Diskette Operating System

Reference Manual

093-000201-00

For the latest enhancements, cautions, documentation changes, and other information on this product, please see the Release Notice (085-series) supplied with the software.

Ordering No. 093-000201

Printed in the United States of America
Revision 00, August 1976

NOTICE

Diskette Operating System
Reference Manual

Original Release - August 1976
Addendum 086-000044-00 - February 1977

DataGeneral

SOFTWARE ADDENDUM

Addendum to Diskette Operating System

Reference Manual

086-000044-00

This addendum updates manual 093-000201-00. See UPDATING INSTRUCTIONS on reverse.

Ordering No. 086-000044

Printed in the United States of America
Revision 00, February 1977

NOTICE

Addendum to
Diskette Operating System
Reference Manual

Original Release - February 1977

UPDATING INSTRUCTIONS

This addendum to the Diskette Operating System Reference Manual supplies new information on system generation.

To update your copy of 093-000201, remove and insert the following pages:

Remove	*Insert*
Title/Notice	Title/Notice, and Addendum Title/Notice
iii	iii
v and vi	v and vi
ix and x	ix and x
3-17 and 3-18	3-17 through 3-18.1
7-19 and 7-20	7-19 and 7-20
E-3 through E-17	E-3 through E-19
F-1	F-1
G-1 and G-2	G-1 and G-2
H-1	H-1 and H-2
I-1	I-1

Insert this sheet immediately behind the new Title/Notice page.

A vertical bar or an asterisk in the margin of a page indicates substantive change or deletion, respectively, from 093-000201-00.

The Addendum number appears on the lower left-hand corner of only those pages changed by this addendum.

PREFACE

Data General's Diskette Operating System is fast, economical, and versatile. Its software was derived from the Real-Time Disk Operating System (RDOS), and it interfaces perfectly with RDOS. DOS is a capable system in its own right, however; it can support some very sophisticated equipment, including 8 diskettes, 8 mag tape units, 2 line printers and a plotter.

This manual covers all features of DOS, from SYSGEN to multitasking, as they apply to NOVA, SUPERNOVA, and microNOVA computers. It assumes that you have some programming experience, but even if you do not, you can operate DOS on a console level with the Command Line Interpreter program.

Chapter 1 introduces the Diskette Operating System, and outlines diskette and memory organization. Files and directories are discussed in Chapter 2, while Chapter 3 presents most of the system calls you will need for I/O in a single-user environment. You will find three tools for extending memory in Chapter 4: program swaps, chains and user overlays. Each of the foregoing chapters will help you with the complexities of multitasking in Chapter 5. Chapter 6 explores user interrupts and power fails.

In Chapter 7, you will find a detailed explanation of the Command Line Interpreter program: its syntax, its many console commands (in alphabetical order), and its error messages. The command section begins with a yellow page.

Appendix A also begins with a yellow page; it lists all system and task commands and error messages; Appendixes B and C contain Hollerith and ASCII character tables, and Appendix D surveys overlay directory structure.

Appendix E explains generating your DOS system -- in steps, from the beginning -- and includes a description of the disk initializer program. Bootstrapping the disk-resident system is covered in Appendix F, Exceptional System Status in Appendix G, and RDOS interface considerations in Appendix H. Appendix I covers maintaining your diskettes, and handling recoverable diskette errors.

Within this manual, we use the term "core" generically, to indicate either semiconductor or core memory.

The following Data General publications offer useful related information.

093-000040	Extended Assembler
093-000065	Extended BASIC User's Manual
093-000106	Catalog of Available Software
093-000044	Symbolic Debugger User's Manual
093-000053	FORTRAN IV User's Manual
093-000074	Library File Editor
093-000081	Macroassembler User's Manual
093-000084	Octal Editor
093-000080	Extended Relocatable Loader User's Manual
093-000041	Relocatable Math Library File
093-000110	Software Summary and Bibliography
093-000111	SUPEREDIT User's Manual
093-000018	Text Editor User's Manual

Note that within this manual, all numbers are decimal unless noted otherwise; e.g., 11_8.

We welcome your suggestions for the improvement of this and other Data General publications. To communicate with us, either use the postpaid remarks form at the end of this manual, or write directly to:

Software Documentation
Data General Corporation
Southboro, Massachusetts 01772

CONTENTS

CHAPTER 1 INTRODUCTION

CHAPTER 2 FILES AND DIRECTORIES

CHAPTER 3 SINGLE TASK SYSTEM COMMUNICATION

CHAPTER 3 continued

CHAPTER 4 SWAPS, CHAINS, AND USER OVERLAYS

CHAPTER 5 MULTITASK PROGRAMMING

CHAPTER 5 continued

CHAPTER 6 USER INTERRUPTS AND POWER FAIL/AUTO RESTART PROCEDURES

CHAPTER 7 THE DOS COMMAND LINE INTERPRETER

APPENDIX A DOS COMMAND ERROR SUMMARY

APPENDIX B HOLLERITH-ASCII CONVERSION TABLE

APPENDIX C ASCII CHARACTER SET

APPENDIX D OVERLAY DIRECTORY STRUCTURE

APPENDIX E HOW TO LOAD AND GENERATE YOUR DOS SYSTEM

APPENDIX F BOOTSTRAPPING DOS FROM DISKETTE

APPENDIX G EXCEPTIONAL SYSTEM STATUS

APPENDIX H RDOS-DOS COMPATIBILITY CONSIDERATIONS

APPENDIX I DISKETTE CONSIDERATIONS

CHAPTER 1
INTRODUCTION

Data General's Diskette Operating System (DOS) combines the advantages of a disk operating system, and the low cost of a diskette system. DOS is real-time oriented, since it can schedule and allocate program control to many tasks within a program. DOS offers maximum system efficiency, economically, to a wide variety of installations.

Some major features of DOS are:

- Disk and Core resident system
- Modular multitask monitor
- Multiple user overlays
- 256 software levels of task priority
- Buffered and nonbuffered I/O
- Flexible file organization
- Real-time support for FORTRAN and BASIC.

At minimum, the Diskette Operating System requires a Data General computer with 16K words of memory, a console teletypewriter or CRT video display, a real-time clock, and a diskette.

Larger versions of DOS include up to 32K of memory, power fail/auto restart, and 2.5 million bytes of diskette storage. DOS supports a line printer, a card reader, a reader/ punch, a plotter, up to 8 mag tape units, and up to 6 additional diskettes.

GENERATING A DOS SYSTEM

Each system installation is unique; it must perform diverse tasks with one of many possible hardware combinations. You can tailor DOS for your own hardware environment with the system generation procedure (SYSGEN), as described in Appendix E.

SYSGEN, the builder of tailored operating systems, is an executable system program which can operate in any installation. A standardized starter (bootstrap) system is delivered with DOS; this starter system and SYSGEN enable you to generate one or more configured systems. If future requirements are known, you can generate other DOS systems to fulfill them.

Generated systems must be bootstrapped into execution via BOOT, the DOS disk bootstrap. Appendix F contains a convenient summary of DOS disk bootstrap procedures.

COMMUNICATING WITH DOS

There are three principal ways to interface with DOS and to make the system work for you. They are:

- via system calls in a program
- via task calls in a program
- through console Command Line Interpreter commands

You issue system and task calls as program instructions, and use the CLI as a dynamic interface to DOS via the system console. System calls and task calls activate logic within either system or task modules.

The multitask monitor has a modular structure: at load time, you inform the DOS Relocatable Loader about the number of tasks and channels your program will require. The loader automatically loads only those task modules needed for execution. This conserves core space, and allows more of your program to stay in core at any given moment.

The Command Line Interpreter (CLI) is a system utility program that accepts command lines from the console and translates the input into commands to DOS. Thus, CLI is an interface between your console and DOS. From your console, you use CLI to load programs, maintain files, and access such system utilities as the Library File Editor and the Extended Relocatable Loader.

The system restores CLI to core whenever the system is idle--after initialization, after a disk bootstrap, after a console break, after the execution of a program, etc. CLI indicates that it is in control by outputting a ready message prompt, "R" and a carriage return.

You activate the CLI by entering a CLI command via the system console. You can interrupt the action of the CLI by depressing the keys CTRL and A, or CTRL and C. Chapter 7 describes the CLI.

DOS ORGANIZATION

The DOS executive is the framework of the operating system, and must be core-resident before any processing can occur. This resident portion of DOS performs interrupt processing, overlay and buffer management, system call processing, and file maintenance operations like opening, closing, renaming or deleting files.

In core, the lowest 16_8 memory locations are used for entry points (interrupt and program) into the second area of DOS. This second area is located at the top of memory. Following DOS in all systems is a series of system buffers. These are used to receive system overlays and disk directories for buffered I/O transfers.

The portion of page zero memory available for your programs begins at physical address 16_8 (labelled USP, for user stack pointer), and extends to location 377. Locations 40-47 are part user address space but they have special meanings to the hardware.

Associated with each user program is a User Status Table, UST. This table starts at address 400_8, and describes, among other things, length, the number of tasks required, and the number of I/O channels needed, for the user program.

If user overlays are defined in a program, an overlay directory is found above the UST. Above the overlay directory, if any, is an area reserved for a pool of Task Control Blocks (TCBs). TCBs store task state information, such as the state of active accumulators and carry. The user program follows the TCB pool.

To load a program, you use the relocatable loader utility supplied with your system. After loading your program, this utility loads all modules referenced by the program. (It extracts these modules from libraries like the system library, SYS.LB). By default, system library modules will be loaded last, after the user program and overlay areas. The Task Scheduler and task processing modules are also extracted from the system library.

Following is a simplified map illustrating the positions of tables and program elements in user address space.

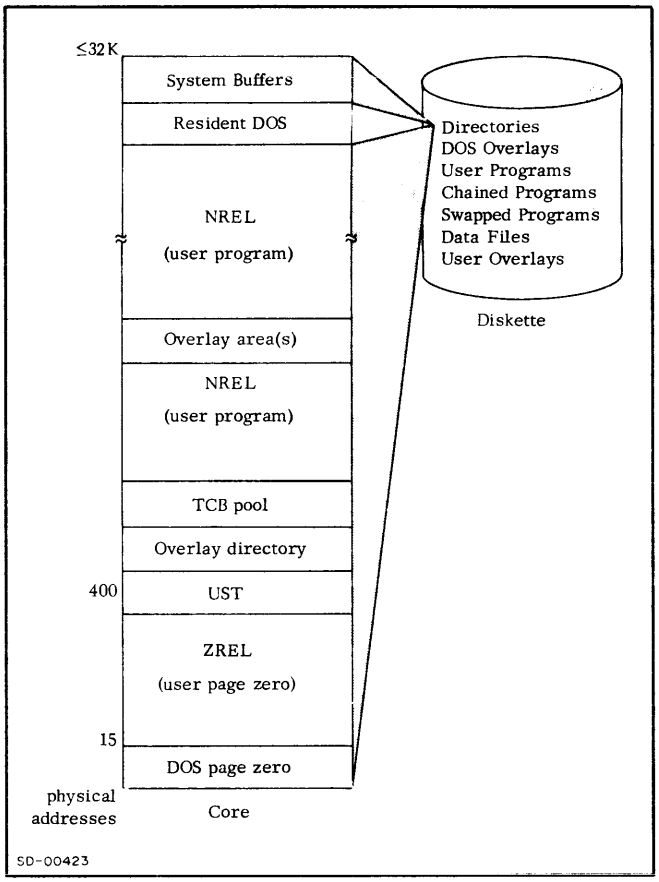


Figure 1-1. DOS Organization

USER PARAMETER FILE (PARU.SR)

User parameters define important system calls, task calls, and mnemonics for user programs. The file PARU.SR, which was delivered with your DOS system, contains all user parameters. File PARS.SR contains all system parameters (addresses, mnemonics, etc.)

END OF CHAPTER

CHAPTER 2

FILES AND DIRECTORIES

DEFINITION OF A FILE

A file is any collection of information or any device receiving or providing the information. Typical examples of both file types are:

Source program file
Relocatable binary file
Core image file (Save file)
Listing file
Teletypewriter keyboard or CRT display
Magnetic tape file

Each of the first four file types has certain qualities, and represents a step in program development. You input a source program file to an assembler (extended or macro) which produces as output a relocatable binary file. You input the relocatable binary file into the relocatable loader, which loads and relocates the file at absolute locations in memory. Once the binary file is loaded, it is stored on disk as a save file. A save file is a core-image file; it is stored word-for-word as it will be loaded into memory. You can create a listing file, to store and/or output the result of any of these steps. DOS executes each step via your CLI (Command Line Interpreter) command.

The teletypewriter or CRT display is the default input and output file; these are discussed below.

Magnetic tape files are also discussed briefly below, and extensively later in this chapter.

FILE OVERVIEW

All devices and disk files are accessible by device or file name; all magnetic tape files are accessible by file number.

A file must be opened (i.e., associated with a DOS channel) before it can be accessed. You can fully open a disk file, allowing several concurrent users to access and modify the file's contents; or open it exclusively, permitting only one user to modify the file but permitting other users to read the file; or you can open it for reading only, by one or more users.

Reserved Device Names

I/O devices have special names which often begin with the character $. Within the limits of the device, you can use each device name exactly as you would a disk file name in a command. You enter each device name as shown below.

$CDR	Punched card reader; mark sense card reader
DPn	Data General diskette unit, number 0, 1, 2, or 3, first controller; number 4, 5, 6, or 7, second controller.
$LPT	80- or 132-column line printer
MTn	9-track magnetic tape transport n (n is in range 0-7).
$PLT	incremental plotter
$PTP	high-speed paper tape punch
$PTR	high-speed paper tape reader
$TTI	teletypewriter or display terminal keyboard*
$TTO	teletypewriter printer or CRT display
$TTP	teletypewriter punch
$TTR	teletypewriter reader
$TTI1	second teletypewriter or display terminal keyboard*
$TTO1	second printer or CRT display

*Input devices other than teletypewriter keyboards automatically provide end-of-file when input ceases for a device-specified time. On TTI line input you must indicate an end-of-file by pressing the CTRL and Z keys.

Disk File Names

A disk file name is string of up to 10 ASCII characters, including upper and lower case letters (DOS converts lower case letters to upper case), numbers, and $. The string is packed left to right, and terminated by a carriage return, form feed, space, or null. You can use any number of characters in a file name, but the system recognizes only the first 10. Moreover, you can use $ wherever you want in a disk file name, if you avoid the reserved 4-character device name combinations.

Each filename in a directory must be unique; if you try to create a file that exists in the current directory, DOS will return an error message. See User Directories, later in this chapter.

You can append an extension to any disk file name. An extension is a string of alphanumeric characters and may include $. The extension can be any number of characters, but the system recognizes only the first two. A period (.) separates the extension from the file name. An example of a file name with an extension is:

FOO.PS

CLI often appends an extension to a filename to indicate the type of information the file contains and to distinguish it from other types of files resulting from the same source file. For example, assume that your source file is named A.SR. CLI would append extensions to different versions of A, as follows:

A.RB	relocatable binary file
A.SV	core image (save file)
A.LS	listing file
A.OL	overlay file

Usually, when you include a file name in a system level command, you need not enter the extension; the command will use a search algorithm to find the file with the correct extension. Occasionally, the system will need extension information from you to find the file you want.

When you choose to add your own extension to a file name, either avoid a CLI extension or use it properly. Don't, for example, confuse the operating system by giving a source file the extension .SV.

File Attributes and Characteristics

A file's attributes protect it; they permit or restrict reading, writing, renaming, deleting, or linking.

The attributes listed below apply primarily to disk files. To protect non-disk files, DOS assigns certain attributes which you cannot change. Of course, you can write-protect a file on magnetic tape by removing the write-enable ring. You can protect a disk file with any of the attributes below. Use either the DOS call .CHATR (Chapter 3) or the CLI command CHATR to alter the attributes of a file.

P	permanent file, which cannot be deleted or renamed.
S	save file (core image).
W	write-protected file, which cannot be written.
R	read protected file, which cannot be read.
A	attribute-protected file. The attributes of such a file cannot be changed. After the A attribute has been set it cannot be removed.
N	no resolution file permitted. This attribute prevents a file from being linked to.
?	first user-definable attribute.
&	second user-definable attribute.

Note that you can assign your own attributes to a file with the characters ? and &; these represent bits 9 and 10 of the attributes word. They are described further under the .CHATR command, Chapter 3.

Disk file characteristics are determined when a file is created, and cannot be changed thereafter. The list of file characteristics is:

C	contiguous file organization.
D	random file organization.
L	link entry. Properly speaking, the L characteristic is given to directory entries rather than to the files themselves.
Y	directory. This characteristic defines a file as being a directory.

The CLI LIST command allows you to obtain information from a file directory about one or more files.

You should avoid giving a file more restrictive attributes than it needs. Note, for example, that a file with attributes AP cannot be deleted in any way except by a full initialization.

File Transfer

You can copy a file from a device to a disk, or write a disk file to a device with the CLI XFER command.

The XFER command transfers the contents of one file to another file. There are two arguments:

XFER sourcefile destinationfile

If you type:

XFER $PTR A)

a file named A is created on disk, and the contents of the paper tape mounted in the paper tape reader are transferred to it. (The symbol) represents a carriage return.) If you type:

XFER P $PTR)

the contents of the file named P are punched out on paper tape.

DISK FILES

DOS supports up to 8 diskettes; its minimum is one diskette. Each diskette has 8 tracks of 77(115_8) disk blocks each; a disk block holds 256 16-bit words. Thus the system's diskette capacity ranges between 315,392 and 1,261,568 words. DOS uses blocks 0 through 17_8 of each diskette for its own files - thus 16 blocks of the total, 4,096 words, are not available for user programs. Usable disk space ranges between 307,200 and 1,228,800 words.

DOS offers you 3 ways to access disk files for I/O. In all but the last mode (which is called Direct Block I/O), files are transferred via system buffers. See Chapter 3, .OPEN command, for the I/O modes.

When DOS writes data into its buffer area, it overwrites the oldest available buffer block first. When all buffers have been used, the least-frequently-used is the first to be overwritten.

After DOS has read a block into its buffers, you can read or write the block's records directly; no further disk access is required. The system keeps track of the blocks that are currently in its buffers, and allows you to access each record within a buffer block.

When you use direct block I/O transfer, DOS transfers by block from disk to the area you specify in core. By avoiding buffering, you save time, but must manage records yourself; you lose the automatic management of the system buffers.

DOS File Organization

Diskette files are organized randomly or contiguously. The maximum length of either kind of file is 65,535 words.

Random Files

All save files employ random organization.

In random files, a file Index contains an entry for each logical block address on diskette. Each entry in the File Index is a word which addresses a disk block. This disk block address may be as high as 65,535. Blocks in the random file are assigned relative block numbers 0 through n, where each number is a sequential unsigned integer. Each index entry has the same relative position as its block has in the file. Thus the logical address of the first block in a DOS file is found at entry number zero in that file's index. All-zero entries in the File Index indicate that the block has not been written.

Generally, no more than two diskette accesses are required to read or write a block: one for the File Index and one for the block of data itself. If the index is core resident (having previously been read into a system buffer), only one access need be made. If the data block itself is resident, no disk accesses at all are required.

Contiguously Organized Files

Contiguously organized files are files whose blocks can be accessed randomly without a random File Index. Contiguous files consist of a fixed number of disk blocks which are located at an unbroken series of disk block addresses. These files can be neither expanded nor reduced in size. Since the data blocks are at sequential logical block addresses, all that DOS needs to access a block within a contiguous file is the address of the first block (or the name of the file) and the relative block number within the file.

All I/O operations permitted on random files can be performed on contiguous files, but the size of the contiguous file remains fixed. Block access is faster in a contiguous file, since there is no need to read a file index.

DISK DIRECTORIES

Each diskette contains its own file directory. The file directory is a list which records all file names on the disk; it is called SYS.DR.

Each diskette also contains its own block allocation map, called MAP.DR. MAP.DR keeps a current record of which blocks are in use and which are free for data storage. MAP.DR is aware of all diskette space except blocks 0 through 5, which contain the bootstrap program. Thus the disk bootstrap can never be destroyed, since the system is unaware of the disk space where it resides.

Initial Disk Block Assignments

On every diskette device blocks 0 through 17_8 have fixed assignments; the remaining blocks are free for system use or user file storage. Blocks 0 and 1 are reserved for the root portion of the disk bootstrap program, BOOT. Blocks 2 through 5 are used by the operating system for recording disk status and marking bad disk blocks. Block 6 is the first index block of SYS.DR, the system directory. Block 7 is reserved for an index of file index blocks used whenever a program swap occurs. Blocks 10 through 17_8 are reserved for swap file indexes. Block 17_8 is reserved for the first block of the MAP.DR file.

Disk Block Number (octal)

0 1	root portion of BOOT
.	unavailable for DOS file space
6	first index block of SYS.DR
7	index of file index blocks used for swap storage
.	swap storage index blocks
17	MAP.DR block
.	free blocks for DOS or user files
1150_8 (maximum)	

As mentioned earlier, the MAP.DR file indicates which disk blocks are currently in use and which are free for assignment. Each bit of each word in MAP.DR indicates whether or not a specific block is in use. Block assignments are from left to right, in ascending block order starting with block 6. MAP.DR is a contiguous file.

Word	Contents
0	block allocation map, 1 bit per block, from left to right in ascending block order
.	starting with block number 6.
.	0 means that block is available,
.	1 means that block is in use.
n-1	n is the size of the partition in blocks/16 (and integer division is used).

System Directory (SYS.DR)

You can create many directories within your DOS system, and create files in each directory. Each disk or diskette has a system file directory, SYS.DR, which maintains information on these directories and files. Each directory also has a SYS.DR, to keep track of the files within it. Each SYS.DR is a random file.

The system directory employs a hashing algorithm to speed up access of directory entries. An initial system directory area is allocated at the time the system is fully initialized. This area (called a frame) is a contiguous set of disk blocks; the set is contiguous to minimize head travel time.

The first word in each block of SYS.DR is the number of files listed in the block. Following this word is a series of 22_8 -word entries, called user file descriptions or UFDs, which describe each file. Each block in SYS.DR looks like this:

Word	Contents
0	Number of files in this block of the directory (16_8 maximum)
1 . . . 22	User file description (UFD)
23 . . . 44	User file description (UFD)
.	
.	
.	

The UFD describes the file's name, its two-character name extension, its size, its attributes and characteristics, the address of the first block, other qualities, and a logical device code describing the device associated with this file.

Word (octal)	Contents
0-4	Filename
5	Extension
6	Attributes and characteristics
7	Link access attributes
10	Block count -1
11	Byte count in last block
12	First address (i.e., logical address of first block in the file.)
13	Year and day last accessed
14	Year and day created or most recently modified
15	Hour and minute created or most recently modified
16	UFD variable info
17	UFD variable info
20	Use count
21	DCT link (device code)

The link access attribute in word 7 permits or restricts links to the file. See Link Entries, below.

A nonzero file use count indicates that one or more users have opened the file. If a hardware malfunction occurs when a file is open, its count will often be wrong; you must clear it to zero (via the CLI command CLEAR) before you close, rename, or delete the file, or RELEASE (.RLSE) its directory (see Chapter 3).

User Directories

You can create a user directory with the CLI command CDIR, or the system command .CDIR. Each directory name on diskette must be unique, as must each file name within a directory. DOS will return an error message if you attempt to create a directory that already exists, or create a file that already exists in a given directory.

Typically, when several people use a DOS system, each user has a personal directory, and unlimited access for reading to several common public files. Access to other directories is often prohibited or severely restricted.

Directories are mutually exclusive subsets of the SYS.DR file space. A directory has no defined amount of file space; it takes file space from the diskette as required and releases the space when it is no longer needed.

A newly-created directory consists of three blocks: SYS.DR's initial index block and data blocks for the SYS.DR and MAP.DR entries. The map directory entry in each subdirectory's SYS.DR points to the parent's MAP.DR.

Initializing and Releasing a Directory

You must initialize a directory before you can access its files. Initialization opens a directory, introduces it to the system, and prepares it for use. This procedure is called initialization.

You can use either of two commands to partially initialize a directory. These are the CLI commands INIT or DIR (or systems commands .INIT or .DIR). Use the first command to initialize any directory:

```
INIT directory)
```

While many directories can be initialized at any moment, you can have only one current default directory. The DIR command changes the current default directory and initializes it at the same time. (As mentioned earlier, the default directory is the one to which all file references without directory specifiers are directed.) For example:

```
DIR directory)
```

During system generation, you specify the maximum number of directories which can be initialized at any moment. The current maximum is 32.

To release a directory, issue the CLI command:

```
RELEASE directory)
```

When you release a directory, you remove its initialization and close all its files. If you release the current default directory, the master directory becomes the current default directory until you specify another current directory. The master directory holds the operating system and the system will shut down if you release it.

At shutdown, of course, you release the master diskette (and with it the master directory). You must release each diskette before physically removing it from its device. For example:

```
RELEASE DP0)
```

Referencing Disk Files

Because a file may be in one of many directories, you must enter both directory name and file name to reference a file (unless the file is in the current directory). This procedure, called directed file access, requires you to enter the directory name, a colon, and the file name (without spaces). For example:

```
ACCTSDUE:SOURCEFILE
```

You can also access a file name by making the file's directory into the current (or default) directory. Use the CLI command DIR, or DOS command .DIR, to select the default directory. For example:

```
DIR ACCTSDUE )
```

After you have established the default directory, all other file references without directory specifiers will be directed to it. If you enter a file name and a directory specifier, DOS will assume that you want a file in another directory, and will make a directed access there.

Because you have more than one diskette in your system, you will often need to reference directories and files with a diskette (or global) specifier. For all 8 permissible diskettes, these specifiers begin with DP0, and end with DP3 for the first controller; they start with 4 and end with 7 for the second controller.

For example, assume DP0 and DP1 are diskettes, on a system. If DP0 and DP1 are both in the system, both units are initialized, and DP1 is the current default directory device, you could use either a directed access,

```
DP0:MAIN )
```

to access directory MAIN on diskette 0; or you could make the diskette the current directory device,

```
DIR DP0 )
```

and then simply reference directory MAIN, or a file in MAIN:

```
DIR DP0:MAIN:SOURCEFILE )
```

If neither device had been initialized, you could initialize each with the INIT command, for example:

```
INIT DP0 )
INIT DP1 )
```

or initialize directory MAIN with one command:

```
INIT DP0:MAIN )
```

Master Diskette

The master diskette is the one on which you booted the current system. If you bootstrap another diskette it becomes the master. The master diskette has the following uses:

1. It becomes the current directory after the release of a current default directory.

2. It contains the system save and overlay files.

3. It contains push space for program swaps.

Disk Bootstrap Device

A disk bootstrap device is a diskette containing a copy of the disk bootstrap program. If you so instruct it, the diskette bootstrap program (Appendix E) will write a copy of the disk bootstrap to blocks 0 and 1 of each bootstrap device; thus any unit can become the bootstrap device.

Link Entries

The link entry permits DOS users to access any disk file or magnetic tape file, by its name or by many different names (called aliases). Moreover, users can access files outside their own directories, and on other diskettes, with link entries.

Link entries save disk file space by allowing users in different directories to access a single copy of a commonly-used disk file; this is their most popular application. Link entries may point to other link entries, with a depth of resolution of up to 10. The entry which is finally linked to is called the resolution entry. You can create a link entry with the CLI LINK command or the system command .LINK.

Creating a link entry is easy - the resolution entry need not even exist when you do it. Your only requirement is that the link entry name be unique within its directory.

To use a link, you must initialize the diskette and directory containing the resolution entry and all intervening directories. (You do this with the call .INIT (Chapter 3), or the CLI command INIT.) Moreover, the attributes of the resolution entry must allow linking.

In Figure 2-3, three links exist to the resolution entry on diskette DP0. The resolution file - ASM.SV - is a useful file: the Extended Assembler supplied with your system. Like other Data General utility programs, it is located on the diskette which holds the operating system (master diskette). It was placed there automatically during system generation. It is not in a user directory, and linking to it is easy. Assume that the system has two diskettes, on one drive. The CLI command sequence which created the structure shown in Figure 2-1 would be:

```
INIT DP0 )
R       (R is the CLI ready prompt)
INIT DP1 )
R
DIR DP1:MYDIR)
R
LINK ASM.SV DP0:ASM.SV)
R
```

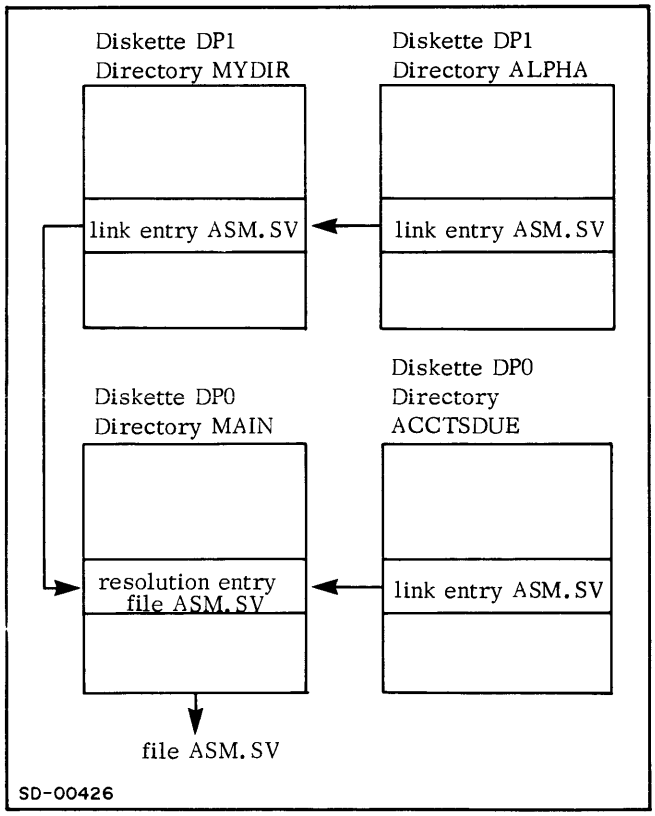


Figure 2-1. Link Entries

The LINK command creates a link entry named ASM.SV in MYDIR to the assembler on DP0. Now that MYDIR is linked to ASM.SV, any user in MYDIR can assemble a file, while ASM.SV occupies disk space on DP0 only.

Note that the LINK command has two arguments:

```
LINK link-entry-name  resolution-file-name
```

The link entry is assumed to be on the current diskette, in the current directory (which can be either the diskette itself or a user directory). The resolution file is also assumed to be on the current diskette, not in a user directory. If the resolution file is elsewhere, you must indicate its location with colon specifiers, as above.

The command sequence to link directory ALPHA would be:

```
DIR ALPHA )
R
LINK ASM.SV DP0:ASM.SV )
```

or

```
LINK ASM.SV MYDIR:ASM.SV)
R
```

(For this chained link example, assume the second link command). To link from ACCTSDUE, you'd type:

```
DIR DP0:ACCTSDUE)
R
LINK ASM.SV ASM.SV  (or LINK ASM.SV/2) )
R
```

The link entry need not have the same name as the resolution file. Link operations are clearer and simpler, however, if the link shares a name with its resolution file. Link entries with different names are called aliases.

Note that both aliases and resolution files that are not on the current diskette require you to input specifier information.

To use the link from ALPHA, you would need to initialize DP1, MYDIR, and DP0 (if any of these had been initialized, DOS would return an error message, but no harm would have been done).

Before you can use a link, all intermediate links must be resolvable. Thus if you removed the link entry from MYDIR (UNLINK or .ULNK) the link in ALPHA would be useless but the link in ACCTSDUE would still work. Note that UNLINK (.ULNK) is the only way to remove a link entry; if you try to DELETE (.DELET) a link, the link will persist and the resolution entry will be deleted.

Each link entry is a filename, whose sole function is to point to the resolution entry (or to another link entry which is closer to the resolution entry). The resolution entry is also a filename, which points to the resolution file.

Like other files, each resolution entry has a user file definition, which includes two sets of attributes: file access attributes (called resolution entry attributes) and link access attributes.

You assign resolution entry attributes to govern direct access to the file and change the attributes via the CLI command CHATR or system command .CHATR (Chapter 3). The attribute N forbids linking; other attributes govern reading, writing, renaming, or deletion. The A attribute makes all other attributes of a resolution entry or file permanent.

Use link access attributes to permit or restrict access to the resolution entry. Again, the N attribute forbids linking. You can use the CLI command CHLAT or DOS command .CHLAT to change these attributes.

Thus, although you can create a link to a resolution file very easily, two sets of resolution entry attributes guard the resolution file. As seen by a link entry, the resolution file has a composite of link access and resolution entry attributes.

More than one link entry may point to a resolution entry, and more than one user at a time may open the resolution file for reading and writing. Single user read-write opens and multiple read-only opens are also allowed.

Whenever you create a file whose name is already linked, the system will create a resolution entry for it. If the resolution entry already exists, then the file must exist, and the system will return error ERCRE (Attempt to create a file that already exists). If you use the CLI to create such a file, it will return the error message: FILE ALREADY EXISTS:filename.

After you create and link a file, you cannot open the resolution file (by a system .OPEN linkname command) until all directories in the path to the resolution file have been initialized. These include the directory containing the link entry and all directories which contain either intermediate links or the resolution entry. The system will return error ERDNI (Directory not initialized) or error ERDSN (Device not in system) from the OPEN command if all intervening directories haven't been initialized.

Link Devices

The links entry offers much more than a simple way to share user files. You can create a link entry for any file - including a reserved device like the $LPT.

If you establish a link to a mag tape resolution file, the device must be initialized before the link will work. A nondisk device name cannot be linked in turn to another file name because it cannot contain a directory.

Directory Command Summary

Following is a list of CLI and .SYSTM commands used to manage disk file directories; see Chapter 3 and Chapter 7 for more information about these commands.

CLI Command	System Command	Meaning
CCONT	.CCONT	Create a contiguously organized file with all data words zeroed.
CDIR	.CDIR	Create a directory.
CHATR	.CHATR	Change file attributes.
CHLAT	.CHLAT	Change a file's link access entry attributes.
CLEAR		Set a file's use count to zero.
CONN	.CONN	Create a contiguously organized file with no zeroing of data words.
CRAND	.CRAND	Create a random file.
CREATE	.CREA	Create a random file.
DELETE	.DELE	Delete a file.
DIR	.DIR	Specify a default directory, initializing it if necessary.
INIT	.INIT	Initialize a directory or device.
LINK	.LINK	Create link entry to a file in any directory.
RELEASE	.RLSE	Release a directory from the system, and make the default or master directory the current directory; or release a diskette unit from the system.
RENAME	.RENAM	Rename a file.
UNLINK	.ULNK	Delete a link entry.

MAGNETIC TAPE FILES

You can access data on magnetic tape by both tape file I/O and direct I/O. DOS permits file access on 9-track magnetic tape, and supports up to 8 magnetic tape drives. For direct block I/O, the tape controller supports reading and writing at any density; other forms of I/O require high density.

The following are the I/O modes generated by the operating system:

Tape File I/O:	9-track NRZI800BIP, ODD Parity 9-track PE 1600BPI, ODD Parity
Free Form I/O:	Parity in any hardware combination except WRITE EOF ODD for 9-track.

If a controller detects a parity error during reading, the system will attempt to reread the data 10 times before issuing error code ERFIL, "file data error." If a file data error is detected and returned to the CLI, the message will be output: PARITY ERROR: FILE MTn:dd, where n is the unit number and dd represents the file number.

If an error is detected after writing, the system will attempt, up to 10 times, to backspace, erase, and rewrite. If the rewrite fails the tenth time, then an error will be signaled.

If an error is received from a magnetic tape unit which does not conform to any predefined system error conditions, the tape status word will be returned as the error code. If this code is returned to the CLI, it will be reported as an unknown error code: UNKNOWN ERROR CODE n. (n is the tape status word.)

Nine Track Data Words

Each data word output to 9-track units, under both file I/O and free format I/O, is written as two successive eight-bit bytes. Data is encoded as in Figure 2-2.

Each tape has a physical end-of-tape (EOT) marker. Whenever access is made beyond this marker, error ERSPC will be returned after the operation is completed. A new file cannot be started beyond the physical end of tape marker.

Upon reaching the physical EOT while writing, you should terminate the tape file to avoid running the tape from its reel.

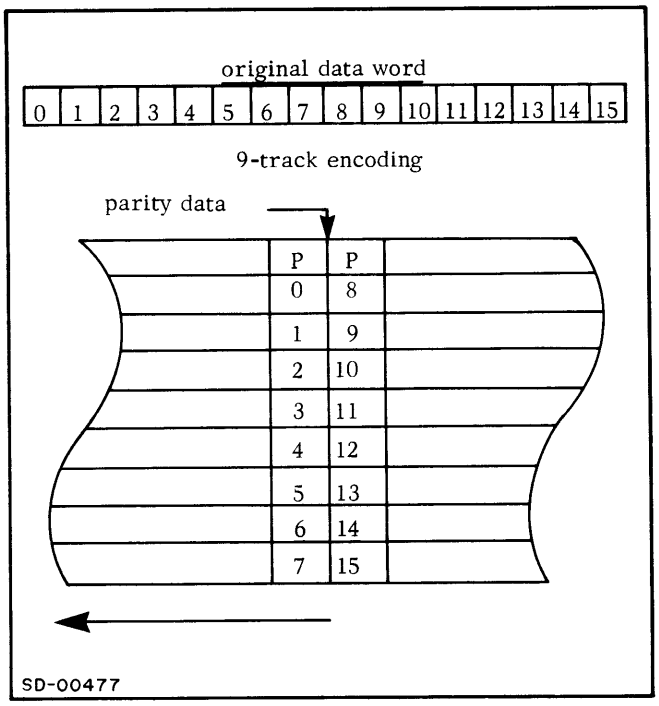
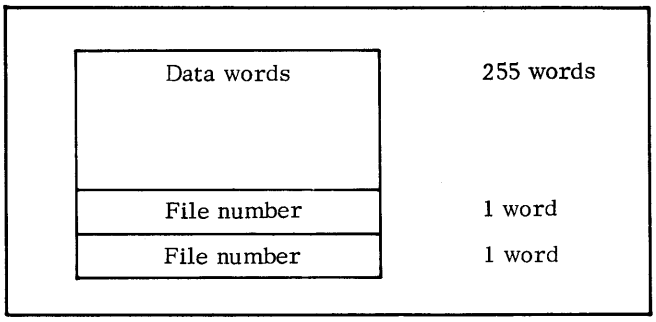


Figure 2-2. Data Encoding (9-track tapes)

Magnetic Tape File Organization

In magnetic tape file format, data is written and read in fixed-length blocks of 257 16 bit words. Data files are variable in length, each one containing as many fixed-length blocks as is required. The first 255 words of each block are user data, and the last two words each contain the file number. The following illustration shows the structure of a data block:



After the first file, a double end-of-file (EOF) mark is written. The system begins writing at the first double EOF it finds, overwrites the second EOF in the pair, writes the file, and signifies the end by writing another double EOF. Files are written in consecutive order, starting with file number 0 and extending through file number 99.

Initializing and Releasing a Tape Drive

To initialize a tape drive, use the CLI INIT command; e.g., INIT MT0). INIT automatically rewinds the tape on that drive to BOT. Full initialization (INIT/F) writes two EOFs, the logical end-of-tape indication, on the beginning of the tape, and rewinds the tape. You must always perform an INIT/F on all new mag tapes before using them. Note that INIT/F effectively erases the tape by permitting DOS to overwrite all files on it.

The CLI RELEASE command rewinds a tape to BOT and releases its drive from the system.

Referencing Files on Magnetic Tapes

Files are placed on tape in numeric order, beginning with file number 0. Up to 100 files may be placed on any given tape, with the last file having number 99.

To access a tape file in a command line, enter the command and the tape specifier, followed by a colon and a file number. For example:

```
PRINT MT0:6 )
```

MT is the specifier for magnetic tape, 0 is the drive unit number, and 6 is the file number. The format and definitions of all magnetic tape specifiers are:

MT*n*:*m*	Magnetic tape unit *n*, where *n* is from 0-7 and has no leading zero, with file number *m* from 0-99.

Either a one-digit or a two-digit number may be used to reference the first ten file numbers. Thus to reference file number 8 on magnetic tape unit 2, you would use the following global specifiers:

MT2:08 or MT2:8

Both the tape global specifier and the file number must be given. Violation of this rule will cause the system to respond: ILLEGAL FILE NAME.

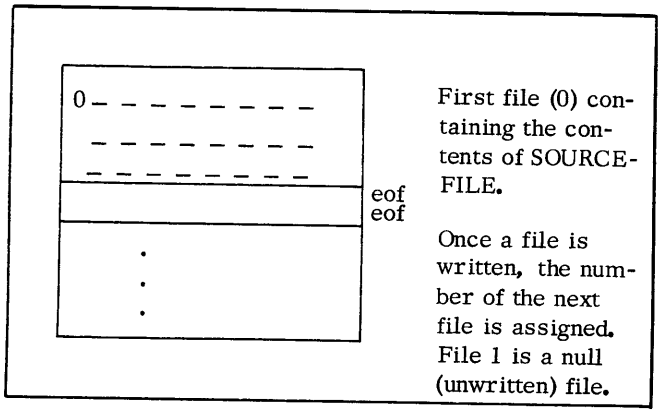
Some examples of references to files on tape and disk are:

DUMP MT0:0)	Dump all nonpermanent files onto tape from disk (this provides a magnetic tape backup). The files become file number 0 of the tape mounted on unit 0.
LOAD MT0:0)	Reload the files onto disk from tape.
ASM MT1:07 MT2:0/B)	Assemble file number 7 on unit 3, and output the relocatable binary to file 0 of the tape mounted on unit 2.

You must write a file on magnetic tape in numeric order. For example, assume that you transfer a disk file to tape unit 0. Tape unit 0 contains a new tape, which has just been fully initialized.

```
XFER SOURCEFILE MT0:0)
```

SOURCEFILE becomes the first file on the new tape, which contains the following:



The system recognizes only files number 0 and 1 on the tape; because numbers are assigned incrementally, only these numbers exist.

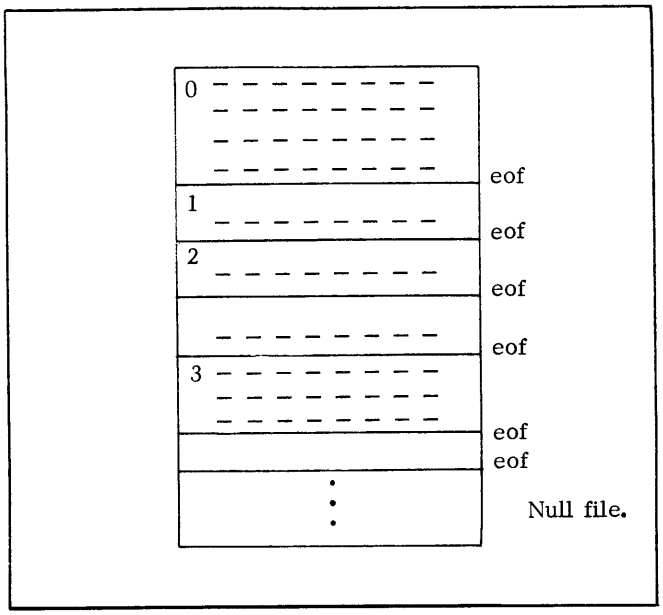
If you try to reference any other file on the tape:

```
XFER MYFILE MT0:2)
```

The system will be unable to find file 2, because file 1 is the last null file. An error message will result:

```
FILE DOES NOT EXIST, FILE: MT0:2
```

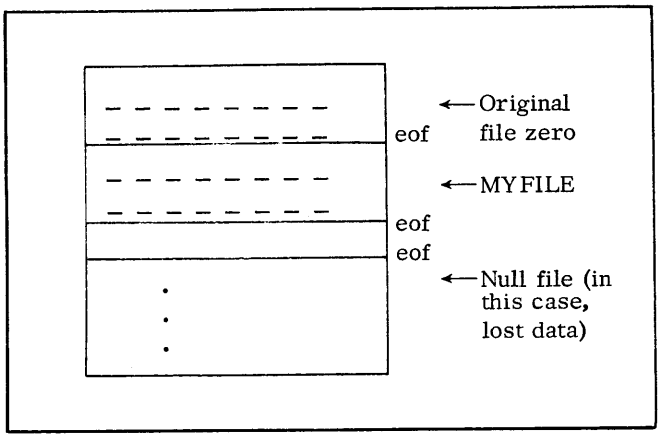
As you write files on tape, you should note their numbers. Otherwise, you could inadvertantly overwrite a file, and thus destroy the overwritten file and all following files. For example, assume a tape on drive 0 contains four files:



The command:

XFER MYFILE MT0:1

overwrites the contents of file 1 with MYFILE, and voids the location data of following files. The original file 1 and all subsequent files are lost.



Before you physically remove a mag tape reel, you must RELEASE or INIT its transport. Either command resets the system tape file pointer to file 0, for correct file access in the future.

You must also note the implications of the logical end-of-tape mark, double EOFs, employed by DOS. For example, if you deliberately write a null file, no further files can be referenced on the tape. Your null file will be the last file.

Linking to Magnetic Tape Files

You can link tape files to disk files with the mechanism described under disk files. Linking disk file A to tape file MT0:0 creates a resolution entry in the current directory for resolution file MT0:0; the link entry to file A is named A. References to file A in the current directory are resolved as references to file MT0:0.

Free Format Tape Reading and Writing

In addition to fixed block-size tape file I/O, DOS allows you to read and write data in free format, word by word, to magnetic tape. You specify free format with the system call .MTDIO, described in Chapter 3, under Input/Output Commands.

Essentially, .MTDIO allows you to read or write from 2 to 4096 words within a data record, and to space forward or backward from 1 to 4095 data records or to the start of a new data file. Additionally, this call allows you to rewind a reel, write an end-of-file mark, to read the transport status, and perform other machine-level operations. The system does not maintain a tape file pointer under free format I/O; this contrasts with tape file I/O.

END OF CHAPTER

CHAPTER 3
SINGLE TASK SYSTEM COMMUNICATION

MULTIPLE AND SINGLE TASK ENVIRONMENTS

A program task is an execution path through user address space demanding use of system resources such as I/O, overlays, or simply CPU control. User address space includes all core memory from 16_8 (the User Stack Pointer) through NMAX-1.

In a multiple task environment, logically distinct tasks compete simultaneously for system resources. Only one task receives program control and the desired resource at any single moment, and resources are awarded to tasks according to their priority and readiness to use the resources. Chapter 5 describes multitasking.

A Task Scheduler supervises all resource allocation. There are two Task Schedulers used by the operating system: TMIN, and TCBMON. Both schedulers were supplied with your system, in your system library, SYS.LB. TMIN oversees single task environments, while TCBMON directs multitask programs. Each scheduler works by modifying Task Control Block (TCB) queues and allocating resources to the highest priority ready task.

A single task environment is a subset of a multitask environment. A TCB queue in a single task environment has only one TCB, and requires little task supervision. Unless you specify more than one task in a program, the basic Task Monitor, TMIN, without any multitask command logic, is loaded by default by the relocatable loader from the system library, SYS.LB.

While this chapter assumes that you are running a single task, you may need more than one I/O channel for this task. The system allocates 8 channels by default, but this number may not be most efficient for your task.

You can specify a number of channels in two ways: at load time, by including with the /C and /K switches in the relocatable loader command line, or by a .COMM TASK statement. The .COMM TASK statement also allows you to specify the number of tasks.

If you choose a .COMM TASK statement, include it in the first user-relocatable binary.

You should omit .COMM TASK from any program which you plan to run under RTOS.

The format of the statement is:

```
.COMM TASK,n*400+m
```

where: n is the number of tasks and
m is the number of channels you want.

CAUTION: In multitask programming, you must use the .COMM TASK statement or equivalent loader switches to avoid fatal load error "LOAD OVERWRITE 17." Chapter 5 gives more detail on TCBMON and TCBs.

SYSTEM AND TASK COMMANDS

DOS system and task commands allow you to communicate directly with the operating system. System calls and task calls are similar, but not identical.

You begin each call with the mnemonic .SYSTM, which assembles a JSR @ 17 instruction. This instruction enables the system to respond to your command in user address space via the Task Scheduler. The command word must be assembled as the word following .SYSTM.

After the system has obeyed a system call, it takes a normal return to the second instruction after the command word. If it detects an exceptional condition, it takes the error return, to the first instruction following the command word. System calls always reserve AC2 for the error code.

The general form of a system call description is:

ACn	- required input to the call .SYSTM command error return (error code in AC2) normal return (each accumulator, except AC3, is restored unless it is used to return output)
ACn	- output from the call
AC3	- the contents of USP (User Stack Pointer) is the default value.

Many system calls require you to input a bytepointer to a specific filename. When you input this bytepointer, you can include a directory specifier as well, if the directory is initialized. All DOS system calls are summarized below.

A task call resembles the system call, with two exceptions:

1. You enter no .SYSTM mnemonic before the task command word.
2. Task calls which cannot take an error return do not reserve an error return location. All system calls reserve an error return location even if no error return is possible. The commands in the chapter are all system calls; Chapter 4 includes two task calls, Chapter 5 consists almost entirely of task calls, and Chapter 6 has one task-oriented call.

COMMAND WORD FORMAT

There are two basic system command word formats: "command C" and "command".

In the first format, C is a digit (from 0 to m of .COMM TASK, page 3-1) representing an I/O channel number. The channel number C* indicates a logical link to a file. The largest value of C is specified in USTCH, and may not exceed 76_8 (see Chapter 5, User Status Table.) For footnotes, see next page.

When no I/O channel is needed for command execution, use the command word alone in the instruction. If the command requires arguments, pass them in the accumulators. Figure 3-1 lists all .SYSTM calls available to you.

.APPEND	Open a file for appending.	.MDIR	Get the logical name of the master device.
.BOOT	Bootstrap a new system.	.MEM	Determine available memory.
.BREAK	Save the current state of memory in save file format.	.MEMI	Allocate an increment of memory by changing NMAX.
.CCONT	Create a contiguously organized file with all data words zeroed.	.MTDIO	Perform free format I/O on tape.
.CONN	Create a contiguously organized file with no zeroing of data words.	.MTOPD	Open a mag tape for free format I/O.
.CDIR	Create a subdirectory.	.ODIS	Disable keyboard interrupts for this console.
.CHATR	Change file attributes.	.OEBL	Enable keyboard interrupts for this console.
.CHLAT	Change link access attributes.	.OPEN	Open a file for reading and writing by one or more users.
.CHSTS	Return file directory information for the file currently-opened on a specified channel.	.OVLOD	Load a user overlay.
.CLOSE	Close a file.	.OVOPN	Open a user overlay file.
.CRAND	Create a random file.	.PCHAR	Output a character from the console.
.CREATE	Create a random file	.RDB	Perform a direct block read.
.DDIS	Provided for RDOS compatibility.	.RDL	Read a line.
.DEBL	Provided for RDOS compatibility.	.RDS	Read sequential bytes.
.DELETE	Delete a file.	.RDSW	Read the console switches.
.DIR	Change the default directory/directory device.	.RENAM	Rename a file.
.DUCLK	Define a user clock.	.RESET	Close all files.
.EOPEN	Open a file for reading and writing by one user only.	.RLSE	Release a directory or device.
.ERTN	Return from program swap with exceptional status.	.ROPEN	Open a file for reading only by one or more users.
.EXEC	Load and execute a program swap.	.RSTAT	Obtain a resolution file's directory status.
.GCHAR	Get character from the console.	.RTN	Return from a program swap.
.GCHN	Get the number of a free channel.	.RUCLK	Remove a user clock.
.GCIN	Get the operator input console name.	.SDAY	Set today's date.
.GCOUT	Get the operator output console name.	.SPDA	Provided for RDOS compatability.
.GDAY	Get today's date.	.SPEA	Provided for RDOS compatability.
.GDIR	Get the current default directory name.	.SPKL	Provided for RDOS compatability.
.GHRZ	Examine the real time clock.	.SPOS	Set the current file pointer.
.GPOS	Get the current file pointer.	.STAT	Obtain file directory information.
.GSYS	Get the name of the current operating system.	.STOD	Set the time of day.
.GTATR	Get file attributes.	.TUOFF	Provided for RDOS compatability.
.GTOD	Get the time of day.	.TUON	Provided for RDOS compatability.
.IDEF	Identify a user device.	.ULNK	Delete a link entry name.
.INIT	Initialize a device or open a directory.	.UPDAT	Update the current file size.
.IRMV	Remove a user device.	.WCHAR	Wait for a character.
.LINK	Create a link entry.	.WRB	Perform a direct block write.
		.WRL	Write a line.
		.WRS	Write sequential bytes.

Figure 3-1. System Command List

STATUS ON RETURN FROM SYSTEM

Status of the accumulators upon return from the system (.SYSTM or task call) is as follows. If the system returns no information as a result of the call, the carry and all accumulators except AC3 will be preserved. On return from all system calls, AC3 will contain the contents of location USP (location 000016). On a NOVA 3 or microNOVA, if the program was loaded with N3SAC3.RB, the value of AC3 will be the contents of the frame pointer register. On exceptional return, AC2 is used to return a numeric error code. Error codes are listed in Appendix A.

DEVICE AND DIRECTORY COMMANDS

DOS can manage many different directory devices simultaneously. You specify the system configuration precisely during system generation.

You specify a directory device by a three-character code, the first two characters of which specify device type and third the unit number. For example, DP0 indicates diskette unit 0, while, DP3 indicates diskette unit 3.

Initialize a Directory or Device (.INIT)

You initialize devices and directories via the following system command:

```
.SYSTM
.INIT
error return
normal return
```

*Any system command requiring a channel number C need not actually contain C. If you specify 77_8 (the device code of the CPU), as the channel number in the instruction, the system will use instead the number passed in AC2. (.RDB and .WRB present a slight variation to this rule.) For example, the following instructions specify a write to channel 3:

```
        LDA     2,C3
        .SYSTM
        .WRS    77
        JSR     ERROR
        .
        .
        .
C3:     3
```

On entry to the system, AC0 contains a bytepointer* to a directory/device specifier character string terminated by a null byte. If AC1 contains 177777 when you invoke .INIT, a full initialization of the device results. Full initialization on a mag tape rewinds the tape and writes two EOF's at the logical beginning-of-tape. All previous files are lost. Full initialization of a diskette overwrites all files on it, and builds a virgin file directory and free storage map. Full initialization of a directory is a no-op.

If AC1 contains 0 when you invoke .INIT, a partial initialization of the device or directory results. The system finds the current default directory on the device and reads it into core, thus allowing subsequent file access on the device. All files in the directory become available to the system software. Partial initialization of a magnetic tape rewinds the tape to BOT, and resets the tape file pointer to file zero.

Any of following error conditions might arise during the execution of the .INIT system command. When an error occurs, the system takes the error return to your program with the error code in AC2.

AC2	Mnemonic	Meaning
1	ERFNM	Illegal file name.
10	ERWPR	Device is write-protected (full initialization only).
12	ERDLE	Directory does not exist.
27	ERSPC	Out of disk space.
31	ERSEL	Unit improperly selected.
36	ERDNM	Device not in system.
45	ERIBS	Device already initialized.
51	ERNMD	Insufficient number of Device Control Blocks (DCBs), specified at SYSGEN time.
52	ERIDS	Illegal directory specifier.
57	ERLDE	Link depth exceeded.
77	ERSDE	Error detected in SYS.DR of nonmaster device.
101	ERDTO	Ten-second disk timeout occurred.
102	ERENA	No linking allowed (N attribute).
112	EROVF	Too many chained directory specifiers caused system stack overflow. This can occur only when links are used in the specifier string.

*In each bytepointer, bits 0-14 contain the word address which holds or will receive the byte. Bit 15 specifies which half (0 left, 1 right).

Change a Default Directory (.DIR)

You can include an optional directory specifier in all file name arguments to system commands. A file name without a specifier is taken to be a file in the current or default directory. During system initialization, or a bootstrap, the default directory becomes the master device.

This system command defines a different directory as the default directory. If the new default directory has not been initialized, this call will initialize it.

Required input to this command is as follows:

AC0 - Bytepointer to new directory name string, terminated by a null.

The format of this call is :

```
.SYSTM
.DIR
error return
normal return
```

If the error return is taken, the current default directory definition remains unchanged and AC2 contains one of the following error codes:

AC2	Mnemonic	Meaning
1	ERFNM	Illegal file name.
12	ERDLE	Directory does not exist.
27	ERSPC	Out of disk space.
36	ERDNM	Device or directory not in system.
51	ERNMD	Attempt to initialize too many directories at one time (not enough DCB's specified at SYSGEN).
52	ERIDS	Illegal directory specifier.
53	ERDSN	Directory specifier unknown.
57	ERLDE	Link depth exceeded.
101	ERDTO	Ten-second disk timeout occurred.
112	EROVF	System stack overflow due to excessive number of chained directory specifiers.

Release a Directory or Device (.RLSE)

Use this command to dissociate a directory or device from the system, and prevent further I/O with it.

```
.SYSTM
.RLSE
error return
normal return
```

On entry to the system, AC0 contains a bytepointer to a directory or device specifier. If the normal return to the user program is taken, it is guaranteed that 1) all I/O activity to and from the device or directory has subsided and 2) no further access can occur until the device or directory has been initialized.

In the case of a removable media directory device, .RLSE should be issued before the pack is physically removed from the unit. (You normally do this with the CLI RELEASE command.) All files within a directory must be closed before that directory may be released. Release of a master directory causes all directories to be released. The master directory is that directory which was specified in a .BOOT/BOOT command or in response to the BOOT query and is available to you via the .MDIR call. After you have released the master directory, you can rebootstrap the system without cycling the disk drive through its load sequence.

If the error return to your program is taken, AC2 will contain one of the following error codes:

AC2	Mnemonic	Meaning
1	ERFNM	Illegal file name.
31	ERSEL	Unit improperly selected.
36	ERDNM	Device not in system.
56	ERDIU	Directory in use.
66	ERDNI	Directory not initialized.
101	ERDTO	Ten-second disk timeout occurred.
114	ERNIR	Attempted release of a unit containing an open file.

Get Current Default Directory/Device Name (.GDIR)

This call returns the name of the current default directory or device. This name is followed by a null (e.g., DP0), not the colon delimiter.

Required input to this call is:

AC0 - Bytepointer to 13_8 -byte area to receive the default directory/device name.

The call format is:

```
.SYSTM
.GDIR
error return
normal return
```

The first 12_8 bytes will contain the name (with trailing nulls, if necessray); byte 13_8 will contain a null terminator. One possible error may result from this call.

AC2	Mnemonic	Meaning
33	ERRD	Attempted to read into system area.

Create a Directory (.CDIR)

This command makes an entry in a diskette's file directory for a directory. If the entry already exists, the error return (ERCRE) is taken.

Required input to .CDIR is:

AC0 - Bytepointer to the directory name.

The format is:

```
.SYSTM
.CDIR
error return
normal return
```

Possible error returns from a .CDIR command are:

AC2	Mnemonic	Meaning
1	ERFNM	Illegal directory name.
11	ERCRE	Attempt to create an existent directory.
53	ERDSN	Directory specifier unknown.
55	ERDDE	Attempt to create a directory within a directory.
57	ERLDE	Link depth exceeded.
66	ERDNI	Directory not initialized.
101	ERDTO	Ten-second disk timeout occurred.

Get the Current Operating System Name (.GSYS)

This call returns the name of the currently executing operating system, its two-character extension (if any), and a null terminator. Required input to this call is:

AC0 - Bytepointer to 15_8 -byte area.

The format of the call is:

```
.SYSTM
.GSYS
error return
normal return
```

The first 12_8 bytes will contain the system name, and bytes 13 and 14 will contain the two-character .SV extension. Byte 15 will contain a null terminator. One possible error return may result from this call:

AC2	Mnemonic	Meaning
33	ERRD	Attempt to read or write into system area.

Get the Logical Name of the Master Device (.MDIR)

Because you can bootstrap from one of several devices, the current master device might not be the master device which was defined at system generation. Use this call to determine the name of the current master device. Required input to this command is:

AC0 - Bytepointer to 13_8 byte area to receive the logical device name.

The format of this call is:

```
.SYSTM
.MDIR
error return
normal return
```

The first 12_8 bytes will contain the name (with trailing nulls, if necessary); byte 13 will contain a null terminator. One error return is possible:

AC2	Mnemonic	Meaning
33	ERRD	Attempt to read or write into system area.

FILE MAINTENANCE COMMANDS

Use the file maintenance commands to enter new file names into a file directory and to perform file maintenance. Each file maintenance command requires you to specify the file name(s) by means of a bytepointer to the file name. The file name is stored as a character string. Each string must consist of characters packed left to right (.TXTM 1) with the high-order bit of each byte equal to 0. The string must have a terminating byte containing one of the following characters: null (000), form feed (014), carriage return (015), or space (040).

If you want to specify an extension, separate it from the filename with a period (.). For example, the word at location BPTR contains a bytepointer to a properly specified file name, MYFILE.SR.

```
BPTR:   2*NAME
        TXTM 1
NAME:   .TXT *MYFILE.SR*
```

File names can include chained directory specifiers.

If you attempt to create a file with the same name as a device in the current system (e.g., $LPT), the system will treat the command as a no-op and take the normal return.

Create a Contiguously Organized File with all Data Words Zeroed (.CCONT)

This command creates a contiguously organized file with all data words initialized to zero. If you create a file whose name exists as a link entry, and if no resolution file exists for this link entry, the resolution file will be created. If the resolution file already exists, the error return (ERCRE) is taken. Required input to this call is:

AC0 - Bytepointer to the file name

AC1 - Number of disk blocks in the file

The call format is:

```
.SYSTM
.CCONT
error return
normal return
```

Possible error returns from a .CCONT call are:

AC2	Mnemonic	Meaning
1	ERFNM	Illegal file name.
11	ERCRE	Attempt to create an existent file.
27	ERSPC	Insufficient disk space to create a SYS.DR entry for this file.
46	ERICB	Insufficient number of free contiguous disk blocks available to create the file.
53	ERDSN	Directory specifier unknown.
57	ERLDE	Link depth exceeded.
66	ERDNI	Directory not initialized
101	ERDTO	Ten-second disk timeout occurred.

Create a Contiguously Organized File with No Zeroing of Data Words (.CONN)

This command creates a contiguously organized file with no zeroing of data words. If you create a file whose name exists as a link entry, and if no resolution file exists for this link entry, the resolution file will be created. If the resolution file already exists, the error return (ERCRE) is taken. Required input to this call is:

AC0 - Bytepointer to filename.

AC1 - Number of disk blocks in the file.

The call format is:

```
.SYSTM
.CONN
error return
normal return
```

Possible error returns from a .CONN call are:

AC2	Mnemonic	Meaning
1	ERFNM	Illegal file name.
11	ERCRE	Attempt to create an existent file.
27	ERSPC	Insufficient disk space to create a SYS.DR entry for this file.
46	ERICB	Insufficient number of free contiguous disk blocks available to create the file.
53	ERDSN	Directory specifier unknown.
57	ERLDE	Link depth exceeded.
66	ERDNI	Directory not initialized.
101	ERDTO	Ten-second disk timeout occurred.

Create a Randomly Organized File (.CRAND)

This command makes an entry for the file name of a randomly organized file in the system file directory. If you create a file whose name exists as a link entry, and if no resolution file exists, the resolution file will be created. If the resolution file already exists, the error return (ERCRE) is taken. Required input to this call is:

AC0 - Bytepointer to the file name

The call format is:

```
.SYSTM
.CRAND
error return
normal return
```

Possible errors resulting from a .CRAND call are:

AC2	Mnemonic	Meaning
1	ERFNM	Illegal file name.
11	ERCRE	Attempt to create an existing file.
27	ERSPC	Insufficient disk space to create the file.
53	ERDSN	Directory specifier unknown.
57	ERLDE	Link depth exceeded.
66	ERDNI	Directory not initialized.
100	ERMDE	Error detected in MAP.DR of nonmaster device.
101	ERDTO	Ten-second disk timeout occurred.

Create a Sequentially Organized File (.CREAT)

In DOS, this call is functionally identical to .CRAND above.

Delete a File (.DELET)

This command deletes a file and its entry in the system file directory. You cannot delete link entry names via this call. If you attempt to delete a link entry name, its resolution file will be deleted unless 1) either the link access or resolution entry attributes words contain the permanent attribute, in which case error ERDE1 is returned, or 2) resolution file doesn't exist, returning ERDLE. Required input to this call is:

AC0 - Bytepointer to filename

The call format is:

```
.SYSTM
.DELET
error return
normal return
```

Possible errors resulting from a .DELET command are:

AC2	Mnemonic	Meaning
1	ERFNM	Illegal file name.
12	ERDLE	Attempt to delete a nonexistent file.
13	ERDE1	Attempt to delete a permanent file.
53	ERDSN	Directory specifier unknown.
56	ERDIU	Directory in use.
57	ERLDE	Link depth exceeded.
60	ERFIU	File in use.
66	ERDNI	Directory not initialized.
100	ERMDE	Error detected in MAP.DR of nonmaster device.
101	ERDTO	Ten-second disk timeout occurred.
102	ERENA	No linking allowed (N attribute).

Rename a File (.RENAM)

This command renames a file. AC0 must contain a bytepointer to the current name of the file, and AC1 must contain a bytepointer to a new name. The format of the .RENAM command is:

```
.SYSTM
.RENAM
error return
normal return
```

After a normal return, the old name no longer exists in the file directory. Possible errors resulting from a .RENAM command are:

AC2	Mnemonic	Meaning
1	ERFNM	Illegal file name.
11	ERCRE	Attempt to create an existent name. (AC1)
12	ERDLE	Attempt to rename a nonexistent file. (AC0)
13	ERDE1	Attempt to rename a permanent file. (AC0)
35	ERDIR	Files specified on different directories.
53	ERDSN	Directory specifier unknown.
60	ERFIU	File in use.
66	ERDNI	Directory not initialized.
101	ERDTO	Ten-second disk timeout occurred.

Get the Current File Pointer (.GPOS)

Use this call when you want to determine the next character position within a file where system processing will occur. DOS indicates a relative character position within a file by a double-precision bytepointer. This is a two-word bytepointer, containing the high-order portion of the byte address in the first word, and the low-order portion of the byte address in the bits 0-15 of the second word. Bit 15 of the second word indicates the byte selection (left or right):

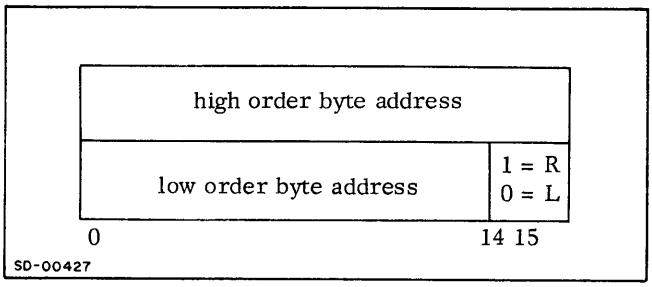


Figure 3-2. Double-precision Bytepointer

The format of this call is:

```
.SYSTM
.GPOSn          ;n IS THE CHANNEL NUMBER OF
                ;THE FILE
error return
normal return
```

Information is returned in the following format:

AC0 - high-order portion of bytepointer.

AC1 - low-order portion of bytepointer.

Zero is returned if a non-disk file is opened on channel n. The following error returns are possible:

AC2	Mnemonic	Meaning
0	ERFNO	Illegal channel number.
15	ERFOP	Attempt to reference an unopened file.

Set the Current File Pointer (.SPOS)

This call sets the current system file pointer to a new character position for future system file operations. DOS indicates the relative character position within a file by the double-precision byte-pointer described in .GPOS. For a magnetic tape, you can specify only position 0 (the file starting location).

This call enables you to randomly access characters and lines within any block of a given file. You can read a character after writing or rewriting it by simply backing up the pointer to its previous position.

If you extend the file pointer beyond the end of the file, DOS automatically extends the length of the file. If the file is contiguous and cannot be extended, a file position error will be signaled when you try to move the pointer past the end of file.

Required input to this call is:

AC0 - high-order portion of bytepointer.

AC1 - low-order portion of bytepointer.

The call format is:

```
.SYSTM
.SPOSn          ;n IS THE FILE's CHANNEL NUMBER
error return
normal return
```

The following error returns are possible:

AC2	Mnemonic	Meaning
0	ERFNO	Illegal channel number.
15	ERFOP	Attempt to reference an unopened file.
64	ERSCP	File position error.

Get the Current File's Directory Status (.RSTAT/STAT)

Use this system call to get a copy of the current directory status information for a file. This call writes a copy of the 22_8 word UFD (as it exists on disk) into the area you specify.

You can then access this information via the indicated displacements defined below. If the file is open, however, the information returned is a snapshot of the UFD as it existed on disk at the time of the most recent .CLOSE or .UPDAT.

System call .RSTAT fetches a copy of the resolution file's UFD if the input argument file name is a link; call .STAT copies the UFD of the argument file whether it is a link, resolution file, or any other file.

Following is a template of a file UFD with displacement mnemonics:

Relative Location or Displacement	Mnemonic	Contents
00000-000004	UFTFN	File name (ASCII file number for open tape file).
000005	UFTEX	Extension.
000006	UFTAT	File attributes.
000007	UFTLK	Link access attributes.
0000010	UFTBK	Number of the last block in the file.
000011	UFTBC	Number of bytes in the last block.
000012	UFTAD	Starting logical block address of the file (the random file index for ran-don files).
000013	UFTAC	Year/day last accessed.
000014	UFTYD	Year/day created, updated, or closed after write.
000015	UFTHM	Hour and minute the file was created, updated, or closed after write.
000016	UFTP1	UFD temporary.
000017	UFTP2	UFD temporary.
000020	UFTUC	User count (1B0 = .EOPEN, or .APPEND or .TOPEN; 1B1 = .OPEN)
000021	UFTDL	DCT link; device code.

Link UFDs assign mnemonics UFLAD and UFLAN to words 7 and 14$_8$. In link UFDs, words 7 - 13 and 14 - 20 are reserved for an alternate directory specifier (if any) and an alias (if any) respectively. Required input to these calls is:

AC0 - Bytepointer to file name string.

AC1 - Starting address of 22$_8$ word UFD data area.

The formats for the calls are:

```
.SYSTM                    .SYSTM
.STAT          or         .RSTAT
error return              error return
normal return             normal return
```

Possible errors resulting from these calls are:

AC2	Mnemonic	Meaning
1	ERFNM	Illegal file name.
12	ERDLE	File does not exist.
33	ERRD	Attempt to read or write into system file space.
36	ERDNM	Device not in system.
53	ERDSN	Directory specifier unknown.
57	ERLDE	Link depth exceeded (.RSTAT only).
66	ERDNI	Directory not initialized.
101	ERDTO	Device timeout.

Get the File Directory Information for a Channel (.CHSTS)

This system call returns a copy of current directory status information for whatever file is currently opened on a specified channel. Directory status information is returned as a copy of the 22$_8$ word UFD, as described in .STAT, except that it shows file status as of last file I/O (by the system, not the user) of this channel. Whereas .STAT/.RSTAT show status, on disk, as of the last open or close.

The required input to .CHSTS is:

AC0 - Starting address of data area. This area must be at least 22$_8$ words long.

The format is:

```
.SYSTM
.CHSTSn      ;n IS THE FILE'S CHANNEL NUMBER
error return
normal return
```

Possible errors resulting from .CHSTS are:

AC2	Mnemonic	Meaning
0	ERFNO	Illegal channel number.
15	ERFOP	No file opened on the given channel.
33	ERRD	Attempt to read into system area.
101	ERDTO	Ten-second disk timeout occurred.

Update the Current File Size (.UPDAT)

This call allows you to update the size information in a file's UFD while the file is open. The UFD contains a file's size, creation date, occurrence date, and attribute information. Specifically, this call updates information in UFTBK and UFTBC in the disk UFD for the file opened on a specified channel, and it flushes all system buffers to ensure that the file contains all information that you have written into it.

This call is particularly useful when a file is open for a long time. Any file that is open during a system failure will have inaccurate size information in its UFD, and you will be unable to read it. By .UPDATing the file frequently you keep its UFD current, and minimize the amount of data which could be lost.

The format of this call is as follows:

```
.SYSTM
.UPDAT n      ;n IS THE FILE'S CHANNEL NUMBER
error return
normal return
```

The following conditions may cause an error return:

AC2	Mnemonic	Meaning
0	ERFNO	Illegal channel number.
15	ERFOP	File not opened.
101	ERDTO	Ten-second disk timeout occurred.

FILE ATTRIBUTE COMMANDS

File attribute commands allow you to check or change the current attributes of a file; you can also use them to check device characteristics. The bit settings of AC0 determine the file attributes; AC1 contains the device characteristics of the file.

Change File Attributes (.CHATR)

This command causes a file's access attributes (or resolution entry link attributes, as viewed from a link entry) to be changed by the contents of AC0. If a link user or a user who has opened via .ROPEN issues .CHATR, his copy of the file attributes is temporarily changed until he closes the file; the true resolution entry attributes persist. You must open a file (OPEN or .OPEN) before you can change its attributes. The format of the .CHATR command is:

```
.SYSTM
.CHATRn          ;n IS THE FILE'S CHANNEL NUMBER
error return
normal return
```

When the .CHATR command is given, AC0 must contain an attribute word having the appropriate bit set for every attribute desired. Set the contents of AC0 according to the bit/attribute relationship given in the following table:

Bit	Symbolic Attribute	Mnemonic	Meaning
1B0	R	ATRP	Read-protected file; cannot be read.
1B1	A	ATCHA	Attribute-protected file. No attribute can ever be changed after this bit is set.
1B2	S	ATSAV	Save file (core image file). Once set, this attribute cannot be removed.
1B7	N	ATNRS	No link resolution allowed.
1B9	?	ATUS1	First user-definable attribute for the file.
1B10	&	ATUS2	Second user-definable attribute, for the file.
1B14	P	ATPER	Permanent file; cannot be deleted or renamed.
1B15	W	ATWP	Write-protected; cannot be written.

The following are disk file characteristics; they are assigned when a file is created, and cannot be changed.

Bit	Characteristic	Mnemonic	Meaning
1B3	L	ATLNK	Link entry.
1B5	D	ATDIR	Directory file.
1B6	-	ATRES	Link resolution file (temporary). Other file attributes persist for the duration of the open.
1B12	C	ATCON	Contiguous file
1B13	D	ATRAN	Random file.

Possible errors resulting from a .CHATR command are:

AC2	Mnemonic	Meaning
0	ERFNO	Illegal channel number.
14	ERCHA	Illegal attempt to change file attributes (file has A attribute).
15	ERFOP	Attempt to change attributes of an unopened file.
101	ERDTO	Ten-second disk timeout occurred.

Get File Attributes and Characteristics (.GTATR)

Use this command to obtain the attributes of a file, and device characteristics. You must give the file's channel number. Before you can obtain attributes the file must be opened (see .OPEN). The format of this command is:

```
.SYSTM
.GTATR n     ;n IS THE FILE'S CHANNEL NUMBER
error return
normal return
```

Upon return, AC0 contains the file attributes. The bit positions that specify attributes are given with the .CHATR command. AC1 contains the device characteristics of the file; these pertain to files on reserved devices; e.g., $LPT. These do not reflect the characteristic inhibit mask supplied when the file was opened. The bit/characteristics correspondence used to interpret the bit configuration returned in AC1 is shown below:

Bit	Mnemonic	Meaning
1B1	DCC80	80-column device.
1B2	DCLTU	Device changes lower case ASCII to upper case.
1B3	DCFFO	Device requiring form feeds on opening.
1B4	DCFWD	Full word device (reads or writes more than a byte.)
1B6	DCLAC	Output device requiring line feeds after carriage returns.
1B7	DCPCK	Input device requiring a parity check; output device requiring parity to be computed.
1B8	DCRAT	Output device requiring a rubout after every tab.
1B9	DCNAF	Output device requiring nulls after every form feed.
1B10	DCKEY	CTRL Z end of file, backslash line delete and rubout character delete suppressed for a keyboard input device.
1B11	DCTO	Equal leader and trailer for $TTP and $PTP.

Bit	Mnemonic	Meaning
1B12	DCCNF	Output device without form feed hardware.
1B13	DCIDI	Device requiring operator intervention.
1B14	DCCGN	Output device without tabbing hardware.
1B15	DCCPO	Output device requiring leader and trailer.

Possible errors resulting from a .GTATR command are:

AC2	Mnemonic	Meaning
1	ERFNO	Illegal channel number.
15	ERFOP	Attempt to get attributes of an unopened file.
101	ERDTO	Ten-second disk timeout occurred.

LINK COMMANDS

As described in Chapter 2, DOS permits you to link files in one directory to files in other directories. The following series of system commands allows you to do these things.

Create a Link Entry (.LINK)

This call creates a link entry in the current directory to a file in the same or in another directory. The link entry may have the same name as the resolution entry, or not; if not, the link entry name is an alias. No attributes restrict a link when you create it, but it cannot reach the resolution file without satisfying both the link entry and the file access attributes of the resolution entry. You can alter the link access rights (but not the file access rights) of each resolution entry by the .CHLAT call.

Required inputs to .LINK are as follows. AC0 must contain a bytepointer to the name of the link entry. AC1 must either be set to zero or contain a bytepointer to a character string. If zero, the link entry must have the same name as the resolution entry, and both must be on the same diskette. That is, if link entry Z is in a directory residing on DP0, link references will be resolved to resolution entry Z on DP0. If the resolution entry name is on some other diskette, (or in a directory on the same diskette) or if the link entry name is an alias (or both), AC1 must contain a bytepointer to the alternate device and/or alias name string. If you place a colon in this string, characters preceding the colon are considered to be an alternate device specifier. If there is no colon in the character string, the string is considered to be an alias name string to be resolved on the same device.

Typical examples of alternate directory/alias name strings are as follows:

Link (AC0)	Character String (AC1)	Meaning
ASM.SV	ASM.SV	Resolution entry is ASM.SV on the current diskette.
ASM.SV	OASM.SV	ASM.SV is linked to file OASM.SV on the current diskette.
ASM.SV	SAM:ASM.SV	ASM.SV is linked to file ASM.SV in directory SAM, on the current diskette.
ASM.SV	DP0:OASM.SV	ASM.SV is linked to file OASM.SV on DP0.

In summary, required inputs to .LINK are:

AC0 - Bytepointer to link entry name string.

AC1 - Zero if link and resolution entry have same name on the current diskette. Bytepointer to name string if the link entry has an alias name, or is on another diskette.

The format of the call is:

```
.SYSTM
.LINK
error return
normal return
```

The error return is taken and an appropriate error code is given in AC2 in the following cases:

AC2	Mnemonic	Meaning
1	ERFNM	Illegal file name.
11	ERCRE	Link entry name already exists.
27	ERSPC	Insufficient disk space to create SYS.DR entry.
53	ERDSN	Directory specifier unknown.
66	ERDNI	Directory not initialized.
101	ERDTO	Disk timeout occurred.

Delete a Link Entry (.ULNK)

This call deletes a link entry (created earlier by LINK or .LINK) in the directory pointed to by the link entry name. This call does not delete other links of the same name in other directories. You must be sure that the link entry being deleted is not between other links

and the resolution entry; if it is, the deletion of this link will render these more remote links unresolvable.

Required input to this call is:

AC0 - Bytepointer to the link entry name string.

The format of this call is:

```
.SYSTM
.ULNK
error return
normal return
```

The error return is taken in the following cases:

AC2	Mnemonic	Meaning
1	ERFNM	Illegal file name.
12	ERDLE	Attempt to delete a nonexistent file.
53	ERDSN	Directory specifier unknown.
66	ERDNI	Directory not initialized.
75	ERNLE	Not a link entry.
101	ERDTO	Disk timeout occurred.

Change Link Access Entry Attributes (.CHLAT)

This command causes a link user's copy of the link access attributes word to be changed by the contents of AC0. When a file is opened via a link entry, the attributes of the file as seen by the link user are a composite of the resolution entry's file attributes and the user's copy of the link access entry attributes. When a file is created, the link entry access attributes are 0 (i.e., there are none).

Required input to .CHLAT is:

AC0 - File attributes word (see .CHATR)

The format is:

```
.SYSTM
.CHLAT n        ;n IS THE CHANNEL NUMBER
error return
normal return
```

The attribute word input in AC0 is identical to .CHATR. Possible errors resulting from a .CHLAT command are:

AC2	Mnemonic	Meaning
0	ERFNO	Illegal channel number.
14	ERCHA	Resolution entry is attribute-protected (has A attribute).
15	ERFOP	File not opened.
101	ERDTO	Ten-second disk timeout occurred.

INPUT/OUTPUT COMMANDS

Use these commands for all system I/O. Generally, you can do nothing with a file until you have opened it and given it a channel number with one of the .OPEN commands: .OPEN, .EOPEN, .ROPEN, .APPEND, or .MTOPD. Each open command opens a file for a specific operation, and you specify the file's channel number in the second field of the command word.

When DOS executes your I/O command, and detects an error, it will retry the command 10 times before reporting the error with code ERFIL.

DOS provides four basic modes for reading and writing files. The modes are:

line
sequential
direct block
free form

This section presents the calls for these modes in order.

Line and sequential* mode are generally used for ASCII character strings and binary files, respectively. Direct-block I/O allows you to transfer a contiguous group of disk blocks without a system buffer. Free form I/O allows you to read or write blocks of data in mag tape files.

In line mode, the system assumes that the data you want to read or write consists of ASCII character strings, terminated by either carriage return, form feed or null characters. File data is processed line by line in sequence from the beginning of the file to its end.

In line mode, the system handles all device-dependent editing at the device driver level. For example, line feeds are ignored on paper tape input devices and are supplied after carriage returns to all paper tape output devices. Furthermore, reading and writing never require byte counts, since reading continues until a terminator is read and writing proceeds until you write a terminator. The line mode commands are Read a Line (.RDL) and Write a Line (.WRL).

*The System Library (SYS.LB) provides a module to speed up line and sequential mode operations. This module is called the Buffered I/O Package; it is described in Application Note #17-000003.

The second mode is unedited sequential mode. In this mode, data is transmitted exactly as read from or written to the file or device. Always use this mode for processing binary files. To use sequential mode, your program must specify the byte count necessary to satisfy your read or write request. The sequential mode commands are Read Sequential (.RDS) and Write Sequential (.WRS).

In line or sequential modes, your position within a file is always the position at the end of the last line or sequential mode call. The first read or write occurs at the beginning of the file, after you create it.

The third mode, direct block I/O, allows you to transfer a continuous group of blocks in a random or contiguous file without using a system buffer. Core locations used in the transfer are sequential; only entire blocks of core area participate in the transfer. You can transfer only an unbroken series of relative block numbers. That is, third, fourth, and fifth blocks in the file may be processed in a single call, but not the third, fifth, and sixth blocks. You can execute direct-block I/O with .RDB and .WRB.

Finally, free form I/O permits you to read or write blocks of data to magnetic tape. With free form I/O, you can read or write from 1- to 4096- word data records, space forward or backward from 1 to 4096 data records, or to the start of a new data file, and read the transport status word. You must specify free-form I/O when you open a file (.MTOPD), and direct its format (.MTDIO).

Open a File (.OPEN)

Before other I/O commands can be used, a file must be linked to a DOS channel number. .OPEN links a file with a channel number and makes the file available to any user for both reading and writing. The .OPEN command does not guarantee exclusive use of the file; other users may also have opened the file via .OPEN and modified its contents. All users of a file must close the file before it can be deleted or renamed. Except for deletions, disk files never shrink in size.

You must pass a bytepointer to the file name in AC0, and pass a characteristic inhibit mask in AC1. For every bit you set in the mask word, the corresponding device characteristic (as defined previously in the .GTATR command) is inhibited. Each characteristic will be inhibited for the duration of the .OPEN. For example, if you want to read an ASCII tape without parity from the paper tape reader, you can inhibit parity checking by the following:

```
        LDA     0,READR
        LDA     1,MASK
        .SYSTM
        .OPEN   3
          .
          .
          .
READR:  .+1*2
        .TXT    *PTR*
MASK:   DCPCK           ;PARITY
                        ;CHARACTERISTIC
```

In general, you will want to preserve all device characteristics defined by the system. To preserve them, insert a SUB 1, 1 instruction before the system call instruction.

The format of the .OPEN command is:

```
.SYSTM
.OPEN n         ;IS THE FILE'S CHANNEL NUMBER
error return
normal return
```

If the file opened requires leader, it will be output on the .OPEN. If the file opened requires intervention, the message "LOAD filename, STRIKE ANY KEY." will be output. Possible errors from a .OPEN command are:

AC2	Mnemonic	Meaning
0	ERFNO	Illegal channel number.
1	ERFNM	Illegal file name.
12	ERDLE	File does not exist.
21	ERUFT	Attempt to use channel already in use.
27	ERSPC	File space exhausted.
31	ERSEL	Unit improperly selected.
36	ERDNM	Device not in system.
53	ERDSN	Directory specifier unknown.
57	ERLDE	Link depth exceeded.
60	ERFIU	File opened for exclusive use (.EOPEN)
66	ERDNI	Directory not initialized.
101	ERDTO	Ten-second disk timeout occurred.
102	ERENA	No linking allowed (N attribute).
111	ERDOP	Attempted open of an open tape file.

Open a File for Exclusive Write Access (.EOPEN)

This command provides write access to a file for a single user only. Thus only one user can modify a given file while it is opened via .EOPEN, although more than one user might gain read access to this file. Required input to this call is identical to that given earlier for .OPEN:

AC0 - Bytepointer to file name.

AC1 - Characteristic inhibit mask.

The format of this call is:

```
.SYSTM
.EOPEN n        ;n IS THE FILE'S CHANNEL NUMBER
error return
normal return
```

Possible errors resulting from an .EOPEN command are:

AC2	Mnemonic	Meaning
0	ERFNO	Illegal channel number.
1	ERFNM	Illegal file name.
12	ERDLE	File does not exist.
21	ERUFT	Attempt to use channel already in use.
31	ERSEL	Unit improperly selected.
36	ERDNM	Device not in system.
53	ERDSN	Directory specifier unknown.
57	ERLDE	Link depth exceeded.
60	ERFIU	File already opened for writing.
66	ERDNI	Directory not initialized.
101	ERDTO	Ten-second disk timeout occurred.
102	ERENA	No linking allowed (N attribute).
111	ERDOP	Attempted open of a file which is already open.

Open a File for Reading Only (.ROPEN)

This call can be used by one or more users to gain read-only access to a file. This type of access may be granted to a file which is also currently open by either .EOPEN, .OPEN, or another .ROPEN. Thus several users may access a file for reading only while one of those users has write access privileges to the file. All users must have closed the file before it can be deleted or renamed. Required input to this call is identical to that given earlier for .OPEN.

AC0 - Bytepointer to file name

AC1 - Characteristic inhibit mask

The format of this call is:

```
.SYSTM
.ROPEN n        ;n IS THE FILE'S CHANNEL NUMBER
error return
normal return
```

Possible errors resulting from an .ROPEN command are:

AC2	Mnemonic	Meaning
0	ERFNO	Illegal channel number.
1	ERFNM	Illegal file name.
12	ERDLE	File does not exist.
21	ERUFT	Attempt to use channel already in use.
31	ERSEL	Unit improperly selected.
36	ERDNM	Device not in system.
53	ERDSN	Directory specifier unknown.
57	ERLDE	Link depth exceeded.
66	ERDNI	Directory not initialized.
101	ERDTO	Ten-second disk timeout occurred.
102	ERENA	No linking allowed (N attribute).
111	ERDOP	Attempt to open an open tape file.

Open a File for Appending (.APPEND)

.APPEND is identical to .EOPEN, except that it opens a file specifically for appending. In a BATCH environment, if a user program is to output to SYSOUT, SYSOUT must be opened for appending (not simply opened). As with .EOPEN, AC0 must contain a bytepointer to the file name, and AC1 must contain the device characteristic inhibit mask. The format of the call is:

```
.SYSTM
.APPEND n       ;n IS THE FILE'S CHANNEL NUMBER
error return
normal return
```

For peripheral devices such as the line printer, the call is identical to .EOPEN. For a diskette, the file is opened, and whatever you write to that file is appended to it. On a magnetic tape device, the tape file is opened and read to the double end-of-file (EOF); writing takes place from that point. If you try to read past the current EOF, (e.g., after an append) the system will return error code EREOF.

If you are running a subsystem under DOS, you can use this call to append to the same output file. Possible errors resulting from .APPEND are:

AC2	Mnemonic	Meaning
0	ERFNO	Illegal channel number.
1	ERFNM	Illegal file name.
3	ERICD	Illegal command for device.
12	ERDLE	File does not exist.
21	ERUFT	Attempt to use channel already in use.
31	ERSEL	Unit improperly selected.
36	ERDNM	Device not in system.
53	ERDSN	Directory specifier unknown.
57	ERLDE	Link depth exceeded.
60	ERFIU	File in use.
66	ERDNI	Directory not initialized.
101	ERDTO	Ten-second disk timeout occurred.
102	ERENA	No linking allowed (N attribute).
111	ERDOP	Attempt to open a file that is already open.

Open a Magnetic Tape Unit for Free Format I/O

The commands .OPEN, .EOPEN, .ROPEN and .APPEND cannot open a file for free format I/O. See .MTOPD and .MTDIO at the end of Input/Output Commands, this chapter.

Get the Number of a Free Channel (.GCHN)

This call enables you to obtain the number of a free channel. You can then open a file on this channel via one of the open file calls. .GCHN does not open a file on a free channel; it merely indicates a channel that is free at the moment. Occasionally, in a multitask environment, you will find that the channel .GCHN indicated is no longer free when you issue your open. If this happens, you will receive error return ERUFT (channel in use) in response to your open. Try re-issuing the call .GCHN, to discover another free channel.

The format of the call is:

```
.SYSTM
.GCHN
error return
normal return
```

Upon a normal return, the information is returned in AC2:

AC2 - Free channel number

One possible error return may occur:

AC2	Mnemonic	Meaning
21	ERUFT	No channels are free.

Close a File (.CLOSE)

You must close a file after use to update its UFD directory information, or to release its directory or device. When you close a file, its channel number becomes available for other I/O. The format of the .CLOSE command is:

```
.SYSTM
.CLOSE n      ;CLOSE CHANNEL n
error return
normal return
```

If the file closed is the high-speed punch, the required trailer will be output on the .CLOSE.

Possible errors resulting from .CLOSE are:

AC2	Mnemonic	Meaning
0	ERFNO	Illegal channel number.
15	ERFOP	Attempt to close a channel not in use.
101	ERDTO	Ten-second disk timeout occurred.

Close all Files (.RESET)

This command closes all open files, and also writes any partially-filled buffers before closing each file. You can issue .RESET in a multitask environment only when no other task is using a channel. The format of the command is:

```
.SYSTM
.RESET
error return
normal return
```

One error return from this command is taken.

AC2	Mnemonic	Meaning
101	ERDTO	Ten-second disk timeout occurred.

Read a Line (.RDL)

This command reads an ASCII line that was written with even parity. AC0 must contain a bytepointer to the starting byte address within the user area into which the line will be read. This area should be 133_{10} bytes long.

Reading terminates normally after the system has encountered either a carriage return, form feed, or null, and transmitted it to you. The system stops reading and takes the error return if it transmits 132 characters without detecting a carriage return, form feed, or null as the 133rd character, or upon detection of a parity error, or upon an end-of-file.

To indicate an end-of-file from the console, enter CTRL Z (unless you have masked DCKEY; (see .GTATR). When the system reads a rubout, it deletes the preceding character from the keyboard stream (unless you masked DCKEY when opening the device).

The number of bytes read (including the carriage return, form feed, or null) is always returned in AC1. If the read terminates because of a parity error, the character having incorrect parity is stored as the last character read; the parity bit is cleared. You can always compute the bytepointer to the bad character as (AC0)*+(AC1)-1. Note: by convention, (AC0) means the contents of AC0.

The format of the call is:

AC0 - Bytepointer to receiving buffer.

```
.SYSTM
.RDL n          ;READ FROM CHANNEL n
error return
normal return
```

AC1 - Byte count of read.

Possible errors resulting from a .RDL command are:

AC2	Mnemonic	Meaning
0	ERFNO	Illegal channel number.
3	ERICD	Illegal command for device.
6	EREOF	End of file.
7	ERRPR	Attempt to read a read-protected file.
15	ERFOP	Attempt to reference a file not opened.
22	ERLLI	Line limit (133 nonterminator characters) exceeded.
24	ERPAR	Parity error.
30	ERFIL	File read error.
33	ERRD	Attempt to read into system area.
34	ERDIO	File accessible by direct block I/O only.
101	ERDTO	Ten-second disk timeout occurred.

Read Sequential (.RDS)

Sequential mode transmits data exactly as read from the file, unless read from a system console. When reading sequentially from a system console, parity bits are set to zero. Note that CTRL Z from the consoles is not recognized as an end-of-file character in this mode. AC0 must contain a bytepointer to the starting byte address within the user area into which the data will be read, and AC1 must contain the number of bytes to be read. Upon detection of an end-of-file, the partial bytecount will be returned in AC1.

```
.SYSTM
.RDS n          ;READ FROM CHANNEL n
error return
normal return
```

Possible errors resulting from a .RDS command are:

AC2	Mnemonic	Meaning
0	ERFNO	Illegal channel number.
3	ERICD	Illegal command for device.
6	EREOF	End of file.
7	ERRPR	Attempt to read a read-protected file.
15	ERFOP	Attempt to reference a file not opened.
30	ERFIL	File read error.
33	ERRD	Attempt to read into system area.
34	ERDIO	File accessible by direct block I/O only.
101	ERDTO	Ten-second disk timeout occurred.

Use of the Card Reader ($CDR) in .RDL and .RDS Commands

When you use the card reader as an input device to .RDL, indicate an end-of-file by punching all rows in column 1 (punch the characters "+", "-", and 0 through 9). Hollerith-to-ASCII translation occurs on a .RDL, not a .RDS. The translation assumes the keypunch codes shown in Appendix B.

A .RDL terminates upon the first trailing blank unless your .OPEN command suppressed DCSTB, causing all 80 characters to be transferred. If all 80 columns are transferred, the system appends a carriage return as the eighty-first character, unless your OPEN command suppressed DCC80 (allowing a maximum of 72 characters to be processed). The system replaces each illegal character with a back slash.

In .RDL calls, columns following the EOF are ignored. The card reader driver permits an unlimited amount of time to elapse until the next card is read. This permits the operator to correct pick errors, or insert new card files. The card reader driver employs double buffering, and premature closing will lose at least one card image; therefore you must wait until the last card or end-of-file has been read to close $CDR.

You can close the reader after it has read an end-of-file card, and reopen it without losing any data. You can also continue card reading. When an end-of-file card is read, the system returns a byte count of 0, and error code EREOF. If you issue another .RDL, the next card will be read normally.

A table of Hollerith-ASCII translation is given in Appendix B.

If .RDS is issued, the card is read in image binary. Each two bytes will be used to read a single column. The packing is done as follows:

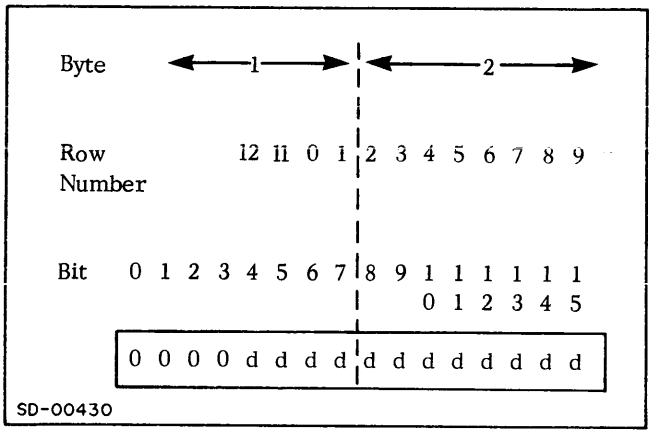


Figure 3-3. Image Binary Card Reading

The "d's" will be 1 for every column punched. In an .RDS, an EOC is signified by a byte pair containing the word 100000. Thus, to read two entire 80-column cards, half a card at a time, you would issue four successive .RDS calls for 82 bytes each. If you requested only 80 bytes for each read, the third .RDS would return the end-of-card word and the first 39 columns of the second card.

Write a Line (.WRL)

This command is the counterpart of .RDL; use it to write a line to an ASCII file. AC0 must contain a bytepointer to the starting byte address within the user area from which characters will be written.

If you have opened the file with .OPEN or .EOPEN, writing starts at the beginning of the file (unless you have moved the file pointer via the .SPOS command). Writing begins at the end of the file if you open it via .APPEND. The system stops writing normally when it detects a null, a carriage return, or a form feed, and abnormally after transmission of 132 (decimal) characters without a carriage return, a null, or a form feed as the 133rd character.

Upon termination AC1 contains the number of bytes written from the user area. The null terminator does not force a carriage return or line feed, and a line feed character will be ignored upon output. Use a carriage return or form feed to generate a line feed upon output (if the device characteristics so dictate).

The format of the .WRL command is:

```
.SYSTM
.WRL n          ;WRITE TO CHANNEL n
error return
normal return
```

Possible errors resulting from the .WRL command are:

AC2	Mnemonic	Meaning
0	ERFNO	Illegal channel number.
3	ERICD	Illegal command for device.
6	EREOF	End-of-file when writing to a contiguous file.
10	ERWPR	Attempt to write to a write-protected file.
15	ERFOP	Attempt to write a file not opened.
22	ERLLI	Line limit (132 characters).
27	ERSPC	Out of disk space.
34	ERDIO	File accessible by direct block I/O only.
101	ERDTO	Ten-second disk timeout occurred.

Write Sequential (.WRS)

The .WRS command is the counterpart of .RDS. Use this sequential mode command to write data exactly as it is in the user area. If you opened the file via .OPEN or .EOPEN, writing starts at the beginning of the file (unless you moved the file pointer by the .SPOS command after opening). If you opened the file via .APPEND, writing starts at the end of the file.

AC0 must contain a bytepointer to the starting address of the data within the user area, and AC1 must contain the number of bytes to be written. The format of the .WRS command is:

```
.SYSTM
.WRSn           ;WRITE TO CHANNEL n
error return
normal return
```

Possible errors resulting from a .WRS command are:

AC2	Mnemonic	Meaning
0	ERFNO	Illegal channel number.
3	ERICD	Illegal command for device.
6	EREOF	End-of-file when writing to a contiguous file.
10	ERWPR	Attempt to write a write-protected file.
15	ERFOP	Attempt to write a file not opened.
27	ERSPC	Out of disk space.
34	ERDIO	File accessible by direct block I/O only.
101	ERDTO	Disk timeout occurred.
113	ERNMC	No outstanding receive request.

Read (or Write) a Series of Disk File Blocks (.RDB, .WRB)

These are the direct block I/O calls. Use these calls in your program to transfer blocks to or from random or contiguous files (.CRAND creates a random file, .CCONT or .CONN a contiguous file). No system buffer is required for the transfer.

Blocks in random and contiguous disk files have a fixed length of 256_{10} words; they are numbered sequentially from 0.

AC0 must contain the beginning core address for the block transfer. AC1 must contain the beginning relative block number in the series to be transferred. The left half of AC2 must contain the number of blocks which will be transferred. If the channel number is set to 77, the right half of AC2 must contain the channel number. The format of the read (.RDB) and write (.WRB) block commands is:

```
.SYSTM
.RDB (.WRB)  ;n IS THE CHANNEL NUMBER
error return
normal return
```

Possible error codes which may be returned are:

AC2	Mnemonic	Meaning
0	ERFNO	Illegal channel number.
3	ERICD	Illegal command for device.
4	ERSVI	Not a randomly or contiguously organized file.
6	EREOF	End of file.
7	ERRPR	File is read-protected (.RDB).
10	ERWPR	File is write-protected (.WRB).
15	ERFOP	File is not opened.
27	ERSPC	Disk space is exhausted.
30	ERFIL	File read error.
33	ERRD	Attempt to read into system area (.RDB).
40	EROVA	File not accessible by direct block I/O.
101	ERDTO	Ten-second disk timeout occurred.

Upon detection of error EREOF or ERSPC, the code is returned in the right byte of AC2; the left byte contains the partial read or write count.

Read (or Write) a Random Record (.RDR or .WRR)

The read and write random record calls allow your program to read (or write) one 64-word record. There are 4 64-word records in a disk block; for the first disk block in a file, these are numbered 0, 1, 2, and 3; for the second block the numbers are 4, 5, 6 and 7, and so on. You need consider record numbers only for the random record calls (to read or write blocks, 4 records at a time, you'd use .RDB/.WRB; to read or write lines you'd use the read/write line or sequential calls).

.RDR reads a random record from the disk file opened on channel n. You must pass the destination core address in AC0, and the record number (record numbers start with 0) in AC1. The disk file must be randomly or contiguously organized.

The read random command allows random reading of records from a file on disk. AC0 must contain a destination core address within the user area, and AC1 must contain the record number. The format of the .RDR command is:

```
.SYSTM
.RDR n          ;READ FROM THE FILE
error return    ;OPENED ON CHANNEL n.
normal return
```

Possible errors resulting from the .RDR commands are:

AC2	Mnemonic	Meaning
0	ERFNO	Illegal channel number.
3	ERICD	Illegal command for device.
6	EREOF	Attempt to read past the end of a contiguous file.
7	ERRPR	Attempt to read a read-protected file. (RDR)
15	ERFOP	No file is open on this channel.
30	ERFIL	File read errors (mag tape or cassette - bad tape).
33	ERRD	Attempt to read into system area.
34	ERDIO	File accessible by direct-block I/O only.
101	ERDTO	Ten-second disk timeout occurred.

Write Random Record (.WRR)

.WRR writes a 64-word record from memory to the randomly - or contiguously organized disk file opened on channel n. Your program must pass the core address in AC0, and the destination record number in AC1. 64 words will be written to the record number in the file, starting from the address passed in AC0.

```
.SYSTM
.WRR            ;WRITE THE FILE
error return    ;OPENED ON CHANNEL n.
normal return
```

Possible errors:

AC2	Mnemonic	Meaning
0	ERFNO	Illegal channel number.
3	ERICD	Illegal command for device.
6	EREOF	Attempt to write past the end of a contiguous file.
10	ERWPR	Attempt to write a write-protected file.
15	ERFOP	Attempt to reference a file not opened.
27	ERSPC	Out of disk space.
34	ERDIO	File accessible by direct block I/O only.
101	ERDTO	Ten-second disk timeout occurred.

Open a Mag Tape Unit for Free Format I/O (.MTOPD)

Before you can read or write in free format on a magnetic tape, the device must be opened and associated with a channel. You must use the .MTOPD command to do this.

When you open a file for free format I/O with .MTOPD it is opened globally; all files on the specified device can be accessed.

This command positions a free format tape to a desired file, since the file name passed to .MTOPD will be of the form MTn:m. The input parameters to .MTOPD are as follows:

AC0 - Bytepointer to magnetic tape file specifier

AC1 - Characteristic inhibit mask (see .GTATR)

Aside from the tape file specifier, these parameters are identical to .OPEN's. If you want to know more about device characteristics, see .OPEN and .GTATR.

The format of the .MTOPD command is:

```
.SYSTM
MTOPD n         ;n IS THE CHANNEL NUMBER
error return
normal return
```

Possible errors resulting from a .MTOPD command are:

AC2	Mnemonic	Meaning
0	ERFNO	Illegal channel number.
1	ERFNM	Illegal file name.
3	ERICD	Illegal command for device.
12	ERDLE	File does not exist.
21	ERUFT	Attempt to use a channel already in use.
27	ERSPC	File space exhausted.
31	ERSEL	Unit improperly selected.
36	ERDNM	Device not in system.

AC2	Mnemonic	Meaning
53	ERDSN	Directory specifier unknown.
57	ERLDE	Link depth exceeded.
66	ERDNI	Directory not initialized.
111	ERDOP	Attempted open of an open file.

Free Format I/O (.MTDIO)

This command gives you a direct interface with magnetic tape units on a machine level. Using .MTDIO, you can read or write data in variable length records from 2 to 4096 words within a data record, space forward or backward from 1 to 4095 data records, or to the start of a new data file, or to perform other similar machine-level operations.

Before you can read or write in free format on a tape unit, you must open the unit for free format I/O with the .MTOPD system command. For more information about the hardware characteristics, see Magnetic Tape, in Programmer's Reference Manual For Peripherals.

The input parameters to .MTDIO are:

AC0 - Core address, if data is to be transferred.

AC1 - Command word, subdivided into the following fields:

bit 0: set to 1 for even parity, 0 for odd parity.

bits 1-3: set to one of the seven command codes which follow:

0 - read (words)
1 - rewind the tape
3 - space forward (over records or over a file of any size up to 4096 records)
4 - space backward (over records or over a file of any size up to 4096 records)
5 - write (words)
6 - write end-of-file (parity: odd for 9 tracks)
7 - read device status word

bits 4-15: word or record count. If 0 on a space forward (or backward) command, and the file is no more than 4096 records, the tape is positioned to the beginning of the next (or previous) file on the tape. If 0 on a read command, words are read until either an end-of-record is detected or 4096 words are read. If 0 on a write command, 4096 words will be written.

AC2 - channel number if n equals 77.

The format of the .MTDIO command is:

```
.SYSTM
.MTDIO n      ;n IS THE CHANNEL NUMBER
error return
normal return
```

Upon a read status command, if no system error is detected, control goes to the normal return and AC2 contains a status word with one or more of the following bits set:

bit 0, error (bit 1, 3, 5, 6, 7, 8, 10, or 14)
bit 1, data late bit 2, tape is rewinding bit 3, illegal command
bit 4, high density if set to 1, otherwise, low density bit 5, parity error bit 6, end-of-tape
bit 7, end-of-file bit 8, tape is at load point bit 9, always set to 1 for 9-track
bit 10, bad tape (or write failure) bit 11, send clock bit 12, first character
bit 13, write-protected or write-locked bit 14, odd character bit 15, unit ready

Figure 3-5. .MTDIO Status Word Bits

When you issue a read, write, space forward, or space backward command, AC1 must contain the number of words written (or read) or the number of records spaced. A word or record count is returned upon a premature end-of-file. The device status word is returned in AC2 upon normal returns and upon detection of hardware errors. (Write end-of-file takes the error return.) Upon detection of a system error, AC2 is set to one of the following:

AC2	Mnemonic	Meaning
0	ERFNO	Illegal channel number.
3	ERICD	Illegal command for device (i.e., improper open).
15	ERFOP	Attempt to reference a file not opened.
40	EROVA	File not accessible by direct I/O.

The following table (Figure 3-6) summarizes the possible returns by .MTDIO and the values returned in AC1 and AC2.

COMMAND	RETURN	AC1	AC2
Any .MTDIO command with a system error detected	Error	Same as input	System error code
Rewind	Normal	Original input lost	Transport Status Word (TSW)
Rewind (tape at load point, etc.)	Error		
Read Status	Normal	Original input lost	TSW (bit zero reset)
Read Status	Error		TSW (bit zero set)
Read, Write, Space Forward, Space Backward; bit 0 in TSW is not set	Normal	Word or record count	TSW
Read, Write, Space Forward, Space Backward; bit 0 in TSW is set	Error (only after 10 retries in read/write)		
Write end-of-file	Error	Original input lost	TSW

SD-00431

Figure 3-6. .MTDIO Values Returned

As with regular magnetic tape I/O, the system will perform 10 read retries before taking the error return. For write errors, the system will perform the following sequence 10 times before taking the error return: backspace, erase a length of tape, and write.

DEVICE ACCESS COMMANDS

Enable User Access of a Device (.DEBL) and

Disable User Access of a Device (.OEBL)

In DOS, these calls are no-ops; they are provided for compatibility with RDOS.

Read the Front Panel Switches (.RDSW)

This system call allows you to read the position of the front panel switches. The switch configuration is returned in AC0. The format of this call is:

```
.SYSTM
.RDSW
error return
normal return
```

The error return is never taken.

CONSOLE I/O COMMANDS

To transfer single characters, with a buffer, between your console and AC0, use commands .GCHAR and .PCHAR. These calls operate like a read or write sequential of one character. They do not affect the column counter, or require special character handling (e.g., of carriage returns). These commands reference $TTI/$TTO; the console is always available to them, and no channel number or open command is required.

Get a Character (.GCHAR)

This command returns in AC0 a character typed from the console. The character (without parity) is right-adjusted in AC0, and the left byte of AC0 is cleared. No channel is required, and the console is always used as input. The format of the .GCHAR command is:

```
.SYSTM
.GCHAR
error return
normal return
```

If no character is currently in the console input buffer, the system waits. One error return is possible:

AC2	Mnemonic	Meaning
3	ERICD	Console not in system.

Put a Character (.PCHAR)

This command transfers a character from AC0 bits 9-15 to the console. The format is:

```
.SYSTM
.PCHAR
error return
normal return
```

One error return is possible:

AC2	Mnemonic	Meaning
3	ERICD	Console not in system.

Get the Input Console Name (.GCIN)

This command returns the name of the current console input device. Required input is a bytepointer to a 6-byte area which will receive the console name; you pass this bytepointer in AC0. The format of the call is:

```
.SYSTM
.GCIN
error return
normal return
```

Errors are:

AC2	Mnemonic	Meaning
33	ERRD	Attempt to read into system area.

Get the Output Console Name (.GCOUT)

This command returns the name of the current output console: $TTO. You must pass a bytepointer in AC0 to the 6-byte area which will receive the console name. The format of the call is:

```
.SYSTM
.GCOUT
error return
normal return
```

One error is possible:

AC2	Mnemonic	Meaning
33	ERRD	Attempt to read into system area.

.SPDA; .SPEA; .SPKL; .TUON; .TUOFF

In DOS, these system calls are no-ops.

MEMORY POINTER COMMANDS

Excluding the Task Scheduler, DOS resides primarily in upper memory. User programs are loaded in lower memory. DOS memory looks essentially as follows:

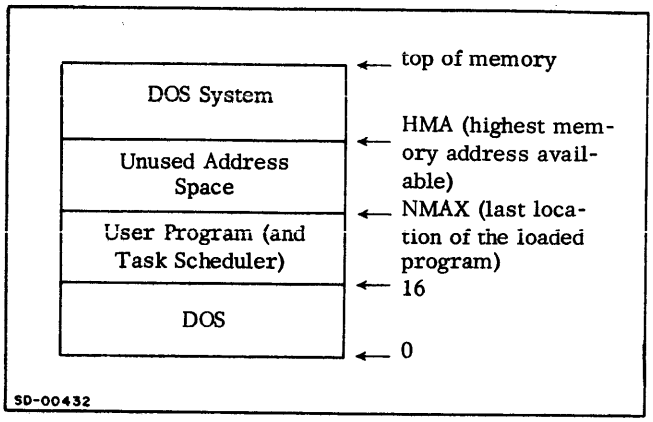


Figure 3-7. Memory Representation

The highest memory address available (HMA) is usually the first word below DOS. If, during loading, RLDR has placed its symbol table at the high end of user memory, HMA will be the first word below the table. The table will be in upper memory only if you ask the relocatable loader to leave it there. (By default, it is loaded just above the user program.)

Determine Available Memory (.MEM)

This command returns the current value of NMAX in AC1, and the value of HMA in AC0. HMA represents the location immediately below either the bottom of DOS or the end of the user symbol table. Use a SUB 1,0 instruction to determine the amount of additional memory available to your program. The format of .MEM command is:

```
.SYSTM
.MEM
error return
normal return
```

There are no error returns from this command.

Change NMAX (.MEMI)

This command allows you to increase or decrease the value of NMAX. You must pass the increment or decrement (in two's complement) in AC0. The command updates the value of NMAX in the UST (in USTNM) and returns the new NMAX in AC1. The format of this call is:

```
.SYSTM
.MEMI
error return
normal return
```

NMAX will not be changed if its new value would be greater than HMA. The system does not check NMAX against its original value (as determined at relocatable load time).

Whenever a user program will require memory space above the loaded program, .MEMI should be invoked before loading to allocate the number of words needed. The value of NMAX is used by the operating system to determine the amount of memory to be saved should a program be suspended. Generally, you should update NMAX even for temporary storage. If you store a program without updating NMAX, the program may be suspended without enough information to continue. This is explained further in the discussion of program swaps, Chapter 4.

Moreover, each user program should request only the memory space that actually needs, and should release memory space when it is no longer required. There is one error from a .MEMI command:

AC2	Mnemonic	Meaning
26	ERMEM	Attempt to allocate more memory than available.

CLOCK/CALENDAR COMMANDS

Four commands are provided to permit the system to keep track of the time of day and the current date. Dates are stored as days from December 31, 1967; (day 1 is January 1, 1968. The time of day is given using a 24-hour clock. All values are passed and returned as binary numbers.

Get the Time of Day (.GTOD)

This command requests the system to pass the current time in hours, minutes, and seconds to the caller. The time will be returned in binary as follows:

AC0 - Seconds

AC1 - Minutes

AC2 - Hours (using a 24-hour clock)

The format of this command is:

```
.SYSTM
.GTOD
error return
normal return
```

No error return is possible.

Set the Time of Day (.STOD)

This command sets the system clock to a specific hour, minute, and second. You pass the initial binary values as follows:

AC0 - Seconds

AC1 - Minutes

AC2 - Hours (using a 24-hour clock)

The format of this command is:

```
.SYSTM
.STOD
error return
normal return
```

A possible error message is:

AC2	Mnemonic	Meaning
41	ERTIM	Illegal time of day.

Get Today's Date (.GDAY)

This command requests the system to return the number of the current month, day, and year. The month is returned in AC1, the day in AC0, and the current year (less 1968) is returned in AC2. The format of the .GDAY command is:

```
.SYSTM
.GDAY
error return
normal return
```

No error return is possible.

Set Today's Date (.SDAY)

This command sets the system calendar to a specific date. The system will increment the date when the time of day passes 23 hours, 59 minutes, and 59 seconds. You pass the number of the month in AC1 (January is number 1), the number of the day within the month in AC0, and the number of the current year -- less 1968-- in AC2. This routine works only on years from 1968 to 2099. The format of the .SDAY command is:

```
.SYSTM
.SDAY
error return
normal return
```

One possible error message is:

AC2	Mnemonic	Meaning
41	ERTIM	Illegal day, month, or year.

KEYBOARD INTERRUPT COMMANDS

You can suspend or halt the current program in two ways: by simultaneously pressing either CTRL and A or the CTRL and C keys.

CTRL A interrupts the current program and gives you program control; CTRL C interrupts the current program, writes a snapshot of the program to disk file BREAK.SV and gives you program control. Alternately you (or your program) can suspend program execution by issuing system call .BREAK; this call produces the effect of a CTRL C interrupt, and is described below.

For each program level, the system creates a User Status Table (UST). Each UST is 24_8 words long, and resides in user address space. Every UST includes two words, USTIT and USTBR, which contain addresses for CTRL A and CTRL C interrupt routines. USTIT contains the address of the routine which will gain control after you enter CTRL A; USTBR holds the address of the CTRL C routine. When you load a program, the relocatable loader initializes both words to -1, and you must change this if you want to specify your own routines.

If USTIT contains -1 when you hit CTRL A, or USTBR holds -1 on a CTRL C or .BREAK, the system closes all channels on the current level and loads the next higher level program. This level's UST is then checked for the address of an interrupt routine. The system continues this process until it finds a program level whose UST contains the address of an interrupt routine. If it reaches CLI on level 0, it uses the CLI's routine. But if you have chained from the CLI, and the new level 0 program contains no interrupt routine address, the system will halt in Exceptional Status (see Appendix G).

During this search, the system checks each program level for a TCB queue. If the queue is missing (perhaps because you accidentally overwrote it), the system skips this program and examines the next higher level program (see Figure 3-8).

After finding a program with USTIT or USTBR (as appropriate) set to an address instead of -1, the system appropriates the TCB of the currently highest priority task (pointed to by USTAC), transfers that task's PC to temporary storage TTMP in the TCB, and places the UST interrupt address in TPC (TPC is the program storage counter in the TCB). Control then goes to the scheduler, which starts the highest priority task. Since the UST interrupt address was placed in TPC of the highest priority task, the interrupt service routine is executed. (In a single task program, the highest priority task is the only task in the system.)

If, upon a CTRL C, the system cannot write the core image file BREAK.SV (possibly because it lacks disk file space), control will go to one less than the address specified in USTBR, and AC2 will contain a system error code.

If you want, you can specify a return from the interrupt service routine to the address of the highest-priority ready task (TTMP). You could use the following instruction sequence for the return:

```
LDA     3,USTP          ;GET UST BASE ADDRESS
LDA     3,USTCT,3       ;GET TCB BASE ADDRESS
LDA     3,TTMP,3        ;GET OLD PC+LINK WORD
MOVZR   3,3             ;RESTORE LINK AND FORM
                        ;THE ADDRESS
JMP     0,3             ;GO TO OLD PC ADDRESS
```

By default, when you start a program, keyboard interrupts are enabled. DOS provides two system calls to disable or re-enable further keyboard interrupts .ODIS and .OEBL. These two calls do not affect the system call .BREAK (below), which performs the same operation sequence as CTRL C.

The BREAK file created by a CTRL C (or .BREAK) is a save file, containing the current state of main memory from SCSTR (the start of save files, location 16) through the higher of NMAX or the start of the symbol table, SST. File space for this file is found in the current directory. The file name used is

BREAK.SV; any previous file BREAK.SV is deleted by the command. Although file BREAK.SV is a snapshot of the current state of main memory, the file is not directly executable. Before you execute it, you must consider how the CTRL C (or .BREAK) interrupt affected the system:

1) It closed all open channels, and you must reopen them.

2) It removed all user-defined clocks and user interrupts; you must re-identify them if desired.

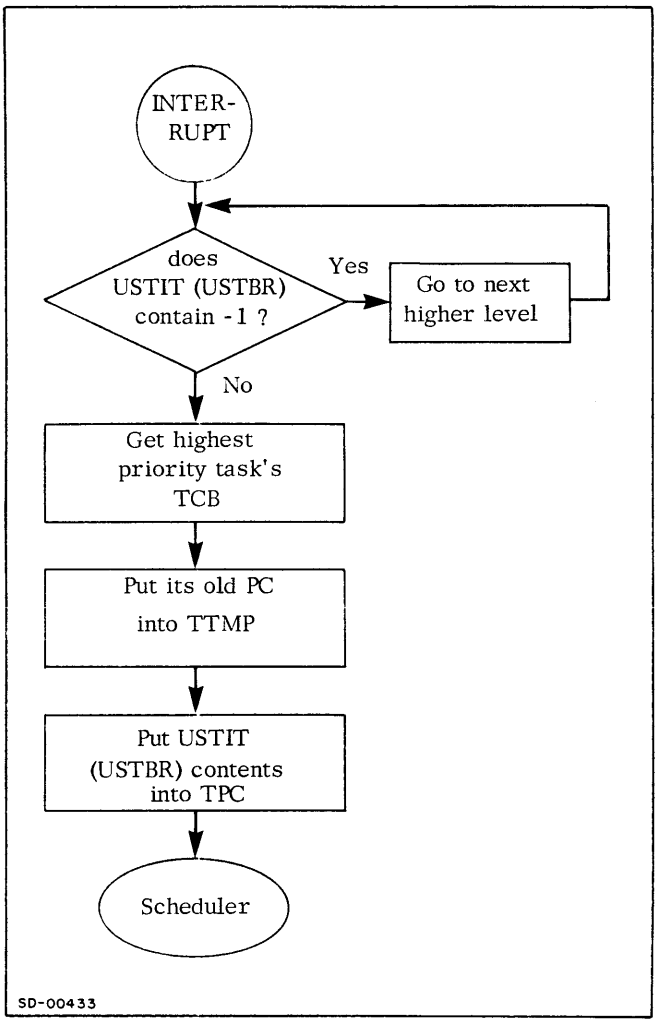


Figure 3-8. Program Interruption Logic Sequence

Save the Current State of Main Memory (.BREAK)

System call .BREAK is operationally equivalent to typing CTRL C on the program console keyboard; it saves memory, in save file format, from location 16 to the higher of NMAX or the start of the symbol table, SST. The file name used is BREAK.SV; any previous file BREAK.SV is deleted by the command. File space used is the current directory. Before you execute BREAK.SV, you should understand how the .BREAK call affected the system. See the preceding paragraph.

Unlike the CTRL C interrupt facility, the .BREAK call is operative at all times and is not disabled by the .ODIS command. The format of this call is as follows:

```
.SYSTM
.BREAK
        ;NO STANDARD ERROR
        ;OR NORMAL RETURNS
```

If USTBR (see preceding section) contains a valid address, control goes to this address after the break file is written. If the contents of USTBR is -1, control returns to the next higher level program and its USTBR will be examined. Control eventually goes to the first higher level program whose USTBR contains a valid address. If the break save file cannot be written (e.g., due to insufficient file space), control goes to one less than the address contained in USTBR. If an error occurs, AC2 will be set to one of the following:

AC2	Mnemonic	Meaning
27	ERSPC	Out of disk space.
60	ERFIN	BREAK.SV is in use.
101	ERDTO	Ten-second disk timeout occurred.

Disable Console Interrupts (.ODIS)

Use this command to disable console interrupts within your program environment. .ODIS disables CTRL A and CTRL C interrupts, but not the system call .BREAK. You can re-enable CTRL A and CTRL C by issuing the system call .OEBL (below) from your program area.

The format of this call is:

```
.SYSTM
.ODIS
error return
normal return
```

The error return is never taken.

Enable Console Interrupts (.OEBL)

By default, when a system is first bootstrapped, console interrupts CTRL A and CTRL C are enabled. If console interrupts have been disabled by system call .ODIS, this call re-enables them within its program environment.

The format of this call is:

```
.SYSTM
.OEBL
error return
normal return
```

The error return is never taken.

END OF CHAPTER

CHAPTER 4

SWAPS, CHAINS, AND USER OVERLAYS

Frequently, a user program will require more memory than is available in the user's address space. The Diskette Operating System provides three ways to segment user programs that need extra address space. These are program swaps, program chains, and user overlays. This chapter explores these DOS mechanisms.

Program swaps, chains and user overlays effectively extend main memory with disk space. They do this by placing segments of programs on disk and calling them into memory when the programs need them. During this process, they may overwrite the same areas of user address space many times with different data from disk.

Program swaps can call these disk files from 1 of 5 different levels of control, where one level often calls to another; chained programs are called in sequence by a program on the same level, and overwrite the calling program. Overlays also operate on one level, but they are called in succession by a root program in core and placed in a reserved area in core.

Swaps and chains are described below; you will find user overlays in the next section.

In general, the information in this chapter applies to both single and multiple task environments; in fact, the mechanisms described below provide an essential tool for the multitasking environment (Chapter 5).

PROGRAM SWAPPING AND CHAINING

This section describes 5 swapping and chaining calls: .EXEC (Swap or chain a program), .RTN (return from a swap or chain), .ERTN (return from a swap or chain with exceptional status), .FGND (describe the level of the current program), and .BOOT (bring in a new operating system).

For a program swap, a core-image file of user address space (from address 16_8, the USP, to NMAX) is stored temporarily on disk. While it is on disk, the current program's task control block (TCB) saves its accumulators, Carry, and PC. After the swap, the TCB restores them from disk. This restoration of a program into the user area is called returning.

Any program executing under the operating system can suspend its own execution and invoke another program or another segment of itself. Every program requested in a swap must exist as a save file on disk.

Program swaps can exist in up to five levels, where one level calls for another and the Command Line Interpreter exists at the highest level, level 0.

The CLI is merely one program executable under the operating system. Its only special property is that it normally executes at the highest level in the system, level 0. Normally, the system utility programs supported by the CLI (e.g., the Text Editor, Assembler, and the Relocatable Loader) execute at level 1. The operating system permits up to five levels of program swaps. This means that a program invoked by the CLI (overwriting the CLI) can in turn invoke a third program (overwriting the second program), the third a fourth, and the fourth program a fifth before the system rejects further requests. Programs on these different levels may communicate via disk files like COM.CM. The CLI at level 0 uses COM.CM to communicate with utility programs on level 1.

On any single level, you can divide a calling program into separate segments whose total size is far larger than your address space. You do this by chaining process, where one program segment calls for another segment of the program, which in turn may call for another segment, etc. The entire program with all its segments exists at the same level. You can divide a single program into a limitless number of segments.

When you plan program swaps or chains, you should ensure that NMAX accurately reflects the core for every program in use; if it does not, part of some calling programs might be lost. Upon a program swap or chain, the current core image is saved up to the higher of NMAX or SST (start of the user symbol table). It is very important that no program use temporary storage above its original value of NMAX at load time without having the system first allocate more memory (see .MEMI, Chapter 3) for this space. If a program exceeds NMAX and invokes another program, part of calling program's memory state will not be saved. Even if the executing program does not call another program, a BREAK from your console may force suspension. To avoid each of these problems, NMAX must always correctly reflect the core in use.

The operations of swapping, chaining, or returning halt activity in the current program. The operating system terminates calls and conditions that would not be appropriate in the new program (most of these involve multitask activity). The following calls and conditions are terminated when a change of program occurs. Many of these calls are detailed elsewhere in this manual, and you should read the references if you want more information.

1) A return or chain closes all channels; see the beginning of Chapter 3.

2) All $TTI ($TTI1) input is halted; this applies to such calls as .GCHAR (Chapter 3).

3) Console interrupts are enabled, removing any outstanding disable calls by .ODIS (Chapter 3).

4) All interrupt message transmissions, .IXMT (Chapters 5 and 6) are removed.

5) If a user clock (.DUCLK, Chapter 5) has been defined, it is removed.

6) All user-defined interrupt service (.IDEF, Chapter 6) is removed.

7) The state of the floating-point unit is not preserved.

When a program's execution resumes after a swap, all channels which were open when the swap occurred will be open. To restore the other conditions (2 through 7) to a program, you must use the appropriate system or task call.

Read in a Save File for Swapping (.EXEC)

This command requests the system to bring in a program swap. The format of the .EXEC command is:

```
.SYSTM
.EXEC
error return
normal return
```

AC0 must contain a bytepointer to save filename of the called program. AC1 must contain an appropriate starting address code. Two possible starting addresses are allowed: the program starting address (USTSA)*, and the Debug III starting address (USTDA)*.

If bit 0 of AC1 is 1, the current level will not be saved, and the operating level will remain unchanged. (Note that this feature provides unlimited program chaining.)

The permissible codes input in AC1 are:

Code	Meaning
0	Swap to user program. Control goes to the highest priority ready task whether within a swap or break save program created by CTRL C.
1B0	Chain to user program.
1	Swap and start at debugger address.
1B0 + 1B15	Chain and start at debugger address.

ERADR status is returned if:

1) No starting address was specified for the save file and code 0 is given (i.e., bit 15 is reset to 0).

2) The Debugger was not loaded as part of the save file and code 1 is given (i.e., bit 15 is set to 1).

*See Chapter 5, User Status Table, for descriptions of USTSA and USTDA.

The contents of AC2 are passed to the new program.

AC2	Mnemonic	Meaning
1	ERFNM	Illegal file name.
4	ERSV1	File requires save attribute (S).
12	ERDLE	File does not exist.
25	ERCM3	More than 5 swap levels.
26	ERMEM	Attempt to allocate more memory than is available.
32	ERADR	Illegal starting address.
53	ERDSN	Directory specifier unknown.
57	ERLDE	Link depth exceeded.
66	ERDNI	Directory not initialized.
73	ERUSZ	Too few channels defined at load time or SYSGEN time.
101	ERDTO	Ten-second disk timeout occurred.
102	ERENA	No linking allowed (N attribute).
125	ERNSE	Program not swappable.

Return from Program Swap (.RTN)

Upon successful completion of a program invoked by .EXEC, this command returns to the calling program at its normal return point and closes all channels. All of the calling program's accumulators are restored, and control passes to the instruction following .EXEC. The format of the .RTN command is:

```
.SYSTM
.RTN
error return
```

The normal return is impossible, since the calling program is restored in memory. The error return is reserved for compatibility with RTOS but is never taken. Error conditions cause Exceptional System Status (see Appendix G).

Return from Program Swap with Exceptional Status (.ERTN)

This command instructs a called program to return error information to the calling program. Use it when you want to know the status of a swapped program. The format of the .ERTN command is:

```
.SYSTM
.ERTN
error return
```

The error return is reserved for compatibility with RTOS but is never taken. Error conditions cause Exceptional System Status (see Appendix G).

This call is identical to .RTN except that normal return is made to the error return of the higher level program: upon return, AC2 contains the lower-level program's AC2 instead of the higher-level program's AC2. A single word of status can, therefore, be returned. If a program issuing a .ERTN has been executing at level 1 (and is returning to the CLI) the CLI will output an appropriate message concerning the status code in AC2. If the code is recognized as a system exceptional status code, a text message will be printed. See Appendix F, Exceptional System Status, for a complete list of exceptional status codes. For example, the code ERDLE (12) would evoke the message FILE DOES NOT EXIST. If "null error" ERNUL is returned in AC2, no error message will be reported by the CLI. If EREXQ is returned in AC2, the CLI will take its next command from disk file CLI.CM. If the code is unrecognized by the CLI, the message UNKNOWN ERROR CODE n will by typed out, where n is the numeric code in octal.

Check the Level of a Running Program (.FGND)

This call returns in AC1 the level at which the current program is running; it always returns 0 in AC0. No inputs are required to the .FGND call.

AC1 returns an integer code indicating the current program level. One of the following codes will be returned:

Code	Meaning
1	Level 0
2	Level 1
3	Level 2
4	Level 3
5	Level 4

The format of this call is:

```
.SYSTM
.FGND
error return
normal return
```

No error condition is currently defined.

Bootstrap a New Operating System (.BOOT)

Use .BOOT to replace the current DOS system with a new system. This system call opens all files, releases all directories, and resets all system I/O. After this orderly shutdown, control is transferred to BOOT which bootstraps a new operating system. This new operating system may exist on any diskette; you specify its name by a byte pointer input in AC0. If the operating system's save file is a link, its overlays must also be linked. All diskettes in the specification must be initialized.

To bootstrap the new system without operator intervention, you must enter -1 in the CPU switches (i.e., all switches up) and the default system must be specified in either a primary or a secondary partition. If this is done, after bootstrapping the new system control will be chained to a user-provided save file named RESTART.SV. In RESTART.SV, you must specify whatever restart procedures are required to resume control of the real time process which was interrupted. The time and date do not get updated and you set them.

Required input to this call is:

AC0 - Bytepointer to operating system specification

The format of this call is:

```
.SYSTM
.BOOT
error return
```

There is no normal return, since upon the normal completion of this call BOOT will receive control and will then pass control to the new operating system.

The following error conditions can occur:

AC2	Mnemonic	Meaning
1	ERFNM	Illegal file name.
12	ERDLE	File name does not exist.
23	ERRTN	File RESTART.SV does not exist, yet data switches were all set for restarting without operator intervention.
53	ERDSN	Unknown specifier.
101	ERDTO	Ten-second disk timeout occurred.

USER OVERLAYS

User overlays are blocks of code, placed in an overlay file, that support a root program. This root program is a save file that remains in core throughout a program level; it extends from location 16_8 to NMAX, and calls overlays into core as required. The overlay file is divided into segments. Each segment contains the overlays which will be loaded into a reserved area of core; this reserved core area is called a node, and it "belongs" to the segment. The RLDR command loads the program, creates the overlay file, loads overlays into segments of the file, and determines the core node size.

Whenever the program needs to use the code in a overlay, it loads the overlay from the overlay segment into the node. When the program has finished using the overlay, it can overwrite this node with another overlay from the same segment. (This process differs slightly for a multitask program; if you plan a multitask program, see USER OVERLAY MANAGEMENT in Chapter 5.)

The size of each node is the smallest multiple of 400_8 words large enough to contain the largest overlay in the node's segment. If any overlay is not exactly the size of its node, it will be padded out with zeros. This means that any segment size equals the node size multiplied by the by the number of overlays within the segment. Each segment is identified on disk by its node number.

Each overlay file is a contiguous disk file, which holds up to 124 overlay segments. You can place no more than 256 overlays in a segment, and no overlay can be larger than 126 disk blocks (31,256 words). If the overlays in a segment differ significantly in size, a lot of disk space will be used to pad out the smaller overlays to the standard size. Therefore, if you can, you should place overlays of about the same size in the same segment.

Directory information for each overlay resides in an overlay directory, which RLDR builds into the program's save file (see Appendix D). Each overlay has a label which the system uses to identify it; this label resolves to a node number and an overlay number, packed by half-words.

The format required for creating an overlay file and associating it with a root program is shown in Chapter 7, under the RLDR command. The following discussion extends the sample RLDR command presented there:

RLDR R0 [A,B,C,D] R1 R2 [E,F G, H])

This command creates a disk save file, R0.SV, and a disk overlay file, R0.OL. The save file contains R0.RB, R1.RB, R2.RB, and 2 overlay nodes; the overlay file has 2 overlay segments containing the binaries enclosed within each pair of brackets. Segment 0 of overlay file R0.OL contains overlay A (number 0, for node 0), overlay B (number 1 for node 0), overlay C (number 2 for node 0), and overlay D (number 3 for node 0). Segment 1 of R0.OL contains overlay E (number 0 for node (0), overlay F and G (number 1 for node 1), and overlay H (number 2 for node 1). Note that the order in which the overlay binaries were given in the command line determines both the overlay number and node number for each overlay.

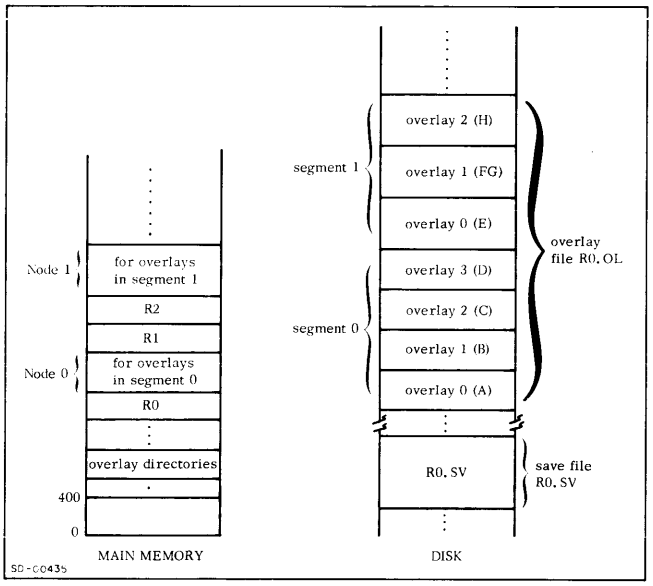


Figure 4-1. User Overlays

You can disregard the loading order of a node's overlays if you use the .ENTO pseudo-op. .ENTO allows you to assign a unique label to each overlay, thus making the order of overlays in the command line unimportant. You do this by assigning each binary a unique label as an argument to .ENTO; you then reference the binary in your program using the unique label, which must be declared by a .EXTN psuedo-op. If you don't use .ENTO, you must ensure that the RLDR command line lists overlay binaries in the proper order.

The sample RLDR command creates two overlay segments. The following illustration (Figure 4-2) shows some possible entry points in the overlays of the second segment.

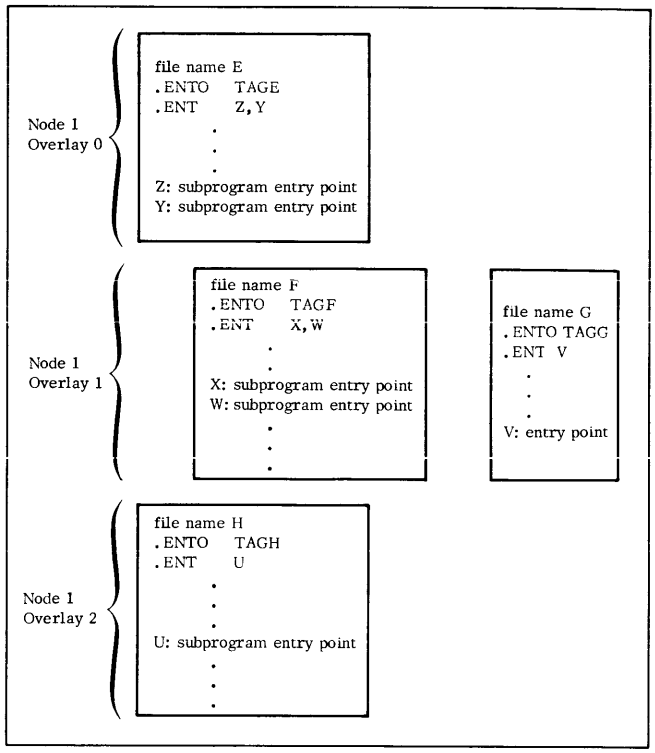


Figure 4-2. Segment 1 of Overlay File R0.OL

To call these overlays, the root program R0.SV must first load the overlay using the overlay label. For instance:

```
.EXTN   TAGE, TAGF, TAGG, TAGH
.EXTN   Z, Y, X, W, V, U

.OVE:   TAGE          ;TAGE IS RESOLVED TO
                      ;NODE 1, OVERLAY 0
                      ;(ENCODED AS 400).
:OVF    TAGF          ;TAGF IS RESOLVED TO
                      ;NODE 1, OVERLAY 1
                      ;(ENCODED AS 401).
:OVG    TAGG          ;TAGG IS RESOLVED TO
                      ;NODE 1, OVERLAY 1
                      ;(ENCODED AS 401).
:OVH    TAGH          ;TAGH IS RESOLVED TO
          .           ;NODE 1, OVERLAY 2
          .           ;(ENCODED AS 402).
          .
        ADC 1,1       ;PREPARE FOR UNCONDI-
          .           ;TIONAL LOAD.
        LDA 0, .OVE   ;GET OVERLAY NUMBER.
        .SYSTM        ;LOAD BINARY E
                      ;UNCONDITIONALLY

        .OVLODn       ;LOAD OVERLAY ON
          .           ;CHANNEL n
          .
```

After the program issues the system call .OVLOD, all of binary E is loaded into core, and routines Z and Y can be used. Because the binary was loaded using a .ENTO label, the user didn't need to know which node or overlay contained the binary; therefore the RLDR sequence could have been:

```
RLDR R0 [A,B,C,D] R1 R2 [H,E,G F] )
```

Open User Overlays for Reading (.OVOPN)

Before you can call an overlay in either a single or multitask environment, the user overlay file must be opened on a user channel. Several users can open an overlay file simultaneously, on different channels. A normal .CLOSE is used to close this channel. AC0 must contain a bytepointer to the name of the user overlay file (including its .OL extension). The format of the .OVOPN command is as follows:

```
.SYSTM
.OVOPN n  ;OPEN CHANNEL n
error return
normal return
```

Possible errors resulting from .OVOPN are:

AC2	Mnemonic	Meaning
0	ERFNO	Illegal channel number
1	ERFNM	Illegal file name
12	ERDLE	Nonexistent file.
21	ERUFT	Attempt to use channel which is already in use.
53	ERDSN	Directory specifier unknown.
57	ERLDE	Link depth exceeded.
66	ERDNI	Directory not initialized.
101	ERDTO	Ten-second disk timeout occurred.
102	ERENA	No linking allowed (N attribute).

Load a User Overlay (.OVLOD)

This command loads an overlay whose node and number word is in AC0. You must pass the node value in the left byte, and the overlay number in the right byte.

There are 2 types of overlay load: conditional and unconditional. An unconditional load loads an overlay whether the overlay is in core or not. This guarantees a fresh copy of the overlay. A conditional overlay request, on the other hand, loads an overlay only if it is not already in core. The conditional request can save you time, but you should use it for reentrant overlays only.

The .OVLOD command will load the overlay conditionally if AC1 is set to 0; or unconditionally if AC1 is set to -1. We recommend that all your overlays be reentrant; if any overlay is not, be sure to load it unconditionally.

The format of the .OVLOD command is:

```
.SYSTM
.OVLOD n  ;LOAD OVERLAY OPENED ON CHANNEL n
error return
normal return
```

In a multitask environment, only one task can be allowed to issue .OVLOD commands. See USER OVERLAY MANAGEMENT, .TOVLD command in Chapter 5 for more on multitasking.

Possible errors resulting from .OVLOD are:

AC2	Mnemonic	Meaning
0	ERFNO	Illegal channel number.
6	EREOF	End of file.
7	ERRPR	Attempt to read a read-protected file.
15	ERFOP	File not opened.
30	ERFIL	Read error.
37	EROVN	Illegal overlay number.
101	ERDTO	Ten-second disk timeout occurred.

END OF CHAPTER

CHAPTER 5
MULTITASK PROGRAMMING

MULTITASK ENVIRONMENT

In a multitask environment, more than one task competes for CPU control with an optimum utilization of this resource resulting. Tasks operate asynchronously and in real time, with CPU control being allocated to the highest priority ready task by the Task Scheduler. A complete list of task calls is given at the end of this chapter.

Task priorities range from 0 through 255, with priority 0 being the highest priority. One task will automatically be created at priority 0 for the task whose starting address is specified by the .END statement at the end of the program.

Several tasks may exist at the same priority. Equal priority tasks receive CPU control on a round-robin basis. This implies that the task which most recently received control will be the last to receive control again, unless other tasks are unable to receive control at the moment that rescheduling occurs. Whenever a task has a priority change (.PRI), the task is placed at the end of the list of all tasks within its new priority.

Task Control Blocks

A task is an asynchronous execution path through user address space demanding use of system resources. Many tasks may be assigned to operate in a single reentrant path, and each of these tasks may be assigned a unique priority. Given the asynchronous nature of tasks, the DOS Task Scheduler must maintain certain status information about each task. This information is retained within an information structure called a Task Control Block (TCB), and there is one TCB for each task. The following illustration describes the structure of TCBs:

Word	Mnemonic	Contents
0	TPC	User PC and Carry.
1	TAC0	AC0.
2	TAC1	AC1.
3	TAC2	AC2.
4	TAC3	AC3.
5	TPRST	Status bits and priority.
6	TSYS	System call word.
7	TLNK	Link word.
10	TUSP	USP.
11	TELN	Extended save area.
12	TID	Task identification, right byte.
13	TTMP	DOS temporary storage.
14	TKLAD	Task kill address.
15	TSP	Stack pointer.
16	TFP	Frame pointer.
17	TSL	Stack limit.
20	TSO	Instruction TRAP PC

TPRST contains information describing the state of the task (discussed following) and the task priority.

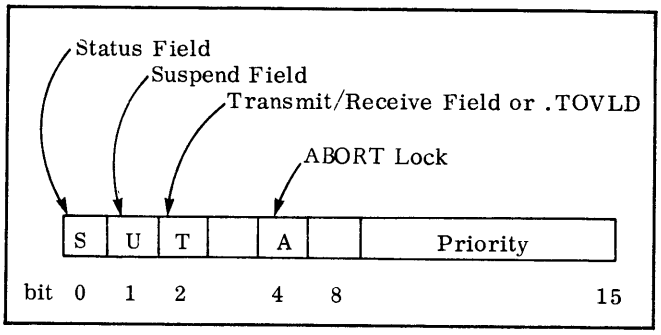


Figure 5-1. Task State/Priority Information (TPRST)

Field S is set to 1 if the task has become suspended by either a system call or by a .XMTW/.REC/.TOVLD task call; field S is set to 0 when the system call or .XMTW/.REC/.TOVLD is completed. Field U is set to 1 only if the task has been suspended by a .SUSP, .ASUSP, or .TIDS task call. Field T is set to a 1 only if the task has issued either .XMTW/.REC or .TOVLD. The task priority is contained in bits 8-15. Bit A is set if the task is being aborted.

TSYS is used by DOS in executing system calls and .XMTW/.REC/.TOVLD. TLNK contains the starting address of the next TCB in the queue. TUSP contains the value of location USP at the time this task last changed from the executing state. USP may be used as a general purpose storage location by each task when it is not otherwise used. The system will restore the USP value for each task that gains CPU control. TELN points to the task's BASIC save area. TID contains the task identification number, if any, in its right byte.

Note: The NOVA 3 and microNOVA hardware stack is moved between task swaps, in multitasking. The Stack Overflow Handler (location 43) is not moved. To return the contents of the hardware frame pointer in AC3, load the program with N3SAC3.

TKLAD contains the address which is to receive control upon a task's being killed, if such an address has been defined via a .KILAD call. The remaining four words contain stack state save information which are reserved for TCBs.

In a multiple task program ready for assembly, you must specify both the number of TCBs and the number of DOS channels which will be required. You can specify these before assembly in your program by means of a .COMM TASK statement or -- more conveniently -- you can specify tasks and channels at load time by means of the /K and /C local switches in the RLDR command line. If a .COMM TASK statement is used, it must appear in the first user relocatable binary loaded since it affects the loading process of the remainder of the program and determines which task scheduler (TMIN or TCBMON) will be loaded by default. If either /C or /K switches are used along with a .COMM TASK statement, the switch information overrides the statement specification. The format of the .COMM TASK statement is:

.COMM TASK,n*400+m

where: n represents the integer number of tasks and m represents the integer number of DOS channels which will be used. Example: .COMM TASK,7*400+20.

TMIN and TCBMON, all task command modules, the interrupt-on symbolic debugger, and BFPKG (See Application Note #017-000003) are found in the system library, SYS.LB. All items in SYS.LB which are loaded into the user address space will be loaded by default at the end of the root program code.

TASK STATES

Tasks may exist in any of three states. Tasks are either ready to perform their functions, they are actually in control of the CPU and are executing their assigned instruction paths, or they are suspended. The Task Scheduler always gives CPU control to the highest priority task that is ready.

Suspended tasks are tasks which have at least one of the three status bits (S, U, T) set to a one. A task may become suspended for one or more of a variety of reasons:

1. It has been suspended by .SUSP, .ASUP, or .TIDS.
2. It is waiting for a message from another task, .REC.
3. It has issued a message-and-wait call, .XMTW.
4. It is waiting for the use of an overlay.
5. It is awaiting the completion of a .SYSTM call.

Just as a number of different events may suspend a ready task, several events can cause a suspended task to be readied:

1. The completion of a .SYSTM call such as a request for I/O).
2. The posting of a message for a suspended task awaiting its receipt.
3. A requested overlay is loaded.
4. The readying of a task by .ARDY or by .TIDR task calls.

If a task is suspended by both a task suspend call and by some other event, the call must be readied both by an .ARDY (or .TIDR) call and by whatever other event is required to ready the task. Thus a task may

be doubly suspended, with both bits S and U set in the task's priority and status word, TPRST, Bits S, U, and T must all be reset in order for the task to be ready.

Tasks may be deleted from the active queue, either separately (.KILL or .TIDK) or as a priority class (.AKILL). Tasks which have been deleted add their empty TCBs to an inactive chain of free element TCBs. When a task is created (.TASK or .QTSK), a TCB is taken from the free element chain and is entered into the active queue. The .TASK or .QTSK command must be used to initiate a multitask environment.

If all tasks are killed, and no task is awaiting creation via .QTSK, the effect is the same as:

```
.SYSTM
.RTN
```

Program control then returns to the next highest program level.

TCB Queues

There is one TCB queue for executing, suspended and ready tasks. This queue consists of a chain of TCBs, connected by the TLNK words of each TCB, and is called the active queue. The first TCB is pointed to by USTAC of the User Status Table. This TCB points to the next TCB, etc. The last TCB in the chain has a TLNK of -1.

The free element TCB chain is a simple queue of dormant TCBs. TCBs in the free element chain are joined by TLNK words; all other words in each of these TCBs are unused. There is no priority among TCBs in the free element chain. The first TCB in the free element chain is pointed to by USTFC of the User Status Table (See Figure 5-2).

Task Synchronization and Communication

DOS permits tasks to communicate with one another by sending and receiving one-word messages. A one-word message is sent to a task in an agreed-upon location in user address space. User address space includes all locations from address 16 through NMAX.

The task sending a message may either return to the Task Scheduler immediately (.XMT) or it may wait (.XMTW) and place itself in the suspended state until a receiving task has issued a receive request (.REC) and has received the message. Receipt of the message includes the resetting of the contents of the message location to zero. Upon receipt of the message, the recipient task has the S and T bits set to zero.

USER STATUS TABLE

The User Status Table (UST) is a 24_8 word table which records information pertinent to the execution of a program level. This table is located at addresses 0400 through 0423* inclusive and has the following structure:

address	label	contents
400	USTPC	Used by the system.
401	USTZM	ZMAX.
402	USTSS	Start of Symbol Table (SST).
403	USTES	End of Symbol Table (EST).
404	USTNM	NMAX after runtime .MEMIs.
405	USTSA	Starting address of Task Scheduler.
406	USTDA	Debugger address; -1 if not loaded.
407	USTHU	USTNM after relocatable load.
411	USTIT	Interrupt address; -1 initially.
412	USTBR	Break address; -1 initially.
413	USTCH	Number of channels and number of TCBs.
414	USTCT	Current TCB pointer.
415	USTAC	Start of active TCB chain.
416	USTFC	Start of free TCB chain.
417	USTIN	Initial start of NREL code (INMAX).
420	USTOD	Overlay directory address.
421	USTSV	Available for use by the system.
422	USTRA	Revision level number, and during execution, the environment state.
423	USTIA	Address of TCB for keyboard interrupt task (For RDOS compatibility only).

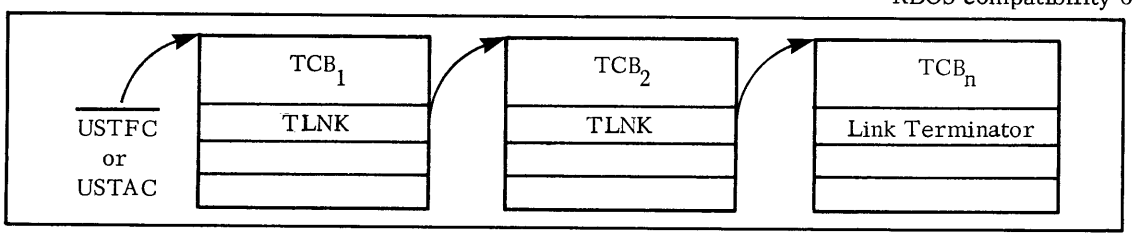


Figure 5-2. TCB Free Element Chain

USTPC is always 0. USTZM contains ZMAX, the first free location in page zero after a relocatable load.

Locations 402 and 403, USTSS and USTES, point to the start and end of the symbol table, respectively. Under default conditions, the loader moves the symbol table at the termination of loading so that the last location in the symbol table +1 coincides with the value of NMAX after all programs are loaded. USTSS, USTES, and NMAX are updated. If you request that the symbol table be placed in upper core (/S switch on a RLDR command), the symbol table is moved so that it will be immediately below the operating system when the save file is executed. If the symbol table has not been loaded, locations 402 and 403 are set to zeros.

USTNM contains the current value of NMAX at run-time. This value changes as NMAX is increased or decreased. Location 407, USTHU, is initialized by the loader to the value of NMAX at the termination of loading. This word is never changed by the operating system during program execution.

USTIT is the interrupt address (CTRL A). At the termination of loading, this address is set to -1. If unchanged at run time, control goes to the next higher level program with USTIT set to a valid address when a CTRL A interrupt occurs. The user core image is not saved. Your program can set USTIT at execution time to an address to which control will be transferred if a CTRL A interrupt occurs.

USTBR is the break address (CTRL C). At the termination of loading, this address is set to a -1. Whenever a CTRL C break occurs, the core image will be written to file BREAK.SV on the default directory device. If unchanged at run time, control goes to the next higher level program with USTBR set to a valid address when a CTRL C interrupt occurs. Alternatively, you can set USTBR to an address to which control will be directed upon the successful creation of the break file. If the creation of the break file is unsuccessful, e.g., due to insufficient file space, control will go to the address specified by (USTBR) -1, one less than the address contained in USTBR, with AC2 = error code.

USTCH contains the number of program tasks in its left byte, and the number of I/O channels in the right byte.

USTRV is reserved for storage of the revision number information for this save file. Revision numbers can extend from 00 to 99; the major revision number is stored in the left byte, and the minor revision number is stored in the right byte of this word. While executing .USTRV indicates the machine and system environment for the program. The user parameter file, PARU.SR, supplied on tape with the system, contains the values of the following symbols: ENUNV, ENNV3, ENSOS, ENRTOS and ENRDOS. USTIA is kept for RDOS compatibility.

SYSTEM AND TASK CALLS

The following sections describe all the system and task calls for use specifically in a multitask environment. You must reference all task calls by their call names in an .EXTN statement. Only those task calls which are so referenced will have the appropriate task call processing modules loaded by the relocatable loader.

Upon return from all task calls, AC3 contains the contents of USP (unless, on a microNOVA or NOVA 3, the program was loaded with N3SAC3, in which case AC3 contains the frame pointer). The frame pointer is maintained in location CSP. Program control returns to the Task Scheduler after all task and system calls (except for specials: .SMSK .IXMT .UIEX .UCEX .UPEX.

The significant differences between a DOS (.SYSTM) call and a task call are as follows:

1. Task calls consist of a one-word call with all parameters passed in the accumulators.

2. Not all task calls have error returns. Those which do not have error returns do not reserve an error return location.

3. Task calls are processed in user address space, while DOS or system calls require system action which occurs in DOS space.

4. Task calls are not preceded by the .SYSTM mnemonic. Task calls are resolved by the loader to be JSR calls to task processing modules.

*Location 12 in ZREL, USTP, points to the start of the UST belonging to the currently executing program. Symbol USTAD, (at interrupt level) created as an .ENTry by the loader, points to the base of a program's UST.

TASK INITIATION

Core resident tasks can be initiated by the .TASK command, and both core-resident and disk resident tasks can be initiated for periodic execution. For a description of tasks queued for periodic execution, see .QTSK in User Overlay Management.

Create a Task (.TASK)

This command initiates a new task at a specified priority in the user environment, and assigns an identification number to the task, if desired. When the program is loaded, only one task exists. This command must therefore be issued to initiate a multitask environment.

AC0 contains in its right byte the priority at which the new task will run and in its left byte the optional task identification number. If the right byte of AC0 is set to zero, the priority of the new task will be identical to the calling task's priority. More than one task with an I.D. number of zero can exist. AC1 contains the address where the new task will begin execution. The contents of AC2 will be passed to the created task. This permits the relaying of an initial one-word message to the newly created task.

```
.TASK
error return
normal return
```

Control will return to the error return if there is no TCB available. AC2 will then contain the error code:

AC2	Mnemonic	Meaning
42	ERNOT	No TCBs.
61	ERTID	A task with the requested I.D. (except 0) already exists.

TASK TERMINATION

Tasks can be killed within a user program in an orderly fashion, or task operations can be halted abruptly (.ABORT).

Tasks can be killed in an orderly fashion either singly by an I.D. number (.TIDK), as a priority group (.AKILL), or the calling task can kill itself (.KILL). To ensure that a task can be terminated in an orderly fashion, DOS provides a facility to define for each task a special routine which will gain control upon an attempted orderly kill.

Upon most orderly task terminations (.AKILL or .TIDK), each task to be terminated is raised to the highest possible priority and is readied unless it was suspended by a .SYSTM call. Thus if it was suspended by a .SYSTM call. Thus if it was suspended by a .REC, .XMTW, .SUSP, or .TIDS task call, the suspension would be lifted. If the task were in suspension due to an outstanding .SYSTM call, that call would be completed before the task was raised to the highest priority ready state. In either case, after the task to be killed has been evaluated to the highest priority ready state, one of two actions then occurs depending upon the intention of the user. First, if no kill-processing address is provided, then the task is simply terminated.

Alternatively, if you wish, you could specify a special kill-processing address for the task. If you specify such an address, then when the task to be killed receives control, control goes to this special processing address. This facility gives you flexibility in providing an orderly release of resources should the task be killed. Moreover, the kill-processing routine can act as a reprieve, since the task executing this routine will not actually be terminated until it itself issues a .KILL call. Thus the kill-processing routine can also be used as a validation procedure to determine whether or not the target task should be terminated.

The case of self-termination, .KILL, is special. If a task attempting to terminate itself has a kill-processing address specified, then upon issuing the .KILL call this task will gain control as the highest priority task at its kill-processing address. If it has defined no such address, then the task will be terminated immediately.

Whenever a task is terminated (by either an orderly kill or abort), its TCB is relinquished to the free TCB pool for possible use in the initiation of other tasks. The following sections describe task calls used to terminate task operations from within a user program.

Define a Kill-Processing Address (.KILAD)

The .KILAD task call permits you to define a special address which will gain control the first time that the termination of a task, the "target task," is attempted. The second time that a termination of this task is attempted, the task will be terminated without control transferring to the kill-processing address.

The kill address can be defined to provide a means of releasing system resources before termination occurs. Such resources as overlay areas, channels, user devices and user clock definitions must be released explicitly by the user. Having released these resources

and performed any other desired functions, the task must then itself issue a .KILL call in order for its termination to occur. Since this would be the second attempt to terminate the task, termination would occur immediately.

If, on the other hand, the target task decides not to terminate itself, then before branching out of the kill-processing routine it should issue a .KILAD call to the same or to a different kill-processing routine. This will ensure that if an attempt is made later to kill this task, it will not be killed immediately but will branch again to its kill-processing routine.

Note that a task in a kill-processing routine is in execution at the highest priority. Thus such routines will retain control until they relinquish this control by a task state transition or by a priority level change.

The format of the .KILAD task call is as follows:

AC0 - Address of the kill-processing routine.

```
.KILAD
normal return
```

There are no error returns from this call.

Delete a Single Task (.KILL)

This command deletes the calling task's TCB from the active queue, and places it in the free element TCB chain. The calling task is the only task that may be deleted via this command. There is no return from this call. If a kill-processing address has been defined for this task, then the task is raised to the highest priority and control goes to this address. Otherwise, control returns to the Task Scheduler which allocates system resources to the highest priority task that is ready.

The format of this call is:

```
.KILL
```

There is no error return nor normal return from this call.

Delete All Tasks of a Given Priority (.AKILL)

This command first raises all tasks of a given priority to the highest priority, and then either kills them or transfers control to their kill-processing address. All TCBs that are deleted from the active queue are placed in the free TCB chain. Tasks in suspension due to .XMTW, .TIDS, .REC, or .SUSP calls will be raised to the highest priority ready state immediately. If an attempt is made to kill a task suspended by an outstanding .SYSTM call, that task will be raised to the highest priority at the completion of the .SYSTM call. The calling task itself may be deleted by this command. The format of this call is:

AC0 - Priority class of task to be killed.

```
.AKILL
normal return
```

There is no error return from this command. If no tasks exist with the priority given in AC0, no action is taken. If all tasks become deleted (and none are awaiting creation via .QTSK), the effect is to cause a program return:

```
.SYSTM
.RTN
```

Abort a Task (.ABORT)

The .ABORT task call readies a specified task immediately, and executes the equivalent of an immediate .KILL task call as soon as it gains control of the CPU. If a .KILL processing address exists, control will be transferred to it. The exact time of completion of the .KILL is dependent on the priority of the aborted task relative to other ready tasks. For example, a task attempting to perform a write sequential of 500 bytes might be aborted after writing any number of bytes. The task which is to be aborted is specified by I.D. number. Thus, the call may abort either itself or some other ready or suspended task.

Task call .ABORT does not release any open channels used by the aborted task, nor does it release any overlays. Outstanding operations performed by the task, like waiting for a message transmission/reception (.XMTW/.REC), are terminated. Likewise, all system calls are aborted.

The format of this call is as follows:

AC1 - I.D. of the task to be aborted.

```
.ABORT
error return
normal return
```

The contents of AC0 is lost upon return.

The error return is taken under one of two conditions:

AC2	Mnemonic	Meaning
61	ERTID	An I.D. of zero was specified, or no such task I.D. was found.
110	ERABT	The specified task is currently being aborted by another task.

TASK STATE MODIFICATION

Change the Priority of a Task (.PRI)

This command changes the priority of the calling task to the value contained in AC0. The calling task will be assigned the lowest priority in its new priority class; all other ready tasks in the same priority class will be allocated CPU control by the Task Scheduler before this task will receive it.

```
.PRI
normal return
```

There is no error return from this command. If a priority greater than 255 is requested, only the value in bits 8 through 15 will be accepted.

Suspend a Task (.SUSP)

This command places the calling task in the suspended state by setting bit U of that task's TCB to the suspended state. There is no error return.

```
.SUSP
normal return
```

The suspended task remains until it is readied by an .ARDY or .TIDR command.

Suspend all Tasks of a Given Priority (.ASUSP)

This command suspends all tasks with the priority given in AC0. The calling task itself may be suspended by this call. All tasks suspended by .ASUSP, even those suspended for other reasons (e.g., an outstanding system call, setting bit S of TPRST) will remain suspended until readied by an .ARDY or .TIDR command and by the reading of bit S.

```
.ASUSP
normal return
```

There is no error return from this command. If no tasks exist with the given priority, no action is taken. The suspended tasks may be readied only by an .ARDY or .TIDR command.

Ready all Tasks of a Given Priority (.ARDY)

This command readies all tasks previously suspended by .ASUSP (.SUSP or .TIDS) whose priority is given in AC0. That is, bit U in word TPRST of each TCB that was set by a previous call to .ASUSP, .SUSP, or TIDS is now reset. Tasks suspended for other reasons too (e.g., outstanding system calls) will only be readied when bit S of TPRST in each of the TCBs is also reset, e.g., by receiving a task message via .REC. In order for a task to be raised to the ready state, both bit S and U of that task's word TPRST must be set to the ready state.

```
.ARDY
normal return
```

There is no error return from this command. If there are no tasks with the given priority in AC0, no action is taken.

Wait for a Keyboard Character (.WCHAR)

This command suspends the caller until either a specified character is typed onto any console keyboard, or the call is reissued by another task to terminate this keyboard wait. Only one task may be suspended for a keyboard character wait at any one moment.

The required input to this call is either the keyboard character which will ready the task, or -1, which terminates the keyboard character wait and readies the suspended task. These parameters are input via AC0. Since the calling task may be readied by either the transmission of a specified keyboard character or by the keyboard wait call being terminated, an appropriate code is returned in AC1 when the normal return is taken. This code will be either the device code of the console keyboard which issued the requested wait character, or -1; -1 indicates that the previous wait was terminated.

Required input to .WCHAR is:

AC0 - Wait character in right byte, or -1 to terminate another task's wait request.

```
.SYSTM
.WCHAR
error return
normal return
```

The error return is taken if a second task tries to suspend itself for a keyboard character while another task is still suspended for a wait character. In this case, AC2 is set to the following:

AC2	Mnemonic	Meaning
47	ERSIM	A previous wait-character request is outstanding.

INTERTASK COMMUNICATION

A mechanism is provided for transmitting and receiving one-word messages between single tasks only. You can also use this mechanism to lock a task process, preventing multiple tasks from entering the process concurrently. One-word messages are deposited in locations whose contents are set to zero when they contain no message. If several tasks attempt to receive message from the same address, only the highest priority task will receive the message.

Transmit a Message (.XMT) and Wait (.XMTW)

These two calls permit the sending of a one word non-zero message by a task to an empty (all-zero) message location for another task. The difference between them is that .XMT simply causes the message to be deposited, while .XMTW deposits the message and suspends the caller. .XMTW will not cause the caller to be suspended if a .REC has already been issued for this message. The message address cannot have bit zero set if the task call .TOVLD is issued in the same program. AC0 contains the address in user address space where the message will be deposited (this address must not have bit zero set to one). AC1 contains the one word non-zero message which will be passed to the receiving task in the address given by AC0.

```
.XMT                .XMTW
error return        error return
normal return       normal return
```

The following conditions will cause the error return to be taken and an appropriate error code to be placed in AC2:

AC2	Mnemonic	Meaning
43	ERXMT	The message address is already in use.
115	ERXMZ	Zero message word.

Transmit a Message from a User Interrupt Service Routine (.IXMT)

Whenever a device requiring special user service generates an interrupt request, the entire task environment becomes frozen until servicing of the special user interrupt is completed. All tasks will resume their former states when the environment is restarted unless the user transmits a message to one of them by means of the .IXMT call* from the interrupt service routine. Rescheduling of the program and task environment may occur when the task environment is restarted, depending upon the exit selected from the user interrupt routine.

If the task for which a message is intended has issued a .REC for the non-zero message that task's state is changed from suspended to ready even though task activity is in suspension. Contents of all accumulators are destroyed upon return from .IXMT, and you must restore AC3 and AC2 before attempting an exit from the service routine (see Chapter 6, Servicing User Interrupts, and .UIEX). As with .XMT, .IXMT deposits the non-zero message in a location specified by AC0. The contents of the location must be zero when you invoke .IXMT. AC1 contains the message which will be transmitted.

```
.IXMT
error return
normal return
```

These error conditions can be signalled in AC2:

AC2	Mnemonic	Meaning
43	ERXMT	Message address is already in use.
115	ERXMZ	Zero message word.

Receive a Message (.REC)

This command returns a message in AC1 that another task or interrupt service routine has posted by a transmit command, and restores the contents of the message address to all zeros. The message address must be lower than 2^{15} (i.e., bit 0 must not be set).

*IXMT and certain other user interrupt calls are not task calls in the strict sense of the word, since there are no TCBs associated with them. (The Task Scheduler and task environments are in suspension.) See Chapter 6.

If the .REC task call has been made but the transmitter has not yet issued the message, the receiving task remains suspended until the message is sent. If the message has already been issued and if the task has not also been suspended by ,ASUSP (or .TIDS), control returns to the task scheduler. Otherwise the task remains suspended until readied by .ARDY. If several tasks attempt to receive the same message, only the highest priority task will receive the message.

Required input to this call is the message address in AC0. The format of the call is:

```
.REC
normal return
```

There is no error return from a .REC command.

Locking a Process via the .XMT/.REC Mechanism

The .REC and .XMT commands can be used to lock and unlock a process or data base which is shared by several tasks, preventing more than one task at a time from accessing the data base or the process path. In essence, the procedure is to define a synchronization word, the message location, which all tasks will attempt to receive. The task in control of the locked resource then issues an .XMT to the synchronization word when the resource is to be made available to the other waiting tasks. The highest priority task waiting to receive (.REC) the synchronization word is then readied and gains unique control of the resource. This task, in turn captures the use of the resource until it unlocks the resource by issuing an .XMT to the synchronization word, etc.

This technique requires that the locking facility be initialized before any tasks use it. Initialization can be performed either by setting the synchronization word initially to a non-zero value, or by having an initialization task issue an .XMT to the synchronization word.

USER OVERLAY MANAGEMENT

The use of user overlays in a multitask environment presents some considerations which were unnecessary in a single task environment. In a multitask environment, .OVLOD may be issued by a task if it is the only task in the environment which will issue overlay requests. Furthermore, .OVLOD and. TOVLD (the multitask overlay load request) command cannot both be issued in a multitask environment. If .TOVLD is used, the overlay numbers range from 0 to 127 (see Appendix D).

As part of its resource management activities, the Task Scheduler maintains a record called the overlay use count (OUC) of the number of tasks using a currently resident user overlay. This count is decremented each time a task issues the overlay release command, .OVREL. When the overlay count goes to zero, some other overlay may then be loaded. As long as any ready task is using a resident overlay and has not yet released this resource, no other user overlays can be loaded. This holds true even if some higher priority task issues an overlay request. Synchronization of tasks waiting for overlays is accomplished via bit T of each task's TPRST. If an overlay use count goes to zero and an overlay load request for the released yet currently resident overlay is issued, the overlay count is then incremented. One of two actions then occurs, depending on whether the overlay load (.TOVLD) request was issued conditionally or unconditionally.

If .TOVLD is issued conditionally, the requested overlay is loaded if it is not core resident, and it is not loaded if it is already core resident. One consequence of this is that overlay code must be reentrant if such overlays are loaded conditionally. This is so since it would be impossible to initialize the overlay code by having it reloaded when it is already core resident.

If the overlay use count has gone to zero and .TOVLD is issued unconditionally, the requested overlay will be loaded regardless of whether or not it is currently core resident. Thus use of the .TOVLD command unconditionally permits the use of non-reentrant overlays, since unconditional .TOVLD requests always cause the desired overlay to be loaded.

Load a User Overlay (.TOVLD)

This command requests the use of the overlay area and the loading of the overlay whose node/number word is in AC0. The node number is in the left byte and the overlay number is in the right byte. The request is issued conditionally if AC1 contains 0 upon entry, and is issued unconditionally if AC1 contains -1 upon entry. AC2 must contain the channel number on which overlays were previously opened by a .OVOPN command (see Chapter 4).

If the load request is conditional and the area is free, the overlay is loaded. If the area already contains the requested overlay, return to the Scheduler is made immediately. Since another task is also using the overlay, this implies that the code must be reentrant. If another overlay is currently in the area and the overlay use count has gone to zero, the caller is suspended until the area becomes free.

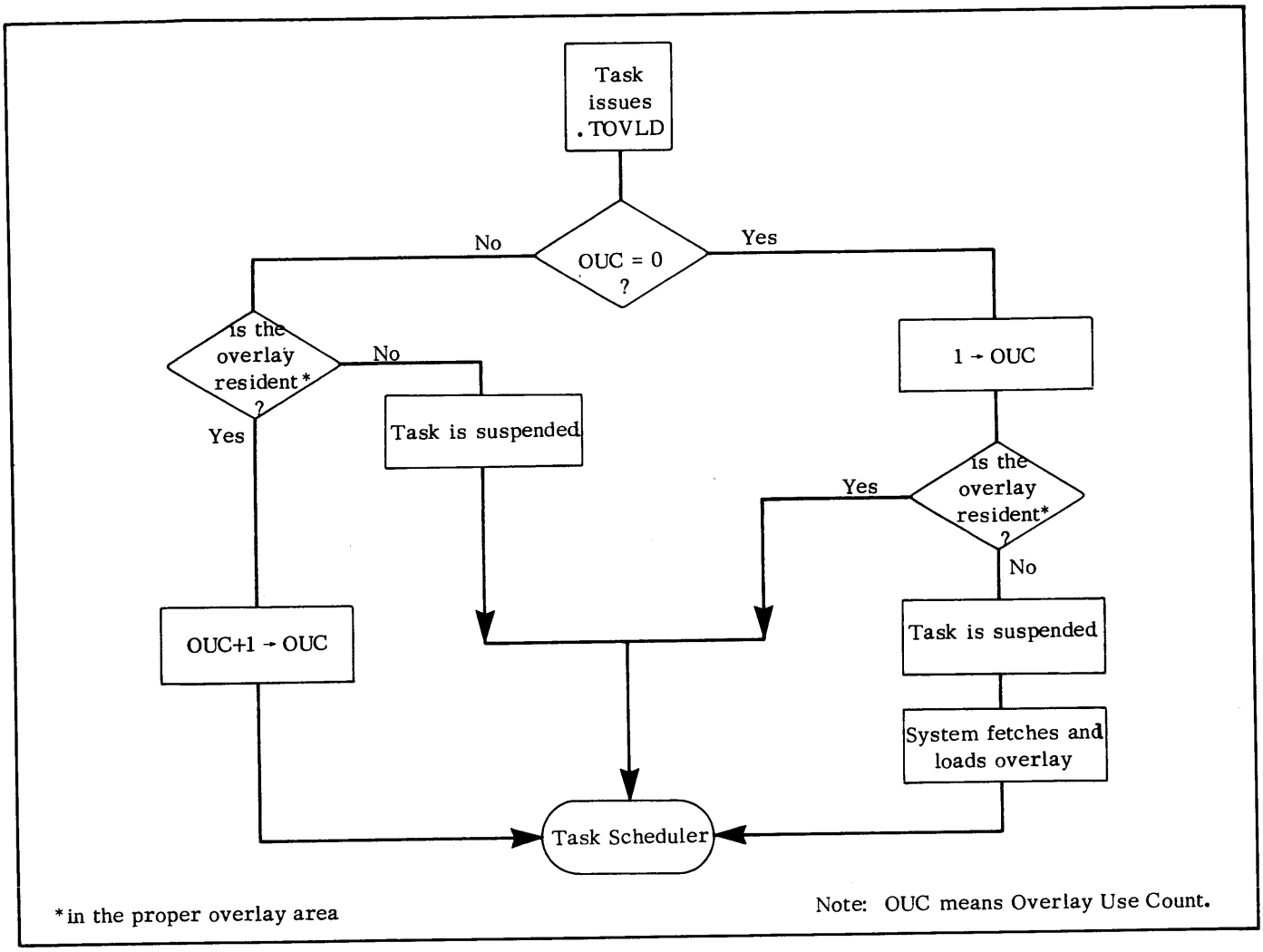


Figure 5-3. .TOVLD Logic Sequence

If the load request is unconditional and the area is free, the overlay is loaded regardless of whether it is currently core resident or not. If the overlay use count has not gone to zero (freeing the area), the caller is suspended (bit T of TPRST) until the area becomes free.

```
.TOVLD
error return
normal return
```

All overlay requests must be paired with an eventual overlay release (.OVREL/.OVEX/.OVKIL) or the area will be reserved indefinitely. An error return is possible, with the following codes given:

AC2	Mnemonic	Meaning
37	EROVN	Invalid (nonexistent) overlay number.
40	EROVA	Overlay file is not a contiguous file.
101	ERDTO	Ten-second disk timeout occurred.

Queue a Core-resident or Overlay Task (.QTSK)

This command periodically initiates a task and queues it for execution. If the task resides within a user overlay, this call loads the user overlay. If there is no TCB currently available for the creation of the new task, this call will be carried out as soon as a TCB becomes available. If two tasks are queued for execution at the same time of day, the higher priority task will receive control first. After each time that a new task is created and activated by this call, you must ensure that this task is killed, suspended, etc. If the task resides within an overlay, you must release the overlay area.

AC2 contains the starting address of a table, QTLN (see file PARU.SR, on the tape supplied with the system) words long, which describes the priority of the task, the time it is to be created, etc. The following task queue table describes QTLN's entries.

User Task Queue Table

Displacement	Mnemonic	Meaning
0	QPC	Starting address of task.
1	QNUM	Number of times to queue the task (-1 if the task is to be queued an unlimited number of times).
2	QTOV	Node number/overlay number (-1 for core-resident tasks).
3	QSH	Starting hour (-1 if the task is to be queued immediately).
4	QSMS	Starting second in hour (reserved but unused if QSH=-1).
5	QPRI	Task I.D./task priority.
6	QRR	Rerun time increments in seconds.
7	QTLNK	System word.
10	QOCH	Overlay channel (unused by core-resident tasks).
11	QCOND	Conditional/unconditional load flag (unused by core-resident tasks).
12	QLDST	System word (load status).

Entry QPC contains the entry point in the user overlay or core-resident task where control will be directed when the task is raised to the executing state. QNUM is an integer value describing the number of times the task will be queued. The task will be queued QNUM times (or without limit if QNUM = -1) unless the task call .DQTSK is issued. This call halts the queuing of the specified task, essentially bypassing the value specified by QNUM.

QTOV contains the node number in the left byte, the overlay number in the right byte for overlay tasks; for core-resident tasks this word must be set to -1. Since node numbers and overlay numbers are assigned at load time (depending upon the order of binary names in the load command) users are cautioned to insure that the values of QTOV correspond to the values assigned at load time.

Entries QSH, QSMS, and QRR all affect the time that the task will be created. QSH contains the hour to execute, and QSMS contains the second within that hour that the task will become created. If QSH contains -1, the task will be created immediately.

If QSH occurs before the current time of day, the task is queued for the next day. If QSH is greater than 24:00 hours and less than 48:00 hours, the task will be queued for the next day. If QSH is equal to (24*d) + h, the task will be queued in d days.

QRR contains the increment in seconds between each time the task will be created.

QPRI contains the task I.D. (if any) in its left byte and task priority in its right byte. If a task with the same I.D. exists at the time this task is activated, this task's I.D. number will be cleared to zero. QTLNK is maintained by the system. QOCH must contain the number of the channel upon which the overlay file was opened by a previous .OVOPN call. QCOND must contain a minus one if the overlay load is to be unconditional. Bit 0 of QLDST is set if the task is currently being loaded; bit 15 is set if a request to dequeue the task has been received. QOCH, QLDST, and QCOND are unused by core-resident queued tasks.

With AC2 containing a pointer to QPC of the User Task Queue Table, the calling sequence of this task call is:

```
.QTSK
error return
normal return
```

If the error return is taken, AC2 will contain the following code:

AC2	Mnemonic	Meaning
50	ERQTS	Illegal information in Task Queue Table.
117	ERQOV	.TOVL not loaded for an overlay queued task.

.QTSK continues on next page.

If the error return is not taken, control returns to the task issuing the call at the normal return based on the task's priority; the calling task does not become suspended. When the queued task gets control, AC2 will contain a pointer to the Task Queue Table.

ERQOV is taken as a .SYSTM.ERTN from the queuing package. To prevent its occurrence when queuing overlay tasks, you must include a .EXTN of .OVKIL, .OVREL, .TOVLD or .OVEX.

Dequeue a Core-resident or Overlay Task (.DQTSK)

This call dequeues a task which has been queued for execution by task call .QTSK. In effect, the .DQTSK call bypasses the value which is currently stored in displacement QNUM of the task's User Task Queue Table. If at some later moment the task is requeued by a call to .QTSK, the queuing process will resume its normal course since .DQTSK does not actually modify the contents of QNUM.

The format of this call is as follows:

AC1 - I.D. of the task to be dequeued.

```
.DQTSK
error return
normal return
```

Upon a normal return, AC2 returns the base address of the task's queue table (QPC). If the error return is taken, the following code is given:

AC2	Mnemonic	Meaning
61	ERTID	Task I.D. error.

Release an Overlay Area (.OVREL)

This command decrements the overlay use count and releases the area if the use count equals zero. This command must not be issued from the overlay which is to be released (see .OVKIL).

Required input to this call is:

AC0 - user overlay area number in left byte;
user overlay number in right byte.

```
.OVREL
error return
normal return
```

An error return from .OVREL is possible with AC2 containing the error code:

AC2	Mnemonic	Meaning
37	EROVN	Invalid overlay number; the overlay area is not being occupied by this user overlay.

Release an Overlay and Return to the Caller (.OVEX)

This command decrements the overlay use count and releases the area if the overlay use count equals zero. Additionally, control returns to an address specified by the caller, typically the return address of the caller if returning from a subroutine with an overlay.

Required input to this call is:

AC0 - left byte contains the overlay area number;
right byte contains the overlay number.
AC2 - Return address upon successful execution of this call.

The format of this call is:

```
.OVEX
error return
```

One error return from a .OVEX is possible, with AC2 containing the following:

AC2	Mnemonic	Meaning
37	EROVN	Invalid overlay number; the overlay area is not being occupied by this user overlay.

Kill an Overlay Task and Release the Overlay (.OVKIL)

.OVKIL kills the calling task and decrements the overlay use count. This is the normal method of terminating a queued, overlayed task. This call may be issued from within the user overlay which is to be released. Required input to this call is:

AC0 - user overlay area number in left byte;
user overlay number in right byte.

The format of the call is:

```
.OVKIL
error return
```

One error return is possible:

AC2	Mnemonic	Meaning
37	EROVN	Invalid overlay number.

USER/SYSTEM CLOCK COMMANDS

All system clock commands can be issued from either a single task or from a multitask environment. Since these commands are of little practical use in a single task environment, the system and user clock commands are presented in this chapter instead of in Chapter 3.

Define a User Clock (.DUCLK)

This command permits the definition of a user clock. When the interval you have defined expires, the Task Scheduler and multitask environment--if any--are placed in suspension, and control goes to the routine you have specified. Each time control goes to your routine, AC0 will contain a value indicating where control came from at the time of the interruption. AC0 will contain 177776 if control was in the system. AC0 will contain -1 if control came from the system while it was in an idle loop (i.e., awaiting an interrupt). AC0 will contain the PC if control was in user space.

To issue the .DUCLK call, you must pass in AC0 the integer number of system RTC cycles which are to elapse between each user clock interruption. AC1 must contain the address of your routine which will receive control when each interval expires. No system or task call (except for .UCEX and .IXMT) may be issued from this routine. Moreover, assembly instruction INTEN must not be issued. The format of this call is:

```
.SYSTM
.DUCLK
error return
normal return
```

If the error return is taken, one of the following error codes are given:

AC2	Mnemonic	Meaning
45	ERIBS	A user clock already exists.

Upon entering the user clock routine, AC3 will contain the address of the return upon entry to the user routine. You must use this address in the .UCEX command to exit from the user clock routine.

Exit from a User Clock Routine (.UCEX)

Upon a user clock interrupt, AC3 will contain the address of the return upon entry to the routine specified in .DUCLK. To return from the user clock routine, you must load AC3 with this return address, and issue .UCEX.

Rescheduling of both the task environment and the program environment will occur upon exit only if AC1 contains some non-zero value.

The format of this call is:

AC1 - Zero only if rescheduling is to be suspended.
AC3 - Return address.

```
.UCEX
```

Control returns to the point outside the user routine which was interrupted by the user clock. No errors are possible from this call. This call can be issued in a single task environment.

Remove a User Clock (.RUCLK)

This system command removes a previously defined user clock from the system. The format of the call is:

```
.SYSTM
.RUCLK
error return
normal return
```

One error return is possible:

AC2	Mnemonic	Meaning
45	ERIBS	No user clock is defined.

Examine the System Real Time Clock (.GHRZ)

This system call permits you to examine the Real Time Clock frequency. The frequency is returned in AC0, in the following manner:

AC0	Meaning
0	There is no Real Time Clock in the system.
1	Frequency is 10 Hz.
2	Frequency is 100 Hz.
3	Frequency is 1000 Hz.
4	Frequency is 60 Hz (line frequency)
5	Frequency is 50 Hz (line frequency)
6	MicroNOVA internal clock.

.GHRZ continues on the next page.

The format of this call is:

```
.SYSTM
.GHRZ
error return
normal return
```

The error return is never taken.

TASK IDENTIFICATION CALLS

Get a Task's Status (.IDST)

This command obtains a code describing a task's status The task whose status is to be obtained is specified by inputting its identification number in AC1. The format of this command is:

```
.IDST
normal return
```

The code describing the task's status is returned in AC0.

0 - Ready
1 - Suspended by a .SYSTM call.
2 - Suspended by a .SUSP, .ASUSP, or TIDS.
3 - Waiting for a message to be sent or received.
4 - Waiting for an overlay area.
5 - Suspended by .ASUSP, .SUSP, or .TIDS and by .SYSTM.
6 - Suspended by .XMTW or .REC and by .SUSP, .ASUSP, or .TIDS.
7 - Waiting for an overlay area and suspended by .ASUSP, .SUSP, or .TIDS.
10 - No task exists with this I.D. number.

The base address (displacement TPC) of the task's TCB is returned in AC2.

There is no error return from this call.

Kill a Task Specified by I.D. Number (.TIDK)

This command kills only that task whose identification number is specified. Tasks suspended by .SUSP, .TIDS, or .ASUSP will be raised to the highest priority and will transfer to a kill processing address or will then be terminated. The format of this command is:

AC1 - I.D. of task to be killed.

```
.TIDK
error return
normal return
```

With the error return, this code is given:

AC2	Mnemonic	Meaning
61	ERTID	Task I.D. error.

Change the Priority of a Task Specified by I.D. Number (.TIDP)

This command changes the priority of that task whose identification is specified by AC1. You must give new priority (from 0 to 255 inclusive) in AC0, bits 8 to 15. The format of this command is:

```
.TIDP
error return
normal return
```

With the error return, this code is given:

AC2	Mnemonic	Meaning
61	ERTID	Task I.D. error.

Ready a Task Specified by I.D. Number (.TIDR)

This command readies only that task whose identification number is input in AC1. That is, this command resets bit U in word TPRST of this task's TCB, which was set by a previous call to .ASUSP, .SUSP, or .TIDS. The format of this call is:

```
.TIDR
error return
normal return
```

With the error return, this code is given:

AC2	Mnemonic	Meaning
61	ERTID	Task I.D. error

If the specified task's bit U of TPRST was already reset, the normal return is taken.

Suspend a Task Specified by I.D. Number (.TIDS)

This command suspends only that task whose identification number is input in AC1. That is, this call sets bit U in word TPRST of the specified task's TCB. The format of this command is:

```
.TIDS
error return
normal return
```

If no task exists with the specified I.D. number, the error return is taken:

AC2	Mnemonic	Meaning
61	ERTID	Task I.D. error

If the task's bit U in word TPRST is already set, the normal return is taken.

DISABLING THE TASK SCHEDULER

Generally the DOS multitask calls permit you to manage a multitask program with complete satisfaction; the task scheduler always gives CPU control to the highest priority ready task. In some instances, however, you may want to suspend briefly the rescheduling function performed by the task scheduler. For example, you might suspend rescheduling to control race conditions between several tasks competing for a single resource. Since the disablement of rescheduling--even briefly--is a drastic step, it should be performed with caution. Note that disabling rescheduling will not affect system activities such as interrupt service. Moreover, the system will reactivate the scheduling function as soon as any system call is issued, even though you may not yet have reenabled rescheduling explicitly.

Disable Rescheduling (.DRSCH)

This task call prevents rescheduling in this program environment until either scheduling is reenabled explicitly or a system call is issued. Task call .DRSCH should be issued only with caution since it disrupts the ordinary management of the multitask environment; the task that issues this call will retain control even though other higher priority tasks may be ready.

The format of this call is as follows:

```
.DRSCH
normal return
```

No errors are returned.

Reenable Rescheduling (.ERSCH)

Normally, the task scheduler is enabled and manages the multitask environment within its program. If task scheduling has been suspended by a call to .DRSCH and no system call has been issued, you can reactivate the scheduler by issuing task call .ERSCH. This call has the following format:

```
.ERSCH
normal return
```

No errors are possible. This call has no effect when scheduling is enabled.

TASK CALL SUMMARY

NOTE: All task names must be declared external.

.ABORT	Terminate a task immediately.
.AKILL	Kill all tasks of a given priority.
.ARDY	Ready all tasks of a given priority.
.ASUSP	Suspend all tasks of a given priority.
.DQTSK	Dequeue a previously queued task.
.DRSCH	Disable the rescheduling of the task environment.
.ERSCH	Reenable the rescheduling of the task environment.
.IDST	Get the status of a task.
.IXMT	Transmit a message from a user interrupt service routine.
.KILAD	Define a kill-processing address.
.KILL	Kill the calling task.
.OVEX	Release an overlay and return to the caller.
.OVKIL	Kill an overlay task and release the overlay.
.OVREL	Release an overlay area.
.PRI	Change the priority of a task.
.QTSK	Queue a core-resident or overlay task.
.REC	Receive a task message.
.SMSK	Modify the current interrupt mask.
.SUSP	Suspend the calling task.
.TASK	Initiate a task.
.TIDK	Kill a task specified by I.D. number.
.TIDP	Change the priority of a task specified by I.D. number.
.TIDR	Ready a task specified by I.D. number.
.TIDS	Suspend a task specified by I.D. number.
.TOVLD	Load a user overlay in a multitask environment.
.UCEX	Return from a user clock routine.
.UIEX	Return from a user interrupt routine.
.UPEX	Return from a user power fail service routine.
.XMT	Transmit a task message.
.XMTW	Transmit a task message and wait for its receipt.

Figure 5-4. Task Command Summary

END OF CHAPTER

CHAPTER 6
USER INTERRUPTS AND
POWER FAIL/AUTO RESTART PROCEDURES

This chapter has two sections. The first describes establishing and referencing user interrupts; the second covers the system's handling of power failures. In some cases, you may want to write your own routine for handling power failures. If so, you can use calls from the first section.

The material in this chapter applies to both single and multitask environments.

SERVICING USER INTERRUPTS

When the system receives an interrupt request, it chooses an interrupt service routine from its device interrupt service table. This table contains pointers to Device Control Tables (DCTs) for devices specified at SYSGEN; it also contains pointers to three-word DCTs built by the user to service interrupts from devices which were not SYSGENed. An application Note called RDOS User Device Driver Implementation describes the SYSGENed DCTs. You use a three-word DCT to provide an interface between the system and your service routine. Your DCT tells DOS how to mask devices and where to find the service routine. It looks like this:

Entry

Displacement	Mnemonic	Purpose
0	DCTBS	(Unused)
1	DCTMS	Interrupt service mask
2	DCTIS	Interrupt service pointer

DCTIS is a pointer to the routine which serves this specific device interrupt request. DCTMS is the interrupt mask that you want to be ORed with the current interrupt mask while DOS is in your interrupt service routine. This mask establishes which devices -- if any -- will be able to interrupt the currently interrupting device. (The interrupts are on when you enter the routine but are masked for this priority device.) The mechanism of device interrupts is covered in the manual How to Use the NOVA Computers, Section 2.4.

After a user interrupt occurs, control goes to your service routine; AC3 contains the return address required for exit from your routine, and AC2 contains the address of the DCT. The task call .UIEX exits from the routine; this call may be issued in both single and multitask environments.

All user devices are removed from the system when either a program swap or chain occurs. When the system receives user interrupt on a program level which has not identified the user device, it issues a NIOC to the device and then returns to normal program execution.

Whenever a device requiring special user service generates an interrupt request, the entire task environment halts until the interrupt has been serviced. All tasks will resume former states when the environment restarts unless you transmit a message to one of them by means of the .IXMT call from the interrupt service routine. See .IXMT, Chapter 5. Rescheduling of the program and task environment may occur upon return from the routine, depending on the contents of AC1 in the return command (.UIEX, below).

In addition to .IXMT, the task calls .SMSK, .UIEX, and .UPEX can be issued by a user interrupt or user power fail routine. You will find them all below.

Identify a User Interrupt Device (.IDEF)

This call introduces to the system a device which was not identified at SYSGEN time, whose interrupts you want the system to recognize. The .IDEF call places an entry in the interrupt vector table.

AC0 must contain the device code of the new device; AC1 must contain the address of the new device's DCT. This address must exceed 400_8. The format of this command is:

```
.SYSTM
.IDEF
error return
normal return
```

Possible error messages are:

AC2	Mnemonic	Meaning
36	ERDNM	Illegal device code (more than 77_8). Device code 77_8 is reserved for the power monitor/auto restart option.
45	ERIBS	Interrupt device code in use.

Exit from a User Interrupt Routine (.UIEX)

This call returns control to a program after a user interrupt; you can use it in both single and multitask environments. When a user device interrupt occurs, AC3 will contain the return address from the user routine and AC2 will contain the address of the user device DCT. If your routice uses AC3, your program must restore the address that AC3 had upon entry to the routine. Without this address, the system cannot return from your routine. Similarly, AC2 must be restored to the address of your DCT.

Upon return, rescheduling of both the task and program environment will occur upon exit only if AC1 contains some non-zero value.

Control returns to the point where the interrupt occurred.

Required input to .UIEX is:

AC1 - Zero only is rescheduling is to be suppressed.
AC2 - Original address of DCT (value at interrupt).
AC3 - Original return address (value at interrupt).

The format of the call is:

```
.UIEX
```

No errors are possible from this call.

Remove a non-SYSGENed Interrupt Device (.IRMV)

To prevent the system's recognition of an interrupt device which was identified by the .IDEF command, issue the .IRMV command. AC0 must contain the user device code which is to be removed from the system.

The format of the .IRMV command is:

```
.SYSTM
.IRMV
error return
normal return
```

One possible error message may be given:

AC2	Mnemonic	Meaning
36	ERDNM	Illegal device code (more than 77_8), or attempt to remove a SYSGENed device.

For more information about communicating with tasks from a user interrupt service routine, see Chapter 5, Transmitting a Message from a User Interrupt Service Routine.

Modify the Current Interrupt Mask (.SMSK)

Use this task call to change the user interrupt mask of a service routine, in both single and multitask environments. Whenever a user interrupt occurs, the interrupt mask is ORed with the mask in DCTMS of the user DCT to produce the current interrupt mask. The .SMSK call allows you to change DCTMS, and produce a new mask which is the logical OR of the old mask (upon entry to the service routine) and a new value. The accumulators are destroyed by .SMSK, and you must restore them for the subsequent .UIEX.

```
AC1 - New value to be ORed with old mask.
.SMSK
normal return
```

There is no error return possible from this call.

POWER FAIL/AUTO RESTART PROCEDURES

DOS provides software support for the power fail/automatic restart option. When the system detects a power loss, it transfers control to a power fail routine which saves the status of all accumulators, the PC, and Carry.

If the console key was in the LOCK position when power failed, the system console will display this message when power returns.

```
POWER RESTORED
```

The system then restores task state variables, resuming operation at the point of interruption.

If the console key was in the ON position when input power failed, you must set all data switches to zero (down) and lift START when power returns. This outputs the POWER message and restores task state variables.

The following system devices are given power-up restart service:

teletypewriters and CRTs
disks
line printer
paper tape reader/punch
card reader
plotter

Character output devices may lose one or more characters during power up. The card reader may lose up to 80 columns of information on a single card. The line printer may lose up to a single line of information. Since power up service for disks includes a complete reread or rewrite of the current disk block, no disk information is lost. Devices requiring operator intervention (like line printers, card readers, etc.) must receive such action if power was lost for an extended period of time.

No power up service is provided for magnetic tape units. You must use the system call .IDEF to provide power up service for special user devices. To do this, pass 77_8 in AC0 and pass the starting address of the user power up routine in AC1. Use the task call .UPEX (below) to exit from the power up routine. .UPEX forces program rescheduling. System and task calls behave the same way in both interrupt and power fail routines.

Exit from a Power Fail Service Routine (.UPEX)

Use .UPEX to return from a user power fail routine in single or multitask environments. When the system enters a user power fail service routine, AC3 contains the address return address required. If AC3 is changed by the routine, you must restore this address before using the task call .UPEX to exit from the routine.

The format of this call is:

AC3 - Return address upon entry to the routine

.UPEX

Control returns to the location which was interrupted by a power failure. No error return or normal return need be reserved.

END OF CHAPTER

CHAPTER 7

THE DOS COMMAND LINE INTERPRETER

The command line interpreter (CLI) is the primary interface between DOS and a user. CLI accepts valid commands entered through the user console, or from some other input file (disk file, tape file, card reader, etc.). All characters in the standard ASCII character set are available to you, although some have special meaning to CLI.

CLI is a system utility of DOS, and most of its commands are implementations of DOS system commands; e.g., RELEASE and .RLSE. You input system commands in assembly language, as part of a program, but you input CLI commands interactively, from a console. The CLI loads the accumulators, sets bits, and executes machine operations for you.

A valid CLI command normally consists of the command name and one or more arguments. In many cases you can append switches to commands and arguments to modify command execution.

CLI's commands can create and maintain files and directories, and execute many other important tasks. For example:

```
CREATE FILE1 FILE2 FILE 3
```

CREATE is the command name. CLI creates three files, with the names entered as arguments. Other commands allow you to:

- delete a file
- rename a file
- concatenate two or more files
- set file use count to zero
- change file access attributes
- chain a program (file) to overwrite CLI

SYSTEM UTILITIES

You have access to Data General system utility programs that perform assemblies, source and disk file editing, program debugging, etc. Whenever you enter the name of a system program or a user program as the first word in a CLI command, that program is loaded and executed by DOS.

Example:

```
MAC YOURPROGRAM
```

The DOS Macroassembler is loaded, and YOURPROGRAM is assembled.

You can access the following utilities through CLI by entering the parenthesized commands:

- Editors - Text(EDIT), Multiterminal Text (MEDIT), Superedit (NSPEED) Octal (OEDIT), and Library File (LFE).
- Assemblers - Extended (ASM), Macro (MAC)
- Loaders - Relocatable(RLDR)

COMMAND LINE FORMAT

CLI reads from an input file and writes to an output file. For interactive devices, the input and output files are the user console. In this chapter, where clarification is necessary in examples, user input is underscored. Entries not underscored represent CLI output to your terminal. The normal format of a CLI command follows:

```
R
Command-name/global-switches,arg1/local-switches,...)
```

where:

R — CLI prompt character (which CLI always follows with a carriage return). Start typing the command on the next line, immediately beneath the prompt. You can append the time to this prompt by the procedure described later in this chapter.

Command-name — Name of the command or system utility.

Global switches

You can often modify a command by appending one or more global switches to a command name. A global switch applies to every argument in the command line; it immediately follows the command name, and consists of a right slash (/) followed by a letter. Generally, the meaning of a global switch depends on the command that it modifies. Each individual command entry in this chapter lists global switches.

Example:

```
R
DELETE/V FILEA
```

DELETE is the command-name, and /V is the switch that modifies the delete function:

/V Write a message to the output file verifying that FILEA has been deleted.

Comma (,) or Blank(s)

A single comma, or one or more blanks serves as a delimiting character between the commands and arguments, or as a delimiter between arguments. Two or more consecutive blanks are treated as a single delimiter.

Thus, the following two commands are equivalent.

```
DELETE FILEA,FILEB,FILEC
            and
DELETE FILEA FILEB  FILEC
```

Arguments and Local Switches

Unless a command is used alone, it is generally followed by one or more arguments. Each of these arguments may have optional switches appended.

COMMAND $argument_1$ ... $argument_n$

$Argument_1$ is the first argument. Ellipses indicate repeating arguments; $argument_n$ is the last argument in the command line.

```
DELETE FILEA FILEB FILEC FILEX
command-name
          argument1          argumentn
```

A switch which follows an argument is a local switch; it modifies that argument only. A local switch consists of a right slash (/) followed by either a number or a letter.

Local Numeric Switches

A numeric switch specifies the number of times the preceding argument is to be performed by CLI.

Example:

```
RLDR $PTR/3
RLDR $PTR $PTR $PTR
```

Each command loads three relocatable binary tapes from the paper tape reader (the RLDR command evokes the Extended Relocatable Loader.).

Numeric switches are cumulative; these commands are equivalent:

```
RLDR $PTR/1/2
RLDR $PTR/3
```

When used alone, the digit 1 in a numeric switch has no effect. The digit zero has no effect when used with other digits.

When used alone, however, the zero has the same effect as one. The following commands are equivalent:

```
RLDR $PTR/1
RLDR $PTR/1/0
RLDR $PTR/0
RLDR $PTR
```

Local Letter Switches

A local letter switch modifies the action of the command line on an argument. The meaning of each switch varies with the argument it modifies and the command line that contains the argument.

Example:

```
ASM BASEA/N BASEB/N  ERRORFILE/E)
```

This command line assembles files BASEA and BASEB, but does not list them because the local /N switches suppress the normal listing. (The global /N switch has a different meaning.) Assembly errors are output to disk file ERRORFILE.

Carriage Return (↵)	A carriage return character (octal 15) denotes the end of a command. The command is executed after you press the carriage return character on the user console.

Combining Commands

You can include two or more commands on the same command line by placing semicolons (;) between the commands.

Example:

```
CREATE FILEA;DELETE FILEX FILEY↵
```

The two commands are sequentially executed after the carriage return character is depressed.

Long Command Lines

A shift N (echoed as ↑ on the user console) typed immediately before a carriage return allows you to span one or more commands over several input lines.

Example:

```
DELETE TESTA TEST01↑↵
TEST01.SV↵
```

The command is executed as if the following had been input:

```
DELETE TESTA TEST01 TEST01.SV↵
```

Arguments or commands can also span two or more lines.

Example:

```
CREATE TESTA TEST01 TEST01.SV.A;DE↑↵
LETE FILEX FILEY↵
```

In both examples the shift N causes the following carriage return to be ignored as a command delimiter.

ASCII Character Set

Appendix C contains a list of ASCII characters used by CLI. Some ASCII characters have special syntactical significance in CLI commands; these are summarized in Table 2-1.

Prompt Messages

The initial CLI prompt is a single "R", output at the completion of each command. You can append the time of day to this prompt (or remove it) by typing the command

```
.↵
```

Example:

```
CREATE A↵
R                 ←initial prompt is a simple "R".
.↵                ←you evoke the expanded version.
9:14:01           ←expanded prompt.
R
LIST A↵           ←you issue a new command.
A.  D0            ←file A is listed.
9:14:37           ←and expanded prompt is output.
R
.↵                ←you remove the expanded prompt.
R
DELETE A↵         ←you issue a new command.
R                 ←simple R response to this
                    command.
```

CLI Punctuation Summary

The following characters serve as delimiters or instructions to CLI. Other special characters, which allow you to manipulate disk files and command lines are described later in this chapter under Disk File Manipulation.

Table 7-1. CLI Punctuation Summary

Symbol	Function	Example
) or ↓	The carriage return key ()) terminates an input command line and activates CLI. The CTRL and L keys (↓) are identical to the carriage return.	CREATE A B) CREATE A B↓
\	The SHIFT and L keys (\) delete an entire line.	CCREAGE\)
←	The RUBOUT key (echoed on TTYs as ←) erases the last character entered. On most CRT terminals the last character will disappear each time you press RUBOUT.	CC←READ←TE
, (space)	You must use a comma or a space to separate arguments. Extra spaces have no effect.	DELETE A,B) DELETE A B) DELETE A B)
/	Use the right slash key before a character to specify a switch.	LIST/A)
;	Use the semicolon (;) to delimit a command. Two or more commands can be on a line separated by semicolons. No commands are executed until you enter a carriage return.	CREATE A;LIST)
↑	Use the shift and N keys (↑) for long command lines. The next carriage return will generate a line feed without terminating your command line. The ↑ must immediately preceed the carriage return.	RENAME A ALPHA↑) B BETA↑) E EPSILON)
.)	Use this combination to add or remove the extended prompt.	.) 14:34:54
( ) < >	Parentheses and angle brackets enclose multiple arguments in a command line. Brackets can be nested, parentheses cannot. See Multiple Arguments.	MAC/V PART:↑) <A,B,C,D>.SR)
(,,,) <,,,>	Use this format for multiple arguments or commands in one line. See Multiple Arguments.	(ASM, RLDR) D)
* -	The asterisk or dash around a file name enable CLI to search for a group of related filenames. See Template Expansion.	LIST .A*) DELETE .A-)
@	A combination of commands. See Grouped Commands.	ASM@FOO@)
[,,]	User overlay definition (see RLDR command).	RLDR R[A,B]

DEVICE AND DISK FILES

A file is a logical collection of information, or a physical device transmitting or receiving information. All device and disk files are accessible by file name; see Disk Files below.

Reserved Device Names

Here is a brief list of reserve device names. You can find more detail at the beginning of Chapter 2.

$CDR	Card reader.
DPn	Diskette unit (not drive) number 0 through 3.
$LPT	Line printer.
MTn	Magnetic tape unit, number 0 through 7.
$PLT	Plotter.
$PTP	Paper tape punch.
$PTR	Paper tape reader.
$TTI	Teletypewriter or console input.
$TTO	Teletypewriter or console output.
$TTP	Teletypewriter punch.
$TTR	Teletypewriter reader.
$TTI1	Second console input.
$TTO1	Second console output.

Reserved Device Operation

Some commands require manual operation of an I/O device. If you issue such a command, you will receive a message telling you what to do. For example, the command:

```
XFER/A $PTR A.SR)
```

requests that a source file be transferred from the paper tape reader to a disk file named A.SR. The system replies:

```
LOAD $PTR, STRIKE ANY KEY.
```

You then load the paper tape reader and strike any key on your console. (The key is not echoed)

Some commands require 2 steps. For example, the command:

```
APPEND NEWFILE $PTR/2 )
```

requests that a file called NEWFILE be created from two files input from the paper tape reader. The following responses will occur:

```
LOAD $PTR, STRIKE ANY KEY.
LOAD $PTR, STRIKE ANY KEY.
```

The second message is typed out after the first file has been transferred.

DISK FILES

DOS is a disk-based system, and its utility programs and most CLI commands work best on disk files. We recommend that you transfer any file to disk before attempting to edit, assemble, compile, load (RLDR), or execute it. You can use the XFER command for the transfer.

A disk file is a collection of information on disk. It can be a directory, which contains many disk files; it can be a data file; it can be a link entry or a resolution entry, which contains nothing, but points to other files; it can be an overlay, swap or chain file, which moves in and out of memory to extend user address space. Each disk file is stored in disk, word for word, exactly as it will be loaded into memory.

CLI has many features to help you manage disk files.

Each disk file has a name, which consists of any number of alphanumeric characters (only the first 10 are significant to the system). Each filename can have an extension, which either you or the CLI assigns. Every file also has one or more attributes (which you can assign or change with the CHATR command) and characteristics (which the system assigns when you create the file).

You will find a description of extensions below; attributes and characteristics are covered in the CHATR command, and at the beginning of Chapter 2.

Directories

Directories contain other files, and allow you to organize information by name and category; there are both system directories and user directories. You can create a directory with the CDIR command, or change the current default directory with the DIR command. Chapter 2 has more information on directories.

Disk File Name Extensions

To make a filename truly unique, you can append an extension to it. For this extension, use two or more alphanumeric characters (only the first two are significant to the system). Separate the filename from the extension with a period (.).

In some cases, CLI itself appends its own key extensions to a file name, to indicate the type of data the file contains. For filename A, the CLI extensions and their meanings are:

A.SR - SouRce file
A.RB - Relocatable Binary file
A.LS - LiSting file
A.SV - core image (SaVe file)
A.OL - OverLay file
A.DR - DirectoRy file

Generally, if a filename has an extension, you must enter both filename and extension to access the file. This applies to any extension that either you or CLI assigns. You can use a template character instead of an extension for some commands (template characters are covered below).

Certain commands, however, will search disk for the filename argument with an appropriate extension, and if unable to find the name with the proper extension, will search for the name without an extension. These are ASM, MAC, MKABS, MKSAVE, RLDR, SAVE, and filename alone, without a command word. For example, the command

```
ASM A )
```

searches for the file named A.SR. If A.SR is found, it is used as the source file for the assembly. Otherwise, CLI searches for A. If you had given sourcefile A the extension 2B, CLI would access A.2B.

When A (or A.SR or A.2B) is found, it is assembled into a relocatable binary file named A.RB. If you specified a listing file, it would be called A.LS. CLI creates these names by dropping the original extensions, if any, and adding the extension .RB.

After A is assembled, you might want to load it into core. The Relocatable Loader could not load the source file A.SR or A.2B (which was in assembly language format), but it can load the relocatable binary A.RB. You load A by calling the loader:

```
RLDR A )
```

CLI searches for A.RB and if not found, for A. The CLI creates an output file for the relocatable loader called A.SV. The extension .SV signifies a save file. If you specified overlays for A, RLDR would also create an overlay file, which CLI would name A.OL.

The commands SAVE and MKSAVE also have a save file as output. For all three commands (RLDR, SAVE, and MKSAVE), CLI adds the extension .SV to the name of the output file. If you attempt to enter another extension, it will be ignored. For example:

```
SAVE A.2B)
```

stores a core image as a save file on disk. CLI names the file A.SV, and ignores extension .2B.

The command MKABS also requires no extension. If you type:

```
MKABS A $PTP )
```

CLI searches for A.SV and, if not found, for A.

To simply execute a save file, you need specify no extension. The command

```
A )
```

executes A.SV (or A).

The .DR extension does not relate to assembly or loading; CLI automatically adds it to any directory name you create.

For most other commands, you must enter the appropriate file name extension. For example, the command:

```
DELETE A )
```

deletes only the file named A. Files named A.SR, A.SV, A.RB, A.2B, or A.33 would not be deleted.

DISK FILE MANIPULATION

CLI offers the following special characters to help you access and modify disk files. Each character evokes a special convention, which is described in this section.

Symbol	Function	Example
(,,,)	Use commas with parentheses for multiple arguments or commands in one line.	(ASM, RLDR) D)
<,,,>	Use angle brackets and commas for in-line expansion of multiple arguments.	MAC/V PART: ↑) <A,B,>
*	The asterisk and dash are template characters in a file name or extension. They help you access many file names from one name.	LIST T**) LIST-)
@	Use the commercial at sign to group commands, and produce indirect commands.	ASM@FOO@)

Multiple Arguments

You can execute a CLI command on several arguments or argument strings by separating the arguments or strings with commas and enclosing them within parentheses. Moreover, you can execute several commands on a single argument by placing the commands within parentheses. For multiple arguments, use the form:

command (arg_1, $arg_2 \ldots arg_n$) ⌈arg_{n+1}⌉

(Modified brackets (⌈ ⌉) are used in this manual to indicate an optional entry in the command line; you need not enter material enclosed in them.)

For example, the command,

```
ASM(FILEA, FILEB, FILEC) $LPT/L )
```

executes as

```
ASM FILE A   $LPT/L )
ASM FILEB $LPT/L)
ASM FILEC $LPT/L )
```

CLI assembles three disk files, and lists the assembly on the lineprinter.

For multiple commands, use the form:

(command$_1$ ... command$_n$) arg

For example, the command line:

```
(ASM, RLDR) MYFILE
```

executes as:

```
ASM MYFILE )
RLDR MYFILE )
```

You can use multiple sets of parentheses in a command line, but you cannot nest them; e.g., ASM(A1,A2(B4)) is illegal. Parentheses must be paired.

The following command shows multiple parentheses:

```
DUMP (FILEA, FILEB) MT0:(0,1) )
```

CLI dumps FILEA on tape file 0, and FILEB on tape file 1, of MT0.

You can also use the comma-parentheses convention to complete and expand an argument name. Thus the command

```
MAC PART(1,2,3).SR )
```

expands to:

```
MAC PART1.SR )
MAC PART2.SR )
MAC PART3.SR )
```

To execute a specifier or other instruction on a group of arguments, enclose the arguments in angle brackets. You can nest angle brackets to any depth. Brackets and parentheses can be used in one command line, but cannot be overlapped. Thus (< >) or <( )> is legal, while <(>) is illegal. Angle brackets are evaluated before parentheses. In the following example:

```
MAC PART:<A,B,C> .SR )
```

CLI directs the Macroassembler to assemble files A, B, and C, within directory PART.

Most of the DOS system utilities produce disk files as output. If you input a file from a nondisk device, the nondisk devicename will become the diskfile name, with possibly unpleasant results. Consider this command:

```
MAC ($CDR,$CDR,$CDR) $LPT/L
```

In this example, three separate files would be printed on the line printer, but only the third source program input via the card reader would exist as an .RB file on disk, $CDR.RB. The previous two disk files with this same name would have been overwritten by the third binary file. You could avoid this by transferring the card reader files to disk and nameing them with the XFER command:

```
XFER $CDR A; XFER $CDR B; XFER $CDR C )
```

and after the transfer, issuing the assembly command:

```
MAC    (A,B,C)
```

Template Expansion

CLI offers two template characters to help you access similar file names. When you use one or both of these characters with a filename in a command, CLI executes the command on all related filenames in the current directory.

A filename template contains either of the following characters:

* (Asterisk)	Represents any single-character, except a period, in a filename or extension.
- (Dash)	Represents any string of characters, except a period, in a filename or extension.

For example, assume the following filenames in the current directory.

```
A
ATOM
A2$M
ADAM.SV
ADAMS
ADAMS.B2
```

The asterisk represents any single character except a period in a filename, thus the command:

```
LIST A**M )
```

would list all 4-letter filenames beginning with A and ending with M. These are ATOM and A2$M. A and ADAMS would not be listed because they are not 4 letters long; the commands LIST*) and LIST *****) respectively would list them. ADAM.SV would not be listed because it has a period and extension; the template A**M.** would access it.

The dash represents any character string of zero or more letters in a filename or extension (except a period). Thus the command

```
LIST A- )
```

would list A, ATOM, A2$M, and ADAMS. The template -M would access ATOM and A2$M, and template A-.- would access all names shown. The command LIST-.-) would list all files in the current directory.

There are some restrictions on the use of templates: you cannot use them with a device specifier (e.g., LIST DP0:-.-)), because CLI would search for the device in the current directory. All file attributes (see CHATR) restrict template access to filenames (for example, you cannot DELETE a permanent file).

When you use a template, the filename must be in the current directory. Templates are permitted in the following commands only:

BUILD	DUMP	MOVE
DELETE	LIST	UNLINK
	LOAD	

In each command description, assume that templates are forbidden unless the text indicates otherwise.

Grouped Commands

You can call a group of different commands simultaneously by giving them a name, and entering that name between commercial at signs (@).

Assume that you regularly conclude each CLI session by deleting listing files, checking the list of nonpermanent files, and determining how much disk space you have left. The command line for this would be:

```
DELETE -.LS;LIST;DISK )
```

You could write these commands into a file called END in the following way:

```
R                          ← CLI prompt
XFER/A $TTI END)           ← this command transfers
                             what you are about to type
                             on TTI to disk file END;
                             the /A switch specifies
                             ASCII transfer.

DELETE-.LS;LIST;DISK)      ← These are the three com-
                             mands you want to comprise
                             file END.

CTRLZ                      ← You enter CTRL Z to term-
                             inate the list of commands,
                             and write the group to the
                             disk file.

R                            CLI returns the prompt.
```

You can now execute the command group by typing:

```
@END@)
```

As another example, assume you have 3 source programs called PART1, PART2, and PART3. You can use the XFER command as shown above to build a file called TEST:

```
R
XFER/A $TTI TEST)
PART1 PART2 PART3)
CTRL Z
R
```

When you issue the command:

```
ASM @TEST@)
```

the 3 files are assembled.

The contents of a file on disk may, in turn, point to another file. As a simple example, suppose:

file A contains L@B@
file B contains I@C@
file C contains ST

Then the command:

```
@A@)
```

is equivalent to the command:

```
LIST)
```

You would enter this sequence this way:

```
XFER/A $TTI A)
L@B@)
CTRL Z
R
XFER/A $TTI B)
I@C@
CTRL Z
R
XFER/A $TTI C)
ST
CTRL Z
R
@A@)          ← system executes LIST command.
```

SYSTEM CONSOLE BREAKS

You can enter two different program breaks from the system console: CTRL A, and CTRL C. CTRL A terminates the current CLI command; CTRL C terminates a program and saves a snapshot of its core image.

The CTRL A interrupt halts user program activity and passes control to a special break processing routine in either the current program (if any) or the closest higher level program with a break processing routine (if any).

The CLI residing at the highest level, 0, has such a processing routine. Thus, if no lower level programs have such a routine, control passes to the CLI upon a CTRL A break; CLI halts the current operation and waits for a new command. If you issue CTRL A in a multitask environment, any tasks which have outstanding system calls will have their system actions aborted.

Pressing CTRL and C on the keyboard causes an eventual interrupt of a program and saves the core image in a file named BREAK.SV. CLI prints the word BREAK at completion of the interrupt.

CTRL C breaks differ from CTRL A interrupts in two ways: first, DOS defers the interrupt until all outstanding task calls have been completed; and second, DOS creates the BREAK.SV file (which you can preserve and rename with the SAVE command). When you execute the save file, the system resumes normal operation, and gives CPU control to the highest priority ready task. CTRL C closes all open files; these must be opened before you can execute BREAK.SV. For more on the intricacies of interrupts, see Keyboard Interrupt Commands, Chapter 3.

NOTE: For obvious reasons, the CTRL A and CTRL C break characters must never be transmitted as input characters to a user program.

ERROR HANDLING

If you issue a command that contains or evokes an error, CLI prints an appropriate error message. A complete list of CLI errors is given at the end of this chapter.

When you enter a command that is legal for some arguments and illegal for others, CLI executes the correct portions of the command, and prints an error message for each illegal argument. For example:

```
R
CRAND A B C)            ←create an entry in
                          SYS.DR for files A,
R                         B, and C.

XFER $PTR A)            ←transfer a file from
                          the paper tape reader
                          PTR to file A
LOAD PTR, STRIKE ANY KEY
R

CRAND A E)
FILE ALREADY EXISTS: A←illegal argument A;
                        legal argument E
R

LIST E)
E.          0           ←E was created
R
```

When the CLI cannot respond to a user command, an error message does not necessarily result. For example, if you command CLI to LIST a nonexistent file, CLI will respond with the prompt message (R) only.

In general, error messages are quite explicit, and give you enough information to correct an error easily.

```
R
CREATE A #A *A)
ILLEGAL FILE NAME: #A
ILLEGAL FILE NAME: *A
R

XFER FOO $PTR)
FILE WRITE PROTECTED: $PTR
R

CREATE TEST; CHATR TEST R)
R

TYPE TEST)
FILE READ PROTECTED: TEST
R

CHATR $LPT 0)
ATTRIBUTE PROTECTED: $LPT
R

MKSAVE $PTR TST.SV)
LOAD $PTR, STRIKE ANY KEY.
FILE SPACE EXHAUSTED
R

XFER NONFILE NEWFILE)
FILE DOES NOT EXIST: NONFILE
R
```

CLI COMMANDS

This section defines and describes each CLI command in alphabetical order.

The following conventions are used to define individual CLI command formats:

All upper case letters represent valid command line elements;

e.g., CREATE A.

Items in a command line printed in lower-case indicate either explanatory information or file names which you must supply. Examples of both: CDIR create a directory; DEB filename.

Elements enclosed in modified brackets, { }, are optional; e.g.,

ENDLOG {password}

Stacked items enclosed in braces, { }, indicate that you must choose one item, e.g.,

LINK { filename/2 / link filename }

The ellipsis (...) indicates that you can repeat the preceding argument as desired; e.g.,

PRINT filename$_1$ {filename$_2$} ...

In the command line formats, the comma (,), plain brackets ([ ]), and right slash (/), up arrow (↑), asterisk (*), dash (-), and parenthesis (( )) are not conventions but are significant and necessary parts of the command structure.

In the examples, where clarification is important, underlined entries represent your input; other entries represent system output; e.g.,

CREATE A)
R

Generally, when a CLI command takes a filename argument, that argument can include directory specifiers (e.g., DP0:ACCTSDUE). A few commands require filenames to be in the current directory; these are noted.

Note that template expansion (characters * and -) are forbidden except as noted, for the following commands: BUILD, DELETE, DUMP, LIST, LOAD, MOVE and UNLINK.

APPEND Combine two or more files

Format:

APPEND newfilename oldfilename$_1$ ... [oldfilename$_n$]

Create a new file, which consists of one or more old files, as entered. The old files are not changed by the command.

Switches: None

Example(s):

APPEND COM.SR COM1 COM2 COM3)

creates file COM.SR, containing the contents of files COM1, COM2, and COM3 in that order.

APPEND DP1:ALL.LB A.LB B.LB DP0:C.LB)

creates file ALL.LB on diskette unit 1, containing the contents of files A.LB and B.LB from the current directory, and C.LB from diskette unit 0.

ASM Assemble a source file

Format: ASM filename$_1$...[filename$_n$]

Assemble one or more source files using the extended relocatable assembler. As output, you can specify a relocatable binary file, a listing file, or both. The relocatable binary will be named filename.RB; the listing file will be filename.LS, unless the switches specify otherwise.

Global Switches:

- /L Produce a listing file (named filename.LS).
- /N Produce no relocatable binary file.
- /U Append user symbols to the relocatable binary output.
- /E Suppress error messages.
- /S Skip pass 2. A BREAK is signaled after pass 1, permitting you to save a version of the assembler that contains your own semipermanent symbols.
- /T Do not produce a symbol table list as part of the listing. (Used when a listing is requested, which produces a symbol table by default.)
- /X Produce a cross reference of symbol table. Symbol table output will contain page number - line number pairs for the symbol definition and every reference to the symbol within the assembly.

Local Switches:

- /B Direct relocatable binary output to this filename instead of filename.RB (overrides global /N).
- /E Direct error messages to this filename.
- /L Direct listing output to this filename instead of filename.RB (overrides global /L).
- /S Skip this file on pass 2 of assembly. (Use this switch only if the file does not assemble any storage words).
- /N Do not list this file. (Used, when a listing is requested, to list a selected number of files for assembly.)

Extensions:

On input, search for filename.SR. If not found, and the filename did not have an extension, search for filename.

On output, produce filename.RB for relocatable binary and filename.LS for listing (global L switch), unless the /S, /L, or /B local switch were given.

Example(s):

```
ASM Z )
```

assembles source file Z, producing a relocatable binary file called Z.RB.

```
ASM/N/L A )
```

assembles file A, producing as output a listing file called A.LS.

```
ASM (A,B,C,D,DP1:E,$PTR SARAH.RB/B) $LPT/L
```

causes separate assemblies of files A, B, C, and D from the default directory, of file E on diskette unit 1, and a paper tape mounted on the high speed reader. (The source program mounted on the reader would need to be reloaded since the assembler requires two passes.) Relocatable binary files for each source file are placed in disk files A.RB, B.RB, C.RB and D.RB. The file in the paper tape reader is placed in DP1, under the name SARAH.RB. Separate assembly listings are produced on the line printer.

CAUTION: issuing the following command would cause the loss of the first relocatable binary disk image:

```
ASM ($PTR, $PTR) $LPT/L )
```

Even though two distinct source files are read by the high-speed reader, each relocatable binary produced is labeled $PTR.RB. Thus the first relocatable binary would be overwritten on disk by the second one.

BASIC Invoke the BASIC interpreter

Format: BASIC

To invoke the BASIC interpreter to execute a BASIC program. A BASIC save file, configured via BASIC System Generation (BSG), must exist before this command can be executed. BASIC configuration and load procedures are described in Chapter 2 of the Extended BASIC System Manager's Guide.

Switches: None

Example(s):

```
R
BASIC)
*
*        } BASIC commands
BYE)     ←(Return to CLI)
R
```

BOOT Perform a disk bootstrap

Format:

BOOT { diskette specifier
{ [diskette specifier:] system-save-filename }

Release the current system and perform a disk bootstrap. The disk bootstrap program, BOOT, must reside on the disk device named in diskette or on the disk unit containing the save file which will be executed. This diskette must be initialized before the BOOT command is issued. No save file can be bootstrapped in a directory.

Use of a device specifier alone (the first format) activates BOOT on that device; thus a file name response will be required when BOOT gains control. The device must be one of the following: DP0...DP7.

If you use a colon in the second format, do not use space separators between the colon and argument. If you omit a system-save-filename, the default operating system, SYS.SV/SYS.OL, will be used. If save filename is specified, it must have no extension other than ".SV". The save file name may be the name of a DOS system, the name of a save file, or a resolution file. If the latter, all intervening link entries must be initialized, and the resolution chain cannot include a directory. If the operating system being run is a link, the diskette containing the resolution entry becomes the master directory; moreover, that system's overlay file must also be linked.

If all front panel switches are up when you give the BOOT command, any save file name given will be ignored and the default system, SYS.SV/SYS.OL, will be bootstrapped. Additionally, the system will attempt to chain to a save file named RESTART.SV (as in power fail-auto restart applications). If RESTART.SV is not found DOS will display error message FILE NOT FOUND: RESTART.SV, and invoke CLI. If you don't want this feature, ensure that the front panel switches are not all in the up position.

Note that there is no switch available in the BOOT command. Thus if you wish to bootstrap an absolute program (which is not in save file format), you must activate BOOT by using the first format, then use the /A switch in response to the file name query by BOOT.

BOOT (Continued)

Switches: None

Example(s):

BOOT DP0)
FILENAME? MYSYS)

This command loads BOOT from diskette unit 0. Upon being loaded, BOOT outputs the query: FILENAME? You then enter the system save filename (or RETURN for the default system, SYS.SV/SYS.OL).

BOOT DP1:SYS32K)

This command loads DOS system SYS32K found on diskette unit DP1. DP1 must have been initialized before the BOOT command. When SYS32K.SV is loaded, time and date information will be requested, after which the operating system's CLI will be active.

BOOT DP0:RTOS)

This command loads the RTOS save file found on diskette DP0. When the execution of RTOS.SV is completed, an RDOS system must be bootstrapped via the front panel switches unless the RTOS program itself activates BOOT.

BOOT DP0)
FILENAME ? DP0:FOO.SV/A)

This command and response to the BOOT filename query executes absolute program FOO.SV. FOO.SV resides on DP0 and does not conform to the conventions required by BOOT (thus the use of the /A switch).

BPUNCH Punch a binary file

Format: BPUNCH filename$_1$ [filename$_2$...]

To punch a given file or files in binary on the high speed punch. The command is the equivalent of a series of XFER commands:

XFER filename$_1$ $PTP;...;XFER filename$_n$ $PTP)

The files may come from any device.

Switches: None

Example(s)

BPUNCH FEE.SR FI.SR FO.RB FUM.RB)

This command punches source files FEE and FI, and relocatable binary files FO and FUM on the high speed punch.

BPUNCH $PTR)

This command punches a paper tape duplicate of the tape in the high speed reader.

BUILD Build a file from filenames

Format: BUILD outputfilename {filename$_1$...}

Build an output file from filenames in the current directory. If you omit an extension from a filename, all matching filenames with no extensions will be incorporated into output-filename.

Global Switches:

/A Include all filenames which have permanent and nonpermanent attributes.

/K Do not include link entries.

/N Do not include extensions to filenames in outputfilename.

Local Switches:

/A Include only files created this day or after, where the preceding argument has the form mm-dd-yy (mm and dd can be one or two digits).

/B Include only files created before this day. The preceding argument has the same form as the /A switch.

/N Build a file to contain any files that do not match this name.

Template Characters: Permitted

Example(s):

BUILD ABC -.SR TEST -.)

Creates file ABC, and writes two categories of files into it: those whose names have the .SR extension, and those whose names begin with TEST. See Template Expansion, in this chapter.

BUILD MYFILE -.RB ABC.RB/N)

Creates MYFILE, and inserts all RBs except ABC into it.

CCONT Create a contiguous file with all data words zeroed.

Format:

CCONT filename$_1$ blockct$_1$ {filename$_n$ blockct$_n$}...

Create one or more contiguous files with all data words zeroed. Each file has only the C attribute and the length in blocks you specify in blkct. A disk block is 256 words long.

Switches: None

Examples:

CCONT ALPHA 20)

Creates the contiguous file, ALPHA, in the default directory, with a length of 20 disk blocks.

CCONT TEST 100 DP1:TEST1 51)

Creates two files, TEST in the default directory and TEST 1 on diskette DP1.

CDIR Create a directory

Format: CDIR directoryname

Create a directory with a .DR extension.

Switches: None

Examples:

```
DIR DP1 )
CDIR BETH )
```

The DIR command makes DP1 the current and default directory device. The second command creates directory BETH.DR on DP1. CDIR DP1:BETH is an equivalent command.

CHATR Change a file's attributes

Format:

CHATR filename$_1$ attrib$_1$ {...filename$_n$ attrib$_n$}

Use CHATR to change, add, or delete access file attributes of a given file. All current access attributes of the file are replaced by those given in the attributes argument. See Chapter 3 for the .CHATR system call.

Switches: None

Attributes:

- N Not a resolution entry; forbids linking.
- P Permanent file. Filename cannot be deleted or renamed while it has this attribute.
- R Read-protected file. Prevents filename from being read.
- S Save file. Having been set, this attribute cannot be removed.
- W Write-protected file. Prevents filename from being altered.
- 0 Removes all removable attributes.
- * Retain existing file attributes (use the asterisk to add attributes).
- ? User-defined attribute.
- & User-defined attribute.

When you want to assign several attributes to a file name, enter them as a single argument. Attributes may be listed in any order in the argument.

The following attributes and characteristics cannot be set or changed by CHATR.

- A Attribute-protected file. (CHATR cannot change any attribute of such a file.)
- C Contiguously organized file.
- D Randomly organized file.
- Y Directory (the file is a directory).

Example(s):

```
CHATR A 0 B R )
```

deletes all attributes of A and causes B to be read-protected.

```
CHATR ALPHA.SV *W )
```

Adds the write-protected attribute to ALPHA.SV's other attributes.

CHLAT Change link access attributes

Format:

CHLAT filename$_1$ attrib$_1$... {filename$_n$ attrib$_n$}

Use CHLAT to change, add, or delete file link access attributes. All current link access attributes of the file are replaced by those given in the attributes argument. See Chapter 2 for a description of linking, and Chapter 3 for system call .CHLAT.

Switches: None

Attributes:

- N Forbids linking to filename.
- P Permanent file. filename cannot be deleted or renamed while it has this attribute.
- R Read-protected file. Prevents filename from being read.
- S Save file. Having been set, this attribute cannot be removed.
- W Write-protected file. Prevents filename from being altered.
- O Removes all attributes of filename.
- * Retain existing file attributes (use the asterisk to add attributes).
- ? User-defined attribute.
- & User-defined attribute.

When you want to assign several attributes to a file name, enter them as a single argument. Attributes may be listed in any order in the argument.

Example(s):

CHLAT EPSILON.SV W)

EPSILON.SV's attributes permit linking and/or reading, but the write-protect attribute prevents it from being altered.

CLEAR Set file use count to zero

Format: CLEAR {filename$_1$...}

Clear the file use count in one or more SYS.DR entries. Use this command after a system failed while files were open. Such files cannot be deleted or renamed until their use counts are set to zero. You can also use CLEAR if your system was shut down before the master device was released. You can correct this oversight by CLEARing the device.

Global Switches:

- /A All files in the current directory (except CLI.OL, CLI.ER, SYS.OL, LOG.CM) have their use counts set to zero.
- /V Verify files cleared on $TTO.

Local Switches: None

Example(s):

CLEAR/A)

If your system failed the command above would help you reinitialize it by setting all file use counts in the current directory to zero.

CLEAR DP0:TEST:TEMP)

sets file TEMP's use count to zero: TEMP is in directory TEST, on DP0.

Note:

While attempting to dump file CLI.OL, you may receive the error message FILE IN USE: CLI.OL. Proceed to clear CLI.OL and CLI.ER by the command CLEAR CLI.<OL,ER>. Now, rebootstrap the system.

CLG Compile, load and execute a FORTRAN IV program

Format: CLG filename$_1$ [filename$_2$][filename]...]

The CLG command allows you to execute one of several different operations on each filename in the command line. CLG can compile files using FORTRAN IV, assemble them with the Extended Assembler, load files as one program with the Relocatable Loader, and execute the final save file. The FORTRAN compilation and assembly steps are optional. In the argument list, you can include FORTRAN IV source files (filename.FR) files in assembly language (filename.SR), and assembled binary files (filename.RB). Output includes one or more intermediate source files, one or more relocatable binary files, and the save file. To load FORTRAN files the loader requires the FORTRAN library, FORT, LB, on disk (this library is produced by merging the four original FORTRAN libraries supplied by Data General; if they have not been merged, you can merge them with the LFE M command). Data General devotes two manuals to FORTRAN IV: THE FORTRAN IV User's Manual and the FORTRAN IV RUN TIME LIBRARY User's Manual.

CLG expands the FORT command, which produces relocatable binary files but cannot produce a save file and execute it. In addition, CLG can treat source files individually, where some require loading, others assembly and loading, and still others compilation as well.

Unless you restrict output with global switches all compilations, assemblies, and the load map will go to the listing file you specify with the /L switch.

In addition to the local switches below, each filename.RB may have switches appropriate to loading (RLDR command). You can create overlays with CLG as in the RLDR command line by enclosing them in brackets.

Global Switches:

/B List only source program.

/E Suppress compiler error messages. Assembler error messages are not suppressed.

/M Suppress loader map.

/T Multitask mode. This switch instructs CLG to load a multitask program (the multitask FORTRAN library, FMT.LB, must be available on disk).

Local Switches:

/A Assemble and load this file; do not compile.

/E Direct error messages to this file name.

/L Direct listing output to this file name.

/O Load this file only; do not compile or assemble. You can use the local switches in ASM and RLDR, as appropriate.

Extensions:

On input, search for filename.FR; if not found, search for filename. If /A is specified, search for filename.SR. If not found, search for filename. If /O is specified, search for filename.RB; if not found, search for filename.

On output, produce temporary assembler source files, filename$_1$.SR(i=1...n). Produce relocatable binary files, filename$_1$.RB(i=1...n). Produce save file filename$_1$.SV (unless local /S was included in the command line).

Examples:

CLG A B C)

Compile A.FR (or A), producing A.SR; assemble A.SR into A.RB and delete A.SR. Do the same with B.SR (or B) and C.SR (or C). Load A.RB, B.RB, C.RB and FORT.LB to produce A.SV. Execute A.SV.

CLG/B MAIN $LPT/L)

Compile MAIN.FR (or MAIN), list it on the line-printer, and produce MAIN.SR. Assemble MAIN.SR into MAIN.RB and delete MAIN.SR. Load MAIN.RB and FORT.LB producing MAIN.SV. Execute MAIN.SV.

CLG/T MAIN RB/O SR/A $LPT/L)

Program MAIN is a file in FORTRAN IV language, RB is in relocatable binary, and SR is in assembly language. The /T switch specifies multitask mode; for multitask compilation FMT.LB is called from disk.

The command instructs the system to: compile MAIN.FR (or MAIN), producing MAIN.SR; assemble MAIN.SR producing MAIN.RB and delete MAIN.SR. Hold RB.RB (or RB) for the loading step. Assemble SR.SR (or SR) into SR.RB. Load all three binaries under the name MAIN.SV; execute MAIN.SV. List all compiler, assembler, and loader output to the line printer.

COPY Copy all Files on One Diskette to Another Diskette

Format: COPY sourcediskette destinationdiskette

Copy all files and directories from one diskette to another. Before you issue this command, sourcediskette must be initialized and destinationdiskette must NOT be initialized. All files in destinationdiskette will be destroyed, and the files from sourcediskette will be moved to it, along with all file directory information. If destinationdiskette is not a Data General diskette, you must format it with the appropriate formatter program and run the diskette reliability test before copying to it.

Because the root portion of BOOT.SV (blocks 0 and 1) is not part of the file structure, it will not be copied. If you will want to bootstrap from a destinationdiskette which lacks a bootstrap, you must install BOOT on it.

After the copy is complete, destinationdiskette will contain the same files as sourcediskette; therefore both diskettes cannot be initialized at one time. DOS doesn't allow two directories with the same name to be initialized at one time; it will recognize only the first directory initialized, until you release the first and INIT (or DIR to) the second. Confusion can result from this, and you should therefore avoid having both an original and copy on the system simultaneously.

Switches: None.

Examples:

```
INIT DP0)
R
COPY DP0 DP1)
R
```

You can now remove the copy from slot DP1 and store it.

```
INIT DP2; INIT DP3)
R
COPY DP2 DP3)
DEVICE ALREADY INITIALIZED: DP3
R
RELEASE DP3)
R
COPY DP2 DP3)
R
```

DOS refused to copy to an initialized diskette.

Note: For COPY to work, the current DOS system must have at least two subdirectories, as determined at SYSGEN.

CRAND Create a randomly organized file

Format: CRAND filename$_1$ { filename$_2$... }

Create a randomly organized file, in the current directory. Each random file has the D attribute and a length of zero; it will grow as required during use.

Switches: None

Example(s):

CRAND RANDFILE)

Creates RANDFILE in the current directory.

CRAND ACCUFILE DP1:VELVET:GROWTHFILE)

Creates ACCUFILE in the current directory, and GROWTHFILE in directory VELVET on DP1.

CREATE Create a randomly organized file

Format: CREATE filename$_1$ {filename$_2$...}

In DOS, CREATE is functionally identical to CRAND.

Switches: None

Example(s):

CREATE ALPHA)

Create a filename, ALPHA, in the default directory.

CREATE TEST TEST1 DP1:TEST2)

Create three file names, TEST, and TEST1 in the current directory and TEST2 in default directory on DP1.

DEB Load a program into memory and go to the debugger

Format: DEBfilename

Debug a program about to be executed. A symbolic debugger must have been loaded as part of the program save file, as described under the RLDR command. The DEB command transfers control to the debugger in this save file.

While debugging, you can examine memory, set break points, run the program, etc. After making any necessary changes in the program, you can save the current core image of the program and return to CLI by issuing the $V Debug command. You can then save the core image under a file name, as described under the SAVE command.

Switches: None

Example(s):

```
DEB A )                    ←Debug A. SV
1004/ADD 0 2 ADD 1 2)      ←Change program.
$V
BREAK                      ←$V break issued.
SAVE A)                    ←Changed version (current
                           core image) saved.
A )                        ←New attempt to execute
```

DELETE Delete one or more files or directories

Format: DELETE filename$_1$ {...filename$_n$}

Delete the files in a directory having names given in the argument list. You may include a directory specifier if the directory has been initialized. If you try to delete a link, the link will persist but the resolution entry will be deleted (attributes permitting). Therefore, to remove a link entry, you must use the UNLINK command.

To delete a directory, RELEASE the directory, then type DELETE directoryname.DR.

Global Switches:

/C Confirm each deletion. Each filename is repeated, while the system waits for you to confirm the deletion by typing a carriage return. To prevent the deletion, press any key other than carriage return.

/L List deleted files on $LPT (overrides /V).

/V List names of deleted files on $TTO.

Local Switches:

/A Delete only files created this date or after, where the preceding argument has the form mm-dd-yy (mm and dd can be one or two digits).

/B Delete only files created before this date. Preceding argument has the form of local switch /A.

/N Do not delete any files that match this name.

Template Characters:

Permitted only when filename argument is in the current directory.

Example(s):

DELETE LIMIT.-)

deletes all files having the name LIMIT and any extension (including null), e.g., LIMIT.SR, LIMIT.RB, LIMIT.SV, LIMIT. .

DELETE/V -.LS)

deletes all files with the .LS extension, and lists their names:

```
DELETED A.LS
DELETED COM.LS
DELETED MAP.LS
```

DELETE A**B)

deletes files AXYB, AXWB, but not AB or AXB.

DELETE A-B)

deletes files AZYB, AXWB, AB and AXB.

DELETE/C A-B)

ask for confirmation of each file before deletion:

```
AZYB:)*      File AZYB is deleted
AXWB:#       File AXWB is not deleted.
```

When you confirm a deletion with a carriage return, the system echoes an asterisk (*). Any other character evokes no echo.

DIR Change the current default directory

Format:

DIR { directory / devicespecifier [:directory] }

Use DIR to change the current directory or default directory device. At bootstrap time, a master device is established as the current directory. The DIR command specifies another device or directory as the default device or directory. If necessary, this command will also initialize the device/directory.

You can use the first format only if directory is on the current diskette.

Switches: None

Example(s):

DIR ACCTSDUE)

Direct file name references to directory ACCTSDUE on the current diskette.

DIR DP1:DEF)

Direct all file name references to directory DEF on diskette 1.

DISK List the number of disk blocks used and remaining

Format: DISK

Return the decimal number of blocks used and the number of blocks left on the current diskette. BOOT requires 2 blocks and the system requires 4, therefore the number of blocks available is always 6 less than the total number on the disk. The sum of blocks reported by DISK is the largest integer multiple of 16 which fits within this figure.

Switches: None

Example(s):

DISK)

LEFT: 520 USED: 88

The response indicates that 520 out of a total of 608 blocks are still available for use.

DUMP Dump one or more files

Format:

DUMP outputfilename {filename ...} {old filename/S new filename}...

Dump files in a directory or diskette onto a given file or device (output filename). filename can be either a directory or a diskette specifier. The directory information for each file -- name, length, attributes, creation and last access time -- is written as a header to each dumped file. You can choose to enter each filename as

old filename/S new filename

where newfilename will be the name of the dumped file and old filename will be retained in the current directory. If you omit specific file names, all nonpermanent files are dumped. By dumping a directory, you dump its contents.

If while dumping all files (DUMP/A) from a diskette that contains more than one directory you abort the dump by typing CTRL A, you should explicitly direct the CLI to a specific directory before making any further file references. This should be done since it is impossible to determine what directory was being dumped (and thus was the current directory) at the abort.

Global Switches:

/A Dump all files, permanent and nonpermanent.

/K Do not dump links.

/L List the dumped files on $LPT (overrides /V).

/S Dump a file on segments of paper tape. The file is punched in segments of up to 20K bytes each. Each tape segment is headed by unique segment number, which enables the system to verify that tapes will be reLOADed in proper sequence.

/V Verify dump by listing names of dumped files on the console.

Local Switches:

/A Dump only files created this day or after. The preceding argument has the form mm-dd-yy (mm and dd can be one or two digits).

/B Dump only files created before this day. The preceding argument has the form of local switch /A.

/N Don't dump files that match this name.

/S Assign this new name to this file in the dump, but retain its old name in the current directory.

Template Characters:

Permitted only when filename argument is in the current directory.

Example(s):

DUMP/A MT0:0 5-20-76/A

dumps all permanent and nonpermanent disk files created on May 20, 1976 or after onto file 0 of magnetic tape unit number 0. This tape file can then be saved as backup for the diskette.

DUMP/L DUMPFI -.SV)

EDIT.SV
ASM.SV
RLDR.SV
} list of files on $LPT

dumps all nonpermanent files on diskette with the extension .SV to disk file DUMPFI, and lists the dumped files on the line printer.

DUMP/A MT0:0 APRIL/S BILLSAPRIL)

File APRIL will be named BILLSAPRIL in the dump (but its name remains APRIL in its directory on this diskette) even if APRIL is permanent. BILLSAPRIL will be dumped on file zero of MT0.

DUMP SOURCE -.SR 7-22-76/A)

Dump all files with the .SR extension created on or after July 22, 1976 to file SOURCE.

EDIT Invoke the Text Editor

Format: EDIT {filename}

The text editor can help you build a new source file or edit existing source files. For more information, see the Text Editor User's Manual. If you include the optional filename argument the editor will automatically execute the command UY filename ESC ESC upon being loaded.

Switches: None

Example(s):

```
EDIT )
*          ← Program is ready to accept commands.
.          ← You issue editing commands.
.
.
*H$$       ← You terminate the editor and return to
               CLI by pressing the H key followed by
               two ESC keys.
R
```

ENDLOG Close the file opened by LOG

Format: ENDLOG {password}

Closes the log file which you opened by a previous LOG command. You must close this file before you can TYPE or PRINT it. If the previous LOG command included a password argument, you must use the password with the ENDLOG command.

This command, ENDLOG password, appears in the log file.

Switches: None

Example(s):

```
ENDLOG GSTONE )
```

The password GSTONE is used since it was specified when the log file was last opened.

Note that you must type the full name of the log file to PRINT or DELETE it; this name is LOG.CM.

FILCOM Compare two files

Format:
FILCOM filename$_1$ filename$_2$ {listing file/L}

Compares two files, word by word, and prints dissimilar word pairs. The displacement and content of the dissimilar words are printed in octal on your console (unless you use the /L switch). File organizations of the two files may differ; i.e., you can compare a random and a contiguous file.

Local Switches:

/L Listing file.

Example(s):

FILCOM YIN YANG $LPT/L)

causes a word-by-word comparison of files YIN and YANG. Any dissimilar word pairs will be printed in octal on the line printer, along with their respective word displacements in the files:

```
025/  044516  042530
141/  000014  020044
142/  000000  046120
143/  000000  052057
144/  000000  046015
145/  000000  000014
```

If YANG were a null file, the entire contents of file YIN would be printed. Dashes would be printed for file YANG.

FORT Compile a FORTRAN IV file

Format: FORT inputfilename {outputfilename.}.

Compile a FORTRAN IV source file. Output may be a relocatable binary file, an intermediate source file, a listing file, or combination of all three. The command name, FORT, cannot be changed.

On input, the command searches for inputfilename.FR; if not found, it searches for inputfilename.

By default, the FORT command produces an intermediate source file, inputfilename.SR (output of assembly). After a successful assembly, the intermediate source file is deleted. No listing is produced by the default command. If you specify a listing it will be named inputfile.LS.

Global Switches:

/A Suppress assembly (intermediate source file is deleted unless you use global/S).

/B List compiler source program input only.

/E Suppress error messages from compiler. (Assembler error messages are not suppressed.)

/F Equivalence FORTRAN variable names and statement numbers to symbols acceptable to the assembler. (Include /U to inform the Debugger of the /F names.)

/L Produce list file (inputfilename.LS).

/N Do not produce relocatable binary file.

/P Compile no more than 72 columns per line (as in punched card images).

/S Save the intermediate source output file.

/U Output user symbols during the assembly phase.

/X Compile statements with X in column 1.

FORT (Continued)

Local Switches:

/B Direct relocatable binary output to given file name (overrides global /N).

/E Direct error messages to given file name. (overrides global /E).

/L Direct listing output to given file name. (overrides default name specified by global /L).

/S Direct intermediate source output to given file name.

Examples:

```
FORT/L MAIN )
```

produce relocatable binary file MAIN.RB with both a compiler and an assembler listing to file MAIN.LS.

```
FORT/F/U DP1:TABLE $LPT/L )
```

compile FORTRAN IV file TABLE.FR (or TABLE) on primary partition DP1 and produce TABLE.RB. Write compiler and assembler listings to the line printer, and output user symbols during assembly.

FPRINT Print a disk file in the specified format

Format:

FPRINT filename {loc/F} {loc/T} {filename/L}

Prints readable disk file in either bytes, decimal, hexadecimal, or octal, with printable ASCII characters on the right side. Any nonprinting characters are reported as periods (.). The location counter is always in octal, and the default listing device is the console. You can specify a first and terminal location for printing.

Global Switches:

/B Byte printout.

/D Decimal printout.

/H Hexadecimal printout.

/L Line printer.

/O Octal printout (default).

/Z File starts at zero.

Local Switches:

/F First location to be printed.

/L Overrides global /L, directs output to file name that precedes it.

/T Last location to be printed.

Example(s):

FPRINT/L TE1)

lists file TE1 on the line printer. The mode is octal by default.

FPRINT MYFILE/B/L 2000/F 3500/T)

Print MYFILE in byte format on the lineprinter, from location 2000 to 3500.

GDIR Return the current directory/device name

Format: GDIR

Switches: None

Example(s):

```
GDIR)
MANHATTAN
```

MANHATTAN is the current directory.

GSYS Return the name of the current operating system

Format: GSYS

Switches: None

Example(s):

```
GSYS)
SYS
```

The current operating system is named SYS. Note that the save file extension, .SV, is not typed.

GTOD Return the time and date

Format: GTOD

Switches: None

Example(s):

GTOD)
02/17/76 21:24:20

The message indicates that the time is 20 seconds after 9:24 p.m., and the date is February 17, 1976.

INIT Initialize a directory or device

Format:
INIT { mag tape specifier | directory | diskette specifier [:directory] }

By initializing a device or directory, you introduce it to the operating system; before the introduction, all of its files are effectively closed. At any moment, many directories on a device can be initialized, while only one remains the current or default directory.

After you INIT a directory or device, you can access all files within it for dumping, deleting, assembly, etc.,; you can use the colon specifier (e.g., DP0:AB) at will. All files on the initialized tape unit, directory or diskette are available until you release (RELEASE command) the device or directory.

Of the formats above, you can use the second only if directory is on the current diskette. When you INIT a diskette using the third format, and omit directory, all directories are initialized (the system does this by reading the diskette's SYS.DR directory).

Global Switches:

/F Full initialization of a tape device rewinds the tape and writes two EOF's at its beginning, effectively erasing the tape. On a diskette, INIT/F destroys all existing files and builds a new file directory and free storage map.

Local Switches: None

Example(s):

INIT DP1)

Initialize diskette number 1.

INIT/F MT1)

Mag tape initialization rewinds the tape to BOT. Full (/F switch) initialization of MT1 rewinds the tape on drive MT1, erases it and writes 2 EOF's at the beginning. (DOS begins writing files on the tape at the double end-of-file.)

INIT DP1:ABC)

All files in directory ABC will now be accessible, provided you issue explicit DIR commands, or direct file accesses with explicit global/directory specifiers, including colon separators.

LFE Update library files

Format:

LFE A inputmaster {arg$_1$ ...} {listing device/L}

LFE A/M inputmaster$_1$ {...inputmaster$_n$}
{listing-device/L}

LFE D inputmaster {outputmaster/O} arg$_1$ {...arg$_n$}

LFE I inputmaster {outputmaster/O} file$_1$ {...file$_n$}

LFE M {outputmaster/O} inputmaster$_1$ {...inputmaster$_n$}

LFE N {outputmaster/O} file$_1$ {...file$_n$}

LFE R inputmaster {outputmaster/O} arg$_1$ file$_1$
{...arg$_n$ file$_n$}

LFE T inputmaster {listing device/L} {inputmaster$_2$...}

LFE X inputmaster$_1$ arg$_1$ {...arg$_n$}

The LFE command calls the Library File Editor. This utility allows you to edit and analyze library files, which are sets of relocatable binary files having special starting and ending blocks and which are usually designated by the extension .LB. See the Library File Editor manual for more information.

A, D, I, M, N, R, T and X are keys designating LFE functions. inputmaster and outputmaster represent library files; args represent logical records on the library files; files are relocatable binary files which update outputmaster.

Action taken by the LFE depends upon the function given in the command:

A Analyze global declarations of one inputmaster, or a series of inputmasters, or of logical records specified from one inputmaster. Output is a listing with symbols, symbol type, and flags; no new output library file is created. Default listing device is $LPT.

Analyze Switches: /B - preceding filename is an .RB, not a library (.LB).

/F Form feed. If you use this switch with local /L (below), the analysis of each logical record will be printed on a separate page. Example: LFE A MATH.LB SIN COS TAN $LPT/L/F.

D Delete logical records, specified by args from inputmaster, producing outputmaster. Default output file is D.L1.

I Insert relocatable binary files, combining them with logical records of inputmaster. If you specify no switches, they will be inserted at the beginning. Default output file is I.L1.

Insert Switches:

/A Insert after. When you append this switch to a logical record in an I function command line, the relocatable binary following the switch will be inserted after the logical record. When you specify neither a /A or /B switch, inserts are made at the beginning of the new library file.

/B Insert before. When you append this switch to a logical record in an I function command line, the relocatable binary following the switch will be inserted before the logical record.

M Merge library files (inputmasters) into a single library file named outputmaster. Default output file is M.L1.

N Create new library file, outputmaster, from one or more relocatable binary files given by files. Output file is N.L1 by default.

R Replace logical records in inputmaster by relocatable binary files, producing outputmaster. You enter arguments in pairs, with the first being the logical record and the second the relocatable binary file that replaces the logical record. Default output file is R.L1.

T Output the titles of logical records on inputmaster to the listing device ($LPT by default).

X Extract from library file inputmaster one or more relocatable binary files given by args. Output is one or more relocatable binary files named args. RB.

Key-switch:

/M Multiple-input library files. The switch modifies the A function and causes all library file names following, except the listing file, to be analyzed as one library.

LFE (Continued)

Local Switches:

/E Direct an error listing to the given filename.

/L Listing file. In the A or T function command line, you must append this switch to the name of the listing file. If you specify no listing file, the $LPT is used by default.

/O Output library file. If you specify an output library file, you must append this switch to its name. The name overrides default names in D, L, M, N, and R commands.

Extensions:

If you specify no extensions for inputmaster or file in the command, LFE searches for inputmaster.LB or file.RB respectively. If not found, LFE searches for inputmaster or file.

Example(s):

```
LFE N $PTP/O A.RB C.RB )
```

Create a library file, output to the punch from two disk files, A.RB and C.RB.

```
LFE A/M ZUT1.LB $PTR ZUT2.RB/B $LPT/L )
```

Analyze as one library ZUT1.LB, the library in the $PTR, and ZUT2.RB. List analysis on the line printer.

```
LFE I MAL.LB MAL1.LB/O R/A HB.RB )
```

In library file MAL.RB, insert relocatable binary HB.RB after relocatable binary R. Name the new output library file MAL1.LB.

LINK Create a link to a file in another directory

Format:

LINK { resfilename/2 | linkentryname [directory specifier:] resfilename }

Use this command to create a link entry to another link or to a resolution file. If your link entry will have the same name as the resolution file (resfilename), and your entry is in the current or default directory, you can create your link with the first command format (the local /2 switch specifies the same filename). Linkentryname will always be created in the current directory unless you specify another directory. When you use the first format, DOS links linkentryname to resfilename on the current diskette.

If the resolution file is in another directory, or if your link name will differ from the resolution file's, use the second command format. A link entry name which differs from the resolution filename is called an alias. The resolution file resfilename may exist either on the current diskette or, if its directory specifier precedes the filename, on some other diskette/directory.

You cannot link to a file whose attributes forbid linking (see CHLAT). Chapter 2 offers more information on linking.

Switches: None (except local switch /2)

Example(s):

```
LINK  ASM.SV/2 )
```

This command creates a link entry named ASM.SV in the current directory to a file named ASM.SV on the current diskette. That is, if the current directory is found on DP1, the resolution filename ASM.SV must be on DP1.

Note that the creation of a link does not require an existing resolution file; you can link to nonexistent files. You will discover an unresolved link only when you try to reference the resolution file, at which time the error message "FILE DOES NOT EXIST" will be given.

LINK (Continued)

LINK A.SV O.SV)

This command creates an alias link entry named A.SV in the current directory to save a file named O.SV on the current diskette.

LINK ASM.SV DP2:OASM.SV)

This command creates a link entry named ASM.SV in the current directory to an alias file named OASM.SV on diskette DP2.

LINK ASM.SV SAM:ASM.SV)

This command creates a link entry named ASM.SV, in the current directory, to a file of the same name in directory SAM.

LIST List file directory information

Format:

LIST { {filename1...} / {device specifier:} {directory:} filename }

Use this command to list information from the current directory or other directories about one or more files or link entries. If you omit the filename argument, all nonpermanent files and link entries in the current directory are listed.

For each file, listed information includes the file name and one or more of the following: file size in bytes, file access attributes, link access attributes, file creation date and time, date last opened, file starting address (UFTAD of UFD), and use count. The link access attributes -- if any -- are, preceded by a right slash. The file starting address is enclosed within brackets, and the file use count is terminated with a period to indicate that it is a decimal figure.

For link entries, the list includes the link name, the resolution entry name, and the directory specifier (if any) given when the link was created. A commercial at sign (@) is printed when the resolution entry was defined to exist on the current diskette.

The following is a list of file attributes (or link access attributes) and their meanings:

Printed Code	Meaning
P	Permanent file: cannot be deleted or renamed.
S	Save file (core image).
W	Write-protected file: cannot be written.
R	Read-protected file: cannot be read.
A	Attribute-protected file, whose attributes cannot be changed. The A attribute cannot be removed.
N	No resolution entry allowed: linking is forbidden.
?	First user-definable attribute (bit 9).
&	Second user-definable attribute (bit 10).

The following is a list of file characteristics and their meanings:

Printed Code	Meaning
D	Random file organization.
C	Contiguous file organization.
L	This file is a link entry
Y	This file is a directory.

LIST (Continued)

Global Switches:

/A List all files within the current directory, both permanent and nonpermanent files, giving file name, byte count, file attributes, link access attributes, and file characteristics.

/B Brief listing; lists only file names.

/C List the creation time (year/month/day hour: minute).

/E List every category of file information (overrides /B, /C, /F, /O, and /U switches).

/F List the first address, i.e., the logical address of the first block in the file; list 0 if unassigned. The address will be enclosed within brackets.

/L Output listing on line printer.

/K Do not list links.

/N List links only.

/O List date file last opened (month/day/year).

/S Sort the output list alphabetically.

/U List file use count (in decimal terminated with a period).

Local Switches:

/A List files created this date or after. The preceding argument has the form mm-dd-yy, where mm and dd can be one or two digits.

/B List files created before this date. The preceding argument has the same form as local /A switch.

/N Do not list files that match this name.

Template Characters:

Permitted only when the filename argument is in the current directory.

Example(s):

LIST/E/A)

This command lists on the console all information for every file and link entry in the current directory. A typical line of information would look like this:

FLI.SV 8160 SD 03/23/76 13:56 03/23/76 [004164] 0

In this example, FLI.SV is the file name, consists of 8,160 bytes, is a randomly organized save file, was created on 1:56 p.m. of the 23rd day of March, 1976, was last accessed on that same date, has a starting logical block address of 4164, and has a file use count of zero.

Typical lines describing link entries would look like the following:

ABC.SV DP0:DEF.SV

In this example, the link entry name is ABC.SV; the resolution file was defined to exist with alias DEF.SV on diskette DP0.

EDIT.SV @:EDIT.SV

In this example, the link entry name is EDIT.SV and the resolution file was defined to have the same name and to reside on the current diskette.

LIST/K/S -.SV 5-2-76/A)

This command lists all save files (.SV extension) created after May 1, 1976. It does not list links; output goes to the console.

LOAD Reload dumped files

Format: LOAD inputfilename {filename$_1$...}

Load a previously dumped file from a given device or directory into the current directory. If you specify no file names or switches, all nonpermanent files in the input file are loaded. With global switches, you can select filenames for LOADing, or you can choose simply to list on $TTO the filenames in the input file.

The LOAD command can load only those files which were previously DUMPed. Files to be loaded must bear different names from files in the current directory (unless you specify the /N or /R switches). Neither DUMPing nor LOADing change a file's directory, access or link characteristics.

If files were dumped in segments using the DUMP/S command, you must follow the DUMP sequence when you LOAD them. Failure to follow the same sequence will evoke a CLI runtime error message.

Global Switches:

/A Load all files, including permanent files.

/B Output a brief listing of loaded files.

/E Suppress nonfatal error messages.

/K Do not load link entries.

/L List loaded file names on the line printer. (Overrides /V switch and listing by /N.)

/N Do not load files; output the file names to the console. If global /R is used with /N, only the most recent version of each will be listed.

/O Delete current file if it exists and replace with file being loaded that has the specified name.

/R Load most recent version of file. When a file to be loaded has the same name as a file in the current directory, the system checks both files' creation dates. If the existing file is older, it is deleted and replaced by the file awaiting loading. If the existing file is not older, it is retained and the dumped file is not loaded.

/V Verify the load by listing filenames loaded on the console. Filenames in a directory are listed before the directory name, and they are indented 2 spaces; directory names are preceded by "**".

Local Switches:

/A Load only files created this day or after, where the preceding argument has the form mm-dd-yy (mm and dd can be one or two digits).

/B Load only files created before this day. The preceding argument has the same form as local /A.

N Don't load files that match this name.

Template Characters:

Permitted only when the filename argument is in the current directory.

Example(s):

```
LOAD MT0:1
```

loads onto disk all previously dumped nonpermanent files on file 1 of the tap on MT0. File name, length, and attributes are entered in the file directory.

```
LOAD/V $PTR -.SV)
LOAD $PTR, STRIKE ANY KEY.
EDIT.SV
ASM.SV
```

loads all files with the extension. SV, and lists files loaded.

```
LOAD/L $PTR -.SV 3-15-76/A TEST -.SV/N )
```

loads from the $PTR all files with the .SV extension (except files whose names begin with the characters TEST and files created before March 15, 1976), and lists them on the line printer.

LOG Open the log file

Format: LOG {password} {directory/O}

The log file records all CLI dialog that appears on the console in a file named LOG.CM. LOG.CM is updated after each line appears on the console. Only one log file may exist at a time. You cannot examine, print, or delete the log file while it is open. You can close it with the ENDLOG command (RELEASing the master device or giving the BOOT command also closes it). Unless it is deleted, the log file will retain all information; when the system is rebootstrapped, it will continue recording dialog after you reopen it.

You can use a password to prevent the log file from being closed inadvertently. The password is an optional argument, of up to 10 alphanumeric characters. If you specify password, the same password must be used in the ENDLOG command to close LOG.CM.

The directory argument indicates a destination for the log file other than the current directory. You must initialize the directory before using its name.

Global Switch:

/H Place a heading at the beginning of the log file. This heading consists of the title

LOG FILE

plus the following information: current directory name, master directory name, current system name (within brackets), and current date and time.

Local Switches:

/O Output the log file to directory.

Example(s):

LOG/H GSTONE)

This command outputs all CLI dialogue, except for prompt messages, to file LOG.CM in the current directory. The password is GSTONE, and on output, the heading of this file will look like:

LOGFILEGATE[DP0:SYS] 6/17/75 3:0:0

GATE is the name of the current directory.

MAC Perform a macroassembly

Format: MAC sourcefilename$_1$ {...sourcefilename$_n$}

Use MAC to assemble one or more source files with the macroassembler. Output may be a relocatable binary file, a listing file, or both. See the Macroassembler User's Manual for more information.

Global Switches:

By default, output of an assembly is a relocatable binary file; there is no listing file.

/A Add semipermanent symbols to cross-reference.

/E Prevent error messages from appearing on the error file (unless there is no listing file). The console is the default error file.

/F Generate or suppress a form feed as necessary to produce an even number of assembly pages. By default, a form feed is always generated.

/K Keep MAC.ST at the end of assembly. By default, MAC.ST is deleted at that time.

/L Produce a listing file. Listings include a cross-reference of symbol table. File MACXR.SV must be available on disk.

/N Produce no relocatable binary file.

/O Override all listing suppression controls.

/S Skip pass 2 and save a version of the assembler's symbol table and macro definitions in file MAC.PS (deleting the old MAC.PS, if any).

/U Include user symbols in the relocatable binary output.

/Z For DGC personnel only: print the DGC proprietary license heading at the top of assembly and cross-reference listing pages. By default, the heading is not printed.

MAC (Continued)

Local Switches:

/B Direct relocatable binary output to this file (overrides global /N).

/E Direct error output to this file.

/L Direct listing output to this file. (overrides global /L).

/S Skip this file on pass 2 of assembly. (You should use this switch only if the file does not assemble any storage words. Macro definition files can be skipped on pass 2.)

Extensions:

If you do not enter an extension for filename, CLI searches for filename.SR. If not found, CLI searches for filename.

On output, CLI produces filename.RB for relocatable binary and filename.LS for listing (global L switch), where filename will be the name portion of the first source file you specified without a /S, /L, or /B local switch.

Example(s):

```
MAC/L Z )
```

assembles of source file Z, producing a listing called Z.LS and a relocatable binary file called Z.RB.

```
MAC LIB/S A B C $LPT/L )
```

Assembles LIB, A, B, and C. File LIB contains macro definitions and thus is skipped during the second pass. Files A, B, and C become relocatable binary A.RB. A listing and cross-referenced symbol table are produced on the line printer.

MDIR Display the master directory name

Format: MDIR

Print the name of the current master directory on the console. This directory contains the system save and overlay files, and push space for program swaps.

Switches: None

Example(s):

```
MDIR)
DP0)
```

MEDIT Call the Multi-user Text Editor

Format: MEDIT terminals {clock-units}

MEDIT allows several users to perform editing at the same time. Each user is served as though he were the only user, except for a possible slight degradation in response time. See the Text Editor User's Manual for more information.

terminals Sets the maximum number (decimal) of text-editing terminals for MEDIT to support. If you specify 8, for example, terminals 0 - 7 will be supported.

clock-units represents the number of system clock-units; it is optional. After this number of units has passed, task rescheduling will be forced.

Switches: None

Example(s):

```
MEDIT 6 )
* ←Program is ready to accept commands,
. ←User issues editing commands.
.
```

MKABS Make an absolute binary file from a disk save file

Format:

MKABS savefilename absolute-binary-filename

Use MKABS to make an absolute binary file from a core image (save) file.

After you convert a disk file into an absolute binary, you can execute it without a disk.

Global Switches:

/S Starting address switch. The starting address of the save file as specified in USTSA of the file will be used as the address for an absolute binary start block. Default is a null start block which halts the binary loader when loading is completed.

/Z Begin save file at core location zero; default is 16_8. (See RLDR global switch Z.)

Local Switches:

/F First address, relative to save file location 0, from which the absolute binary file is to be created.

/S Starting address switch. The absolute binary start block will have the address specified by the octal number that precedes this switch.

/T last address, relative to save file location 0, to become a part of the absolute binary file.

Extensions:

CLI searches for save filename .SV; if not found, CLI searches for save filename.

Example(s):

```
MKABS FOO $PTP )
```

punches an absolute binary file on the paper tape punch from file FOO.SV or, if not found, from FOO.

```
MKABS FOO $PTP 1000/S )
```

punches an absolute binary file with a start block that specifies 1000 as the starting address.

MKSAVE Make a disk save file from an absolute binary file

Format:

MKSAVE absolute-binary-filename savefilename

Use MKSAVE to create a core image (save) file from an absolute binary file. The disk file will automatically be given the S attribute.

Global Switches:

/Z Start save file beginning at core location zero rather than 16_8. See RLDR global switch Z.

Extensions:

MKSAVE produces save-filename.SV as output, regardless of the extension you specify in the save file argument.

Example(s):

MKSAVE/Z $PTR DP1:A)

Makes a disk save called A.SV on diskette unit 1 from the absolute binary file loaded in the paper tape reader. A.SV begins at core location zero, and you can execute it by typing BOOT DP1) , and responding to the FILENAME? query with A.SV/A).

MOVE Move files from one directory to another

Format:

MOVE destination-directoryname {filename$_1$...}†)
{ old filename/S new filename}...

Move a given file or files in the current directory onto a given file or device. destination-directory can be a diskette or directory; filename cannot be a directory. The directory information for each file is preserved -- name, length, attributes, creation and last access time. If you specify no file names, all nonpermanent files in the current directory are moved. All filenames must be on the current diskette, and you cannot precede a name by a device specifier.

You can choose to enter each filename as oldfilename/S newfilename, where newfilename will be the name of the moved file, and oldfilename will be retained in the current directory.

Global Switches:

/A Move all files, including permanent files.

/D Delete original files once transfer is complete.

/K Do not move links.

/L List moved filenames on the line printer (overrides /V switch and console listing by /N).

/R Move most recent version of the file. When a file to be moved has the same name as a file in destination-directory, the system checks both files' creation dates. If the file in destination-directory is older, it is deleted and filename replaces it. Otherwise it is not moved, and the newer file is retained.

/V Verify the move by listing the names of the moved files on the console.

MOVE (Continued)

Local Switches:

/A Move any file created this day or after, where the preceding argument has the form mm-dd-yy (mm and dd may be one or two digits).

/B Move any file created before this date. The preceding argument has the same form as local switch /A.

/N Do not move files that match this name.

/S Assign this new name to a moved file (but retain its old name in the current directory).

Template Characters: Permitted.

Example(s):

MOVE/D/K MYDIR -.SR)

Moves all nonpermanent files in the current directory with .SR extension, (except link entries) into destination-directory MYDIR, and deletes the original files after the transfer.

MOVE DEBT JOE.SR/S JOEJR.SR -.- 3/1/76/A)

Moves JOE.SR under the name JOEJR.SR, and all files created after March 1, 1976, into directory DEBT.

NSPEED Invoke the Supereditor

Format: NSPEED {filename}

Edit ASCII text. Superedit has the following capabilities: multibuffer editing, multiple I/O files, macroprogramming, and numeric variables. For more information, see the Superedit User's Manual.

Switches: None

Example(s):

NSPEED)

```
!       Program is ready to accept commands.
.
.       User issues editing commands.
.
!H$$    User terminates SUPEREDIT and returns to CLI.
```

OEDIT Edit a disk file

Format: OEDIT filename

Invoke the octal editor to examine and modify in octal, decimal, or ASCII any location in any user file. For more information, see the Octal Editor User's Manual.

Switches: None

Extensions:

The octal editor searches for whatever file name and extension are given.

Example(s):

```
OEDIT FOO.SV )      ← If OEDIT finds FOO.SV, the
                      editor gives a carriage re-
                      turn/line feed.
14/01672            ← User proceeds with editing.
.
.
.
$Z                  ← To return to CLI, user types
                      ESC Z (echoed as $Z).
R                     OEDIT then returns to
                      command level.
```

You can find the current value of NMAX by issuing the OEDIT command

404-16/nnnnnn

The value nnnnnn is returned by OEDIT, and indicates the contents of USTNM of the User Status Table; USTNM contains the current value of NMAX (See Chapter 5). Note that all save files are core images minus the first 16_8 words of memory (system save files are an exception). Thus to reference a memory location n in a save file, you must subtract 16 from n.

POP Return to the next higher level program in this program environment

Format: POP

Use this command for program swaps, to return to the next higher level program in this program environment. DOS permits fives levels of hierarchy in program swaps, where one level calls to another and is swapped to disk while the other executes. CLI is on level 0. See Chapter 4 for more information.

Oftentimes a user program may wish to suspend its operation temporarily by pushing the CLI into operation at the next lower program level (via a .EXEC system call). The POP command provides a means to return from the lower level CLI to resume the user program's operation.

An attempt to issue this command from a CLI residing at program level zero will be rejected with a CLI runtime error.

Switches: None

Example(s):

RESTORE.SV, a user program running at level 1, wishes to suspend its operation temporarily so that it can enlist the aid of the CLI to perform some routine maintenance function. However, upon the completion of this maintenance function, RESTORE wishes to resume its own operation. Thus a return (via CTRL A or any other means) to the level zero CLI would be inadequate, since it would not be possible to resume the operation of RESTORE at the point where it was suspended. To enlist the support of the CLI temporarily, RESTORE issues system call .EXEC, causing itself to be swapped to disk and the CLI to be invoked at level 2.

After using the CLI at level 2 to perform whatever funtions were needed, the console operator merely issues the command

POP)

RESTORE will be restored in main memory and its execution will resume at the point of interruption.

PRINT Print a file on the line printer

Format: PRINT filename$_1$ {...filename$_n$}

This command is the equivalent of a series of XFER commands:

XFER/A filename$_1$ $LPT;...;XFER/A filename$_n$ $LPT

The source files may come from any device. If the system detects a parity error, it prints a left slash (\) in place of the bad character, and outputs the message "PARITY ERROR" on the console; printing then continues.

Switches: None

Example(s):

PRINT FOO.SR DP2:COM.SR EXT.SR)

Prints source files FOO, COM, and EXT, on the line printer.

PRINT ARREARS:JOEJR)

prints file JOEJR in subdirectory ARREARS.

PUNCH Copy an ASCII file on the paper tape punch

Format:

PUNCH filename$_1$ {...filename$_n$}

Translates a given ASCII file or files to paper tape on the high-speed punch. This command is the equivalent of a series of XFER commands:

XFER/A filename$_1$ $PTP;...;XFER/A filename$_n$ $PTP

The source files may come from any device. If the system detects a parity error, it punches a left slash (\) in place of the bad character, and outputs the message "PARITY ERROR" on the console; punching continues.

Switches: None

Example(s):

PUNCH DP0:ALPHA.SR BETA.SR)

punches files ALPHA.SR and BETA.SR, on the high-speed punch.

RELEASE Release a device from the system

Format: RELEASE { directory / device-specifier }

The RELEASE command dissociates a directory or device from the system, and prevents further access to it. When you RELEASE a mag tape unit, the tape is automatically rewound. After release, a directory or device is closed to access until you initialize it with an INIT or DIR command. Be sure to release a diskette before physically removing it. To shut down the system, release the master device (which might not be the current diskette). A release of the master directory releases all initialized devices in the system, including tape transports. Releasing a diskette causes its directories to be released.

After you have released the master disk, the message "MASTER DEVICE RELEASED" will appear on the console, and all core memory within the program area will be set to HALT.

If you have been working on a system level, you must close all files (see .CLOSE and .RESET, Chapter 3) before releasing their directory. Each file's use-count must be zero for the release.

Switches: None

Example(s):

RELEASE DP1)

This command releases DP1 and all its directories and permits the diskette to be removed from unit 1. DP1 was not the master device.

RELEASE MT0)

MT0 will be rewound.

RENAME Change a file's name

Format:

RENAME oldname$_1$ newname$_1$ [... oldname$_n$ newname$_n$]

Switches: None

Example(s):

DELETE Q.SV)
R

RENAME QTEST.SV Q.SV)
R

The above commands replace the old version of Q.SV with a new version, one previously named QTEST.SV.

RENAME DP0:A1.DR DP0:A.DR B1 B)

Rename directory A1 to A on diskette unit 0; rename file B1 to B in the current directory.

REV Obtain the revision level of a save file

Format: REV filename [.SV]

Display the revision level of a save file. The save file must have the "S" attribute. The system will return the major revision number followed by a period and a minor revision number. Both major and minor revision levels can be in the range 0 through 99, and include "PR" if the save file is a pre-release revision.

The .REV pseudo-op is used to assign a major and minor revision level number to a save file. If this pseudo-op is not used in a save file, then the revision level of this save file will be displayed as "00.00".

Switches: None

Example(s):

```
REV TEST.SV )

03.07
```

This response indicates that the major revision level of TEST.SV is 03, and the minor revision level is 07.

RLDR Perform a relocatable load

Format: RLDR { binary1... / root binary1...[overlay binary1...]... }

Use this command to load relocatable binary files and create a save file from the load. If you specify overlays, the command creates a corresponding overlay file. By default, the save file is named for the first relocatable binary in the command line, and receives the extension .SV; the overlay file (if any) is also named for the first relocatable binary, and receives the extension .OL.

Use the second command format to load overlays. The root binary is the binary which will remain in core, while it calls each overlay binary into the overlay node beneath it, and overwrites one overlay binary with another according to the needs of your program. The left bracket ([) defines the start of an overlay segment on disk. For example, in the command line RLDR ABLE [A,B] , binary ABLE will be loaded as a root program, and binaries A and B loaded as its overlays. Use a comma to delimit each overlay (or overlay group) within the brackets. For more on overlays, see Chapter 4; for more on RLDR, see the Extended Relocatable Loader Manual.

RLDR always uses its own save and overlay files to load a file. Whenever you load a program to be run under DOS, RLDR also uses the system library, where it selects the proper task monitor and modules for the load. If you have specified overlays, other modules from the system library are placed at the end of each root program.

During each load, RLDR produces a symbol table (also called a core map) for debugging the files it has loaded. Unless you specify otherwise with global switches, this table will not be printed, and it will be positioned immediately above the highest address of the loaded files. This table won't be written to disk unless you append the global /D or /K switches, below. By default, User symbols will not be loaded; see local /U switch.

In multitask programming, you must use the local /C and /K switches or a .COMM TASK statement to avoid the fatal load error "LOAD OVERWRITE 17".

Global Switches:

/A Produce an additional symbol table listing with symbol ordered alphabetically. (You must also include the local switch /L.)

/C Load for compatibility with RTOS; INMAX is set at 440, the save file starts at 0 (as with /Z global switch), USTSA contains the program starting address, and SYS.LB is not searched unless you specify otherwise.

/D Load symbolic debugger DEBUG III from SYS.LB. The symbolic debugger IDEB will be loaded instead of DEBUG III only if you include IDEB.RB somewhere in the RLDR command line. The symbol table will not be written to the save file unless you include the /D switch.

/E Output error messages to the console when a listing file has been specified (local /L). By default, when a listing file is specified, error messages to the console are suppressed. If you specify no listing file, error messages are always output to the console.

/H Print numeric output in hexadecimal (radix 16). By default, output is printed in octal.

/K Keep the symbol table for this load in savefilename.ST. Normally, this file is deleted at the end of the load.

/M Suppress load map and all console output. This speeds loading on systems with teletypewriter consoles, but you should use it carefully since it suppresses all load error messages.

/N Inhibit search of the system library SYS.LB. By default, the search is performed.

/S Leave symbol table at the high end of memory (must be used with /D).

/X Used for system loads during SYSGEN.

/Y See global /X.

/Z Start save file at location zero. Use the /Z switch with caution. A save file produced using the /Z switch cannot be executed under DOS. The switch's primary purpose is to permit loading of routines that use page zero locations 0 - 15. The save file can then be output using MKABS/Z to produce an absolute binary that can be read in the stand-alone mode, using the binary loader or BOOT.

Local Switches:

/C Preceding octal value specifies the number of channels required. This value is placed in USTCH of the User Status Table and overrides any channel number you specified in a .COMM TASK statement (See Chapter 3).

/E Output error messages output to given file name.

/K Preceding octal value specifies the number of tasks required. This value overrides any task number you specified in a .COMM TASK statement.

/L List the symbol table on the output file whose name precedes the switch. The table will list symbols in numeric order.

/N NMAX, the starting address for loading a file, is forced to an absolute address given by the octal number preceding the switch. The specified value must be higher than the current value of NMAX when this command line is executed.

/S Give save file the preceding file name with the .SV extension. Overlay file is also given the preceding file name but with the .OL extension.

/U Load user symbols from the relocatable binary file whose name precedes the switch.

Extensions:

A search is made for each input file with the name filename.RB. If not found, then a search is made for filename.

Example(s):

```
RLDR A B C DP2:D )
```

loads files A, B, and C from the default directory, and D from diskette unit 2, and produces save file A.SV on the default directory.

```
RLDR A/S $PTR )
```

loads the file in the paper tape reader to produce a save file names A.SV.

```
RLDR/D A B C )
```

loads files A, B, and C together with the symbolic debugger, to produce save file A.SV.

RLDR (Continued)

RLDR $LPT/L A 4440/N B)

loads A and B; loading of B starts at 4440_8. A numeric core map is printed on the line printer.

RLDR Q)

File in the current directory is loaded to produce save file Q.SV. All loader messages (except a load map) are output to the console; there is no load map.

RLDR ZUT $LPT/L)

File ZUT is loaded to produce ZUT.SV. All loader messages (including the load map) go to the line printer.

RLDR/E X $LPT/L)

File X is loaded to produce X.SV. All loader messages go to $LPT. Additionally, all messages (except the load map) go to the console.

RLDR R0 [A, B, C, D] R1 R2 [E, F G, H])

loads R0, R1, and R2 to a root program with two overlay segments on disk. The overlays in segment 0 (A, B, C, and D) can be called one-by-one by program R0 into the core node reserved for R0; the overlays in segment 1 (E, F G, and H) can be called by R2 into the core node reserved for R2. The overlay file contains both overlay segments and is named R0.OL. (See Chapter 4 for more on overlays.)

SAVE Save the core image as a save file

Format: SAVE filename

Create a save file from the file named BREAK.SV on the default directory diskette. SAVE is commonly used to save the core image of a program interrupted by a CTRL C break or by the debugger $V command. The command renames the core image file BREAK.SV to filename.SV. If filename already exists on the default diskette it is deleted. You cannot precede filename with device specifier. For more information on BREAK.SV, see Keyboard Interrupts, Chapter 3.

Switches: None

Extensions:

Output always has the .SV extension. If the filename argument already has an extension, the extension will be ignored, e.g., either

SAVE GAMMA) or SAVE GAMMA.YY)

would produce the save file GAMMA .SV.

Example(s):

```
DEB ALPHA)                  ←enter debugger to
                              correct location PP
PP/LDA 2@0 LDA 2 @0 3)
$V
BREAK                       ←exit from debugger
SAVE ALPHA)                 ←save core image as
                              ALPHA.SV.
```

SDAY Set today's date

Format: SDAY month day year

Set the system calendar. The year information may be either two or four digits (e.g., 76 or 1976). Use spaces or commas to separate the date arguments.

Switches: None

Example(s):

```
SDAY 4 17 1976)
```

sets the system calendar to April 17, 1976.

STOD Set the time

Format: STOD hour minute second

Set the system clock (which is a 24 hour clock). Use spaces or commas to separate the time arguments.

Switches: None

Example(s):

```
STOD 21 24 0 )
```

Sets the system clock to 9:24:0 p.m.

SYSGEN Generate a New System

Use the SYSGEN command to generate an operating system (read Appendix E first). All SYSGEN dialog will be output on TTO. The new system will be loaded automatically (saving you the BOOT step) unless you specify the global /N switch.

Format:

SYSGEN {sysname/S} {dialogfile/V} {listingfile/L}

SYSGEN {newsysname/S} {olddialogfile/A newdialogfile/V} ↑)
{listingfile/L}

Use form 1 of the command to generate a brand-new system, or a system which differs in any way from an existing system. In the first format, you choose the sysname which will be used to boot the system; the /S switch will save your system in disk file sysname.SV. (If you omit a name, the system will assign the default name of SYS000.SV.) The name you specify for dialogfile, and the /V switch, will allow you to copy the system easily in the future (form 2, below). The load map, which you may use for system diagnosis or to update your system, will be output to TTO unless you specify a different listingfile.

Use form 2 when you want to copy an existing system. The copy will be saved under newsysname.SV. Olddialogfile is the dialog file name of any existing system; the /A switch directs the current system to generate the copy from olddialogfile. Newdialogfile is your choice of a name for the new dialog file (you can append the /V switch if you want to copy this system). After SYSGEN, the load map will be output to TTO unless you choose another listingfile.

Global Switches:

/N Do not load the generated system.

/A Generate a system from preceding dialog file name.

/L Output load map to the specified listing file. (Default is TTO.)

/S Create save file for the finished system and rename it to the specified filename.

/V Save the SYSGEN dialog for the system being generated in the specified filename.

Example(s):

SYSGEN FIRSTSYS/S FIRSTSYS.SG/V)

This command activates the system generation program, which allows you to generate a new DOS system by answering a series of questions. (These are described in Appendix E). The new system will be named FIRSTSYS, and reside in file FIRSTSYS.SV and FIRSTSYS.OL. The dialog file will be FIRSTSYS.SG. The /N switch is not specified, therefore FIRSTSYS will be loaded automatically; load errors and the load map will go to TTO.

The following example presumes that FIRSTSYS is running.

SYSGEN SECONDSYS/S FIRSTSYS.SG/A ↑)
SECONDSYS.SG/V $LPT/L)

This command line generates a system from the FIRSTSYS.SG dialog file. The copy is named SECONDSYS and stored as SECONDSYS.SV. The new dialog file is named SECONDSYS.SG; the load map and loading errors go to the line printer.

After you have generated a system, certain SYSGEN files, like SYS000.RB and SYSGEN.CM will remain on diskette. You can delete them at will.

TYPE Output file contents on the system console

Format: TYPE filename$_1$ {...filename$_n$}

Copy an ASCII file or files on the program console. The command is the equivalent of a series of XFER commands:

XFER/A filename$_1$$TTO;...;XFER/A filename$_n$$TTO)

The source files may come from any device. When the system detects a character with bad parity, it types a left slash (\) and outputs the message "PARITY ERROR"; typing continues.

Switches: None

Example(s):

TYPE A.SR B.SR DP1:XX.SR)

This command displays or types the following disk files on the program console: A.SR and B.SR, in the current directory, and source file XX on diskette DP1.

UNLINK Delete a link entry name

Format:

UNLINK linkname$_1$ {...linkname$_n$}

Unlink files by deleting one or more link entrynames. This does not affect the resolution file.

Global Switches:

/C Confirm each removal. The system displays each link entry name on the console, and waits for you to confirm the deletion by typing a carriage return. To prevent the deletion or unlinking, type any key other than a carriage return.

/L List the removed files on $LPT (overrides /V).

/V Verify the removal of a list of deleted link entry names.

Template Characters:

Permitted only when the filename argument is in the current directory.

Example(s):

UNLINK/C TEST.-)

This command asks for a confirmation before each link entry is removed:

```
TEST.SR )  *     Link TEST.SR is removed
TEST.RB )  *     Link TEST.RB is removed
TEST.SV %        Link TEST.SV is not removed
```

When you confirm a deletion with a carriage return, the system echoes with an asterisk (*). Otherwise, it echoes nothing.

XFER Copy the contents of a file into another file

Format: XFER sourcefile destinationfile

Use XFER to transfer a file on any device to another file on any other device. You can select a different file organization with local switches. If destinationfile does not exist, XFER creates it. By default, the destination file is organized randomly. The source file can be organized contiguously or randomly.

The system will detect parity errors in ASCII files, and it will detect parity errors in binary files on magnetic tape. When a parity error is detected in the input file, the message "PARITY ERROR, FILE:xxx" will be output to the console. If one or more parity errors are detected in an ASCII paper tape file, a backslash will be substituted for each bad character. If a parity error is detected in a magnetic tape file, the transfer is terminated.

If you want to type input into a DOS file directly, you can do so using the form:

XFER/A $TTI filename)

and typing the text or commands you will transfer to the file. When you are done, get out of input mode by entering CTRL Z. You can then type or print filename.

Global Switches:

By default, files are transferred sequentially without alteration. There are two switches:

/A ASCII transfer. Transfer the file line by line taking appropriate read/write action, such as inserting line feeds after each carriage return when transfer is from disk to line printer.

/B Append the source file to the end of the destination file.

Local Switches:

By default, the destination file will be organized randomly.

/R Organize the destination file randomly. (/R is not used on the source file.)

/C Organize the destination file contiguously. (/C is not used on the source file, and may be used only if the source file is a disk file.)

Example(s):

XFER $PTR Q)

Transfers the file in the paper tape reader to a disk file named Q.

XFER/A ALPHA.SR $LPT)

Prints ASCII file ALPHA.SR on the line printer.

XFER $PTR $PTP)

Punches a duplicate tape the paper tape reader.

XFER DP0:MYFILE DP1:MYFILE)

transfers MYFILE from diskette unit 0 to diskette unit 1.

```
R
XFER/A $TTI NOVEL)
AT THE BEGINNING...)
.
.

THE END)
CTRL Z
R
```

This sequence creates the file NOVEL, which consists of input from the console.

CLI ERROR MESSAGES

ATTEMPT TO READ INTO SYSTEM SPACE
Attempted access of system area.

ATTEMPT TO WRITE AN EXISTING FILE
Attempt to write an existing file.

ATTEMPT TO RELEASE AN OPEN DEVICE
Attempt made to release an open device.

BRACKET ERROR
Nested or unmatched brackets.

CHANNEL ALREADY IN USE
Attempt to open a channel which is already in use.

CHANNEL CLOSED BY ANOTHER TASK
Two tasks share a common I/O channel. One task closed the channel before the other task was able to complete its I/O.

CHECKSUM ERROR
Checksum error detected during input.

COMMAND LINE TOO LONG
Command line exceeds limit.

DEVICE ALREADY IN SYSTEM
Illegal (and unnecessary) attempt to re-initialize a a directory device, or to identify a system device as a user device.

DEVICE NOT IN SYSTEM
Attempt to reference an unitialized directory or device.

DEVICE PREVIOUSLY OPENED
The device is already open for I/O.

DEVICE TIMEOUT
10 second disk timeout occurred on a nonmaster device.

DIRECT I/O ACCESS ONLY
Attempt to perform sequential I/O on a file with the I attribute.

DIRECTORY DEPTH EXCEEDED
Attempt to create directory within a directory.

DIRECTORY IN USE
Attempt to assign a logical name to a directory or device which has already been initialized. Alternatively, an attempt to release a directory which has open files in it.

DIRECTORY NOT INITIALIZED
Attempt to reference an uninitialized directory or device.

DIRECTORY SIZE INSUFFICIENT
A CPART command specified less than 48 disk blocks.

END OF FILE
End of file detected.

ERROR IN USER TASK QUEUE TABLE
Bad table input to .QTSK.

FATAL SYSTEM UTILITY ERROR
File read error.

FILE ALREADY EXISTS
Attempt to create a file with a name which is already in use.

FILE ATTRIBUTE PROTECTED
Attempt to CHATR an attribute-protected (A) file.

FILE DATA ERROR
File read error.

FILE DOES NOT EXIST
Attempt to reference a nonexistent file.

FILE IN USE
Attempt to modify a file which is in use.

FILE NOT OPEN
Attempt to access an unopened file.

FILE POSITION ERROR
Attempt to access an illegal position in a file.

FILE READ PROTECTED
Attempt to read a read-protected file.

FILE SPACE EXHAUSTED
Out of User disk space, or mag tape EOT reached while writing.

FILE WRITE PROTECTED
Attempt to modify a write-protected file.

FILES MUST EXIST IN SAME DIRECTORY
Improper use of RENAME command.

ILLEGAL ARGUMENT
Illegal character in an argument.

ILLEGAL ATTRIBUTE
Undefined attribute specified.

ILLEGAL BLOCK TYPE
Attempted LOAD of a file which is not in DUMP format.

ILLEGAL CHANNEL NUMBER
Channel number exceeding 77_8 was input to a system command.

ILLEGAL COMMAND FOR DEVICE
Attempt to perform illegal I/O (e.g., direct I/O on disk data).

ILLEGAL DIRECTORY NAME
Illegal character in link resolution name string was given in a LINK command.

ILLEGAL FILE NAME
Undefined character in file name string.

ILLEGAL NUMERIC ARGUMENT
Nonnumeric character in a numeric argument.

ILLEGAL OVERLAY NUMBER
The specified overlay does not exist.

ILLEGAL SYSTEM COMMAND
Attempt to reference an uninitialized directory or device, or to release an uninitializable device.

INSUFFICIENT CONTIGUOUS BLOCKS
Insufficient number of free contiguous disk blocks to create the desired file.

INSUFFICIENT MEMORY TO EXECUTE PROGRAM
Attempt to allocate more memory than is available.

INVALID TIME CHANGE
Attempt to set illegal time or date.

LINE TOO LONG
Line limit exceeded on read or write line I/O.

LINK DEPTH EXCEEDED
Attempt to reference a resolution file via more than 10 levels of links.

LINK NOT ALLOWED
Attempt to link to a file which has the no link resolution (N) attribute set.

MAP.DR ERROR
File system MAP.DR inconsistency detected in a directory device.

NO DEBUG ADDRESS
The debugger was not loaded with this save file and the DEB command was issued.

NO DIRECT I/O
Direct I/O allowed only on magnetic tape files.

NO FILES MATCH SPECIFIER
No match on template characters (* or -).

NO MORE DCBS
Attempt to initialize too many devices and/or directories. Release one or more currently initialized devices or directories, or SYSGEN a new system and specify a greater number of subdirectories/subpartitions to be accessible at one time.

NO ROOM FOR UFTS
Insufficient number of channels specified at SYSGEN time.

NO SOURCE FILE SPECIFIED
The CLI command requires a source file.

NO STARTING ADDRESS
Attempt to execute a nonsave file.

NO SUCH DIRECTORY
Directory specifier unknown.

NOT A COMMAND
Fatal CLI error due to modification of CLI.SV or CLI.OL.

NOT A LINK ENTRY
Attempt to delete a link entry. Use the UNLINK command (see DELETE).

NOT A SAVED FILE
The command requires a disk file in .SV format.

NOT ENOUGH ARGUMENTS
More arguments are required by this command.

OUT OF TCB'S
Task Control Blocks all used.

PARITY ERROR
Parity error on read line or read sequential of mag tape.

PERMANENT FILE
Attempt to delete or rename a permanent file; attempt to LOAD or DUMP a permanent file without using the /A global switch.

PHASE ERROR
Attempt to reset location counter back over current value; you can reset this counter only in the MKSAVE command.

POP ERROR
Attempt to return from level zero.

PUSH DEPTH EXCEEDED
Attempt to push too many levels (limit of 4).

REPEAT ERROR
An odd number of parentheses in a single command line.

SYS.DR ERROR
File system SYS.DR inconsistency detected.

SYSTEM DEADLOCK
System has run out of buffers.

SYSTEM STACK OVERFLOW
System stack capacity exceeded.

STACK OVERFLOW
Fatal CLI runtime error, generally caused by one of the following: (1) CLI.OL has been modified; (2) the creation dates of CLI.SV and CLI.OL do not match; (3) command string too long. The second condition might be caused if you tried to replace a CLI while it was running, an impossible task since CLI.OL would be open and could not then be replaced.

TASK ID ERROR
Task ID violates syntax requirement.

TASK NOT FOUND FOR ABORT
Attempt to abort a task that does not exist.

TOO MANY ACTIVE DEVICES
Too many devices active at the same time.

TOO MANY ARGUMENTS
Too many arguments input to a command where the number or position of arguments is significant.

UNIT IMPROPERLY SELECTED
Magnetic tape controller not turned on, tape unit not ON LINE. Disk unit adapter not turned on.

UNMATCHED @
Command line has an odd number of @ signs.

YOU CAN'T DO THAT
Attempt to end console logging without using password. Attempt to run an ECLIPSE program on a NOVA. Alternatively, this message is given to the CLI by the system for grossly bad input, such as lower case or non-ASCII characters.

END OF CHAPTER

APPENDIX A
DOS COMMAND AND ERROR SUMMARY

COMMAND SUMMARY

After a task or system call, AC3 contains the User Stack Pointer (USP). Error codes, if any, are returned in AC2. Task calls sometimes destroy ACs, as noted. SYSTM calls preserve ACs if not specifically returning values.

CALL	AC0	AC1	AC2
.ABORT	Destroyed.	Bits 8-15: task I.D. number.	
.AKILL(1)	Priority of tasks to be killed.		
.SYSTM .APPENDn	Bytepointer to file name.	Device characteristic mask (see .GTATR).	Channel number (if n = 77)
.ARDY(1)	Priority of tasks to be readied.		
.ASUSP(1)	Priority of tasks to be suspended.		
.SYSTM .BOOT(2)	Bytepointer to diskette specifier:filename.		
.SYSTM(1) .BREAK(2)			
.SYSTM .CCONT	Bytepointer to file name.	Integer number of disk blocks:	
.SYSTM .CDIR	Bytepointer to new directory name.		
.SYSTM .CHATRn	1B0: read protected. 1B1: attribute protected. 1B7: no link resolution. 1B9: user attribute. 1B10: user attribute. 1B14: permanent file. 1B15: write protected.		Channel number (if n = 77)
.SYSTM .CHLATn	same as .CHATR		Channel number (if n = 77)

(1) no error return

(2) no normal return

CALL	AC0	AC1	AC2
.SYSTM .CHSTSn	Starting address of 22_8 word area.		Channel number (if n = 77)
.SYSTM .CLOSEn			Channel number (if n = 77)
.SYSTM .CONN	Bytepointer to file name.	Integer number of disk blocks.	
.SYSTM .CRAND	Bytepointer to file name.		
.SYSTM .CREAT	Bytepointer to file name.		
.SYSTM .DELET	Bytepointer to file name.		
.SYSTM .DIR	Bytepointer to directory/ directory device specifier.		
.DQTSK		bits 8-15: task I.D. number.	(returned) Base address of released queue area.
.DRSCH(1)			
.SYSTM .DUCLK	Number of RTC ticks.	Address of user interrupt routine.	
.SYSTM .EOPENn	Bytepointer to file name.	Characteristic inhibit mask (see .GTATR). 0 leaves previous characteristics unchanged.	Channel number (if n = 77)
.ERSCH(1)			
.SYSTM .ERTN(2)			Data word to be passed to next higher level.

(1) no error return

(2) no normal return

CALL	AC0	AC1	AC2
.SYSTM .EXEC	Bytepointer to save file name.	0: swap to user program. 1B0: chain to user program. 1: swap to debugger. 1B0+1: chain to debugger.	Message to new program.
.SYSTM .FGND	0	(returned) Program level code (1 = level 0...5 = level 4)	
.SYSTM .GCHAR	(returned) bits 9-15: character bits 0-8: cleared		
.SYSTM .GCHN			(returned) Free channel number.
.SYSTM .GCIN	Bytepointer to 6-byte area receiving the input console name.		
.SYSTM .GCOUT	Bytepointer to 6-byte area receiving the output console name.		
.SYSTM .GDAY	(returned) Day	(returned) Month	(returned) Year - 1968
.SYSTM .GDIR	Bytepointer to 13_8-byte area.		
.SYSTM .GHRZ	(returned) 0: no RTC 1: 10 HZ 2: 100 HZ 3: 1000 HZ 4: 60 HZ 5: 50 HZ 6: microNOVA internal clock		
.SYSTM .GPOS<u>n</u>	(returned) High order portion of byte-pointer.	(returned) Low order portion of byte-pointer.	Channel number (if <u>n</u> = 77)

CALL	AC0	AC1	AC2
.SYSTM .GSYS	Bytepointer to 15_8 byte area.		
.SYSTM .GTATRn	(returned) 1B0: read protected. 1B1: attribute protected. 1B2: save file 1B3: link entry* 1B5: directory file* 1B6: link resolution entry* 1B7: no link resolution allowed. 1B9: user attribute. 1B10: user attribute. 1B12: contiguous file* 1B13: random file* 1B14: permanent file 1B15: write protected	(returned) 1B1: 80-column card. 1B2: lower-to-upper case. 1B3: form feed on open. 1B4: full word device. 1B6: LF after CR. 1B7: parity check/generation. 1B8: rubout after tab. 1B9: null after FF. 1B10: keyboard input. 1B11: TTY output. 1B12: no FF hardware. 1B13: operator intervention needed. 1B14: no TAB hardware. 1B15: leader/trailer.	Channel number (if n = 77)
.SYSTM .GTOD	(returned) Seconds	(returned) Minutes	(returned) Hours (using a 24-hr. clock)
.SYSTM .IDEF	Device code	DCT. User power restart address if AC0 = 77_8	
.IDST(1)	0: ready. 1: suspended by .SYSTM call. 2: suspended by .SUSP, .TIDS, .ASUSP. 3: waiting for .XMTW/.REC. 4: waiting for overlay area. 5: suspended by .SUSP, .ASUSP, or .TIDS and .SYSTM call. 6: suspended by .XMTW/.REC and .SUSP, .ASUSP, or .TIDS. 7: suspended by .ASUSP, .SUSP, or .TIDS and waiting for overlay area. 10: no such task exists.	bits 8 - 15: task I.D. number	(returned) Base address of task's TCB
.SYSTM .INIT	Bytepointer to directory/ global device specifier.	-1: full (tape or diskette) 0: partial	

*cannot be set by user

(1)no error return

(2)no normal return

CALL	AC0	AC1	AC2
.SYSTM .IRMV	Device code.		
.IXMT	Message address (destroyed).	Nonzero message (destroyed)	(destroyed)
.KILAD(1)	Address of kill-processing routine.		
.KILL(1)(2)			
.SYSTM .LINK	Bytepointer to link name	0: link will be resolved in diskette of link entry's residence. non-0: byte pointer is either to an alternate directory alias name or to an alias name string.	
.SYSTM .MDIR	Bytepointer to 13_8 byte area		
.SYSTM .MEM	HMA	NMAX	
.SYSTM .MEMI	NMAX increment or decrement (2's complement).	(returned) new NMAX (after change)	
.SYSTM .MTDIOn	Core data address, if a data transfer	bit 0: 1, even parity; 0, odd parity. bits 1 - 3: 0, read (words); 1, rewind tape; 3, space forward; 4, space backwards; 5, write (words); 6, write EOF; 7, read device status word. bits 4-15: word or record count. If 0 on space command, position tape to new file if it is less than 4096 records away. (returned) number of words read/written or number of records spaced.	Channel number (if n = 77_8). Status word or system error code if error returns; status word if read status normal return). Returned: 1B0: error. 1B1: data late. 1B2: tape rewinding. 1B3: illegal command. 1B4: high density if 1; low density if 0. 1B5: parity error. 1B6: end of tape. 1B7: end of file. 1B8: tape at load point. 1B9: 9-track if 1. 1B10: bad tape; write failure. 1B11: send clock. 1B12: first character. 1B13: write protected or write locked. 1B14: odd character. 1B15: unit ready.

CALL	AC0	AC1	AC2
.SYSTM .MTOPDn	Bytepointer to tape global specifier.	Characteristic inhibit mask (see .GTATR)	Channel number if n = 77_8
.SYSTM .ODIS			
.SYSTM .OEBL			
.SYSTM .OPENn	Bytepointer to file name.	Characteristic inhibit mask (see .GTATR); 0 leaves previous characteristic unchanged.	Channel number (if n = 77)
.OVEX(3)	bits 0 - 7: area number (node). bits 8 - 15: overlay number.		Return address.
.OVKIL(4)	bits 0 - 7: area number (node). bits 8 - 15: overlay number.		
.SYSTM .OVLODn	bits 0 - 7: area number (node). bits 8 - 15: overlay number.	-1: unconditional 0: conditional	Channel number (if n = 77)
.SYSTM .OVOPNn	Byte pointer to overlay file name (with .OL extension)		Channel number (if n = 77)
.OVREL	bits 0 - 7: area number (node). bits 8 - 15: overlay number		
.SYSTM .PCHAR	bits 9 - 15: character; bits 0 - 8: ignored.		
.PRI(1)	bits 8 - 15: new task priority.		
.QTSK			Address of User Task Queue table.

(1) no error return

(2) if error EREOF, error code in bits 8-15, partial read count in bits 0-7.

(3) normal return through AC2

(4) no normal return.

CALL	AC0	AC1	AC2
.SYSTM .RDBn	Starting core address to receive data.	Starting disk relative block number	bits 0 - 7: number of blocks to be read(2) bits 8 - 15: channel number (if n = 77)(2)
.SYSTM .RDLn	Byte pointer to user core area.	(returned) Read byte count (including terminator).	Channel number (if n = 77)
.SYSTM .RDSn	Bytepointer to core buffer.	Number of bytes to be read (if EOF detected, partial byte count returned).	Channel number (if n = 77)
.SYSTM .RDSW	(returned) Console switch position.		
.REC(1)	Message address.	Message.	
.SYSTM .RENAM	Bytepointer to old name.	Bytepointer to new name.	
.SYSTM .RESET			
.SYSTM .RLSE	Bytepointer to directory or global device specifier.		
.SYSTM .ROPENn	Bytepointer to file name.	Characteristic inhibit mask (see .GTATR); 0 preserves characteristics without change.	Channel number (if n = 77)
.SYSTM .RSTAT	Bytepointer to file name string.	Starting address of 22_8 word area.	
.SYSTM .RTN(2)			
.SYSTM .RUCLK			

(1)no error return

(2)no normal return

CALL	AC0	AC1	AC2
.SYSTM .SDAY	Day	Month	Year - 1968
.SMSK(1)	Lost	New interrupt mask to be ORed with old mask.	Lost
.SYSTM .SPOSn	High order portion of byte-pointer.	Low order portion of byte pointer.	Channel number (if n 77)
.SYSTM .STAT	Bytepointer to file name string.	Starting address of 22_8 word area.	
.SYSTM .STOD	Seconds	Minutes	Hours
.SUSP(1)			
.TASK	bits 0 - 7: task I.D. number; bits 8 - 15: task priority.	New task entry point address.	Message to new task.
.TIDK		bits 8 - 15: task I.D. number.	
.TIDP	bits 8 - 15: new priority.	bits 8 - 15: task I.D. number.	
.TIDR		bits 8 - 15: task I.D. number	
.TIDS		bits 8 - 15: task I.D. number.	
.TOVLD	bits 0 - 7: area number; bits 8 - 15: overlay number.	-1: unconditional; 0: conditional.	Channel number on which overlays were .OVOPNed.
.UCEX(1)(2)(3)		Any nonzero value if re-scheduling to occur.	

(1) no error return

(2) no normal return

(3) return address is input in AC3

CALL	AC0	AC1	AC2
.UIEX(1)(2)(3)		Any nonzero value if re-scheduling to occur.	As passed to interrupt handler.
.SYSTM .ULNK	Bytepointer to link entry name.		
.SYSTM .UPDATn			Channel number (if n = 77)
.UPEX(1)(2)(3)			
.SYSTM .WRBn	Starting core address.	Starting relative block number.	bits 0 - 7: number of disk blocks(4) bits 8 - 15: channel number (if n = 77)(4)
.SYSTM .WRLn	Bytepointer to core buffer.	Write byte count, including terminator, returned at end of write.	Channel number (if n = 77)
.SYSTM .WRS n	Bytepointer to core buffer.	Number of bytes to be written.	right byte: Channel number (if n = 77)
.XMT	Message address.	1-word message for receiving task.	
.XMTW	Message address.	1-word message for receiving task.	

(1) no error return

(2) no normal return

(3) return address is input in AC3

ERROR MESSAGE SUMMARY

Applicable commands are arranged alphabetically in columns, in descending order.

CODE	MNEMONIC	MEANING	APPLICABLE COMMANDS			
0	ERFNO	Illegal channel number.	.APPEND .CHATR .CHSTS .CHLAT .CLOSE .EOPEN	.GPOS .GTATR .MTDIO .MTOPD .OPEN .OVLOD	.OVOPN .RDB .RDL .RDS .ROPEN .SPOS	.WRB .WRL .WRS .UPDAT
1	ERFNM	Illegal file name.	.APPEND .BOOT .CCONT .CDIR .CONN .CRAND	.CREAT .DELET .DIR .EOPEN .EXEC .INIT	.LINK .MTOPD .OPEN .OVOPN .RENAM .ROPEN	.RLSE .RSTAT .STAT .UNLK
2	ERICM	Illegal system command.	Any unimplemented command passed to system.			
3	ERICD	Illegal command for device.	.APPEND .GCHAR .MTDIO	.MTOPD .PCHAR .RDB	.RDS .RDL .WRB	.WRL .WRS
4	ERSV1	File requires the Save attribute or the ranDom characteristic.	.EXEC	.RDB	.WRB	
6	EREOF	End of file.	.OVLOD .OVOPN .RDB	.RDL .RDS .WRB	.WRL .WRS	
7	ERRPR	Attempt to read a read-protected file.	.RDS .RDB	.OVLOD .OVOPN	.RDL	
10	ERWPR	Attempt to write a write-protected file.	.INIT .WRS	.WRB .WRL		
11	ERCRE	Attempt to create an existent file.	.CCONT .CDIR .CONN	.CRAND .CREAT .LINK .RENAM		

CODE	MNEMONIC	MEANING	APPLICABLE COMMANDS			
12	ERDLE	Attempt to reference a nonexistent file.	.APPEND .BOOT .DELET .DIR	.EOPEN .EXEC .INIT .MTOPD	.OPEN .OVOPN .ROPEN .RSTAT	.STAT .ULNK
13	ERDE1	Attempt to alter a permanent file.	.DELET	.RENAM	.ULNK	
14	ERCHA	Illegal attempt to change file attributes.	.CHATR	.CHLAT		
15	ERFOP	Attempt to reference an unopened file.	.CHATR .CHLAT .CHSTS .CLOSE	.GPOS .GTATR .MTDIO .OVLOD	.RDB .RDL .RDS .SPOS	.UPDAT .WRB .WRS .WRL
16	ERFUE	Fatal utility error.	.EXEC (argument to .ERTN)			
17	EREXQ	Execute CLI.CM on return to CLI.				
20	ERNUL	Null error code.	.EXEC (argument to .ERTN)			
21	ERUFT	Attempt to use a channel already in use.	.APPEND .EOPEN	.GCHN .MTOPD	.OPEN .OVOPN	
22	ERLLI	Line limit exceeded on read or write line.	.RDL	.WRL		
23	ERRTN	Attempt to restore a non-existent image.	.BOOT			
24	ERPAR	Parity error on read line. Magnetic tape parity.	.RDL	.RDS		
25	ERCM3	Trying to push too many levels.	.EXEC			
26	ERMEM	Attempt to allocate more memory than available.	.EXEC	.MEMI	.OVOPN	

CODE	MNEMONIC	MEANING	APPLICABLE COMMANDS			
27	ERSPC	Out of disk space. Magnetic tape - EOT	.BREAK .CCONT .CDIR .CRAND	.CRAND .CREAT .DIR	.INIT .OPEN .LINK	.MTOPD .WRL .WRS
30	ERFIL	File read error. Magnetic tape - bad tape - odd count.	.EXEC .OVLOD	.OVOPN .RDB		
31	ERSEL	Unit improperly selected.	.APPEND .DIR .EOPEN	.INIT .OPEN .MTOPD	.ROPEN .RLSE	
32	ERADR	Illegal starting address.	.EXEC			
33	ERRD	Attempt to read into system area.	.CHSTS .GCIN .GCOUT	.GDIR .GSYS .MDIR	.RDB .RDL .RDS	.RSTAT .STAT
34	ERDIO	Attempt to perform direct block I/O on a sequentially organized file.	.RDB	.WRB		
35	ERDIR	Files specified on different directories.	.RENAM			
36	ERDNM	Device not in system or illegal device code.	.APPEND .DIR .EOPEN	.IDEF .INIT .IRMV	.MTOPD .OPEN .RLSE	.ROPEN .RSTAT .STAT
37	EROVN	Illegal overlay number.	.OVEX .OVKIL	.OVLOD .OVREL	.TOVLD	
40	EROVA	File not accessible by direct (free form) I/O.	.MTDIO .OVLOD	.OVOPN .RDB	.TOVLD .WRB	
41	ERTIM	Attempt to set illegal time or date.	.SDAY	.STOD		
42	ERNOT	Out of TCB's.	.TASK			
43	ERXMT	Message address is already in use.	.IXMT	.XMT	.XMTW	

CODE	MNEMONIC	MEANING	APPLICABLE COMMANDS			
45	ERIBS	Interrupt device code in use.	.DUCLK	.IDEF	.RUCLK	
46	ERICB	Insufficient number of free contiguous disk blocks.	.CCONT	.CONN		
50	ERQTS	Illegal information in task queue table.	.QTSK			
51	ERNMD	Attempt to open too many devices or directories.	.DIR	.INIT		
52	ERIDS	Illegal directory specifier.	.DIR	.INIT		
53	ERDSN	Directory specifier unknown.	.APPEND .BOOT .CCONT .CDIR .CONN	.CRAND .CREAT .DELET .DIR .EOPEN	.EXEC .LINK .MTOPD .OPEN .OVOPN	.RENAM .ROPEN .RSTAT .STAT .ULNK
55	ERDDE	Directory depth exceeded.	.CDIR			
56	ERDIU	Directory in use.	.INIT	.DELET	.RLSE	
57	ERLDE	Link depth exceeded.	.APPEND .CCONT .CDIR .CONN .CRAND	.CREAT .DELET .DIR .EOPEN .EXEC	.INIT .LINK .MTOPD .OPEN .OVOPN	.RENAM .ROPEN .RSTAT .STAT .ULNK
60	ERFIU	File is in use.	.APPEND .BREAK	.DELET .EOPEN	.OPEN .RENAM	
61	ERTID	Task I.D. error.	.ABORT .DQTSK	.TASK .TIDK	.TIDP .TIDR	.TIDS
64	ERSCP	File position error.	.SPOS			
66	ERDNI	Directory/device not initialized.	.APPEND .BOOT .CCONT .CDIR .CONN	.CRAND .CREAT .DELET .DIR .EOPEN	.EXEC .LINK .MTOPD .OPEN .OVOPN	.RENAM .ROPEN .RSTAT .STAT .ULNK

CODE	MNEMONIC	MEANING	APPLICABLE COMMANDS			
73	ERUSZ	Not enough room for UFTs within USTCH, or attempting to execute a file loaded with /Z or /C global switch.	.EXEC			
75	ERNLE	Attempt to delete an entry lacking the link characteristic.	.ULNK			
77	ERSDE	Error detected in SYS.DR.	.CCONT .CDIR	.CONN .CRAND	.CREATE .INIT	.LINK .RENAM
100	ERMDE	Error detected in MAP.DR	.BREAK .CCONT	.CDIR .CRAND	.CREAT .DELET	
101	ERDTO	Device timeout.	.APPEND .BOOT .BREAK .CCONT .CDIR .CHATR .CHLAT .CHSTS .CONN	.CRAND .CREAT .DELET .DIR .EOPEN .EXEC .GTATR .INIT .LINK	.MTOPD .OPEN .OVOPN .RDB .RDL .RDS .RENAM .RESET .ROPEN	.RSTAT .STAT .TOVLD .ULNK .WRB .WRL .WRS
102	ERENA	Link not allowed.	.APPEND .DELET	.EOPEN .EXEC	.INIT .OPEN	.OVOPN .ROPEN
110	ERABT	Task abort is not allowed.	.ABORT			
111	ERDOP	Attempt to open a magnetic tape unit that is already open.	.APPEND .EOPEN	.MTOPD .OPEN	.ROPEN	
112	EROVF	System stack overflow (the current system command is aborted).	.DIR	.INIT		
114	ERNIR	Attempt to release a tape unit with a currently open file.	.RLSE			
115	ERXMZ	Attempt to transmit a zero-word message.	.IXMT	.XMT	.XMTW	

CODE	MNEMONIC	MEANING	APPLICABLE COMMANDS	
117	ERQOV	.TOVL not loaded for overlay task.	.QTASK	
121	ERFMT	Disk format error.	.DIR	.INIT
122	ERBAD	Disk has bad block table.	.DIR	.INIT
124	ERZCB	Attempt to create a contiguous file of zero length.	.CCONT	.CONN
125	ERNSE	Program is not swappable.	.EXEC	

END OF APPENDIX

APPENDIX B

HOLLERITH - ASCII CONVERSION TABLE

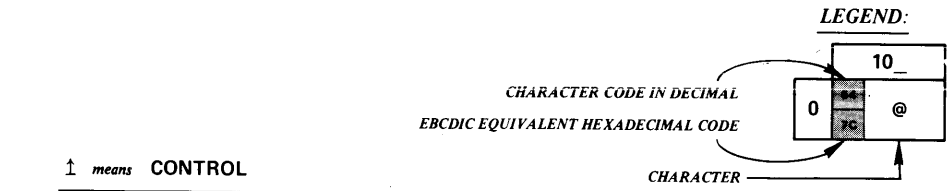
CHAR.	CARD CODE	ASCII CODE	CHAR.	CARD CODE	ASCII CODE
NUL	12-0-9-8-1	000	SPACE	NO PUNCHES	040
SOH	12-9-1	001	!	11-8-2	041
STX	12-9-2	002	"	8-7	042
ETX	12-9-3	003	#	8-3	043
EOT	9-7	004	$	11-8-3	044
ENQ	0-9-8-5	005	%	0-8-4	045
ACK	0-9-8-6	006	&	12	046
BEL	0-9-8-7	007	'	8-5	047
BS	11-9-6	010	(	12-8-5	050
HT	12-9-5	011	)	11-8-5	051
LF or NL	11-9-5	012	*	11-8-4	052
VT	12-9-8-3	013	+	12-8-6	053
FF	12-9-8-4	014	,	0-8-3	054
CR	12-9-8-5	015	-	11	055
SO	12-9-8-6	016	.	12-8-3	056
SI	12-9-8-7	017	/	0-1	057
DLE	12-11-9-8-1	020	0	0	060
DC1	11-9-1	021	1	1	061
DC2	11-9-2	022	2	2	062
DC3	11-9-3	023	3	3	063
DC4	4-8-9	024	4	4	064
NAK	9-8-5	025	5	5	065
SYN	9-2	026	6	6	066
ETB	0-9-6	027	7	7	067
CAN	11-9-8	030	8	8	070
EM	11-9-8-1	031	9	9	071
SUB	9-8-7	032	:	8-2	072
ESC	0-9-7	033	;	11-8-6	073
FS	11-9-8-4	034	<	12-8-4	074
GS	11-9-8-5	035	=	8-6	075
RS	11-9-8-6	036	>	0-8-6	076
US	11-9-8-7	037	?	0-8-7	077

CHAR.	CARD CODE	ASCII CODE	CHAR.	CARD CODE	ASCII CODE
@	8-4	100	\	8-1	140
A	12-1	101	a	12-0-1	141
B	12-2	102	b	12-0-2	142
C	12-3	103	c	12-0-3	143
D	12-4	104	d	12-0-4	144
E	12-5	105	e	12-0-5	145
F	12-6	106	f	12-0-6	146
G	12-7	107	g	12-0-7	147
H	12-8	110	h	12-0-8	150
I	12-9	111	i	12-0-9	151
J	11-1	112	j	12-11-1	152
K	11-2	113	k	12-11-2	153
L	11-3	114	l	12-11-3	154
M	11-4	115	m	12-11-4	155
N	11-5	116	n	12-11-5	156
O	11-6	117	o	12-11-6	157
P	11-7	120	p	12-11-7	160
Q	11-8	121	q	12-11-8	161
R	11-9	122	r	12-11-9	162
S	0-2	123	s	11-0-2	163
T	0-3	124	t	11-0-3	164
U	0-4	125	u	11-0-4	165
V	0-5	126	v	11-0-5	166
W	0-6	127	w	11-0-6	167
X	0-7	130	x	11-0-7	170
Y	0-8	131	y	11-0-8	171
Z	0-9	132	z	11-0-9	172
[	12-0-5-8	133	{	12-0	173
\	0-8-2	134	¦	12-7-8	174
]	12-11-5-8	135	}	11-0	175
↑ or ¬	11-8-7	136	~	11-0-1	176
← or _	0-8-5	137	DEL	12-9-7	177

END OF APPENDIX

APPENDIX C

ASCII CHARACTER SET



↥ means CONTROL

OCTAL	00_	01_	02_	03_	04_	05_	06_	07
0	NUL	BS (BACK-SPACE)	↥P	↥X	SPACE	(	Ø	8
1	↥A	HT (TAB)	↥Q	↥Y	!	)	1	9
2	↥B	LINE FEED	↥R	↥Z	" (QUOTE)	*	2	:
3	↥C	VT (VERT. TAB)	↥S	ESC (ESCAPE)	#	+	3	;
4	↥D	FF (FORM FEED)	↥T	↥\	$	, (COMMA)	4	<
5	↥E	CR (RETURN)	↥U	↥]	%	–	5	=
6	↥F	↥N	↥V	↥↑	&	. (PERIOD)	6	>
7	BELL ↥G	↥O	↥W	↥←	' (APOS)	/	7	?

OCTAL	10_	11_	12_	13_	14_	15_	16_	17_
0	@	H	P	X	` (GRAVE)	h	p	x
1	A	I	Q	Y	a	i	q	y
2	B	J	R	Z	b	j	r	z
3	C	K	S	[	c	k	s	{
4	D	L	T	\	d	l	t	|
5	E	M	U	]	e	m	u	}
6	F	N	V	↑ or ^	f	n	v	~ (TILDE)
7	G	O	W	← or _	g	o	w	DEL (RUBOUT)

CHARACTER CODE IN OCTAL AT TOP AND LEFT OF CHARTS.

SD-00476

END OF APPENDIX

APPENDIX D

OVERLAY DIRECTORY STRUCTURE

Every save file with user overlays has an overlay directory which describes each overlay. This directory resides in user address space, is pointed to by USTOD of the User Status Table, and can be examined by the user. Each overlay directory in a multitask environment has the following structure:

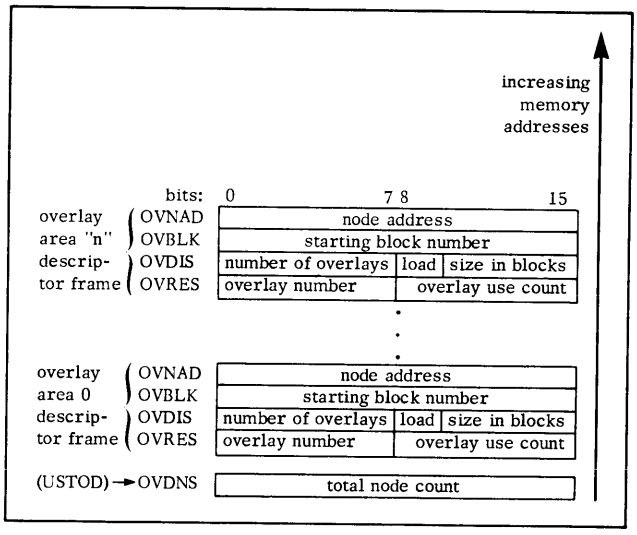


Figure F-1. Overlay Directory Structure (multitask)

Each overlay area in the save file has a corresponding four-word descriptor frame. The load bit (bit 8) of OVDIS is set to 1 during a multitask overlay load. Each overlay number, during and after loading, is bits 0-7 of OVRES. Finally, the overlay use count (OUC), bits 8-15 of OVRES, describes the number of tasks using or requesting the resident overlay. This count must go to zero before a new overlay can be loaded in this area.

Bits 0 to 7 of OVDIS describe the number of overlays which can be loaded into this overlay area. Bits 9 to 15 of this word describe the memory size (integer multiples of decimal 256, the size of each disk block) of this overlay area. OVBLK contains the starting logical disk block address of this area's segment of the overlay file, and OVNAD contains the core address for the start of this overlay area.

The overlay directory built by the relocatable loader for a single task environment is identical to that described above except that the load bit is ignored. Overlay areas hold a maximum of 256 overlays in both multitask and single task environments. The maximum number of one-block overlay areas is 124. Page zero and task scheduler space requirements limit the maximum overlay size (in a single overlay system) to 126 disk blocks.

END OF APPENDIX

APPENDIX F

BOOTSTRAPPING DOS FROM DISKETTE

FROM A COLD SYSTEM

Ensure that the write-protect hole of your system diskette is covered. Insert the system diskette into slot DP0, and make sure that no other slot has the same number. Turn DP0 ON, and power up all other equipment. Note: never change the number of a diskette while it is initialized. RELEASE it before you change its number.

The following steps apply to the first diskette controller on your system - DP0, DP1, DP2, and DP3. If your system has a second controller, and you want to bootstrap from it, use device code 73_8 instead of 33_8.

On a microNOVA with hand-held console, press RESET, CLR D, enter 33, and press PR LOAD. If you are using the console debug option to the terminal asynchronous controller board then:

- Type in: 33L

If neither of the above is present the program load option to the CPU board must be present.

- Set the jumpers for device code 33 (jumpers W4, W8, W12, W13, W15, W16 and other jumpers out).
- Hit the FRONT PANEL ROCKER SWITCH to the 'PL/START' position.

Go to step 2.

On other NOVAs:

1a. If your computer has automatic Program Load hardware: set the data switches to 100033_8 (switches 0,11,12,14 and 15 up, the others down), lift RESET, then PROGRAM LOAD. (On the SUPERNOVA, put 000033_8 in the switches, lift RESET and press CHANNEL START.) Proceed to step 2.

1b. If your computer lacks automatic Program Load hardware:

- Set the data switches to 000376_8 (switches 8 through 14 up, others down) and lift EXAMINE.
- Set the data switches to 060133_8 (switches 1, 2, 9, 11, 12, 14 and 15 up, others down) and lift DEPOSIT.
- Set the data switches to 000377_8 (switches 8 through 15 up, others down) and depress DEPOSIT NEXT.
- Set the data switches to 000376_8 (put down switch 15) and lift RESET, then START. Proceed to step 2.

2. Your manipulation of the data switches brings the Disk Bootstrap Loader into execution. BOOT then displays this query on the console:

   FILENAME?

   You must now respond with the name of your DOS system save and overlay files. This name was determined at SYSGEN; if it is the default name (SYS), you can simply press RETURN, and the system will be bootstrapped into execution. If it has any other name than SYS, you must type its name and RETURN. You can use a directory specifier to bootstrap a system not on DP0; e.g.

   ```
   FILENAME? DP1:SYS)
   ```

   DOS will now display its revision name and number, and ask log-on questions about date and time. After you have answered these, DOS will output the CLI prompt "R".

FROM A RUNNING SYSTEM

You can use the BOOT command to replace the current DOS system with a different system, on the same or on another diskette. All diskettes involved must be initialized. For example:

```
BOOT DP1:SECONDSYS)
MASTER DEVICE RELEASED
(DOS displays revision data)
DATE (M/D/Y)? 12/12/76)
TIME (H:M:S)? 14:32:30)
R
```

See the BOOT command for more information.

To sign off the current system, type:

```
RELEASE system-diskette-name)
```

END OF APPENDIX

APPENDIX G

EXCEPTIONAL SYSTEM STATUS

Certain serious error conditions can either halt the system entirely in a crash, or cause the system to suspend processing and display an exceptional status message. The message returned from exceptional status will help identify the error; no information will return from a crash.

In both situations, if you selected the core dump feature at SYSGEN (Appendix E) you can dump a core image of address space on the line printer or on diskette. This dump will help identify the error, and we recommend that you select it during system generation.

EXCEPTIONAL STATUS MESSAGES

In an exceptional status, the system will output the contents of the accumulators and an error code on the console, for example:

000015	177777	000011	037500	100010
AC0	AC1	AC2	AC3	error code

if you selected the core dump feature, the AC data will be followed by this message:

DUMP TO $LPT (TYPE 0), DP0, (TYPE 1) OR QUIT (TYPE CR)

The AC and error message data are always output; procedures to follow after you receive the DUMP message are described below. Note that if a SYSTEM error caused the exceptional status, bit 0 of the error code will be set to 0, and the rest of the code word contain a system error number, which is explained in Appendix A. The dump procedures described below apply to both kinds of error. If bit 0 is set to 1, the contents of the rest of the error code have the following meanings:

1 File system inconsistency detected. DOS tried to return a master device block which had no record in MAP.DR. DUMP the system diskette immediately, and run the diskette initializer program (DOSINIT.SV) on it. DOSINIT.SV is described in Appendix E.

2 SYS.DR error detected while accessing a directory on the system diskette. Either the entry count in a block of the directory exceeds 16 octal, or a free entry in the block was indicated but could not be found. If AC0 contains 16, AC2 contains the illegal count; if AC0 doesn't contain 16, a free entry was expected but not found. In both cases, reformat the diskette with the initializer program, DOSINIT.SV.

3 Interrupt stack overflow. The low order bits of AC0 contain the address of the overflowed interrupt stack. The stack overflow is generally caused a continually-interrupting device, which, in turn, if often caused by the wrong mask in a user-defined device. Locate and remove the problem device, and rebootstrap the system diskette.

4 Inconsistent system data, such as illegal device address or partial overwrite by a user program. Rebootstrap the system diskette.

5 System diskette data error; run diskette reliability test.

6 System diskette timeout. Make sure that the system diskette is ON, and on-line. If there are no obvious errors, run diskette reliability test.

7 Illegal device address on the system diskette. This can be caused by misreading of the diskette. Run reliability test.

10 An undefined interrupt has been detected and cannot be cleared via an NIOC. The cause is often a faulty connection. The right byte of AC2 contains the code of the device. Try to correct the problem and rebootstrap the system.

13 Attempted .RTN from level 0 (CLI level). Remove this instruction from the level 0 program, or execute it at a lower level. Rebootstrap the system.

CONTROLLING EXCEPTIONAL STATUS

You can write your own routine to handle exceptional status situations. The address of your routine must be stored in location 11, at run time, since save files begin at location 16. Your routine will then gain control at an exceptional status; the console will not display the accumulator/error code message, but AC0, AC1, and AC3 will retain the contents they had at the error, and AC3 will contain the address of the error code.

PRODUCING A CORE DUMP

If you chose the core dump feature at SYSGEN, you can dump core on the line printer and/or diskette after an exceptional status or system crash. The $LPT dump can help you pinpoint the problem; the diskette dump will help us at Data General to improve the DOS system, and minimize future errors. The line printer dump has three parts: the left column shows a memory address, the middle 8 columns show the contents of each word in the address, and the right column shows the ASCII value (if any) of each byte in the address. The figure below contains a sample line printer dump.

Before proceding with a core dump, you can select either $LPT or the diskette in DP0 to receive the dump data; you can also dump in sequence to each device. Note that if either an exceptional status or crash recurs, you should dump to both devices (if either condition recurs on the same diskette, after it has been reformatted, you should shelve the diskette).

On exceptional status, the console will show the accumulators and an error code, followed by the message:

DUMP TO $LPT (TYPE 0), DP0 (TYPE 1) OR QUIT (TYPE CR)

To dump on diskette, place a formatted diskette in DP0 and type 1. The entire address space will then be dumped to DP0, and the message will reappear.

To dump on the lineprinter, type 0; then, if you want to dump all address space, press the CONTINUE switch twice. The dump will execute, and the message reappear. If you want to dump selected portions of memory, place the starting address in the data switches, and press CONTINUE; the CPU will halt. Enter the ending address in the switches, and press CONTINUE again; the dump will proceed, and the message return. To dump another section of memory, type 0 and repeat the sequence with the data switches. You can abort the $LPT dump at any time by striking a neutral key on the console; the message will then occur again.

To quit, press the RETURN key; you can then proceed to rebootstrap DOS on either the original system diskette or a backup diskette, depending on the severity of the error. If the error recurs without a plausible explanation, please arrange to deliver the dump diskette and a software trouble report (STR) to your Data General representative.

After a system crash, the console will display nothing. Lift RESET, and enter 11 (octal) in the data switches. Lift EXAMINE, (on microNOVA, press RESET, CLRD, enter 000011 in the keys, press MEM and START) and note the number returned in the data lights. Enter this number in the data switches, lift RESET, then START. The console will then display the contents of the accumulators, an error code, and the DUMP query:

nnnnnn nnnnnn nnnnnn nnnnnn eeeeee

DUMP TO $LPT (TYPE 0), DP0 (TYPE 1), OR QUIT (TYPE CR)

Disregard the error code, and proceed with the sequence described for exceptional status dumps.

```
01020  060277 014510 060277 014506 060277 014504 060277 014502 ***H***F***D***B
01030  060277 014500 060277 014476 060277 040532 044532 050532 ***@***>**AZIZQZ
01040  054532 176660 054524 024466 125224 000420 034012 020742 YZ**YT)6****8*|*
01050  163000 042741 025415 021414 101005 045414 020454 025405 **E***#***K*|,**
01060  106414 000404 025406 041406 045405 034012 025405 044546 ******C*K*8***I*
01070  030436 051405 025377 044017 025414 044537 030431 051414 1*8***H***I*1*8*
01100  060177 024016 045010 024426 044505 020536 040422 040422 **(*J*)*IE|*A*A*
01110  034502 152120 021420 043410 175400 153102 000774 137000 9B*P**G****B****
01120  025776 044511 014467 000410 176440 002466 005731 005756 **II*7*** *6****
01130  000011 000011 001014 015766 000406 011766 030453 004434 ************1***
01140  004404 000763 034450 000535 054455 006447 126400 044450 ****9(**Y-*'**I(
01150  006446 024446 034475 137000 025400 006443 024441 010440 *&)&9******#)|*
01160  102120 107037 125401 002436 006423 000763 000000 000013 *P***************
01170  006326 000005 001014 054411 006420 020411 143000 006406 ******Y***|*****
01200  006407 006414 005124 002401 000000 002014 041057 003146 *****T******B/**
01210  001400 177777 003777 002076 002013 002617 003137 000000 *******>********
01220  001631 000000 002001 100000 002031 000000 060000 003257 ****************
01230  002000 000405 002032 000000 000424 000714 002367 177770 ****************
  :      :      :      :      :      :      :      :      :        :
```

END OF APPENDIX

APPENDIX H

RDOS - DOS COMPATIBILITY CONSIDERATIONS

Read this appendix before you try to access diskettes that were created on an RDOS system. RDOS supports certain commands and features which DOS does not. Generally, if you try to use an RDOS-only command or feature, it will behave as a no-op, and you'll receive an error message.

Mag tape files are 100% compatible between the two operating systems, therefore need no further consideration.

Diskettes created under DOS are entirely compatible with RDOS. Diskettes created under RDOS may not be compatible with DOS if certain RDOS features not supported under DOS are used. These features are:

1. Diskettes to be used under DOS must not contain partitions. Any attempt to reference a partition will result in the loss of data on the diskette.

2. Diskettes used under DOS must not have any blocks in the RDOS bad block table. DOS maintains its own bad block recording scheme. When a diskette is initialized under DOSINIT (the diskette initializer program), no bad block list is created. If you have a diskette produced under RDOS and you are not sure if it has bad blocks, you can run the DOSINIT LIST command. If there are any bad blocks, LIST will report them. Your remedy is to copy the diskette to a new diskette using the DOSINIT COPY command. This will produce a new diskette with no bad block table, and copy the blocks out of the remap area to their proper places on the new diskette. The new diskette will then be usable under both DOS and RDOS.

3. DOS has a fixed hash frame size of 5 blocks. This frame size is produced by DOSINIT.SV. Diskettes created for RDOS using DKINIT.SV have a default frame size of 5, and are therefore compatible. If you chose a frame size other than the default when you INITed the diskette using RDOS DKINIT, the diskette cannot be used with DOS. The only remedy is to copy the diskette using RDOS DKINIT to a diskette that has the correct frame size.

DETAILS

Largely because diskette storage space is limited, DOS does not support:

- Any disk type other than 6030 series diskettes.
- Memory mapping. DOS has no memory write-protection (.WRPR), expanded memory calls (.MAPDF), or virtual overlays.
- Foreground/Background programming. DOS runs in the background only; EXFG, .EXFG, GMEM, SMEM, and .EXBG return error messages. System call .FGND returns the level of the program which issues it.
- Spooling, or its associated calls and commands (.SPEA, SPKILL, etc.)
- Tuning or its calls or commands (.TUON, TPRINT, etc).
- Task-operator messages (.TRDOP), or system operator messages (.RDOP), or the OPCOM feature.
- The Batch monitor, or the RDOSSORT program.
- Secondary partitions (see note above).
- Common-area related operations (.ICMN, .RDCM, etc.)
- Task call .DELAY.
- The Multiprocessor Communications adapter, MCA, or its calls.
- Device renaming call .EQIV, or command EQUIV.
- The overlay replace commands: .OVRP, OVLDR, or REPLACE.
- The user-defined interrupt procedure enabled by .INTAD.
- Cassettes.

DOS and RDOS differ in that:

Data General supplies DOS software on diskettes, in mag tape DUMP format, which you address as DISKn instead of DPn; as with mag tape, you access files on these by number; e.g., DISK0:0.

On a microNOVA, the real-time clock is internal, and its frequency is fixed; this frequency code, as returned by system call .GHRZ, is 6.

RDOS subdirectories are called directories in DOS. DOS system directories, (SYS.DRs) are copied into each diskette and directory, as in RDOS, but have no device entries; the device entries (e.g. $LPT.) remain in memory.

DOS cannot tolerate a hard error (surface inconsistancy or flaw) on a diskette.

DOS offers two copying commands: the CLI COPY, and the DOSINIT COPY.

The DOS initializer, DOSINIT, differs from the RDOS initializer, DKINIT, as described above. You can configure any 4234, 4233 or similar disk as a psuedo-diskette by answering 6030 to the RDOS DKINIT query, but you'll be limited to 616 blocks on it.

END OF APPENDIX

APPENDIX I

DISKETTE CONSIDERATIONS

Unlike other disks, which are protected by a plastic cartridge or isolated in sealed drives, diskettes are protected only by paper envelopes, and they require careful handling.

Data General supplies each diskette in an outer sleeve and an inner envelope. You must remove the sleeve to use the diskette, but you should never remove the inner envelope, which has a small write-protect hole in a corner. When this hole is open, DOS will not write data to the diskette. To write to the diskette, cover this hole with one of the pieces of adhesive tape supplied. Note that you must never alter the drive number of a diskette drive while it is on-line (initialized); RELEASE it first.

When you insert a diskette in a drive, the write-protect hole should be at the left. This will ensure that the proper diskette surface faces the moving head. After you have removed a diskette from its drive, return it to its outer envelope.

Once you have initialized a diskette, via INIT or DIR, do not open its door until you have RELEASED it.

Occasionally, when you bootstrap DOS, or a stand-alone program like DOSINIT via the front panel switches, the computer will not respond. Open the door of DP0, close it, and try the switches again.

Temperature extremes - both hot and cold - can warp diskettes, and you can prolong their lives by storing them at room temperature. Folding will ruin a diskette. The data on a diskette can be destroyed by grease, therefore avoid touching the diskette surface with your hands. You should also keep your diskettes at least a foot away from strong magnetic fields.

RECOVERABLE ERRORS

If you selected soft error-reporting at SYSGEN, DOS itself will help you detect problems with a diskette before they disable further processing. If you omitted soft error reporting, DOS will ignore all soft errors and you'll be unaware of problems until a hard error develops. When DOS detects a write error, it attempts to rewrite the data correctly. If it succeeds, and if you chose soft error-reporting, DOS notes the error in the recoverable-error table on diskette, and prints the message:

```
SOFT ERROR ON DPn/BLOCK bbbbbb, r RETRIES
```

on the console, and adds the error to the diskette's error count. In the message, n indicates the diskette, bbbbbb the block number, and r the number of retries necessary to write the data correctly. If you receive this message several times for one diskette, copy the diskette onto a new, formatted diskette, using the CLI COPY command. The copy's error count will then be set to zero.

If you selected soft error-reporting, when the recoverable-error count reaches 30 for any track of a diskette, DOS will not permit you to initialize (or bootstrap) the diskette. Instead, when you attempt to bootstrap, INIT, or DIR to the diskette, DOS will print the message:

```
TOO MANY SOFT ERRORS
```

You must now precede to copy the faulty diskette with the DOSINIT COPY command. The DOSINIT COPY command will produce a perfect copy, with all track error counts set to zero - thus enabling you to use all the material on the faulty diskette. DOSINIT is described in Appendix E.

END OF APPENDIX

INDEX

NOTE: The legend "F" means "and the following page"; "FF" means "and the following pages". Primary command references are underlined. Generally, capitalized entries indicate commands. If they follow a period, they are system-level calls (e.g., .ABORT); if they lack a period, they are CLI commands (e.g., APPEND), or acronyms (e.g., CLI).

END OF INDEX

DataGeneral

Software Documentation Remarks Form

How Do You Like This Manual?

Title _____ No. _____

We wrote the book for you, and naturally we had to make certain assumptions about who you are and how you would use it. Your comments will help us correct our assumptions and improve our manuals. Please take a few minutes to respond.

If you have any comments on the software itself, please contact your Data General representative. If you wish to order manuals, consult the Publications Catalog (012-330).

Who Are You?

- ☐ EDP Manager
- ☐ Senior System Analyst
- ☐ Analyst/Programmer
- ☐ Operator
- ☐ Other _____

What programming language(s) do you use? _____

_____

How Do You Use This Manual?

(List in order: 1 = Primary use)

- _____ Introduction to the product
- _____ Reference
- _____ Tutorial Text
- _____ Operating Guide
- _____ _____

Do You Like The Manual?

Yes	Somewhat	No	
☐	☐	☐	Is the manual easy to read?
☐	☐	☐	Is it easy to understand?
☐	☐	☐	Is the topic order easy to follow?
☐	☐	☐	Is the technical information accurate?
☐	☐	☐	Can you easily find what you want?
☐	☐	☐	Do the illustrations help you?
☐	☐	☐	Does the manual tell you everything you need to know?

Comments?

(Please note page number and paragraph where applicable.)

From:

Name _____ Title _____ Company _____

Address _____ Date _____

FOLD DOWN FIRST FOLD DOWN

FIRST
CLASS
PERMIT
No. 26
Southboro
Mass. 01772

BUSINESS REPLY MAIL

No Postage Necessary if Mailed in the United States

Postage will be paid by:

Data General Corporation

Southboro, Massachusetts 01772

ATTENTION: Software Documentation

FOLD UP SECOND FOLD UP

STAPLE