU was late in accepting a byte from the FDC data register. For more information, consult the reference manual for the floppy disk controller used in your disk controller.

4 Parity error during read X'Ø4'

During a read sector disk I/O request, the FDC sensed a CRC error. Possible media failure would be indicated. The Drive hardware could also be at fault.

5 Data record not found during read X'Ø5'

A disk sector read request was generated with a sector number not found on the cylinder referenced.

6 Attempted to read system data record X'Ø6'

A read request for a sector located within the directory cylinder was made without using the directory read routines. Directory cylinder sectors are written with a data address mark that differs from the data sector's data address mark. Consult the controller reference manuals for information concerning address marks.

7 Attempted to read locked/deleted data record X'Ø7'

In systems that support a "deleted record" data address mark, this error message will appear if a "deleted" sector is requested to be read. LDOS currently does not utilize the "deleted" sector data address mark.

8 Device not available X'Ø8'
A reference was made for a logical device that could not be located in the device control blocks. Most likely your device specification was in error. A DEVICE command can be used to display all devices available to the system.

9 Parity error during header write X'Ø9'
This is the same type of error as error 1 except that the operation requested was sector WRITE.

1Ø Seek error during write X'ØA'
This is the same type of error as error 2 except that the operation requested was sector WRITE.

11 Lost data during write X'ØB'
During a sector write request, the CPU was not fast enough in transferring a byte to the FDC so it could be written to the disk.

12 Parity error during write X'ØC'
A CRC error was generated by the FDC during a sector write operation.

13 Data record not found during write X'ØD'
Similar to error 5, the sector number requested for the write operation could not be located on the cylinder being referenced. Either the request is erroneous, or the cylinder is improperly formatted.

14 Write fault on disk drive X'ØE'
This error message results when the FDC returns a "write fault" error. Consult your FDC reference manual.

15 Write protected disk X'ØF'
A write request was generated to a disk which had a write protected diskette, or was software write protected. On 5-1/4" diskettes, a covered up notch will protect the diskette from being written. On 8" media, an exposed notch will perform the same thing. If you want to write on a diskette, you must observe the proper notch condition.

16 Illegal logical file number X'1Ø'
A bad Directory Entry Code (DEC) was found in the File Control Block (FCB). This usually indicates that your program has altered the FCB improperly.

17 Directory read error X'11'
Any disk error sensed during the reading of directory information will result in this error. It could be media failure, hardware failure, or program crashes.

18 Directory write error X'12'
Similar to error 17 but the error condition is sensed while attempting to write a directory sector back to the disk. The integrity of the directory is now suspect. This error can also occur if a disk is write protected when attempting a directory write.

19 Illegal file name X'13'
The file specification provided to the system has a character not conforming to the file specification syntax. See the reference manual section on filespecs.

20 GAT read error X'14'
Any disk error sensed during the reading of the granule allocation table will result in this error. It could be media failure, hardware failure, or program crashes.

21 GAT write error X'15'
Similar to the above except that the error was sensed during a WRITE request. The integrity of the GAT is suspect. This error may also occur if the disk was write protected during a GAT write attempt.

22 HIT read error X'16'
Similar to error 20 but occurring during the reading of the Hash Index Table.

23 HIT write error X'17'
Similar to error 21 but occurring during the writing of the Hash Index Table.

24 File not in directory X'18'
You referenced a file specification that could not be located in the directory. Note that if your request was to LOAD or RUN the file, the error message displayed would be "Program not found". Most likely the cause was a misspelled filespec.

25 File access denied X'19'
The requested file was password protected and the password used was neither the ACCESS nor UPDATE password. This will also occur if a password is specified for a non-password protected file.

26 Full or write protected disk X'1A'
An open of a new file was requested and the target disk's directory was entirely in use. Use another diskette or kill off unwanted files. The error could also result if the disk was protected from write requests.

27 Disk space full X'1B'

While a file was being written, all available space on the disk was allocated before the file was completely written. Whatever space was already allocated to the file will still be allocated although the file's end of file pointer will not be updated. It may be desirable to kill the file to recover the space after writing the file to another diskette.

28 End of file encountered X'1C'

You attempted to read past the end of file pointer. The file was smaller than your application thought. This error can also be used within an application to determine the end of a sequentially read file.

29 Record number out of range X'1D'

A request to read a record within a random access file (see the @POSN vector) provided a record number that was beyond the end of the file.

3Ø Directory full - can't extend file X'1E'

This will result whenever a file has in use all extent fields of its last directory record, and must find a spare directory slot but none is available. All available file positions are in use. See the technical section on directory records for more information. The solution would be to repack the disk by individually copying its files to a freshly formatted diskette.

31 Program not found X'1F'

The load request for a file can not be completed because the file was not located in the directory. Either the filespec was misspelled or the disk that contained the file was not mounted.

32 Illegal drive number X'2Ø'

This error will occur whenever a reference is made to a disk drive that is not included in your system (see the DEVICE command) or the drive requested was not ready for access (no diskette, drive door open, etc.).

33 No device space available X'21'

The SET command was used to establish a new device in the system. Unfortunately, all of the resident system area reserved for Device Code Block tables is already in use. It is suggested that you use the DEVICE command to see if any currently defined non-system devices can be eliminated.

34 Load file format error X'22'

An attempt was made to LOAD a file that did not conform to the format structure for a program file capable of being loaded by the system loader. Most likely, the file referenced is a data file or a BASIC program file.

35 Memory fault X'23'

During the process of loading a program file, the integrity of the load is monitored to ensure that each memory position loaded stores the proper byte value. In cases of partial memory failure, one or more bits of a memory cell will not replicate the value being loaded. This error will then be displayed. If this condition repeats, it is suggested that you subject your hardware to a thorough memory test to locate the root cause.

36 Attempted to load read only memory X'24'

The program file being loaded referenced a memory cell that could not be altered. Either the cell was part of the read only memory (ROM), or the address was referencing an area of the machine not containing any read/write memory (RAM). Use CMDFILE to locate the address loading information of the program file and verify the memory available in your CPU.

37 Illegal access attempted to protected file X'25'

The ACCESS password was given for an operation that required the UPDATE password.

38 File not open X'26'

An I/O operation was requested using a File Control Block that indicated a closed file. See the section on File Control Blocks and the field FCB+Ø.

39 Device in use X'27'

A request was made to KILL a device (remove it from the Device Control Block tables) while it was in use. It is necessary to first RESET a device in use.

4Ø Protected system device X'28'

You cannot kill any of the following devices: *KI, *DO, *PR, *JL. If you try, you will get this error message.

-- Unknown error code

Any time the error routine is called with a error number not in the acceptable range, this message will be displayed. It most likely indicates a software problem.

G L O S S A R Y

The following terms are used throughout this manual, and are fully described here. All of the descriptions pertain to the user sections of this manual; most also pertain to the Technical section.

abbr: - The abbreviation for "abbreviation". It is used at the bottom of each "syntax" block and is followed by the allowable abbreviations for the parameters and switches involved with the command.

ALPHANUMERIC - consisting of only the letters A-Z, a-z, and the numerals Ø-9.

ASCII - The alphanumeric representation of controls and characters as a single byte, falling within a range from 1 to 127 (sometimes including Ø).

ASCII files - Files generally containing only ASCII characters.

BACKGROUND TASK - A job that the computer is doing that is not apparent to the user or does not require interaction with the user. Some examples are the REAL TIME CLOCK, the SPOOLer and the TRACE function.

BAUD - A term that refers to the rate of serial data transfer.

BIT - One eighth of a byte, one binary digit.

BOOT - The process of reseting a computer and loading in the resident operating system from the system drive.

BUFFER - An area in RAM that will temporarily hold information that is being passed between devices or programs.

BYTE - The unit that represents one character to the TRS-8Ø. It is composed of eight binary "bits" that are either ON (1) or OFF (Ø). One byte can represent a number from Ø to 255.

CONCATENATE - To add one variable or string onto the end of another.

CONFIGURATION - The status of the system and physical devices that are available to it. This configuration may be dynamically changed with several library commands and the SYSTEM command, and may be saved with a SYSGEN. If the system is SYSGENed, that configuration will be re-established each time the machine is re-booted or re-started.

CURSOR - The location on the video display where the next character will be printed. It will be marked by the presence of a cursor character.

CYLINDER - All tracks of the same number on a disk drive. On single sided drives, cylinders will synonomus with tracks. On a double sided drive, there will be two tracks per cylinder, one on the front and another on the back of the diskette.

:d - This is used to indicate that a drive spec (number) may be inserted where this is used. A drive spec must always be preceded immediately by a ":" as shown. If a drive spec is not to be given, then the ":" must not be used.

DAM (Data Address Mark) - An identifying data byte put on a diskette to mark the difference between data and directory cylinders.

DCB - Device Control Block, a small piece of memory used to control the status and the input and output of data between the system and the devices.

DCT - Drive Code Table, a piece of memory containing information about the disk drives and/or diskettes in them.

DENSITY - Refers to the density of the data written to a diskette. Double density provides approximately 80% more capacity than single density.

DEVICE - A physical device located outside of the CPU (Central Processing Unit), whose purpose is to transmit/receive data to/from the operating system. The operating system is in total control of any activity directed to/from a DEVICE.

devspec - The name associated with a device by which it is referenced. A "devspec" will ALWAYS consist of three characters; the first of which is an asterisk, followed by two upper case alphabetic characters.

DIRECTORY - A cylinder on a diskette used to store information about a diskette's free and used space and file names.

DRIVER - A machine language program used to control interactions between the operating system and a DEVICE.

*DO - An LDOS system device, the Video Display.

EOF - End Of File, a marker use to denote the end of a program or data file.

/ext - The extension of a filespec. The use of /ext is sometimes optional. An extension (if used), must contain as its first character a "/" (slash), and may be followed by one to three alphanumeric characters.

FCB - File Control Block, a small piece of memory used to control the status and the inputting and outputting of data between the operating system and disk files.

filespec - The name by which a disk file is referenced. A "filespec" consists of four fields and two switches, of which the first field is always mandatory. A filespec is designated by the following format:

!filename/ext.password:d! - - where

"!" - (preceding filename) is an optional switch. If this switch is set, filespec is taken to be absolute. This allows the accessing of a filespec that would otherwise be inaccessible (i.e. a filespec that is the same as an LDOS library command).

filename - The mandatory name of the file

/ext - The optional file extension

.password - The optional file password

:d - The optional drive specification

"!" - (following :d) is an optional switch. If this switch is set, the end of file marker for file (filespec) will be updated after every write to the file.

filename - The mandatory name used to reference a disk file. A filename consists of one to eight alphanumeric characters, the first of which must be alphabetic.

FILTER - A machine language routine which monitors and/or alters I/O that passes through it. "FILTER" is also the LIBrary command which establishes a FILTER routine.

/FIX - The desired file extension for a PATCH file.

FOREGROUND TASK - Jobs the computer does that are apparent to the user, such as running an applications program or a utility and interacting directly with the user.

GRAN - An abbreviation used for the term GRANule. A GRAN is the minimum amount of storage used for a disk file. As files are extended, file allocation is increased in increments of GRANs. The size of a gran varies with the size and density of a diskette.

HIGH$ - High Dollars, a pointer used to mark the highest unused memory address available for use. Any machine language programs loaded above HIGH$ will never be destroyed by the LDOS system.

I/O - The abbreviation for Input/Output.

INTERRUPT - an interruption of the system generated by a hardware clock. The interrupt periods are used by LDOS with such functions as type ahead and the printer spooler.

/JCL - The desired file extension for a DO file. "JCL" is an abbreviation for Job Control Language.

*JL - An LDOS system device, the Joblog.

*KI - An LDOS system device, the Keyboard.

/KSM - The desired file extension for a KSM file. "KSM" is an abbreviation for Key-Stroke Multiply.

LCOMM - A sophisticated communications program capable of interacting with disk, printer, video, keyboard and the RS232 interface. LCOMM will dynamically buffer all the system devices. LCOMM is provided with the LDOS system.

LIBRARY - A set of commands used to perform most operating system functions.

load module format - A file format that loads directly to a specified RAM address.

LSB - The Least Significant Byte in a hexadecimal word, sometimes referred to as the "low order byte".

MACRO - Statements or verbs used in JCL.

MSB - The Most Significant Byte in a hexadecimal word, sometimes referred to as the "high order byte".

MOD DATE - The date a file was last written to.

MOD FLAG - A "+" sign place after a filename to indicate that the file was written to since its last backup.

NIL - A setting of a device, indicating an inactive state. All I/O to/from this device will be ignored.

NRN - Next Record Number.

PACK I.D. - A diskette's name and master password assigned during formatting.

PARSE - The breaking up of a parameter line into its individual parameter values.

partspec - An abbreviation representing "PARTial fileSPEC". A partspec may be used with certain LDOS LIBRARY commands in lieu of a filespec. A partspec may be composed of any combination of the four fields defining a filespec. No switches (i.e. the leading and trailing "!" in a filespec) may be contained in a partspec.

In addition, the filename and /ext fields of a partspec may be abbreviated with leading information and/or "wildcarded". Examples of partspecs are given in the LDOS manual where applicable.

-partspec - Identical to a partspec, except that it is used as exclusion criteria during certain functions.

PARAMETER - The information that follows a library command or a utility, on the command line. This information is passed to the job that will be executed to tell the job how you wish execution to take place. Parameters usually follow the command and are enclosed in parentheses.

parm - The abbreviation for parameter described above.

.password - The password associated with a filespec, the use of which is optional. A password (if used), must contain as its first character a "." (period) , and may be followed by one to eight alphanumeric characters, the first of which must be alphabetic.

PATCH - A utility to make minor alterations to disk files.

*PR - An LDOS system device, the Line Printer.

RAM - Random Access Memory. In the TRS-8Ø, the free user memory in the Computer unit.

ROM - Read Only Memory. In the TRS-8Ø, the BASIC language and drivers stored in the Computer unit.

SECTOR - A contiguous block of disk storage, defined to be 256 bytes, where each byte within the sector has an absolute location and byte identification number. All sectors have a predefined, absolute starting and ending location.

*SI - An LDOS system device, the Standard Input. It is not presently used by the LDOS system.

*SO - An LDOS system device, the Standard Output. It is not presently used by the LDOS system.

SWITCH - A parameter with a definite setting, such as ON/OFF.

TOKEN - A variable used in JCL.

UTILITY - A program that provides a service to the user. Utility programs usually run "outside" of the operating system itself.

wcc - The abbreviation for WildCard Character. In LDOS, the replacement of filespec characters with <$> during certain LDOS commands.

WORD - Two bytes in HEXadecimal format X'nnnn'. Usually entered in reverse notation: low byte, then high byte (LSB,MSB).

IN CASE OF DIFFICULTY

Your LDOS operating system was designed and tested to provide you with trouble free operation. If you do experience problems, there is a good chance that something other than the LDOS system is at fault. This section will discuss some of the most common user problems, and suggest general cures for these problems.

PROBLEM 1) ... The system seems to access the wrong disk drives, or cannot read the diskettes.

There are two main causes of this problem. If you have special hardware, it must be configured properly with the SYSTEM (DRIVE=,DRIVER) command. Check the drive table display with the DEVICE command and make sure that it shows the correct drive configuration.

If you have trouble reading diskettes created on other operating systems, refer to the REPAIR and CONV Utilities. Those sections will explain what is needed to make these types of disks readable.

Problem 2) ... RS-232 communications do not work, or function incorrectly.

If you experience RS-232 problems, the first thing you should do is to make sure both "ends" are operating with the same RS-232 parameters (baud rate, word length, stop bits, and parity). If these parameters are not the same at each end, the data sent and received will appear scrambled.

Some hardware, such as serial printers, require handshaking when running above a certain baud rate. It may be necessary to hook the hardware's handshake line (such as the BUSY line) to an appropriate RS-232 lead, such as CTS.

Problem 3) ... Random system crashes, re-occurring disk I/O errors, system lock up, and other random glitches keep happening.

If you encounter these types of problems, the first thing to check is the cable connections between the TRS-8Ø and the peripherals. The contacts can oxidize, and this can cause many different random problems. Clean the edge card connectors on the CPU unit and the peripherals, and be sure all other cable connections are secure.

If you experience constant difficulty in disk read/write operations, chances are that the disk drive heads need cleaning. There are kits available to clean disk heads, or you may wish to have the disk drive serviced at a repair facility. If you need to frequently clean the disk heads, you might be using some defective disk media. Check the diskettes for any obvious signs of flaking or excess wear, and dispose of any that appear even marginal. Tobacco smoke and other airborne contaminants can build up on disk heads, and can cause read/write problems. Disk drives in "dirty" locations may need to have their heads cleaned as often as once a week.

One common and often overlooked cause of random type problems is STATIC ELECTRICITY. In areas of low humidity, static electricity is present, even if actual static discharges are not felt by the computer operator. Be aware that static discharges can cause system glitches, as well as physically damage computer hardware and disk media.

CUSTOMER SERVICE

The LDOS development and support team is committed to the needs of our customers. Customer service may be obtained by writing LDOS Support services, or by calling the customer service number. The normal customer service hours are from 9 AM to 5 PM Central time, Monday through Friday, excluding holidays. The address and telephone number are as follows:

Logical Systems, Inc.
Customer Support Services
1152Ø N. Port Washington Rd.
Mequon, WI 53Ø92

(414) 241-3Ø66

Note: As of March 1st, 1983, Logical Systems will be moving to a new location. The address and telephone number are as follows:

Logical Systems, Inc.
Customer Support Services
897Ø N. 55th St.
P.O. Box 23956
Milwaukee, WI 53223

(414) 355-5454

All LDOS owners may send in their properly packaged master disk and a $1Ø.ØØ update fee, and receive a copy of the latest LDOS 5.1 version.

Extended Support Agreement

An extended support agreement contract is provided with your copy of LDOS. If signed and returned with the extended support fee, you will receive the following additional support FOR ONE YEAR:

SOFTWARE MAINTENANCE allows you to send, at any time, your ORIGINAL LDOS Master Diskettes to LDOS Support and receive a copy of the latest version of LDOS-5.1. Unless instructed differently, send in only the LDOS-5.1 disk - the LDOSXTRA disk will not normally require updating. The update fee will be $5.ØØ dollars for the LDOS-5.1 disk, and an additional $2.5Ø dollar if both disks are returned. The disk(s) should be sent to the above mentioned support address. If you send in your Master Disk for an update, be sure it is properly packaged.

THE LDOS QUARTERLY NEWSLETTER will be sent to you, containing useful information from other LDOS users as well as technical information and editorial content from our development and support team.

MICRONET BULLETIN BOARD service sponsored by L.S.I. and available for you, if you are also a Micronet subscriber. The LDOS registration card and the Extended Support card contain a space for you to register your Micronet user number with our customer service department. You will then be logged into our bulletin board as a valid user. The bulletin board will be maintained by our customer service personnel on a regular basis, to answer questions and provide timely information.

IF YOU HAVE PROBLEMS

LDOS has been created as a powerful, flexible, and user-oriented system. If you do run into problems, before you pick up the phone, do this:

1) Read the IN CASE OF DIFFICULTY section and do the checks indicated there.

2) READ THE MANUAL.
   Check syntax and spelling carefully.
   Review notes and technical information.
   Verify your understanding of the purpose of the command.
   Check if any updated version is available that deals with the problem.

3) RETRY THE OPERATION.
   Repeat the procedure again.
   Reset (BOOT) and repeat the procedure again.
   Retry the operation using your Master Diskette, if possible.
   Perform the same function in a different manner, if possible (let the Utility prompt for information rather than putting it in the command line or vice-versa, don't abbreviate the parameter, remove unnecessary system options, etc.)

4) THINK.
   Did it work last time? If so, - what has changed since then?
   Could it be a faulty diskette? Maybe another copy would work.
   Is everything turned on, plugged in, etc?
   Is a needed file not present on the disk, such as a system file, data file, etc?

5) WRITE IT DOWN!
   Make notes on the problem, the things you have tried, and the exact steps that led to the problem. The more detailed the notes, the better!

6) CALL.
   Call the Customer Service number during the proper hours.

LDOS LIMITED WARRANTY

Every effort has been made to assure the high quality and reliability of the LDOS product. With the purchase of LDOS, the user is granted certain customer support privileges. This support shall be limited to the privilege of having the master disk updated as often as desired for the current update fee. This is limited to updates within the 5.1 series. Logical Systems, Inc. will also provide a lifetime warranty on the physical diskette media of the original serialized LDOS master diskette. If the diskette media physically fails to retain the original LDOS operating system, replacement media will be provided at no charge. This does not include media that has been damaged in shipment from the user to Logical Systems, or media that has been damaged by the user or his equipment. To receive this support, the user MUST fill out and return the registration card within 30 days of purchase. Should a user find a valid error in the LDOS system, and clearly define it in writing to LDOS SUPPORT, every effort will be made to correct the error. All support shall apply only to registered LDOS owners.

Logical Systems Incorporated and its associates on the LDOS product, assume no liability whatsoever, with regard to the reliablity and/or fitness of the LDOS product for any application. All data and/or programs entrusted to the LDOS system and the computer that it is operating on are the sole responsibility of the user. Under no circumstances will Logical Systems, Incorporated or its associates be held liable for the loss of TIME, DATA, PROGRAMS or for any consequential damages incurred by the user. This warranty and support information refers to the LDOS 5.1.x product only.

FOR LDOS USER SUPPORT CALL.... 414 - 241-3066

LDOS PROBLEM REPORT FORM

Date ___/___/___ Customer Name _____

Serial # _____ Version _____ Model _____

Address _____

City _____ State _____ Zip _____

Country _____ Phone # (_____) _____-_____

CPU: (____) TRS-80 (____) LNW (____) PMC (____) Video Genie

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

LOWER CASE: (____) RS (____) PENCIL (____) Other _____

E.I.: (____) RS (____) LX80 (____) LNW (____) OMIKRON

(____) Other _____

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

(____) Other _____

CLOCK SPEED UP (____) _____ RATE in MHZ _____

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

(____) Other _____

MODEM: (____) _____ RS232 _____

VIDEO: (____) STANDARD (____) Other _____ HI-RES (____)

DRIVES: 5''(____) _____ 8''(____) _____

5''(____) _____ 8''(____) _____

5''(____) _____ 8''(____) _____

5''(____) _____ 8''(____) _____

HARD DRIVE: 5''(____) 8''(____) Controller _____

PRINTER: Parallel (____) Serial (____) Brand _____

(OVER)

LDOS PROBLEM REPORT FORM

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