Disk Operating System (DOS)

Reference Manual

093-000201-02

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 02, November 1978

DataGeneral
SOFTWARE DOCUMENTATION

NOTICE

Disk Operating System
(Was Diskette Operating System)
Reference Manual
093-000201

Revision History:

093-000201

Original Release - August 1976
First Revision - May 1978
Second Revision - November 1978

This document has been extensively revised from revision 01; therefore, change indicators have not been used.

PREFACE

Data General's Disk Operating System is complete, 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, a plotter, and process I/O and communications hardware.

DOS originally ran from diskette only, hence the name Diskette Operating System. Because the system can now support dual-plotter 10 megabyte subsystems, we have changed its name to Disk Operating System.

This manual covers all features of programming DOS from files to multitasking, as they apply to microNOVA and other NOVA computers.

Note that the DOS system-generation process which was covered in Chapters 7 and 8 of the last revision of this manual is now detailed in the manual How to Generate Your DOS System (093-000222).

We have organized this, the DOS manual, as follows:

- Chapter 1 introduces the Disk Operating System and outlines diskette and memory organization.
- Chapter 2 explains files and directories.
- Chapter 3 presents most of the system calls you will need for I/O in a single-user environment.
- Chapter 4 describes three tools for extending memory: program swaps, chains, and user overlays.
- Chapter 5 covers multitasking; it builds on the topics covered in Chapters 2, 3, and 4.
- Chapter 6 explores user interrupts and power fails.

Appendix A 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 contains a listing of DOS user parameters - file PARU.SR. 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.

Appendix J discusses advanced multitask programming. Appendix K presents two multitask programming examples.

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

NOTE: The term "disk" means either hard disk or diskette; "diskette" means only diskette.

The following Data General publications offer useful related information. You received some of them with your system.

14-000065	Technical Reference, Model 6030 Diskette Subsystem (NOVA DOS only).
14-000073	Technical Reference, MicroNOVA Computer Systems (microNOVA DOS only).
15-000050	Programmer's Reference Manual microNOVA Computers (microNOVA DOS only).
093-000040	Extended Assembler User's Manual
093-000065	Extended BASIC User's Manual
093-000106	Software Catalog
093-000044	Symbolic Debugger User's Manual
093-000053	FORTRAN IV User's Manual
069-000022	Learning to Use Your RDOS/DOS System
093-000074	Library File Editor User's Manual
093-000081	Macroassembler User's Manual
093-000084	Octal Editor User's Manual
093-000080	Extended Relocatable Loader User's Manual
093-000041	Relocatable Math Library File User's Manual
093-000105	RDOS/DOS User's Handbook
093-000109	RDOS/DOS Command Line Interpreter User's Manual
093-000110	Software Summary and Bibliography
093-000111	SUPEREDIT User's Manual
093-000160	Symbolic Editor User's Manual
093-000018	Text Editor User's Manual
093-000222	How to Generate Your DOS System

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
Westboro, Massachusetts 01581

READER, PLEASE NOTE:

We use certain symbols in special ways:

Symbol	Means
)	Press the RETURN key on your terminal's keyboard.
□	Be sure to put a space here. (We use this only when we must; normally, you can see where to put spaces.)

All numbers are decimal unless we indicate otherwise; e.g., 35_8.

Finally, we usually show all examples of entries and system responses in CAPITAL LETTERS. But, where we must clearly differentiate your entries from system responses in a dialog, we will underline your entry.

CONTENTS

CHAPTER 1 INTRODUCTION

CHAPTER 2 FILES, DIRECTORIES, TAPES, AND MULTIPLEXORS

CHAPTER 3 SINGLE TASK PROGRAMMING

CHAPTER 3 continued

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 GENERATING A DOS SYSTEM ON A MICRONOVA

This information now appears in How to Generate Your DOS System (093-000222)

CHAPTER 8 GENERATING A DOS SYSTEM ON A NOVA

This information now appears in How to Generate Your DOS System (093-000222)

APPENDIX J continued

APPENDIX K MULTITASK PROGRAMMING EXAMPLES

ILLUSTRATIONS

TABLES

CHAPTER 1
INTRODUCTION

Data General's Disk 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 Memory 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 Disk 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 drive or a dual-platter hard-disk subsystem.

Larger versions of DOS include up to 32K of memory, power fail/auto restart, 2.5 million bytes of diskette storage, and/or 20 million bytes of hard-disk storage. DOS can also handle ALM-driven multiplexor boards, and up to two line printers. NOVA-based DOS can support a card reader, a reader/punch, a plotter, and up to 8 mag tape drives.

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 How to Generate Your DOS System.

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:

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

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

The multitask monitor has a modular structure: at load time, you tell the Relocatable Loader utility the number of tasks and channels your program will require. The loader automatically loads only those task modules needed for execution. This conserves memory space, and allows more of your program to stay in memory 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, the CLI is an interface between your console and DOS. From your console, you use CLI to create, organize, and maintain files, and access such system utilities as the Library File Editor and the Extended Relocatable Loader.

The system restores CLI to memory 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.

DOS ORGANIZATION

The DOS executive is the framework of the operating system, and must be memory-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 memory, 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. The highest portion of DOS is a series of system buffers. These are used to receive system overlays and disk files 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_8. Locations $40\text{-}47_8$ are part of 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.

Above the UST 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. If you defined overlays in this program, an overlay directory is found above the TCBs. Your program follows the TCB pool.

To make a program executable, you use the relocatable loader utility supplied with your system. After loading your program onto diskette, this utility loads all modules referenced by the program. (It extracts these modules from libraries like the system library, SYS.LB). 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.

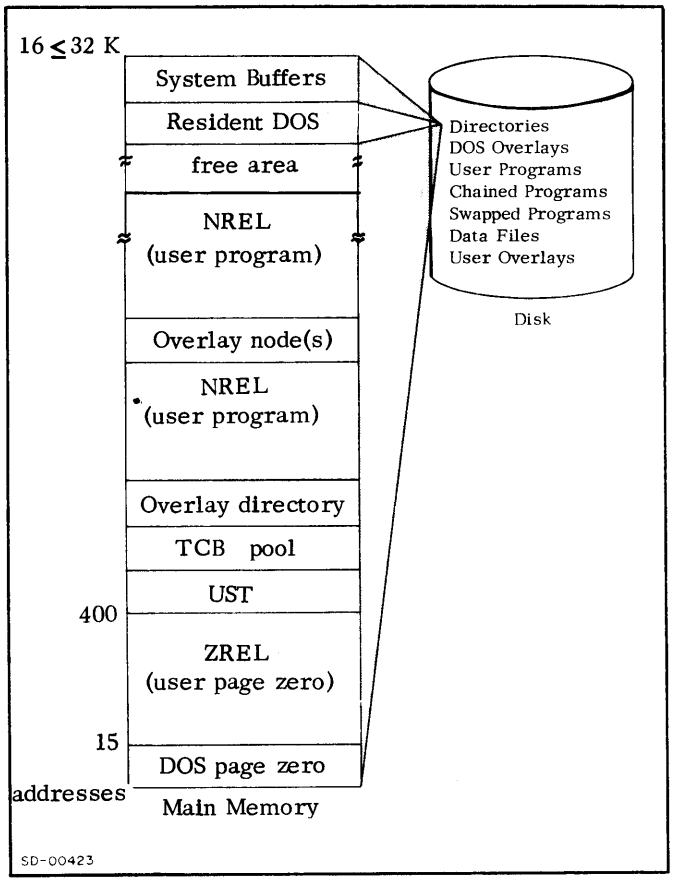


Figure 1-1. DOS Organization

END OF CHAPTER

CHAPTER 2
FILES, DIRECTORIES, TAPES, AND MULTIPLEXORS

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 a compiler or an assembler which produces as output a relocatable binary file. You input the relocatable binary file into the relocatable loader, which produces a core-image (save) file on disk for execution at absolute locations in memory. A save file is stored word-for-word as it will be executed in memory. You can designate 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 Filenames

I/O devices have special filenames 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	Disk or diskette drive n, range 0-7. Each microNOVA diskette controller handles one or two drives (slots). The first controller manages drives DP0 and DP1; the second manages drives DP2 and DP3; the third handles DP4 and DP5; and the fourth controls DP6 and DP7. Each NOVA controller handles one to four drives. Each "drive" is either one dual-platter hard-disk subsystem or one diskette slot. The first controller manages drives DP0, DP1, DP2, and DP3; the second controller manages DP4, DP5, DP6, and DP7. If any of these is a hard-disk, "DPn" indicates the removable cartridge. The fixed disk has the suffix "F"; i.e., DP0F, DP1F, DP2F, or DP3F.
DHn, DHnF	Model 6095 hard disk (microNOVA DOS only). For the first controller, n is 0, for the second controller, n is 1. The top (removable) disk is DHn; the bottom (fixed) disk is DHnF.
$LPT	First 80- or 132-column line printer
$LPT1	Second 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

QTY:n	ALM or QTY multiplexed line. n is in the range 0-7 for microNOVA, 0-63 for NOVA.
$TTI	Teletypewriter or display terminal keyboard* (see footnote next page)
$TTO	Teletypewriter printer or CRT display (80- or 132-columnd).
$TTP	teletypewriter punch
$TTR	teletypewriter reader
$TTI1	Second teletypewriter or display terminal keyboard*
$TTO1	Second console printer or CRT display (80- or 132-column).

*Input devices other than console keyboards and card readers 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 a string of up to ten ASCII characters, including upper and lowercase letters (DOS converts lowercase letters to uppercase), 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 ten. Moreover, you can use $ whenever you want in a disk file name, but you must avoid using reserved file names of devices on your system.

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

The 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. The CLI would append extensions to different versions of A, as follows:

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

Usually, when you specify a file name to a system utility, you need not enter the extension; the command will use a search algorithm to find the file with 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. Never give a source file the extension .SV, .OL, or .RB, because system utility programs may delete this file when they produce an assembled or save version of it.

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 or write-protect a complete diskette by uncovering the write-protect hole. 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 (memory 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.

DISK FILES

DOS supports up to 8 diskettes or two dual-platter hard-disk subsystems; its minimum is one diskette or one hard-disk drive. A system's diskette capacity ranges between 157,696 and 1,261,568 words; its hard-disk capacity ranges from 5 to 10 million words. DOS uses blocks 0 through 17_8 of each diskette or disk for its own files - thus 16 blocks (or 4,096 words) are not available for user storage.

DOS offers you four 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-recently-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 memory. By avoiding buffering for any specific file you save time, but must manage records yourself; you lose the automatic management of the system buffers for that file.

DOS File Organization

Disk files are organized randomly or contiguously. The maximum length of either kind of file is the amount of space remaining on the current disk.

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 608. 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 disk 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 memory 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 disk contains its own file directory. This file directory is a file which records all file names on the disk; it is called SYS.DR.

Each disk also contains its own block allocation file, called MAP.DR. MAP.DR records blocks which are in use and which are free for data storage. MAP.DR is aware of all disk space except blocks 0 through 5 which contain the bootstrap root, and other disk information. Thus these blocks can never be destroyed, since the system is unaware of the disk space where they reside.

Initial Disk Block Assignments

On every disk, 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. 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	first 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 disk 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 UFDs 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. A UFD template appears next.

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	Device code (DCT link)

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 the system fails when a file is open, its count will often be wrong; you must clear it to zero (via the CLI command CLEAR) before you can rename or delete the file.

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.

User 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.

Accessing Directories

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.

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 )
```

Releasing the master directory releases all active directories and initialized devices in the system.

See Learning to Use Your RDOS/DOS System or the CLI manual for more on directory access.

Master Disk

The master disk is the one which contains the current system. If you bootstrap another disk, it becomes the master. The master disk 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. Thus if a user program swaps (Chapter 4), DOS writes its current core image to the master directory. This temporarily requires extra space in the master directory.

Bootstrap Disk

A bootstrap disk can start up a DOS system. Such a disk must have both the bootstrap root (on blocks 0 and 1), and a copy of the bootstrap program, BOOT.SV. You can copy BOOT.SV to any diskette with the CLI MOVE command; and you can use BOOT.SV to install the root.

A bootstrap disk can start up a DOS system that exists on any disk in the system; it need not contain a DOS system itself.

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.) The attributes of the resolution entry must allow linking.

The section "Managing Diskette Space", in Chapters 7 and 8, shows some practical uses of the CLI LINK command. For other examples, see Learning to Use Your RDOS/DOS System or the CLI manual.

Directory Command Summary

Following is a list of CLI and .SYSTM commands used to manage disk file directories; see Chapter 3 and the CLI manual 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	not available	Set a file's use count to zero.
CCONT	.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 (NOVA SYSTEMS ONLY)

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 if on a dual-density drive.

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

Tape File I/O:	9-track NRZI800BPI, 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 2-1.

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.

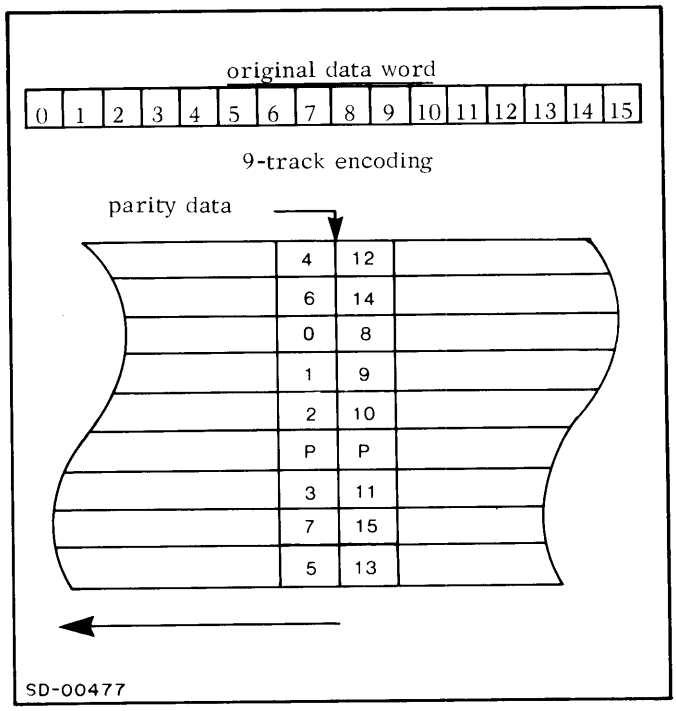
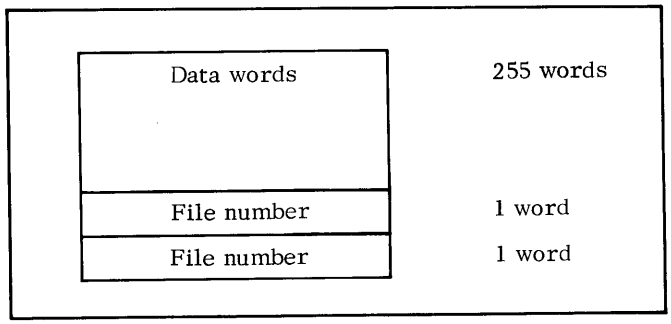


Figure 2-1. Data Encoding (9-track tapes)

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 should 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<u>n</u>:<u>m</u>	Magnetic tape unit <u>n</u>, where <u>n</u> is from 0-7 and has no leading zero, with file number <u>m</u> 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.

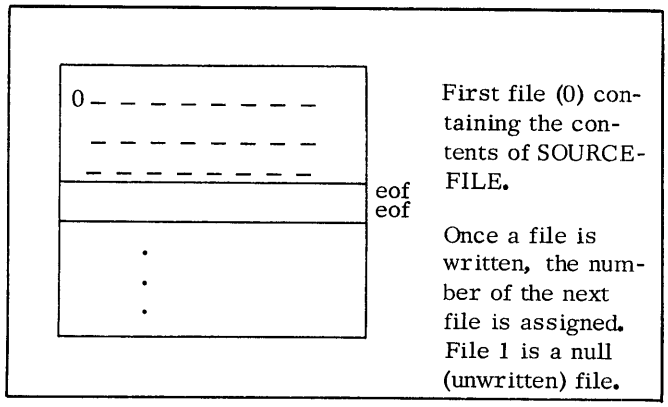
Some examples of references to files on tape and disk are:

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 MYFILE MT0:4/L)	Assemble MYFILE, and send assembly listing to tape file 4 of tape on MT0.

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.

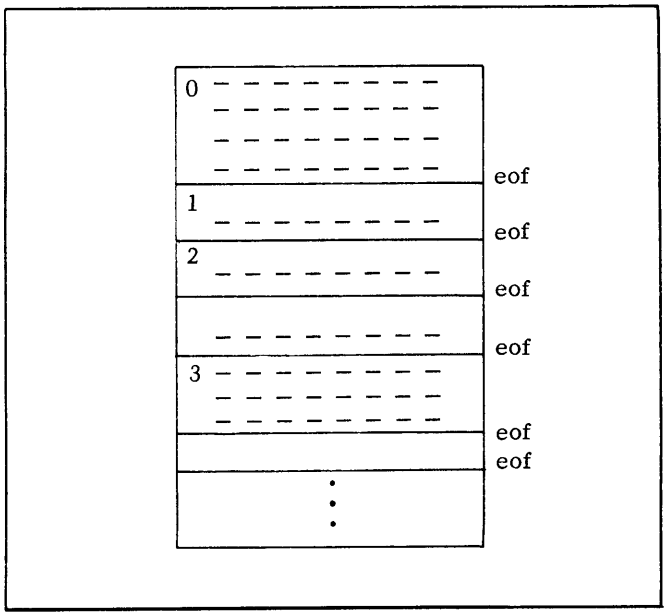
If you try to reference any other file on the tape:

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

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

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

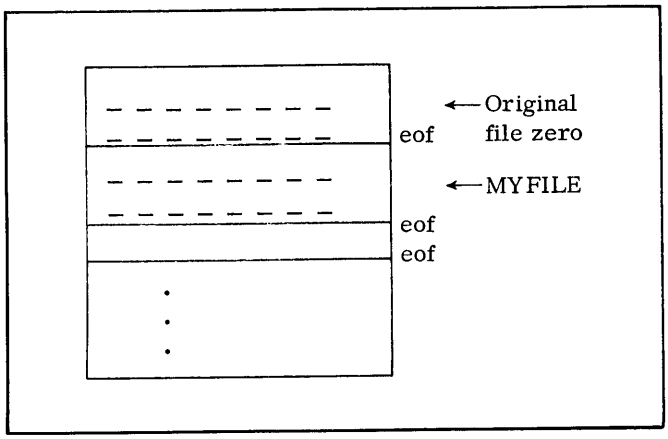
As you write files on tape, you should note their numbers. Otherwise, you could inadvertantly overwrite a file, and thus destroy the overwritten file and all following files. For example, assume a tape on drive 0 contains four files:



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 its transport. This 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.

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 after it locates the file you specified in .MTOPD; this contrasts with tape file I/O.

MULTIPLEXORS

A NOVA DOS system can support either a type 4255-series Asynchronous Line Multiplexor (ALM) or a 4060-series Asynchronous Communications Multiplexor (QTY). A microNOVA DOS system can support either a series 4226-4228 ALM, or one or more Asynchronous Interface boards (type 4207) configured as a multiplexor. (See microNOVA Technical Reference 014-000073 for details on wiring an Asynchronous PC board.)

In this section, "ALM" always means "Asynchronous Line Multiplexor"; "QTY" means either an 4060-series multiplexor (NOVA DOS), or one or more Asynchronous Interface boards wired as a multiplexor (microNOVA DOS).

The SYSGEN program allows you to specify either an ALM or QTY multiplexor; if you specify a QTY for a microNOVA system, it then asks about the jumper-selected device codes for the receive and transmit function of each asynchronous board. You'll use these codes for I/O to the multiplexed lines; they must not conflict with other device codes on your system.

SYSGEN uses file ALMSPD.SR (provided by Data General) to set characteristics for ALM lines; you can change this by editing ALMSPD.SR and assembling it with the Macroassembler, then executing SYSGEN again. ALMSPD.SR is described later in this section.

Either the ALM or QTY can support from one to eight full- or half-duplex lines (one to 64 on a NOVA). A full-duplex line allows data to flow two ways simultaneously; users can transmit to DOS over it, and DOS can transmit to users' terminals. DOS assumes full-duplex lines, but you can set up half-duplex protocols if you want.

Each ALM or QTY line is a filename, of the form

QTY:x

where x is a number from 0 to 63. For the microNOVA QTY, each asynchronous interface board is a QTY line. You can open a multiplexed line on any DOS I/O channel (channels are described in Chapter 3). After you have opened a line on a channel, you can use system calls .RDL/.WRL and .RDS/.WRS to read and write to it. In Chapter 3, I/O Channel Numbers describes selecting a channel number, and Input/Output Commands describes opening a file, and the read/write calls. No more than one read and one write can be outstanding on any one line. To close a line and abort I/O, you must .CLOSE its channel (because the .ABORT task call doesn't affect QTY/ALM I/O).

When you .OPEN a multiplexed line (or any file), the contents of AC1 determine what operations DOS will allow on that line. AC1 acts as a characteristic disable mask, as described under .OPEN (Chapter 3). The following characteristic bits affect multiplexors:

AC1	Meaning
DCCRE=1B4	Masking disables carriage return echoes on line reads (CR then acts as enter key).
DCLAC=1B6	Masking disables line feed after CR.
DCXON=1B8	Masking enables XON/XOFF protocol for $TTR. (This prevents the teletypewriter reader from overflowing the multiplexor read buffer.)
DCNAF=1B9	Masking disables 20 nulls after form feed.
DCKEY=1B10	Masking disables echo, CTRL Z end-of-file, and line and character rubout.
DCTO=1B11	Masking enables backspacing for rubout (CRT displays only).
DCLOC=1B13	Masking makes this a modem line. Default is local.
DCCGN=1B14	Masking disables TAB expansion.
DCNI=1B15	Masking enables multiplexor interrupts.

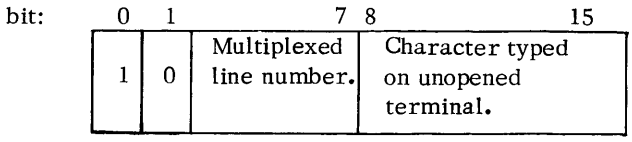
When AC1 equals 0 on the .OPEN, the multiplexed console has the following default characteristics:

1. line feeds after carriage returns
2. 20 nulls after form feed
3. during line reads: characters are echoed.
   SHIFT-L deletes line.
   RUBOUT deletes character and is echoed as backarrow.
   CTRL-Z results in an end-of-file error.
   ESCAPE also results in an end-of-file error.
4. this is a local line.
5. TABs are expanded as spaces.

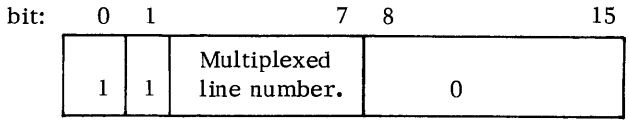
CHECKING MULTIPLEXED LINES FOR ACTIVITY OR INTERRUPTS

Line 64 Reads

DOS allows you to monitor both activity on all unopened QTY/ALM lines and console interrupts from all opened ALM/QTY lines. If a task opens QTY:64 and issues a read line or read sequential call, DOS will suspend this task until someone either presses a key at the end of an unopened line, or hits an interrupt on an opened line. When DOS receives the character typed, it readies the task, takes the normal return from the read call, and passes the following data in AC2:

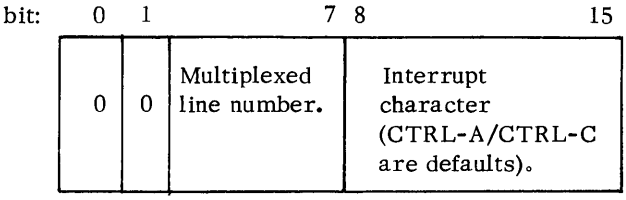


When DOS receives and answers a ring from a modem (ALM only), it will send the following data to line 64, in AC2:



This allows your program to detect a service request from a distant terminal. If the request comes from an unopened line, your program can then .OPEN the line for communications.

If an open line receives an interrupt (CTRL-A and CTRL-C are defaults), DOS will ready the task which .OPENed line 64, and pass the following data in AC2:



At SYSGEN, you can select interrupts other than CTRL-A and CTRL-C.

Line 64 Writes

DOS allows you to change the device characteristic disable mask on any line, and the line speed, or modem state on any ALM line. Just issue a .WRL to a channel opened on QTY:64, and pass the following data:

To change the mask (on .OPENED lines only)

AC0 = W64DC + line number
AC1 = new mask

To change line speed:

AC0 = W64LS + line number
AC1 = new line speed (0, 1, 2, or 3)

To change modem state:

AC0 = W64MS + line number
AC1 = [W64DTR] [+] [W64RTS]

(Entries in brackets are optional.) W64DTR raises Data Terminal Ready; if you omit it, DTR is lowered. W64RTS raises Request To Send; if you omit it, RTS is lowered.

These symbols are defined in the user parameter file PARU.SR. Appendix E contains a PARU listing.

Multiple Channels

A program can have several channels opened to the same line. The first channel opened on a line becomes the master channel, and all other channels opened on it become subordinate; if you close the master, the subordinate channel numbers will be unable to use the line. Before you can reassign (.OPEN) the subordinate channel numbers on another line, you must close each one. If you .OPEN a new channel on a line after .CLOSing the master, the new channel becomes the master channel.

ALM Modem Support

The ALM supports modems with the following six signals:

DTR - Data Terminal Ready (either DOS or you set this)

RTS - Request to Send (set either by DOS or yourself).

DSR - DataSet Ready

RD - Ring Detect (handled by the hardware)

CD - Carrier Detect (handled by the hardware)

CTS - Clear to Send (handled by hardware)

When you bootstrap DOS, it raises DTR and RTS, unless you have changed the ALM parameter file (ALMSPD.SR) to specify low DTR and/or RTS. On a disconnect, if DSR goes low, it lowers both DTR and RTS. Note that certain modem disconnects may cause multiple indications of line drop. On a ring interrupt, DOS raises both DTR and RTS. On a disconnect, if DSR is low, it lowers both DTR and CTS.

When a modem's DSR (DataSet Ready) is low, it cannot communicate; DOS will take the error return on all reads/writes to its modem line, and it will place code ERRDY in AC2. Note however, that the error return occurs only if you defined the line as a modem line by masking DCLOC on the .OPEN. On power fail, local lines are restored when power returns, and modem lines are restored when the user dials in.

MULTIPLEXOR ERROR MESSAGES

The following errors relate to reads/writes on multiplexed lines. For other read/write errors, see .RDL/.RDS or .WRL/.WRS in Chapter 3. On the error return, AC2 may contain one of the following codes:

AC2	Mnemonic	Meaning
6	EREOF	End of file detected while reading.
24	ERPAR	Hardware parity error on read.
47	ERSIM	Duplicate read or duplicate write.
127	ERRDY	Line not ready; modem's DSR is low.
130	ERINT	Console interrupt received.

The following errors clear the read buffer and error the read request:

131	EROVR	Hardware overrun error on read.
132	ERFRM	Hardware framing error on read.

ALMSPD.SR

The ALM source file, ALMSPD.SR, defines the line characteristics of each line of the ALM. You can edit this source file and assemble it with MAC (the macroassembler) to tailor your multiplexed lines for specific applications. You can then generate a new DOS system, during which SYSGEN will include the new ALMSPD.RB. If you do not define a line in this module, or if you set its characteristics at default, then it has the following characteristics:

1. clock frequency as set in SYSGEN
2. 1 stop bit
3. 7 bits per character
4. even parity
5. no loopback
6. DTR + RTS raised on initialization

You can define these characteristics for any line by inserting the line

```
LNDEF   xx,DEFAULT
```

in ALMSPD.SR where xx is the two digit number for the line you want to set. If you want to define different line characteristics, then insert a line of the form

```
LNDEF   xx,spd,stop,bits,par,loop
```

or

```
LNDEF   xx,spd,stop,bits,par,loop,dtr,rts
```

where

xx is the two-digit decimal line number;

spd is the clock frequency (may be 0, 1, 2, or 3);

stop is the number of stop bits per character (may be 1 or 2);

bits is the number of bits per character (may be 5, 6, 7, or 8, not including the parity bit);

par defines whether you wish no parity to be generated or checked (specify NO), even parity (EVEN), or odd parity (ODD);

loop tells whether you want to enable loopback (specify LOOPBACK or NOLOOPBACK);

dtr defines the state of Data Terminal Ready on initialization (DTRHIGH or DTRLOW); and

rts defines the state of the Request To Send on initialization (RTSHIGH or RTSLOW).

Note that you may omit the arguments for dts and rts if you wish to set their states as high.

For example, to set line 3 to have clock frequency 1, 2 stop bits, 7 bits per character, even parity, and no loopback, you'd insert the following line. Both Data Terminal Ready and Request To Send will be initialized high.

```
LNDEF   03,1,2,7,EVEN,NOLOOPBACK
```

After defining ALMSPD.SR, type

```
MAC ALMSPD $TTO/L
```

or

```
MAC ALMSPD $LPT/L
```

if you have a line printer before performing a new DOS SYSGEN. (Before doing this, you must build MAC's permanent symbol file, MAC.PS, as described in the Macroassembler User's Manual.)

End of Chapter

CHAPTER 3
SINGLE TASK PROGRAMMING

This chapter describes most of the system calls you will need to program in DOS in a single-task environment. It explains system and task command structures, summarizes the most commonly-used system calls, and then lists complete descriptions of single-task calls, under the headings:

DEVICE AND DIRECTORY COMMANDS
FILE MAINTENANCE COMMANDS
LINK COMMANDS
FILE I/O COMMANDS
CONSOLE I/O COMMANDS
MEMORY ALLOCATION COMMANDS
DEVICE ACCESS COMMANDS
CLOCK/CALENDAR COMMANDS
KEYBOARD INTERRUPT COMMANDS

For important single-task material on program swaps and overlays, read Chapter 4; for user interrupts read Chapter 6. Chapter 5 covers tasks and multitasking; it includes system clock commands which you can use in a single-task environment.

MULTIPLE AND SINGLE TASK ENVIRONMENTS

A program task is an execution path through user address space which demands use of system resources such as I/O, overlays, or simply CPU control. User address space includes all memory from location 16_8 through NMAX-1.

In a single-task environment, the program itself is the only task. A program creates a multitask environment by creating a task via task calls .TASK or .QTSK. If you plan a multitask program, you must specify multiple tasks either with an assembly-language .COMM TASK statement or with RLDR switches. If you do so, RLDR will load the multitask scheduler (called TCBMON) into your program, and allot the specified number of Task Control Blocks (TCBs).

The format of the .COMM TASK statement is:

.COMM TASK,k*400+c

where k is the octal number of tasks, and c is the octal number of channels for the program to use.

If you omit both a .COMM TASK statement and task/channel switches, RLDR assumes a single-task program, and loads the single-task scheduler into your program. RLDR also allots eight channels for the program - enough for most single-task programs. Either a single or multitask program can use all system calls in this chapter. For more on multitasking, see Chapter 5.

SYSTEM AND TASK CALLS

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

You begin each system call with the mnemonic .SYSTM, which assembles as a JSR @ 17 instruction. This instruction enables the system to respond to your command.

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 location 16 (the User Stack Pointer) is the default value.

There are two basic types of system calls: those which require a channel number, and those which don't. Channel numbers are described below.

Many system calls require you to include a byte pointer to a specific filename. When you use a byte pointer to a disk filename, you can include a directory specifier as well, if you have initialized the directory. All DOS system calls are summarized in Table 3-1, below.

A task call resembles a system call, with these exceptions:

1. You enter no .SYSTM mnemonic before the task command word.

2. RDOS executes task calls in user address space, not in system space.

3. 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 this chapter are all system calls.

See Chapter 5 for more detail on the differences between system and task calls.

STATUS ON RETURN FROM SYSTEM CALLS

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 are preserved. For certain calls, the system returns information in AC0, AC1, and/or AC2.

By default, on return from any system call, AC3 contains the contents of location 16_8 (the USP), unless you specified N3SAC3 in the RLDR command line, as shown below. On an error return, DOS uses AC2 to return a numeric error code. Appendix A lists the error codes.

AC3 on Return

If you loaded your program with module:	then (upon return from call) AC3 contains contents of:
NSAC3 (any machine; used by default)	USP (location 16_8).
N3SAC3 (NOVA3s and microNOVAs only)	Frame Pointer register.

Table 3-1. System Command List

.APPEND	Open a file for appending.
.BOOT	Bootstrap a new system. Chapter 4.
.BREAK	Interrupt the current program; save the current state of memory in save file format.
.CCONT	Create a contiguously orgainzed file with all data words zeroed.
.CONN	Create a contiguously organized file with no zeroing of data words.
.CDIR	Create a subdirectory.
.CHATR	Change file attributes.
.CHLAT	Change link access attributes.
.CHSTS	Get the status of the file currently-opened on a specified channel.
.CLOSE	Close a file.
.CRAND	Create a random file.
.CREAT	Create a random file.
.DDIS	(Provided for RDOS compatibility.)
.DEBL	(Provided for RDOS compatibility.)
.DELET	Delete a file.
.DIR	Change the current directory.
.DUCLK	Define a user clock. Chapter 5.
.EOPEN	Open a file for reading and writing by one user only.
.ERTN	On an error, return from program and describe error.
.EXEC	Swap or chain in a new program. Chapter 4.
.FGND	Check the levels of the current program. Chapter 4.

Table 3-1. System Command List (continued)

.GCHAR	Get character from the console.
.GCHN	Get the number of a free channel.
.GCIN	Get the operator input console name.
.GCOUT	Get the operator output console name.
.GDAY	Get today's date.
.GDIR	Get the current directory name.
.GHRZ	Examine the real time clock, Chapter 5.
.GPOS	Get the current file pointer.
.GSYS	Get the name of the current operating system.
.GTATR	Get file attributes.
.GTOD	Get the time of day.
.IDEF	Identify a user device.
.INIT	Initialize a device or a directory.
.IRMV	Remove a user device, Chapter 6.
.LINK	Create a link entry.
.MDIR	Get the logical name of the master device.
.MEM	Determine available memory.
.MEMI	Change NMAX.
.MTDIO	Perform free format I/O on tape.
.MTOPD	Open a mag tape for free format I/O.
.ODIS	Disable keyboard interrupts for this console.
.DEBL	Enable keyboard interrupts for this console.
.OPEN	Open a file for reading and writing by one or more users.
.OVLOD	Load a user overlay into memory. Chapter 4.
.OVOPN	Open a user overlay file. Chapter 4.
.PCHAR	Write a character to the console.
.RDB	Read one or more disk blocks.
.RDL	Read a line.
.RDR	Read a random record.
.RDS	Read sequential bytes.
.RDSW	Read the console switches.
.RENAM	Rename a file.
.RESET	Close all files.
.RLSE	Release a directory or device.
.ROPEN	Open a file for reading only by one or more users.
.RSTAT	Get a resolution file's statistics.
.RTN	Return from a program to a higher-level program. Chapter 4.
.RUCLK	Remove a user clock. Chapter 5.
.SDAY	Set today's date.
.SPDA	(Provided for RDOS compatability.)
.SPEA	(Provided for RDOS compatability.)
.SPKL	(Provided for RDOS compatability.)
.SPOS	Set the current file pointer.
.STAT	Get a file's statistics.
.STOD	Set the time of day.
.TUOFF	(Provided for RDOS compatability.)
.TUON	(Provided for RDOS compatability.)
.ULNK	Delete a link entry.
.UPDAT	Update the current file size.
.WRB	Write one or more 256-word blocks to disk.
.WRL	Write a line.
.WRR	Write a random record.
.WRS	Write sequential bytes.

I/O CHANNEL NUMBERS

Before you can access a file for I/O, you must give it an I/O channel number in your open call. While the file is open, it retains this channel number, and you must access it via the number instead of the filename. When you close the file, the number is released. The number immediately follows the call word in your program; if the channel number is n, the I/O calls for a file could run:

```
open n
  .
  .
  .
file reads/writes n
  .
  .
  .
close n
```

The maximum number of channels for a program is 77_8.

For a single-task program, RLDR allots eight I/O channels, numbered 0 through 7. Usually, this is enough. If you want to specify more channels, use either the RLDR /C switch, or the assembler pseudo-op .COMM TASK.

Selecting a Channel

There are two ways to assign a channel number to a file: either directly when you open, e.g.,

```
.OPEN 3
```

or via AC2. If you specify number 77 on your open, DOS will open the file on the channel number contained in the right byte of AC2. To open a number above 77 (assuming that your program permits one), you must open on 77 and pass the number in AC2. The major advantage to opening on 77 is that you can use system call .GCHN to find a free channel for your open.

.GCHN returns the number of a free channel in AC2, and you can give this number a name, and use the name for all I/O to the file. This method can provide a free channel for file I/O (unless all channels are in use, or another task has opened the channel between the .GCHN call and this task's OPEN attempt). Here is an example:

```
   .
.SYSTM
.GCHN
JMP ER
STA  2, FILE1        ;STORE THIS CHANNEL
   .                 ;NUMBER UNDER "FILE1".
   .
   .
LDA  2, FILE1
.SYSTM
.OPEN  77            ;OPEN "FILE1" FOR ANYTHING.
JMP ER
.SYSTM
.WRS 77              ;WRITE TO "FILE1".
   .
   .
.SYSTM
.GCHN
JMP ER
STA  2, FILE2        ;STORE NUMBER UNDER "FILE2".
.SYSTM
.APPEND 77           ;OPEN "FILE2" FOR APPENDING.
JMP ER
   .
   .
   .
```

CAPSULE COMMAND SUMMARY

As you write different programs for your application, you will use certain system calls quite frequently, and others rarely or not at all. The following table attempts to summarize the most useful calls, in the sequence which you might use in a program. It gives the call name, format, accumulator data, and possible error code. It assumes that you will use the CLI to create and initialize directories, and to execute mag tape I/O, and that your program won't do such esoteric things as alter file attributes, create link entries, or manage a multitask environment. Of course, you can do all of these things via DOS calls if you choose.

The summary also assumes a single-task environment; it does not cover multitasking (Chapter 5). Many commands not given below are included in the rest of this chapter (or manual). Each call has the form:

```
.SYSTM
call name
error return to program
normal return
```

For example:

```
                .TXTM 1
                .
                .
                LDA 0,BTPTR
                .SYSTM
                .DIR
                JSR ERROR
                .
                .
                .
BTPTR:  .+1*2
        .TXT "DP1:SUBDIR"
ERROR:  .SYSTM
        .ERTN
        JMP .+1
```

Each file I/O command requires a channel number, as noted. The term "btptr" means byte pointer.

Table 3-2. Common Call Summary

Call	Purpose	Required Input
.CRAND	Create a random file.	AC0:btptr to filename. All filenames must be terminated by a carriage return, form feed, or a null.
.CCONT	Create a contiguous file.	AC0:btptr to filename. AC1:number of disk blocks for the file.
.OPEN n	Open a file for I/O on channel n.	AC0:btptr to filename. AC1:characteristic disable mask. You can specify the system default mask (normal procedure) by passing 0 via a SUB 1,1 instruction before the .OPEN.
.APPEND n	Open a file for appending, on channel n. Set position for writing at the end of the file.	AC0:btptr to filename. AC1: characteristic disable mask. As with .OPEN you can use the default mask by inserting a SUB 1,1 before the .APPEND.
.RDL n	Read an ASCII line on channel n. Counterpart of .WRL.	AC0:btptr to 133 byte area. AC1 returns the count of characters read.
.RDS	Read sequentially from the file .OPENed on channel n. Sequential mode is required for binary data.	AC0:btptr to starting address of data. AC1:number of bytes to be read.
.WRL n	Write an ASCII line to the file OPENed on channel n. Writing begins at start of file if you .OPENed the file; at end if you .APPENDed the file. Limit is 132 characters, terminated by a CR, null, or form feed.	AC0:byte pointer to user area which holds the ASCII line.
.WRS n	Write sequential bytes to the file on channel n. See .WRL for position information.	AC0:btptr to starting address of data. AC1:number of bytes to be written.
.WRB n, .RDB n	Direct-block I/O calls. Write or read a series of disk blocks to or from the random or contiguous file on channel n.	AC0:starting source address for the block write or read. AC1:starting relative block number in the series. AC2:left byte-number of 256-word blocks to be written or read to the file.
.CLOSE n	Close the file opened on channel n. DOS then updates the file's UFD information. (.ERTN and .RTN close all channels in the current program.)	
.DELET	Delete a file.	AC0:btptr to filename.

The following calls control NMAX (the top of user NREL+1), execute and return from program swaps or chains, and load overlays.

.MEM	Return the current program's NMAX value in AC0, and the value of the Highest Memory Address available for user programs in AC1.	
.MEMI	Raise NMAX by the value entered in AC0, or lower NMAX by the value entered in two's complement in AC0. DOS returns the new value in AC1.	AC0: new NMAX
.ERTN or .RTN	Close all channels in the current program and return to (resume execution of) the next higher-level program (usually the CLI). .ERTN returns an error code in AC2; if return is to the CLI, it also prints an error message on the console.	
.OVOPN n	Open user overlays, for reading, on channel n. Before your program can use overlays, you must open them on a channel. You close the channel via a .CLOSE n.	AOS:btptr to overlay filename, including .OL extention.
.OVLOD n	Load an overlay from the overlay file opened on channel n into its reserved memory node.	AC0:overlay descriptor. AC1:conditional load flag.

If your program takes the error return from any of the calls above, AC2 will usually contain one of the following error codes:

AC2	Mnemonic	Meaning
0	ERFNO	Illegal channel number (legal range: 0 through 377$_8$.
1	ERFNM	Illegal filename (only alphanumeric or $ characters are permitted).
3	ERICD	Illegal command for device (for example, trying to read from the line printer).
6	EREOF	End of file detected while reading; or attempt to write beyond the end of a contiguous file.
7	ERRPR	The file is read-protected.
10	ERWPR	The file is write-protected.
11	ERCRE	The file already exists.
12	ERDLE	The file (directory) does not exist.
13	ERDEI	The file cannot be deleted because it has the permanent attribute.
15	ERFOP	The file hasn't been opened.
21	ERUFT	This channel is in use.
22	ERLLI	Line limit (132 characters) exceeded.
26	ERMEM	Attempt to allocate more memory than is available.
27	ERSPC	Current disk file space exhausted.
33	ERRD	Attempt to read or write into system space.
36	ERDNM	Device not in system.
37	EROVN	Illegal overlay number.
40	EROVA	File not accessible by direct-block I/O.
47	ERSIM	Simultaneous reads or writes attempted to same QTY/ALM line.
52	ERIDS	Illegal directory specifier.
66	ERDNI	Directory not initialized.
101	ERDTO	10-second disk timeout occurred.
104	ERSRR	A short receive request terminated the MCA transmission.
106	ERCLO	QTY/ALM/MCA output terminated by channel close.

DEVICE AND DIRECTORY COMMANDS

This section describes the DOS system commands which pertain to opening and releasing disks, mag tape drives, and disk directories; it also covers disk partition and directory creating commands. It includes these commands:

.INIT	Initialize a directory/device.
.DIR	Select a different current directory.
.RLSE	Release a directory/device.
.GDIR	Get the current directory's name.
.CDIR	Create a subdirectory.
.GSYS	Get the current RDOS system's name.
.MDIR	Get the master directory's name.

Commands for individual files are covered in the following section, File Maintenance.

DOS can support many directory devices simultaneously. During SYSGEN, you configured your system for specific disk and tape devices, and you can address any of these by a 3- or 4-character code as shown in Chapter 2.

Initialize a Directory or Device (.INIT)

Your program can initialize devices and directories via the system command .INIT.

If AC1 contains anything but -1 when you invoke .INIT, a partial initialization of the device or directory results; this makes all files in the directory available to the system software. Partial initialization of a magnetic tape rewinds the tape and resets the tape file pointer to file zero. 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 to signify the logical end-of-tape. You lose all previous files on that tape. Full initialization of a disk overwrites all files on it, and builds a virgin SYS.DR and MAP.DR. DOS treats full initialization of a directory as a partial initialization.

Required input:

AC0 - Byte pointer to a directory/device specifier character string terminated by a null byte.

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

Format:

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

Possible errors:

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	Insufficient number of Device Control Blocks (DCBs), specified at SYSGEN time.
51	ERNMD	Insufficient number of Device Control Blocks specified at SYSGEN.
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.
121	ERFMT	Disk format error.

Change the Current Directory (.DIR)

When you bootstrap a DOS system, the directory which holds the system becomes the current directory. The .DIR command selects a different current directory -- if the new current directory hasn't been initialized, .DIR will initialize it.

After you .DIR to a directory, you can access all files in it without using directory specifiers.

.DIR is not mandatory for file access in nonmaster directories, because DOS permits directory specifiers in all filename arguments to system commands. For example, both of the following examples could access MYFILE in DP1, from master directory DP0 if DP1 had been initialized:

```
                .
                .
1.  LDA 0,      .MYFILE
    .SYSTM
    .OPEN
                .
                .
.MYFILE:            .+1*2
                    .TXT "DP1:MYFILE"
                .
                .
2.  LDA 0,          .DP1
    .SYSTM
    .DIR
                .
                .
    LDA 0,          .MYFILE
    .SYSTM
    .OPEN
                .
                .
.DP1:           .+1*2
                .TXT "DP1"
.MYFILE:        .+1*2
                .TXT "MYFILE"
```

In the first example, DP0 remains the current directory; in the second, DP1 becomes the current directory.

Required input:

AC0 - Byte pointer to directory name string (must be terminated by a null).

Format:

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

If DOS takes the error return, the current directory definition remains unchanged.

Possible errors:

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.
121	ERFMT	Disk format error.

Release a Directory or Device (.RLSE)

This command dissociates a directory or device from the system, and prevents further I/O with it.

You should always release a disk via either the CLI command RELEASE (or .RLSE) before removing it from the unit. (If you forget, disk integrity may be destroyed, as described under "Releasing the System" in Chapter 7 or 8.) You must also close all files within a directory before you can release it. Release of a master directory releases all directories. The master directory is the directory which holds the current DOS system. You can get its name by using the .MDIR call or the MDIR command.

Required input:

AC0 - Byte pointer to a directory or device specifier.

Format:

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

Possible errors:

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 tape unit containing an open file.

Get Current Directory Name (.GDIR)

This call returns the name of the current directory or device. This name is followed by a null (e.g., DP0); it doesn't include the names of superior directories, or colon specifiers. For the current directory DP0:PART2, it would return PART2.

Required input:

AC0 - Byte pointer to 15_8-byte area to receive the current directory/device name.

Format:

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

The first 14_8 bytes will contain the name and extension (with trailing nulls, if necessary); byte 15_8 will contain a null terminator.

Possible error:

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

Create a Directory (.CDIR)

This call creates an entry for a directory name in the current partition's system directory (SYS.DR). The directory will automatically receive the .DR extension. If the entry already exists, DOS will return the error code ERCRE.

Required input:

AC0 - Byte pointer to the directory name (directory specifiers permitted).

Format:

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

Possible errors:

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 .SV extension, and a null terminator.

Required input:

AC0 - Byte pointer to 15_8-byte area.

Format:

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

Possible error:

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

Get the Name of the Master Directory (.MDIR)

.MDIR returns the name of the master directory, which holds the current DOS system.

Required input:

AC0 - Byte pointer to 15_8 byte area to receive the directory name.

Format:

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

The first 14_8 bytes will contain the name (with trailing nulls); byte 14_8 will contain a null terminator.

Possible error:

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

FILE MAINTENANCE COMMANDS

The commands in this section relate to individual files; they enable you to create, delete, set position, and check the status of files. The file maintenance commands are:

.CCONT	Create a contiguous file with data words zeroed.
.CONN	Create a contiguous file with no data words zeroed.
.CRAND	Create a random file.
.DELET	Delete a file.
.RENAM	Rename a file.
.GPOS	Get the current file pointer.
.SPOS	Set the current file pointer.
.STAT	Get a file's status.
.RSTAT	Get a link entry's resolution file status.
.CHSTS	Get a channel's file information.
.UPDAT	Update an open file's size information.

Each file maintenance command requires you to specify the file name(s) by means of a byte pointer to the file name. In the byte pointer, bits 0-14 contain the word address which holds or will receive the first byte. Bit 15 indicates which half: 0 is left, 1 is right.

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

```
          .TXTM 1
            .
            .
            .
BPTR:   .+1*2
        .TXT "MYFILE.SR"
```

File names can include 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 call creates a contiguously-organized file with all data words initialized to zero. If the file's name exists as a link entry, and if no resolution file exists for this link entry, DOS will create a contiguous resolution file. If the file already exists, DOS returns the error code ERCRE.

Required input:

AC0 - Byte pointer to the file name.
AC1 - Number of disk blocks in the file.

Format:

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

Possible errors:

AC2	Mnemonic	Meaning
1	ERFNM	Illegal file name.
11	ERCRE	File already exists.
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)

.CONN creates a contiguously-organized file; it is faster than .CCONT because DOS doesn't need to zero the data words. If the file's name exists as a link entry, and if no resolution file exists for this link entry, DOS will create a contiguous resolution file. If the file already exists, DOS returns the error code ERCRE.

Required input:

AC0 - Byte pointer to filename.
AC1 - Number of disk blocks in the file.

Format:

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

Possible errors:

AC2	Mnemonic	Meaning
1	ERFNM	Illegal file name.
11	ERCRE	File already exists.
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 (SYS.DR). If the file's name exists as a link entry, and if no resolution file exists, DOS will create a random resolution file. If the file already exists, DOS will return the error code ERCRE.

Required input:

AC0 - Byte pointer to the file name.

Format:

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

Possible errors:

AC2	Mnemonic	Meaning
1	ERFNM	Illegal file name.
11	ERCRE	File already exists.
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 non-master device.
101	ERDTO	Ten-second disk timeout occurred.

Delete a File (.DELET)

Use this command to delete a file and its entry in the system file directory. Do not delete link entry names with 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 DOS returns error ERDE1), or 2) a resolution file doesn't exist (ERDLE returned).

Required input:

AC0 - Byte pointer to filename.

Format:

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

Possible errors:

AC2	Mnemonic	Meaning
1	ERFNM	Illegal file name.
12	ERDLE	File does not exist.
13	ERDE1	File is permanent.
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	Link access not allowed (N attribute).

Rename a File (.RENAM)

This call renames a file. You may rename a file in a different directory, as long as you use the same directory specifier in both the current name and new name.

Required input:

AC0 - Byte pointer to the current filename.
AC1 - Byte pointer to the new name.

Format:

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

After a normal return, the old name no longer exists in the file directory.

Possible errors:

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 in 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's Directory Status (.STAT/.RSTAT)

Use either of these system calls to get a copy of the current directory status information for a file. These calls write 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, the information returned is a snapshot of the UFD as it existed on disk at the time of the most recent .CLOSE or .UPDAT.

Use system call .STAT to return the UFD of a file. Use .RSTAT to find the UFD of a link's resolution file. .RSTAT and .STAT have the same effect on a nonlink file.

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

Offset or Displacement	Mnemonic	Content
00000-000004	UFTFN	File name (ASCII file number for open tape file).
000005	UFTEX	Extension.
000006	UFTAT	File attributes.
000007	UFTLK	Link access attributes.
000010	UFTBK	Number of the last block in the file (i.e., block count - 1).
000011	UFTBC	Number of bytes in the last block.
000012	UFTAD	Starting logical block address of the file (the random file index for random files).
000013	UFTAC	Year/day last accessed.
000014	UFTYD	Year/day created, update 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	Use count (1B0 = .EOPEN or .APPEND or .ROPEN; 1B1 = .OPEN)
000021	UFTDL	DCT link; device code.

If you issue .STAT to a link entry, DOS returns the link's UFD. In a link UFD, words 7 and 14_8 have mnemonics UFLAD and UFLAN; words $7\text{-}13_8$ and $14\text{-}21_8$ contain the link's alternate directory specifier (if any) and an alias (if any), respectively.

Required input:

AC0 - Byte pointer to file name string.
AC1 - Starting address of 22_8 word UFD data area.

Format:

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

Possible errors:

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)

.CHSTS returns a copy of current directory status information for whatever file is currently open on a specified channel. DOS returns directory status information 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 by you) of this channel. For example, .CHSTS would return the status after a .WRL, whereas .STAT/.RSTAT would show status, on disk, as of the last open, close, or update.

Required input:

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

Format:

```
.SYSTM
.CHSTS n          ;n is the file's channel number
error return
normal return
```

Possible errors:

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, attributes, and other information. Specifically, this call updates information in UFTBK and UFTBC in the disk UFD for the file opened on a specified channel, and it writes all system buffers to ensure that the file contains all information that your program has 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 may have inaccurate size information in its UFD; if so you will be unable to read new data. By .UPDATing the file frequently, you keep its UFD current and minimize the amount of data which could be lost.

Format:

```
.SYSTM
.UPDAT n        ;n is the file's channel number
error return
normal return
```

Possible errors:

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

This section describes the following calls:

.CHATR	Change the attributes of the file opened on channel n.
.GTATR	Get the attributes or characteristics of the file opened on channel n.

Note that these calls work only on an open file. For link commands, see the next section.

Change File Attributes (.CHATR)

This command changes the access attributes of an open file (or the resolution entry attributes, as viewed from a link entry), according to the contents of AC0.

When you create a file, it has no attributes. If a link user or a user who has opened via .ROPEN issues .CHATR, DOS temporarily changes his/her copy or the file attributes until he/she closes the file; however, the true resolution entry attributes persist. You must open a file (.EOPEN or .OPEN) before you can change its attributes via .CHATR.

Note that DOS provides two special attributes bits; you can use these to define your own unique file access specifications.

Format:

```
.SYSTM
.CHATR n        ;n is the file's channel number
error return
normal return
```

Required input:

AC0 - An attribute word which contains bits set according to the attributes you want. Set the contents of AC0 according to the following bit/attribute relationships:

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 you set this bit.
1B2	S	ATSAV	Save file (core image file).
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, DOS assigns them when you create a file; you cannot change them.

Bit	Characteristic	Mnemonic	Meaning
1B3	L	ATLNK	Link entry.
1B5	Y	ATDIR	Directory file (except MAP.DR).
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:

AC2	Mnemonic	Meaning
0	ERFNO	Illegal channel number.
14	ERCHA	Illegal attempt to change file attributes (file has A attribute).
15	ERFOP	No file open on this channel.
101	ERDTO	Ten-second disk timeout occurred.

Get the File Attributes and Characteristics (.GTATR)

Use this command to obtain the attributes or device characteristics of a file.

Format:

```
.SYSTM
.GTATR n        ;n is the file's channel number
error return
normal return
```

When DOS returns, AC0 will contain the file attributes. See the .CHATR command for a description of the bit positions that specify attributes.

AC1 will contain the device characteristics of the file. These pertain to files on reserved devices, e.g., $TTO. These do not reflect the characteristic disable mask supplied when the file was opened. Use this bit/characteristics table to interpret the bit configuration returned in AC1:

Bit	Mnemonic	Meaning
1B1	DCC80	80-column device.
1B2	DCLTU	Device changes lowercase ASCII to uppercase.
1B3	DCFFO	Device requiring form feeds on opening.
1B4	DCFWD	Full word device (reads or writes more than a byte.)
1B5	DCSPO	Spoolable device.
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 are disabled for this keyboard input device.

Bit	Mnemonic	Meaning
1B11	DCTO	Teletypewriter output device or equal leader and trailer for $TTP and $PTP.
1B12	DCCNF	Output device without form feed hardware.
1B13	DCIDI	Input device requiring operator intervention.
1B14	DCCGN	Output device without tabbing hardware.
1B15	DCCPO	When file is $TTR/$TTP: output device requiring leader and trailer.

For multiplexor mask bits, see "Multiplexors" in Chapter 2.

Possible errors:

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

LINK COMMANDS

As we described in Chapter 2, DOS permits you to link files in one directory to files in other directories. Either directory can be a diskette or user directory. The link commands are:

.LINK	Create a link entry.
.UNLK	Delete a link entry.
.CHLAT	Change the link access attributes of a file.

Create a Link Entry (.LINK)

This call creates a link entry in the current directory to a file in the same or another directory. This link entry may or may not have the same name as the resolution file; 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. Your program can alter the link access rights (but not the file access rights) of any nonlink file by using the .CHLAT call.

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

Link	Character String	Meaning to DOS
LFE.SV	— —	Create link entry LFE.SV in the current directory; link it to resolution file LFE.SV on the current directory's parent directory.
NLFE.SV	DP1:LFE.SV	Create link NLFE.SV in the current directory; link it to resolution file LFE.SV in DP1.

Required Input:

AC0 - Byte pointer to link entry name string.
AC1 - Zero if the link and resolution file have same name, and if the resolution file is in the parent diskette. Byte pointer to the name string if the link entry has an alias name, or is not on the parent directory. You can omit a directory specifier from the resolution file name if the resolution file is on the link entry's parent directory.

Format:

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

Possible errors:

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 to which the link entry name points. This call does not delete other links of the same name in other directories. You must be sure that the link entry you are deleting does not also exist between other links and the resolution entry; if it does, you will not be able to resolve these more remote links after this deletion.

Required input:

AC0 - Byte pointer to the link entry name string.

Format:

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

Possible errors:

AC2	Mnemonic	Meaning
1	ERFNM	Illegal file name.
12	ERDLE	File does not exist.
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

This section describes the calls your program can use to write data to, and read data from, an existing, open file. It begins by describing the five I/O modes available, and proceeds to explain the calls which open and close a file.

It then covers the calls you can use to change position in a file, and finally lists the different writing/reading calls themselves.

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.

Remember that a file can be a device (e.g., $TTI, console input; or QTY:xx, multiplexor line) or a disk file (e.g., MYFILE.SR), which can include a directory specifier (e.g., DP1:MYFILE.SR) if you have initialized the directory.

These are the file I/O calls:

.OPEN n	Open a file for I/O on channel n.
.EOPEN n	Open a file for exclusive writing on channel n.
.ROPEN n	Open a file for reading only on channel n.
.APPEND n	Open a file for appending on channel n.
.GCHN	Get the number of a free channel.
.CLOSE n	Close the file on channel n.
.RESET	Close all files.
.GPOS n	Get the position of the file pointer.
.SPOS n	Set the position of the file pointer.
.RDL n	Read an ASCII line from the file on channel n.
.WRL n	Write an ASCII line to the file on channel n.
.RDS n	Read sequential data from the file on channel n.

.WRS n	Write sequential data to the file on channel n.
.RDR n	Read a 64-word record from the file on channel n.
.WRR n	Write a 64-word record to the file on channel n.
.RDB n .WRB n	Read (or Write) a series of disk blocks from or to the file or channel n, without a system buffer.
.MTOPD n	Open a mag tape or cassette file on channel n for free-form I/O.
.MTDIO n	Write or read data to or from the mag tape file on channel n in free form.

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

DOS provides five basic modes for reading and writing files:

- line
- sequential
- random record
- direct block
- free form (tape)

This section presents the calls for these modes in order.

You will generally use line and sequential* mode for ASCII character strings and binary files, respectively. Random record mode allows you to read or write 64-word records.

Direct-block I/O allows you to transfer one or more disk blocks without a system buffer. Free form I/O allows you to directly control a mag tape drive.

In line mode, the system assumes that the data you want to read or write consists of ASCII character strings, terminated by either a carriage return, a form feed, or a null character. DOS processes file data 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, it ignores line feeds on paper tape input devices and supplies them after carriage returns to all paper tape output devices. Furthermore, reading and writing never require byte counts, since reading continues until DOS reads a terminator and writing proceeds until you write a terminator. The line mode commands are Read a Line (.RDL) and Write a Line (.WRL).

The second mode is the unedited sequential mode. In this mode, DOS transmits data exactly as it reads it from or writes it to the file or device. You can use this mode for processing data byte-by-byte. 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 your last line or sequential mode call, or .SPOS call. The first read or write occurs at the beginning of the file, unless your program opened the file for appending.

The third mode, random record, permits random access to fixed-length records within random or contiguous disk files. The fixed length of a random record is 100_8 words. The random calls are .RDR and .WRR.

The fourth 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. DOS uses sequential memory locations in the transfer, and it transfers only 512-byte blocks of data between memory and disk. You can transfer only an unbroken series of relative block numbers; i.e., you may process the third, fourth, and fifth blocks in a file 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 free form blocks of data to magnetic tape. With free form I/O, you can read or write from one to 4096-word data records, you can space forward or backward through one to 4096 data records or to the start of a new data file, and you can read the transport status word. To use free form I/O you must open a file via .MTOPD, and direct its operation via .MTDIO. You cannot mix .MTDIO with .WRL, or .WRS on the same tape drive.

*The DOS System Library contains a module to speed up line and sequential mode operations. This module is called the Buffered I/O Package, and is described in that Application Note.

Open a File (.OPEN)

Before your program can issue other I/O commands, it must link a file to a DOS channel number. .OPEN links a file with a channel number and makes the file available to anyone for both reading and writing. The .OPEN command does not guarantee exclusive use of the file; other program tasks may also have opened the file via .OPEN and modified its contents. Every task using a file must close it before anyone can delete or rename it. In DOS, there is no command to reduce the size of a file. This means that files never shrink, and they maintain space for all material written to them by any task. If you want to remove redundant or useless material from a file, you can either edit it with a text editor utility, or you can overwrite the useless data with nulls or new material, using file position and system write calls.

Required input:

AC0 - Byte pointer to the filename
AC1 - Characteristic disable mask. For every bit you set in the mask word, DOS disables the corresponding device characteristic for the duration of the .OPEN. (See .GTATR in the File Attributes Commands section of this chapter.)

For example, if you want to read an ASCII tape without parity checking from the paper tape reader, you can disable checking and the operator intervention message by the following:

```
        LDA  0,READR
        LDA  1,MASK
        .SYSTM
        .OPEN  3
          .
          .
          .
READR:  .+1*2
        .TXT "$PTR"

MASK:   DCPCK+DCIDI     ;DISABLE PARITY
                        ;CHECKING AND OPER.
                        ;INTERVENTION MESSAGE.
```

To use system mnemonics like mask and error words, you should assemble your program with the macroassembler, and the assembler's symbol table file must include PARU.SR.

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 .OPEN call.

Format:

```
        .SYSTM
        .OPEN n         ;n becomes the channel
                        ;number of the file
                        ;until n is closed.
        error return
        normal return
```

Note that DOS will interleave line printer output if multiple tasks in the same program open and write to the printer.

If the file opened requires leader, DOS will output it on the .OPEN. If the file opened requires intervention, DOS will display the message

LOAD filename, STRIKE ANY KEY.

Possible errors:

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 gives you exclusive write access to a file. Thus only you can modify a given file when you open it via .EOPEN, although other users may gain read access to this file via .ROPEN.

Required input:

AC0 - Byte pointer to file name.
AC1 - Characteristic disable mask.

Format:

```
.SYSTM
.EOPEN n        ;n is the file's channel number
error return
normal return
```

Possible errors:

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	Attempt to open a file which is already open.

Open a File for Reading Only (.ROPEN)

This call opens a file for reading only. Your program can gain read-only access to a file which is 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 anyone can delete or rename it.

Required input:

AC0 - Byte pointer to file name
AC1 - Characteristic disable mask

Format:

```
.SYSTM
.ROPEN n        ;n is the file's channel number
error return
normal return
```

Possible errors:

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	ERDLE	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.

If your program tries to read a file which you have opened for appending, DOS will return error code EREOF (end-of-file).

Required input:

AC0 - Byte pointer to the filename
AC1 - Device characteristic disable mask

Format:

```
.SYSTM
.APPEND n       ;n is the file's channel number
error return
normal return
```

On a disk, DOS opens the file, and appends whatever you write to that file. On a magnetic tape device, DOS opens the tape file and reads to the end-of-file (EOF); it then writes from that point. On a line printer, DOS opens the printer without a form feed.

Possible errors:

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 (opened by .EOPEN).
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 tape 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 returns (in AC2) the number of a free channel. Your program can then use AC2 to open a file 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; reissue the call .GCHN, to obtain another free channel.

Format:

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

Upon a normal return, DOS returns the free channel number in AC2.

Possible error:

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 delete it, or release its directory or device. When you close a file, its channel number becomes available for other I/O. The calls .RTN, ERTN, .BREAK or .RESET automatically close all channels.

Format:

```
.SYSTM
.CLOSE n        ;Close channel n
error return
normal return
```

Whenever you .CLOSE a high-speed punch file, DOS will output the required trailer.

Possible errors:

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 after writing any partially-filled system buffers. You should issue .RESET in a multitask environment only when no other task is using a channel.

Format:

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

Possible error:

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

Get the Current File Pointer (.GPOS)

Use this call to determine the next character position within a file where program writes or reads will occur. DOS indicates a relative character position within a file by a double-precision byte pointer. This is a two-word byte pointer containing the high-order portion of the byte address in AC0, and the low-order portion of the byte address in AC1. Bit 15 of AC1 indicates the byte selection (left to right):

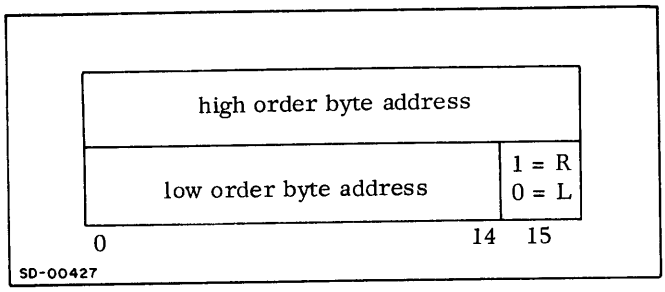


Figure 3-1. Double-Precision Byte Pointer

Format:

```
.SYSTM
.GPOS n         ;n is the file's channel number
error return
normal return
```

DOS returns zero if you open a nondisk file on channel n.

Possible errors:

AC2	Mnemonic	Meaning
0	ERFNO	Illegal channel number.
15	ERFOP	No file is open on this channel.

Set the Current File Pointer (.SPOS)

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

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

If you set the file pointer beyond the end of file, DOS automatically extends the length of the file. If the file is contiguous - hence cannot be extended - DOS will take the error return, and pass ERSCP in AC2.

Required input:

AC0 - High-order portion of byte pointer.
AC1 - Low-order portion of byte pointer.

Format:

```
        .SYSTM
        .SPOS n          ;n is the file's channel number
        error return
        normal return
```

Possible errors:

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

Read a Line (.RDL)

.RDL reads an ASCII line from a file to your specified area. AC0 must contain a byte pointer to the starting byte address within user memory into which DOS will read the line. This area should be 133 bytes long.

Reading terminates normally after DOS has read either a carriage return, form feed, or null, and transmitted it to your program. The system stops reading and takes the error return if it transmits 133 characters without detecting a carriage return, form feed, or null, or upon detection of a parity error, or end-of-file.

If DOS is reading through a multiplexor, it also terminates reading if it reads an ESC.

If the file you are reading from is the keyboard ($TTI, $TTI1), keyboard controls work as usual (unless you have masked DCKEY in AC1 -- see .GTATR). Rubout deletes the preceding character, and backslash (SHIFT-L) deletes the preceding line, from the keyboard stream. DOS echoes all printing characters and ignores line feeds. You can indicate an end of file by pressing CTRL-Z. Note that when you are reading from a multiplexed line, ESC also indicates an end of file.

DOS will always return the number of bytes read (including the carriage return, form feed, or null) in AC1. If the read terminates because of a parity error, DOS stores the character having incorrect parity. as the last character read and clears the parity bit. You can always compute the byte pointer to the bad character as (AC0) + (AC1)-1. (Note: (AC0) means the contents of AC0.)

Required input:

AC0 - Byte pointer to receiving buffer.

Format:

```
        .SYSTM
        .RDL n           ;Read from channel n
        error return
        normal return
```

After a normal return, AC1 will contain the number of bytes read.

Possible errors:

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 open.
22	ERLLI	Line limit (133 nonterminator characters) exceeded.
24	ERPAR	Parity error (tape - possibly dirty heads).
30	ERFIL	File read error (bad tape; possibly dirty heads).
33	ERRD	Attempt to read into system area.
34	ERDIO	File accessible by direct block I/O only.
47	ERSIM	Simultaneous reads from the same multiplexor (ALM/QTY) line.
101	ERDTO	Ten-second disk timeout occurred.
106	ERCLO	Channel closed by another task.

Write a Line (.WRL)

.WRL is the counterpart of .RDL; it writes an ASCII line to the file open on the specified channel. AC0 must contain a byte pointer to the starting byte address within user memory from which characters will be written.

When you have opened a file via .OPEN or .EOPEN, writing begins at the beginning of the file if this is the first write since the open. For second and subsequent writes after the open, writing begins at the file position pointer, which is altered by every I/O access. Writing begins at the end of the file if you open it via .APPEND. Normally, the system stops writing when it detects a null, a carriage return, or a form feed. Abnormally, it stops writing after transmitting 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 your area to the file. The null terminator does not force a carriage return or line feed. A carriage return generates a line feed upon output if the device characteristics so dictate.

Required input:

AC0 - Byte pointer to starting byte address.

Format:

```
        .SYSTM
        .WRL n          ;Write to the file on channel n
        error return
        normal return
```

Possible errors:

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.
47	ERSIM	Simultaneous writes to the same multiplexor (ALM/QTY) line.
101	ERDTO	Ten-second disk timeout occurred.
106	ERCLO	Channel closed by another task.

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 on a .RDS. The translation assumes the keypunch codes shown in Appendix B.

A .RDL terminates upon the first trailing blanks unless your .OPEN command suppressed DCSTB, thus causing DOS to transfer all 80 characters. If DOS transfers all 80 characters, it will append a carriage return as the eighty-first character, unless your .OPEN command suppressed DCC80 (allowing DOS to process a maximum of 72 characters). The system replaces each illegal character with a backslash.

In .RDL calls, DOS ignores all columns following the EOF. The card reader driver permits an unlimited amount of time to elapse until it reads the next card, thus permitting the operator to correct pick errors, or insert new card files. The card reader driver employs double buffering, and you will lose at least one card image if you close prematurely; therefore your program must wait until DOS reads the last card or end-of-file to close $CDR.

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

If you issue .RDS (see below), DOS reads the card in image binary. It uses each two bytes to read a single column, packing them as follows:

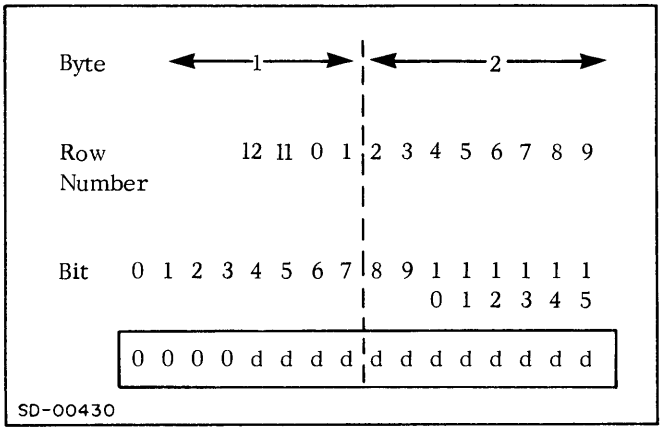


Figure 3-2. Image Binary Card Reading

Each "d" will be 1 for every punched hole in the column. In .RDS, you signify an end-of-card (EOC) by a byte pair containing the word 100000. Thus, to read two entire 80-column cards, one card at a time, you would issue two successive .RDS calls for 162 bytes each. If you requested only 160 bytes for each read, the second .RDS would return the first end-of-card word, and the first 79 columns of the second card.

Read Sequential (.RDS)

In the sequential mode, DOS transmits data exactly as it reads it to or writes it from a file. You can use this mode for binary data.

The .RDS call tells DOS to read data exactly as it is in the file, unless it is reading from the system console. When reading sequentially from a system console, DOS sets the parity bits to zero. Note that DOS does not recognize CTRL-Z from the console as an end-of-file character in this mode. Upon detection of an end-of-file, DOS will return the partial bytecount in AC1.

Required input:

AC0 - Byte pointer to the starting byte address within user memory into which DOS will read the data.

AC1 - Number of bytes to be read.

Format:

```
.SYSTM
.RDS n          ;Read from channel n
error return
normal return
```

Possible errors.

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 open.
24	ERPAR	Parity error (tape). Often caused by dirty heads.
30	ERFIL	File read error (bad tape or dirty tape heads).
33	ERRD	Attempt to read into system area.
34	ERDIO	File accessible by direct block I/O only.
47	ERSIM	Simultaneous reads from same multiplexed line.
101	ERDTO	Ten-second disk timeout occurred.
106	ERCLO	Channel closed by another task.

Write Sequential (.WRS)

.WRS is the counterpart of .RDS; it writes data verbatim from memory to a file. Note that DOS recognizes no character as an end-of-file in this mode.

If you open a file via .OPEN or .EOPEN, and if this is the first write since the open, DOS starts writing at the beginning of the file (unless you moved the file pointer by the .SPOS command after opening). Subsequent writes after the open begin at the file position pointer. If you opened the file via .APPEND, DOS starts writing at the end of the file.

Required input:

AC0 - Byte pointer to the starting address of the data within user memory.
AC1 - Number of bytes to be written.

Format:

```
.SYSTM
.WRS n          ;Write to channel n
error return
normal return
```

Possible errors:

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 open.
27	ERSPC	Out of disk space.
34	ERDIO	File accessible by direct block I/O only.
47	ERSIM	Simultaneous writes to the same QTY/ALM line.
101	ERDTO	Disk timeout occurred.
106	ERCLO	Channel closed by another task.
113	ERNMC	No outstanding receive request.

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

These calls allow your program to read (or write) one 64-word record in either a random or contiguous disk file. There are four 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, 7, and so on.

Required input:

AC0 - Destination memory address.
AC1 - Record number (record numbers start with 0).

Format:

```
.SYSTM
.RDR n          ;Read from 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 read past the end of a contiguous file.
7	ERRPR	Attempt to read a read-protected file.
15	ERFOP	No file is open on this channel.
30	ERFIL	File read errors (mag tape or cassette - probably a 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 a randomly- or contiguously- organized disk file. DOS will write 64 words to the record number you specify, starting from the address you pass in AC0.

Required input:

AC0 - Memory address.
AC1 - Destination record number.

Format:

```
.SYSTM
.WRR n          ;Write to 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.

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

These are direct block I/O calls. Use these calls in your program to transfer blocks to or from random or contiguous files. DOS uses no system buffers for the transfer.

Blocks in random and contiguous disk files have a fixed length of 256_{10} words; they are numbered sequentially from 0. A .RDB for the first block in a file would transfer the 64-word records numbered 0, 1, 2, and 3.

Required input:

AC0 - Starting memory address for the block transfer.
AC1 - Starting relative block number in the series to be transferred.
AC2 - The left half of AC2 must contain the number of blocks which you want DOS to transfer. If you set the channel number to 77, the right half of AC2 must contain the channel number.

Format:

```
.SYSTM
.RDB (.WRB) n    ;n is the channel number
error return
normal return
```

Possible errors:

AC2	Mnemonic	Meaning
0	ERFNO	Illegal channel number.
3	ERICD	Illegal command for device.
4	ERSVI	Not a random or contiguous 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 open.
27	ERSPC*	Disk space is exhausted.
30	ERFIL	File read error, mag tape. Probably a bad tape or dirty head.
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, DOS returns the code in the right byte of AC2; the left byte contains the partial read or write count.

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

Before you can read or write in free format on a magnetic tape, you must open the device and associate it with a channel. Use the .MTOPD command to do this. After you have finished with the drive, release it.

.MTOPD is a global call; after you use it, you can access all files on the specified device.

To position a free format tape to a specific file, pass the file name to .MTOPD in the form MTn:m.

Required input:

AC0 - Byte pointer to the magnetic tape file specifier.
AC1 - Characteristic inhibit mask (see .GTATR).

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

Format:

```
.SYSTM
.MTOPD n        ;n is the channel number
error return
normal return
```

Possible errors:

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

Perform 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 long, you can space forward or backward from 1 to 4095 data records or to the start of a new data file, or you can 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 information about the hardware characteristics, see Magnetic Tape in the Programmer's Reference Manual for Peripherals.

Required input (to read device status word):

AC1 - Command word - bits 1-3 set, other bits 0.
AC2 - Channel number if n equals 77.

Required input (for other .MTDIO operations):

AC0 - Memory address for data transfer.
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 these seven command codes:

- 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 track, even for 7-track)
- 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, DOS positions the tape to the beginning of the next (or previous) file on the tape. If 0 on a read command, DOS reads words until it encounters either an end-of-record or 4096 words. If 0 on a write command, the system will write 4096 words.

AC2 - channel number if n equals 77.

Format:

```
.SYSTM
.MTDIO n        ;n is the channel number
error return
normal return
```

Upon a read status command, if DOS detects no system error, control will go to the normal return and AC2 will contain a device 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 3-3. Device Status Word on Normal Return .MTDIO

When your program issues a read, write, space forward, or space backward command, the command word in AC1 contains 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.

Possible errors:

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 free form I/O.

Figure 3-4 summarizes the possible returns by .MTDIO and the values returned in AC1 and AC2. On hardware errors, DOS sets bit 0 of TSW (in AC2); on system errors it clears this bit.

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 3-4 .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.

CONSOLE I/O COMMANDS

To transfer single characters between your primary ($TTI/$TTO) 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, nor do they provide special character handling (e.g., for carriage returns). These commands reference $TTI/$TTO; the console is always available to them, and you need no channel number or open command.

Get a Character (.GCHAR)

This command places a character typed on the console in AC0. DOS right-adjusts the character (without parity) in AC0, and clears the left byte of AC0. You need no I/O channel for .GCHAR; DOS always uses the console for input. DOS will not echo the character on the console.

Format:

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

If no character is currently in the console input buffer, the system waits.

Possible errors: none.

Put a Character (.PCHAR)

This command types the character in bits 9-15 of AC0 on the console.

Format:

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

Possible errors: none.

Get the Input Console Name (.GCIN)

This command returns the name of the console input device. This name is always $TTI.

Required input:

AC0 - Byte pointer to a six-byte area which will receive the console name.

Format:

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

Possible error:

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

Get the Output Console Name (.GCOUT)

This command returns the name of the output console. This name is always returned as $TTO.

Required input:

AC0 - Byte pointer to the six-byte area which will receive the console name.

Format:

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

Possible error:

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

MEMORY ALLOCATION COMMANDS

Excluding the Task Scheduler, and locations 0-15$_8$, DOS resides in upper memory. It executes your program in lower memory. DOS memory looks essentially like this:

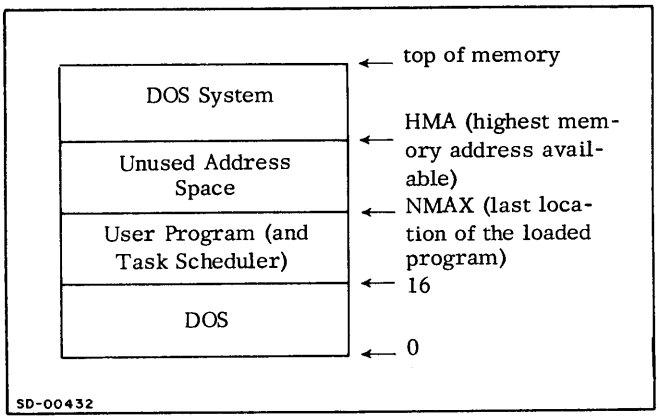


Figure 3-5. Memory Allocation

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 include the global switch /S in the RLDR command. (By default, DOS loads it just above your 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 the bottom of DOS (or the bottom of the symbol table, if the program was loaded with global /S).

Follow .MEM with a SUB 1,0 and INC 0,0 instruction to determine the amount of additional memory available to your program.

Format:

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

Possible errors: none.

Change NMAX (.MEMI)

This command allows your program to increase or decrease the value of NMAX. The command updates the value of NMAX in the UST (in USTNM) and returns the new NMAX in AC1.

DOS will not change NMAX if its new value would be greater than HMA + 1. The system does not check NMAX against its original value (as determined by RLDR).

Whenever one of your programs will require memory space above its NMAX, it can invoke .MEMI to allocate the number of words needed. DOS uses the value of NMAX to determine the amount of memory to save if it suspends a program. Generally, you should update NMAX even for temporary storage above the current NMAX. If a program swaps 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.

However, each of your programs should request only the memory space it actually needs, and should release memory space when it no longer needs it.

Required input:

AC0 - The increment or decrement (as a signed, two's complement figure) of NMAX.

Format:

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

Possible error:

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

DEVICE ACCESS COMMANDS

The RDOS device access commands, .DEBL and .DDIS are no-ops under DOS, and take the normal return. They are provided for RDOS compatibility.

Read the Front Panel Switches (.RDSW)

This system call allows your program to read the position of the front panel or hand held console switches. DOS returns the switch configuration in AC0. Bit 0 equals switch 0, etc.

Format:

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

Possible errors: none.

CLOCK/CALENDAR COMMANDS

DOS provides four commands to keep track of the time of day and the current date. It stores dates as days from December 31, 1967 (day 1 is January 1, 1968). DOS uses a 24-hour clock.

Get the Time of Day (.GTOD)

This command requests the system to pass you the current time in hours, minutes, and seconds. DOS will return the time in binary as follows:

AC0 - Seconds
AC1 - Minutes
AC2 - Hours (24-hour clock)

Format:

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

Possible errors: none.

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 (24-hour clock)

Format:

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

Possible error:

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. DOS returns the month in AC1, the day in AC0, and the current year (less 1968) in AC2.

Format:

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

Possible errors: none.

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. This routine works only on years from 1968 to 2099.

Required input:

AC0 - Number of the day within the month.
AC1 - Number of the month (January is month 1).
AC2 - Number of the current year, less 1968.

Format:

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

Possible error:

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

SPOOLING COMMANDS

The RDOS spooling commands, .SPKL, .SPDA, and .SPEA are no-ops in DOS, and take the normal return. They are provided for RDOS compatibility.

CONSOLE CONTROL CHARACTERS

You can suspend display on any console by typing CTRL-S, and restore display by typing CTRL-Q. Output to the console after the CTRL-S is not lost; it is merely suspended until you type CTRL-Q. This feature, which works with most Data General programs, is useful for long displays on CRT terminals.

CONSOLE INTERRUPTS

You can interrupt the current program from the console by typing either CTRL and A or CTRL and C. CTRL-A works abruptly; it halts program execution, saves nothing, and gives control to a higher-level program - generally the CLI. CTRL-C writes the current core image to disk file BREAK.SV and then does the same thing as CTRL-A. After DOS has executed CTRL-A, the message - INT will appear on the console; after CTRL-C, the message BREAK will appear.

If you want to program an interrupt, use the system call .BREAK; this produces the same effect as CTRL-C. If, upon any of these interrupts, you want any program other than the CLI to gain control, you must set up its User Status Table as 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, starting at location 400_8. 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 loader initializes both words to -1, and you must change this if you want to specify your own routines. Chapter 5 describes the UST in detail.

IF USTIT contains -1 when you hit CTRL-A, or if 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 the 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 may halt with 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 (it is in user address space), the system skips this program and examines the next higher level program (see Figure 3-7).

After finding a program with USTIT or USTBR (as appropriate) containing an address instead of -1, the system appropriates the TCB of the current 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, DOS executes the interrupt routine. (In a single task program, the program is the highest priority task.) Figure 3-6 shows a program with an interrupt handler.

```
START: LDA 2, USTP  ;Put UST addr in AC2.
       LDA 0, .BRKA ;Pointer to addr of
                    ;CTRL-A handler.
       STA 0, USTIT, 2 ;Store CTRL-A
                       ;addr in USTIT.

;The main program follows here.

MAIN:   ....
        ....
        ....

;CTRL-A handler- this code will be
;executed on CTRL-A.

BRKA:   ....
        ....
        ....
.BRKA:  BRKA
        .
```

Figure 3-6. Program with Interrupt Handler

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 highest of NMAX or the start of the symbol table, SST. DOS places the break file in the current directory and uses the file name BREAK.SV; it deletes any existing BREAK.SV first. If the system cannot write file BREAK.SV (possibly because it lacks disk file space), control will go to one location before the address specified in USTBR, and AC2 will contain a system error code.

Although the BREAK.SV is a snapshot of the current state of main memory, the file is not directly executable; it is generally useful for debugging. Before you try to 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 if the breakfile requires them.

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

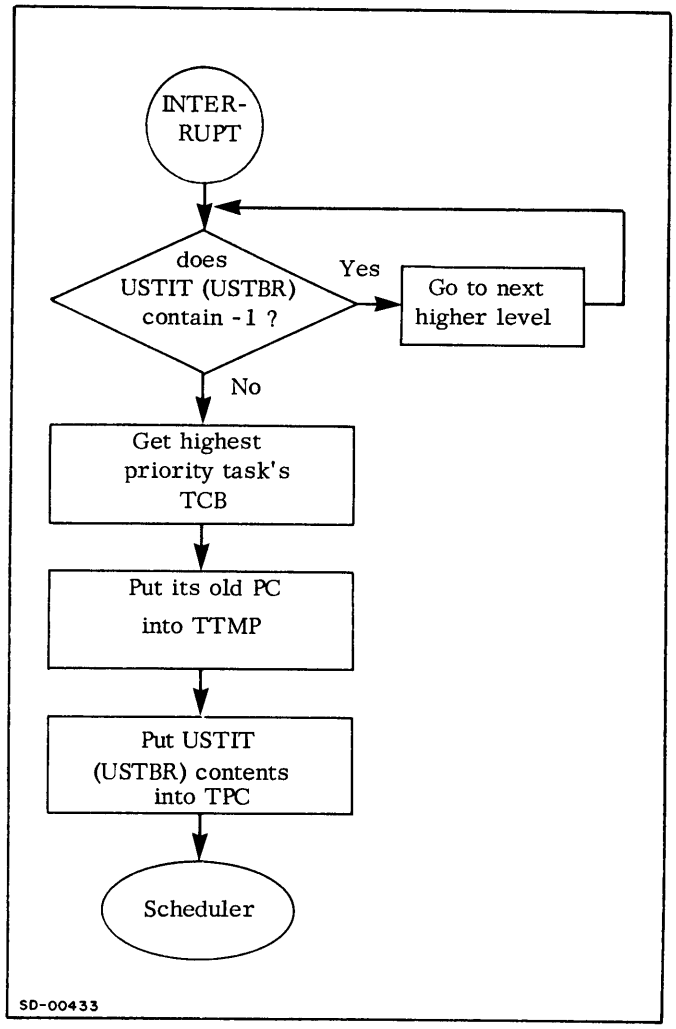


Figure 3-7. Program Interruption Logic Sequence

By default, when you execute 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.

Interrupt Program and Save Main Memory (.BREAK)

System call .BREAK is operationally equivalent to typing CTRL-C on the console; it saves the state of 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 version of BREAK.SV is deleted, and the breakfile is written to the current directory, where you can retain it, SAVE (CLI command) it under another name, or delete it. Generally, because system breaks close all channels, the breakfile is useful only for debugging with a disk editor, such as OEDIT or SEDIT.

The memory image file created by CTRL-C and .BREAK saves the program in the following state:

1. all open channels are closed;

2. user-defined clocks, user interrupts and user device enables (.DEBL) are removed;

Unlike the CTRL-C interrupt mechanism, the .BREAK call is operative at all times and is not disabled by the .ODIS command.

If USTBR (see preceding section) contains a valid address, control goes to this address after DOS writes BREAK.SV to disk. If USTBR contains -1, control will return to the next higher level program and DOS will examine its USTBR. Control eventually goes to the first higher level program whose USTBR contains a valid address. If DOS cannot write the break save file (e.g., due to insufficient file space), control goes to one location before the address contained in USTBR.

Format:

```
.SYSTM
.BREAK      ;No standard error
            ;or normal returns
```

Possible errors:

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 CTRL-C and CTRL-A console interrupts within your program. However, you can never disable the .BREAK system command with this command. You can re-enable console interrupts by issuing system call .OEBL from your program.

Format:

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

Possible errors: none.

Enable Console Interrupts (.OEBL)

When you first bootstrap a system DOS enables console interrupts CTRL-A, and CTRL-C. If you disable console interrupts by system call .ODIS this call re-enables them.

Format:

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

Possible errors: none.

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, 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, the current core image is saved up to the higher of NMAX of 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 .NEMI, 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.

Note: A program cannot swap or chain if multiplexor lines are open.

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
0B0	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.
1B15	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 Error 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 it returns an error code to the higher-level program's AC2. 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 error code, a text message will be printed. 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 be typed out, where n is the numeric code in octal. This mechanism is described in an appendix of the CLI manual.

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)

The .BOOT call is operationally equivalent to the CLI BOOT command. It closes all files, releases the master directory, and invokes BOOT.SV; BOOT.SV then looks for the DOS system or program whose name you indicate in AC0. BOOT.SV then loads the new system, which asks date/time questions and then invokes its CLI.

The new system or program can be in any initialized disk, but not in a user directory. It can also be the name of a link to a system or program, if all disks in the resolution chain are initialized (the overlay file, if any, must also be linked).

For automatic start or restart features of BOOT.SV, see Chapter 6, Automatic Restarts.

Required input to .BOOT is:

AC0 - Byte pointer to system or program name.

The format is:

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

There is no normal return, because a successful call invokes BOOT.SV, which then passes control to the new system.

Possible errors:

AC2	Mnemonic	Meaning
0	ERFNM	Illegal file name.
12	ERDLE	File does not exist.
23	ERRTN	File RESTART.SV does not exist, yet the jumper or switches specified a search for it.
53	ERDSN	Directory specifier unknown.
101	ERDTO	Ten-second disk timeout.

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 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.

For an example of all this, take the following RLDR command:

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

(A left bracket specifies the start of an overlay node, and a right bracket specifies the end of this node. For more on formats, see the Extended Relocatable Loaders manual or RLDR in the CLI manual.)

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 1), 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.

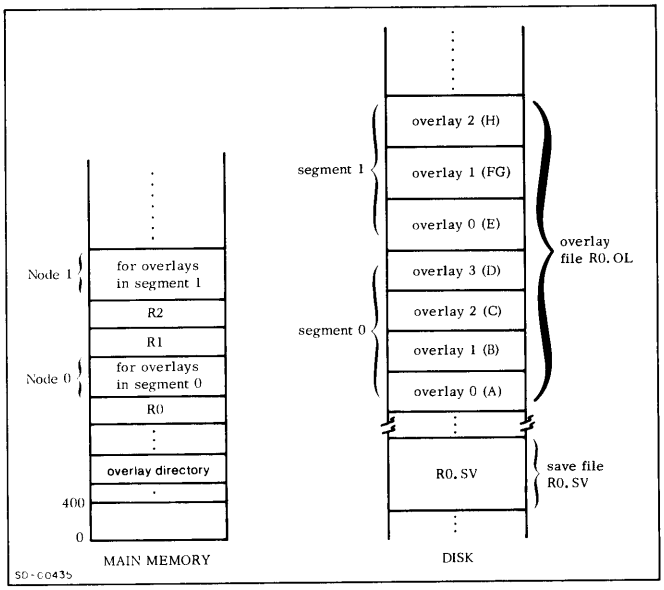


Figure 4-1. User Overlays

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.

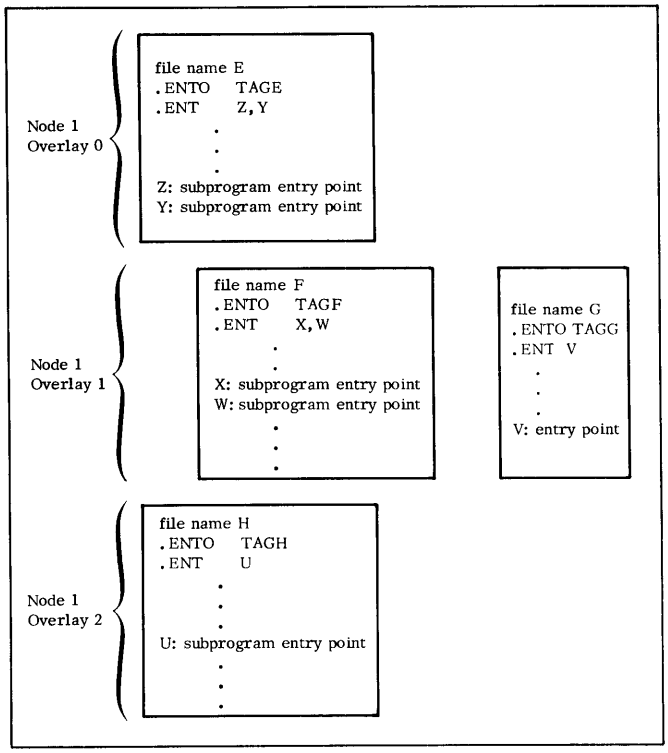


Figure 4-2. Segment 1 of Overlay File R0.OL

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
        .OVLOD n        ;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; or, if you used .ENTO in the overlay, pass a byte pointer to the .ENTO-assigned label.

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.

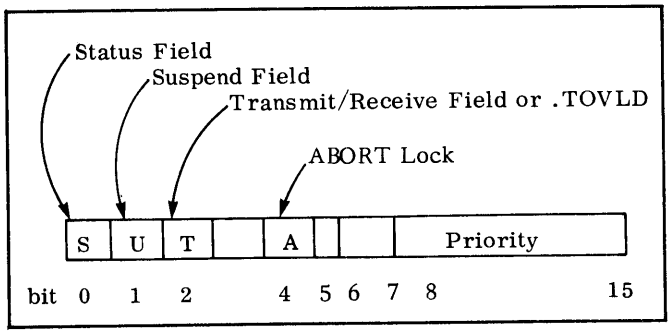


Figure 5-1. Task State/Priority Information (TPRST)

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. Bit A is set if the task is being aborted. Bit 5 is reserved. Bits 6 and 7 can help you to expand the DOS task-handling mechanism, as explained in Appendix J. The task priority is contained in bits 8-15.

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 higher-language save area; if this doesn't apply, the system sets TELN to 0. TID contains the task identification number, if any, in its right byte.

Note: The NOVA 3 and microNOVA hardware stack, the Trap PC, and the stack limit are 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 after a system or task call, 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. Bit 0 is set if a .KILL or .ABORT of the task has been issued. 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. A task can also be dormant; i.e. not initiated by a .TASK or .QTSK call.

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 transmit-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.
5. The reception of a message sent by .XMTW.

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.
410	USTCS	FORTRAN common area size.
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	USTRV	Revision level number, and during execution, the environment state.
423	USTIA	Address of TCB for keyboard interrupt task (For RDOS compatibility only).

*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.

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 NOVA 3 or microNOVA, the program was loaded with N3SAC3, in which case AC3 contains the frame pointer). 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.

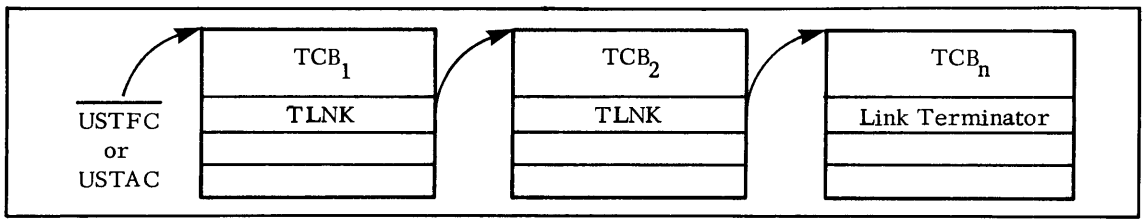


Figure 5-2. TCB Free Element Chain

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 .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, with one exception: calls performing multiplexor I/O.

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, or it was performing multiplexor I/O.

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 readying 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.

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. 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.

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 node.

AC2 contains the starting address of a table, QTLN (see the PARU.SR listing in Appendix E), words long, which describes the priority of the task, the time it is to be created, etc. The following task queue table describes the queue table'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	QAC2	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 node 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 within an overlay.

Required input to this call is:

AC0 - left byte contains the overlay node 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 node number in left byte; user overlay number in right byte.

The format of the call is:

```
.OVKIL
error return
```

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 overlay 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 node 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 node 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 node and the overlay use count has gone to zero, the caller is suspended until the node becomes free.

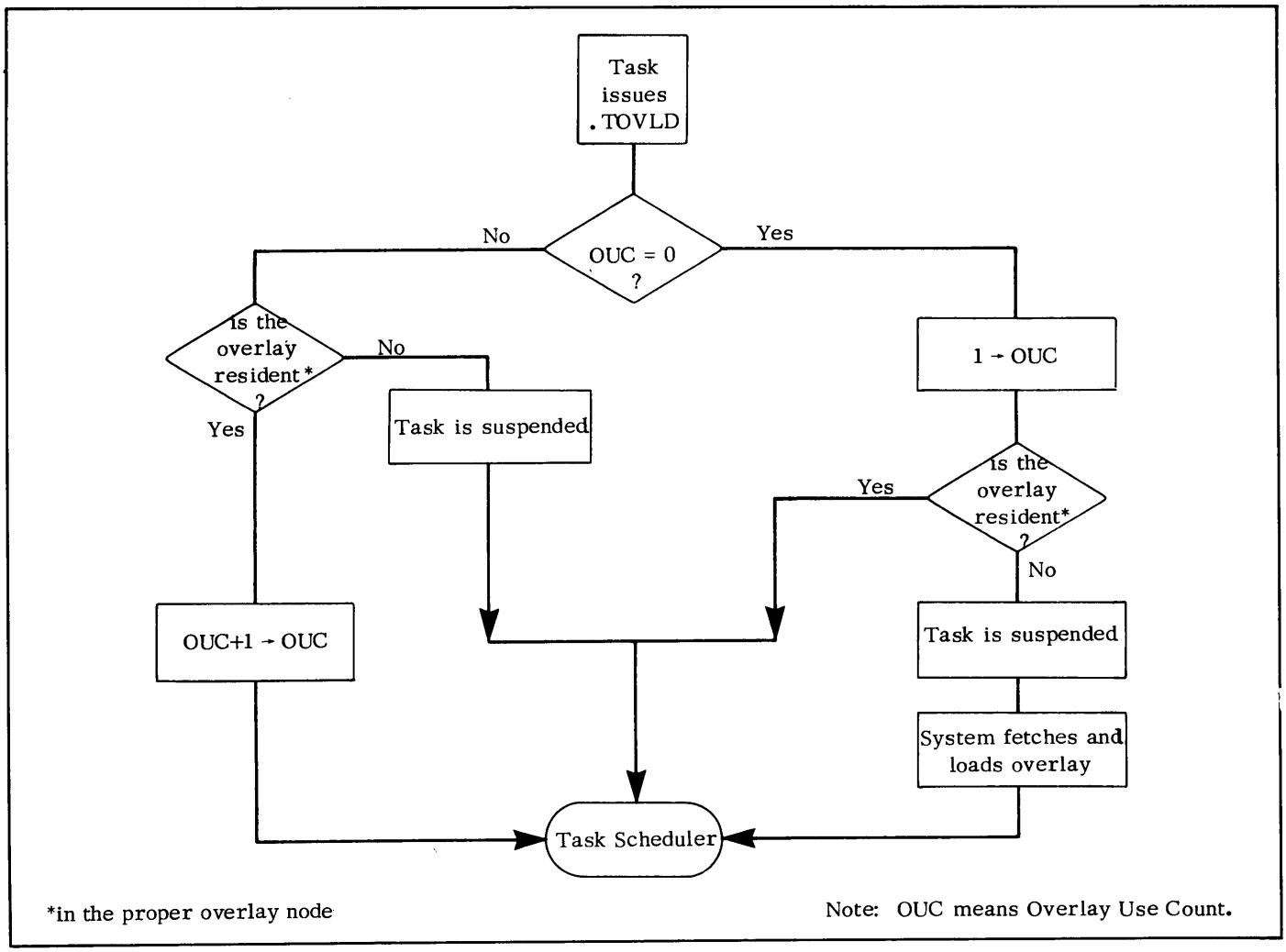


Figure 5-3. .TOVLD Logic Sequence

If the load request is unconditional and the node 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 node becomes free.

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

All overlay requests must be paired with an eventual overlay release (.OVREL/.OVEX/.OVKIL) or the node 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.

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 interrupts 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
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

These calls work only with tasks which have non-zero IDs.

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 AND ENABLING THE MULTI-TASK ENVIRONMENT (.SINGL AND .MULTI)

In a normal multitask environment, ready tasks compete for CPU control according to their relative priority. Although you can assign the highest priority (0) to one or more tasks, rescheduling occurs on each system interrupt, or when the executing task issues a system or task call -- thus, in a multitask environment, even the highest priority task may be suspended. Under some circumstances, you may want a task to retain CPU control continuously. To give a task such control, DOS provides the task call

.SINGL

When a task issues .SINGL, it disables the multitask environment and retains CPU control despite system calls and most task calls it issues; although interrupts continue, the Scheduler will allow the task to retain control. However, user interrupt routines defined via .IDEF continue to execute as usual. The privileged task retains CPU control until it restores the multitask environment by issuing task call

.MULTI

The multitask environment is also restored if the task suspends or kills itself.

Generally, a task should not disable the environment unless it must be absolutely autonomous; certainly it should not do so if it relies on other tasks. If you must deny other tasks access to a critical resource, like a database, use the .XMT/.REC mechanism.

As with other task calls, you must declare .SINGL and .MULTI external (.EXTN) in a source program if you want to use them.

Disable the Multitask Environment (.SINGL)

This call disables the multitask environment, and gives the issuing task continuing CPU control, despite its priority or any system calls (and most task calls) it issues. This can be useful for operations outside of user state (see Appendix J).

There is no required input to .SINGL; there is no error return.

Format:

```
.SINGL
normal return
```

Restore the Multitask Environment (.MULTI)

This call enables normal Scheduler operations and the multitask environment after they have been disabled by call .SINGL. There is no required input, nor an error return from .MULTI.

Format:

```
.MULTI
normal return
```

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.
.MULTI	Restore the multitask environment.
.OVEX	Release an overlay and return to the caller.
.OVKILL	Kill an overlay task and release the overlay.
.OVREL	Release an overlay node.
.PRI	Change the priority of a task.
.QTSK	Queue a core-resident of overlay task.
.REC	Receive a task message.
.SINGL	Disable the multitask environment.
.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:

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

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.) Make sure this mask masks out this priority levice. The mechanism of device interrupts is covered in the Programmer's Reference Manual for Peripherals.

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 Routine or Power-Fail Restart Routine (.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. You can also use .IDEF to identify your own power-up routines. Use .UIEX, below, to exit from the routine.

For User Devices:

AC0 must contain the device code of the new device; AC1 must contain the address of the new device's DCT.

For Power-up Routines:

Pass 77_8 in AC0 and the starting address of your power-up routine in AC1. 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 if rescheduling is to 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 or returns 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 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

If you chose the AUTO RESTART ON POWER FAIL option during system generation, your DOS system includes support for the power fail/auto restart option. When the system detects a power loss, it transfers control to a power fail routine which saves the contents of all accumulators, the PC, carry, and FPU.

For all semiconductor memory, the auto restart routine requires a working battery backup. If there is no battery, or if it fails during the power loss, the state of any program in semiconductor memory will be lost.

Automatic Start

If you wish, you can bring up a DOS system without operator intervention. This is most useful for a dedicated system which lacks a console, or for restart after a power failure where a backup battery fails, (power fails are described below).

For a start without operator intervention, the system disk must be in the primary drive on the controller (DP0, DP2, DP4, or DP6 on microNOVA, DP0 or DP4 on NOVA). It must also have the proper bootstrap root, BOOT.SV, and a DOS system named SYS.SV/SYS.OL (or links named SYS.SV/SYS.OL to a system) on it.

For a microNOVA DOS system, the computer must have the CPU program load option, and jumper W10 must be in on the CPU control board. For a NOVA, all data switches must be up.

If all of these conditions exist when you execute the program load steps, BOOT.SV will invoke SYS; then SYS will attempt to chain to a file named RESTART.SV instead of the CLI. File RESTART.SV is not provided with DOS; you must create it to tell SYS what to do when it gets control. If the DOS system cannot find RESTART.SV, it displays a FILE NOT FOUND error message, then asks date/time questions. After you answer these, it invokes its CLI.

If you provide auto start (or restart) via RESTART.SV, remember that RESTART.SV is chained into execution on level 0. It should either swap to a dedicated program on level 1, or chain to CLI.SV, or release the master device to shut down. Also, when RESTART.SV gets control, the time and date remain 00:00:00, January 1, 1968, and you must update them if your application requires this.

microNOVA Power Fail and Auto Restart

The power fail rules assume that this DOS system has the hardware power fail option, and that it was SYSGENed to include AUTO RESTART ON POWER FAIL. The system must also have a backup battery.

When power returns:

1. If the battery backup works throughout the outage, and if the front panel POWER switch is in LOCK when power returns, DOS will continue the current program from the point of interruption. If you defined a power fail routine with .IDEF, DOS will execute the .IDEF routine after restarting all SYSGENed devices.

   If the backup battery works throughout the outage, and if the POWER switch is in RUN when power returns, execute the following steps to have DOS execute the power-up routine above:

   - If you have a hand-held console, press CONT.
   - If you have the console debug option, type "P."
   - If you have the CPU program load option, press the front panel rocker switch to CONTINUE.

2. If the battery fails and if this microNOVA system lacks the CPU program load option, then control will go to either the hand-held console or console debug routine (whichever is present), when power returns. Rebootstrap the system and CLEAR all files (see the CLI manual).

   If you have the CPU program load option and the battery backup fails, DOS can automatically restart. (Naturally, you must have executed the steps below before the battery failure.) For an automatic restart, jumper W10 must be in on the CPU control board, and the POWER switch must be in LOCK when power is restored. Under these conditions, the hardware program loads the device specified in the program-load jumpers. This brings in BOOT.SV, which searches for a DOS system named SYS.SV/SYS.OL (or links named SYS.SV/SYS.OL to a system) when power returns. If BOOT finds SYS.SV, it executes SYS.SV, which then types this message on the console:

   ```
   MICRONOVA DOS REV x.xx
   MANUAL START (STRIKE ANY KEY) ?
   ```

The system then waits 15 seconds. If a key is struck on $TTI during this time, DOS asks the normal date/time questions, then invokes its CLI as described under Automatic Start, above. If 15 seconds pass, DOS attempts to chain to file RESTART.SV. If DOS can't find RESTART.SV, it invokes its CLI. See Automatic Start, above, for details on RESTART.SV.

If jumper W10 is not in on the CPU board, the power fail routine will invoke BOOT (as above) when power returns; but BOOT will ask:

FILENAME?

and wait for someone to bring up a system normally.

In summary, if you want the system to resume some operation automatically after a battery backup failure during a power outage:

- The system must have the CPU program load option, you must insert jumper W10 on the CPU board.
- The POWER key must be in LOCK when the power is restored.
- You must have a DOS system (or links) named SYS.SV/SYS.OL on DP0 (or equivalent in other enclosure).
- You must build file RESTART.SV to tell DOS what to do when it regains control.

NOVA Restart Procedures

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

POWER RESTORED

The system then restores the current program, which resumes from the point of interruption.

If the console key is in the ON position when power returns, set all data switches to zero (down), and lift START when power returns. This outputs the POWER RESTORED message and restores the current program.

Power-up on Devices

All system peripherals (except mag tape drives) receive power-up service when power returns. Power-up service for disks includes a complete reread or rewrite of the current disk block, thus no disk information is lost.

Character output devices may lose one or more characters during power up. The card reader may lose up to eighty columns of information on a single card: a line printer may lose as much as a line.

No power up service is provided for user devices. You must use .IDEF to identify a user power-up routine, and .UPEX, below, to exit from such a routine.

Exit from a User Power-up Routine (.UPEX)

.UPEX returns from a user power-up routine (defined by .IDEF). When the system enters a user power-up routine, AC3 contains the return address, and your program should save AC3 and restore it before issuing .UPEX.

.UPEX is a task call, which means that you must insert a .EXTN .UPEX statement in any program which includes it.

The format of .UPEX is

AC3 - Return address upon entry to the user routine.

```
.UPEX
```

Control returns to the address which the original program was executing when the power failure interrupted it. The .UPEX call takes neither an error nor normal return.

End of Chapter

CHAPTER 7
GENERATING A DOS SYSTEM ON A microNOVA

This information now appears in How to Generate Your DOS System (093-000222).

CHAPTER 8
GENERATING A DOS SYSTEM ON A NOVA

This information now appears in How to Generate Your DOS System (093-000222).

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 .GPOSn	(returned) High order portion of byte-pointer.	(returned) Low order portion of byte-pointer.	Channel number (if n = 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 fail 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
.MULTI(1)(4)			
.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
.SINGL(1)(2)			
.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

(4) on error ERSPC, partial write count returns in bits 0-7, and error code in bits 8-15.

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.	.EXEC (argument to .ERTN)			
20	ERNUL	Null error code.				
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 invalid 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
127	ERRDY	Line not ready (multiplexors).	
130	ERINT	Console interrupt received (multiplexors).	
131	EROVR	Character overrun error (mux hardware).	
132	ERFRM	Character framing error (mux only).	
133	ERSFT	Too many soft errors on diskette.	

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	!	12-8-7	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	11-9-5	010	(	12-8-5	050
HT	12-9-5	011	)	11-8-5	051
LF or NL	0-9-5	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	9-8-4	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
@	8-4	100
A	12-1	101
B	12-2	102
C	12-3	103
D	12-4	104
E	12-5	105
F	12-6	106
G	12-7	107
H	12-8	110
I	12-9	111
J	11-1	112
K	11-2	113
L	11-3	114
M	11-4	115
N	11-5	116
O	11-6	117
P	11-7	120
Q	11-8	121
R	11-9	122
S	0-2	123
T	0-3	124
U	0-4	125
V	0-5	126
W	0-6	127
X	0-7	130
Y	0-8	131
Z	0-9	132
[	12-8-2	133
\	0-8-2	134
]	11-8-2	135
↑ or ¬	11-8-7	136
← or _	0-8-5	137

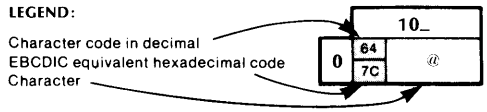
CHAR.	CARD CODE	ASCII CODE
\	8-1	140
a	12-0-1	141
b	12-0-2	142
c	12-0-3	143
d	12-0-4	144
e	12-0-5	145
f	12-0-6	146
g	12-0-7	147
h	12-0-8	150
i	12-0-9	151
j	12-11-1	152
k	12-11-2	153
l	12-11-3	154
m	12-11-4	155
n	12-11-5	156
o	12-11-6	157
p	12-11-7	160
q	12-11-8	161
r	12-11-9	162
s	11-0-2	163
t	11-0-3	164
u	11-0-4	165
v	11-0-5	166
w	11-0-6	167
x	11-0-7	170
y	11-0-8	171
z	11-0-9	172
{	12-0	173
|	12-11	174
}	11-0	175
~	11-0-1	176
DEL	12-9-7	177

END OF APPENDIX

APPENDIX C

ASCII CHARACTER SET

To find the *octal* value of a character, locate the character, and combine the first two digits at the top of the character's column with the third digit in the far left column.



OCTAL	00_		01_		02_		03_		04_		05_		06_		07_	
0	0 00	NUL	8 16	BS (BACK-SPACE)	16 10	↑P	24 18	↑X	32 40	SPACE	40 4D	(	48 F0	0	56 F8	8
1	1 01	↑A	9 05	HT (TAB)	17 11	↑Q	25 19	↑Y	33 5A	!	41 5D	)	49 F1	1	57 F9	9
2	2 02	↑B	10 15	LINE FEED	18 12	↑R	26 3F	↑Z	34 7F	" (QUOTE)	42 5C	*	50 F2	2	58 7A	:
3	3 03	↑C	11 0B	VT (VERT. TAB)	19 13	↑S	27 27	ESC (ESCAPE)	35 7B	#	43 4E	+	51 F3	3	59 5E	;
4	4 37	↑D	12 06	FF (FORM FEED)	20 3C	↑T	28 1C	↑\	36 5B	$	44 6B	, (COMMA)	52 F4	4	60 4C	<
5	5 2D	↑E	13 0D	CR (RETURN)	21 3D	↑U	29 1D	↑]	37 6C	%	45 60	-	53 F5	5	61 7E	=
6	6 2E	↑F	14 0E	↑N	22 32	↑V	30 1E	↑↑	38 50	&	46 4B	. (PERIOD)	54 F6	6	62 6E	>
7	7 2F	BELL ↑G	15 0F	↑O	23 26	↑W	31 1F	↑←	39 7D	' (APOS)	47 61	/	55 F7	7	63 6F	?

OCTAL	10_		11_		12_		13_		14_		15_		16_		17_	
0	64 7C	@	72 C8	H	80 D7	P	88 E7	X	96 79	` (GRAVE)	104 88	h	112 97	p	120 A7	x
1	65 C1	A	73 C9	I	81 D8	Q	89 E8	Y	97 81	a	105 89	i	113 98	q	121 A8	y
2	66 C2	B	74 D1	J	82 D9	R	90 E9	Z	98 82	b	106 91	j	114 99	r	122 A9	z
3	67 C3	C	75 D2	K	83 E2	S	91 8D	[	99 83	c	107 92	k	115 A2	s	123 C0	{
4	68 C4	D	76 D3	L	84 E3	T	92 E0	\	100 84	d	108 93	l	116 A3	t	124 4F	|
5	69 65	E	77 D4	M	85 E4	U	93 9D	]	101 85	e	109 94	m	117 A4	u	125 D0	}
6	70 C6	F	78 D5	N	86 E5	V	94 5F	↑ or ^	102 86	f	110 95	n	118 A5	v	126 A1	~ (TILDE)
7	71 C7	G	79 D6	O	87 E6	W	95 6D	← or _	103 87	g	111 96	o	119 A6	w	127 07	DEL (RUBOUT)

SD-00476 Character code in octal at top and left of charts.

↑ means CONTROL

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:

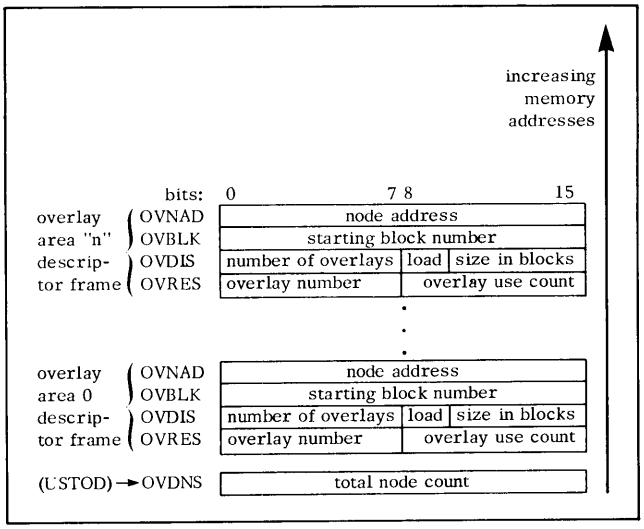


Figure D-1. Overlay Directory Structure (multitask)

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 E
USER PARAMETERS

This appendix lists file PARU.SR, which describes all DOS user parameters. These parameters define important system calls, task calls, and mnemonics for user programs. PARU.SR was delivered with your DOS system. File PARS.SR contains all system parameters.

```
        .TITL   PARU


;
; USER FILE TABLE (UFT) TEMPLATE
;

; USER FILE DEFINITION (UFD) OF UFT

.DUSR UFTFN=0           ;FILE NAME
.DUSR UFTEX=5           ;EXTENSION
.DUSR UFTAT=6           ;FILE ATTRIBUTES
.DUSR UFTLK=7           ;LINK ACCESS ATTRIBUTES
.DUSR UFLAD=7           ;LINK ALTERNATE DIRECTORY
.DUSR UFTBK=10          ;NUMBER OF LAST BLOCK IN FILE
.DUSR UFTBC=11          ;NUMBER OF BYTES IN LAST BLOCK
.DUSR UFTAD=12          ;DEVICE ADDRESS OF FIRST BLOCK (0 UNASSIGNED)
.DUSR UFTAC=13          ;YEAR-DAY LAST ACCESSED
.DUSR UFTYD=14          ;YEAR-DAY CREATED
.DUSR UFLAN=14          ;LINK ALIAS NAME
.DUSR UFTHM=15          ;HOUR-MINUTE CREATED
.DUSR UFTP1=16          ;UFD TEMPORARY
.DUSR UFTP2=17          ; "     "
.DUSR UFTUC=20          ;USER COUNT
.DUSR UFTDL=21          ;DCT LINK

; DEVICE CONTROL BLOCK (DCB) OF UFT

.DUSR UFTDC=22          ;DCT ADDRESS
.DUSR UFTUN=23          ;UNIT NUMBER
.DUSR UFCA1=24          ;CURRENT BLOCK ADDRESS (HIGH ORDER)
.DUSR UFTCA=25          ;CURRENT BLOCK ADDRESS (LOW ORDER)
.DUSR UFTCB=26          ;CURRENT BLOCK NUMBER
.DUSR UFTST=27          ;FILE STATUS
.DUSR UFEA1=30          ;ENTRY'S BLOCK ADDRESS (HIGH ORDER)
.DUSR UFTEA=31          ;ENTRY'S BLOCK ADDRESS (LOW ORDER)
.DUSR UFNA1=32          ;NEXT BLOCK ADDRESS (HIGH ORDER)
.DUSR UFTNA=33          ;NEXT BLOCK ADDRESS (LOW ORDER)
.DUSR UFLA1=34          ;LAST BLOCK ADDRESS (HIGH ORDER)
.DUSR UFTLA=35          ;LAST BLOCK ADDRESS (LOW ORDER)
.DUSR UFTDR=36          ;SYS.DR DCB ADDRESS
.DUSR UFFA1=37          ;FIRST ADDRESS (HIGH ORDER)
.DUSR UFTFA=40          ;FIRST ADDRESS (LOW ORDER)

; DCB EXTENSION

.DUSR UFTBN=41          ;CURRENT FILE BLOCK NUMBER
.DUSR UFTBP=42          ;CURRENT FILE BLOCK BYTE POINTER
.DUSR UFTCH=43          ;DEVICE CHARACTERISTICS
.DUSR UFTCN=44          ;ACTIVE REQ COUNT
```

```
.DUSR UFTEL=UFTCN-UFTFN+1        ;UFT ENTRY LENGTH
.DUSR UFDEL=UFTDL-UFTFN+1        ;UFD ENTRY LENGTH

.DUSR UDBAT=UFTAT-UFTDC ;NEGATIVE DISP. TO ATTRIBUTES
.DUSR UDDL=UFTDL-UFTDC  ;NEGATIVE DISP. TO FIRST ADDRESS (HIGH ORDER)
.DUSR UDBAD=UFTAD-UFTDC ;NEGATIVE DISP. TO FIRST ADDRESS (LOW ORDER)
.DUSR UDBBK=UFTBK-UFTDC ;NEGATIVE DISP. TO LAST BLOCK
.DUSR UDBBN=UFTBN-UFTDC ;POSITIVE DISP. TO CURRENT BLOCK


; FILE ATTRIBUTES  (IN UFTAT)

.DUSR ATRP =1B0          ;READ PROTECTED
.DUSR ATCHA=1B1          ;CHANGE ATTRIBUTE PROTECTED
.DUSR ATSAV=1B2          ;SAVED FILE
.DUSR ATNRS=1B7          ;CANNOT BE A RESOLUTION ENTRY
.DUSR ATUS1=1B9          ;USER ATTRIBUTE # 1
.DUSR ATUS2=1B10         ;USER ATTRIBUTE # 2
.DUSR ATPER=1B14         ;PERMANENT FILE
.DUSR ATWP =1B15         ;WRITE PROTECTED


; FILE CHARACTERISTICS  (IN UFTAT)

.DUSR ATMSK=7B7          ;TO GET HIGH ORDER PART OF 3330
                         ; ADDRESSES OUT OF UFTDL
.DUSR ATLNK=1B3          ;LINK ENTRY
.DUSR ATPAR=1B4          ;PARTITION ENTRY
.DUSR ATDIR=1B5          ;DIRECTORY ENTRY
.DUSR ATRES=1B6          ;LINK RESOLUTION (TEMPORARY)
.DUSR ATCON=1B12         ;CONTIGUOUS FILE
.DUSR ATRAN=1B13         ;RANDOM FILE

;
; DCT PARAMETERS.
;

.DUSR DCTBS=0            ;1B0=1 => DEVICE USES DATA CHANNEL
.DUSR DCTMS=1            ;MASK OF LOWER PRIORITY DEVICES
.DUSR DCTIS=2            ;ADDRESS OF INTERRUPT SERVICE ROUTINE
```

```
; DEVICE CHARACTERISTICS  (IN UFTCH)

.DUSR   DCSTB=  1B15    ; SUPPRESS TRAILING BLANKS $CDR CNLY
.DUSR   DCCPO=  1B15    ; DEVICE REQUIRING LEADER/TRAILER
.DUSR   DCSTO=  1B15    ; USER SPECIFIED TIME OUT CONSTANT (MCA)
.DUSR   DCCGN=  1B14    ; GRAPHICAL OUTPUT DEVICE WITHOUT TABBING
                        ; HARDWARE
.DUSR   DCIDI=  1B13    ; INPUT DEVICE REQUIRING OPERATOR INTERVENTION
.DUSR   DCCNF=  1B12    ; OUTPUT DEVICE WITHOUT FORM FEED HARDWARE
.DUSR   DCTC=   1B11    ; TELETYPE CUTPUT DEVICE
.DUSR   DCKEY=  1B10    ; KEYBOARD DEVICE
.DUSR   DCNAF=  1B09    ; OUTPUT DEVICE REGLIRING NULLS AFTER FORM FEEDS
.DUSR   DCRAT=  1B08    ; RUBOUTS AFTER TABS REQUIRED
.DUSR   DCPCK=  1B07    ; DEVICE REQUIRING PARITY CHECK
.DUSR   DCLAC=  1B06    ; REQUIRES LINE FEEDS AFTER CARRIAGE RTN
.DUSR   DCSPO=  1B05    ; SPOOLABLE DEVICE
.DUSR   DCFWD=  1B04    ; FULL WORD DEVICE (ANYTHING GREATER THAN
.DUSR   DCFFO=  1B03    ; FORM FEEDS CN OPEN
.DUSR   DCLTU=  1B02    ; CHANGE LCWER CASE ASCII TC UPPER
.DUSR   DCC80=  1B01    ; READ 80 CCLLMS
.DUSR   DCDIO=  1B00    ; SUSPEND PROTOCOL CN TRANSMIT (MCA)
.DUSR   DCBDK=  1B00    ; DISK CHARACTERISTIC (SET NON-PARAMETRICALLY)
                        ; SET MEANS ITS 3330
.DUSR   DCSPC=  1B00    ; SPOOL CONTRCL
                        ; SET = SPCCLING ENABLED
                        ; RESET = SPCCLING DISABLED


;
; DEVICE CHARACTERISTICS FOR QTY AND ALM (PARU.SR)
;

 .DUSR  DCNI=   1B15    ;(MASKING ENABLES) CONSOLE INTERRLPTS
;.DUSR  DCCGN=  1B14    ;(MASKING DISABLES) TAB EXPANSICN
 .DUSR  DCLCC=  1B13    ;LOCAL LINE (MASKING MAKES MODEM LINE)

;               1B12    ;SAVE FOR 3 MCDEM PROTOCALS
;.DUSR  DCTC=   1B11    ;_ FOR RUBCLT (MASKING GIVES BACKSPACE)
;.DUSR  DCKEY=  1B10    ;(MASKING DISABLES) INPUT ECHOING,
                        ;       LINE EDITS, AND ↑Z EOF

;.DUSR  DCNAF=  1B9     ;(MASKING DISABLES) 20 NULLS AFTER FORM FEED
 .DUSR  DCXCN=  1B8     ;(MASKING ENABLES) XON/XOFF PROTCCALL FOR $TTR
;               1B7     ;SAVE FOR FLTURE USE

;.DUSR  DCLAC=  1B6     ;(MASKING DISABLES) LINE FEED AFTER CARRAIGE RETLRN
;.DUSR  DCSPO=  1B5     ;(MUST BE OFF) SPCCLING
 .DUSR  DCCRE=  1B4     ;CARRAIGE RETLRN ECHO
                        ;       (MASKING ENABLES CR AS ENTER KEY)
;
; .WRL TO LINE 64
;
;       AC0= CODE+LINE #
;       AC1= DATA

.DUSR   W64DC=  087             ;CHANGE DEVICE CHARACTERISTIC MASK (AC1)
.DUSR   W64LS=  187             ;CHANGE LINE SPEED (AC1= 0 -> 3)
.DUSR   W64MS=  287             ;CHANGE MCDEM STATE (AC1) AS FOLLOWS
.DUSR     W64DTR= 1B15          ;  RAISE DATA TERMINAL READY
                                ;       ELSE LCWER
.DUSR     W64RTS= 1B14          ;  RAISE REQUEST TO SEND
                                ;       ELSE LCWER
```

```
;
; SWITCHES
;

.DUSR   A.SW=   1B00
.DUSR   B.SW=   1B01
.DUSR   C.SW=   1B02
.DUSR   D.SW=   1B03
.DUSR   E.SW=   1B04
.DUSR   F.SW=   1B05
.DUSR   G.SW=   1B06
.DUSR   H.SW=   1B07
.DUSR   I.SW=   1B08
.DUSR   J.SW=   1B09
.DUSR   K.SW=   1B10
.DUSR   L.SW=   1B11
.DUSR   M.SW=   1B12
.DUSR   N.SW=   1B13
.DUSR   O.SW=   1B14
.DUSR   P.SW=   1B15
.DUSR   Q.SW=   1B00
.DUSR   R.SW=   1B01
.DUSR   S.SW=   1B02
.DUSR   T.SW=   1B03
.DUSR   U.SW=   1B04
.DUSR   V.SW=   1B05
.DUSR   W.SW=   1B06
.DUSR   X.SW=   1B07
.DUSR   Y.SW=   1B08
.DUSR   Z.SW=   1B09
```

```
;
; SYSTEM CONSTANTS
;

.DUSR SCWPB=255.          ;WORDS PER BLOCK
.DUSR SCDBS=256.          ;SIZE OF DISK BLOCK
.DUSR SCRRL=64.           ;WORDS PER RANDOM RECORD
.DUSR SCLLG=132.          ;MAX LINE LENGTH
.DUSR SCAMX=24.           ;MAX ARGUMENT LENGTH IN BYTES
.DUSR SCFNL=UFTEX-UFTFN+1 ;FILE NAME LENGTH
.DUSR SCEXT=UFTEX-UFTFN   ;EXTENSION OFFSET IN NAME AREA
.DUSR SCMER=10.           ;MAX ERROR RETRY COUNT
.DUSR SCSTR=16            ;SAVE FILE STARTING ADDRESS
.DUSR SCTIM=-80.          ;RINGIO 1 MS. LOOP TIME (SN)
.DUSR SCPPL=0             ;PRIMARY PARTITION LEVEL
.DUSR SCPPA=6             ;PRIMARY PARTITION BASE ADDRESS
.DUSR SCDSK=3             ;ABSOLUTE ADDRESS OF DISK INFORMATION BLOCK
.DUSR SCBAD=4             ;ABSOLUTE ADDRESS OF BAD BLOCK TABLE BLOCK
.DUSR SCSYS=0             ;SYS.DR ADDRESS OFFSET
.DUSR SCPSH=1             ;PUSH DIRECTORY OFFSET
.DUSR SCPNM=4             ;MAX NUMBER OF PUSH LEVELS
.DUSR SCMAP=SCPNM*2+SCPSH ;RELATIVE BASE ADDRESS OF MAP.DR
.DUSR SCBPB=1             ;RELATIVE BACKROUND PUSH BASE
.DUSR SCFPB=SCBPB+SCPNM   ;RELATIVE FOREGROUND PUSH BASE
.DUSR SCFZW=SCPNM*4+SCBPB ;FRAME SIZE WORD (SKIP DOUBLE WORD PUSH INDICES)
.DUSR SCNVW=SCFZW+1       ;NUMBER-OF-SYSTEM-OVERLAYS WORD
.DUSR SFINT=1B0           ;INTERRUPT FLAG
.DUSR SFBRK=1B15          ;BREAK FLAG
.DUSR SCNSO=64.           ;NUMBER OF SYSTEM OVERLAYS


; SYSTEM BOOTSTRAP CONSTANTS

.DUSR   SCTBP=0           ;TEXT STRING BYTE POINTER
.DUSR   SCINS=1           ;SWITCHED FULL/PARTIAL-OVERLAYS ADDRESS
.DUSR   SCPSA=2           ;PROGRAM START ADDRESS
.DUSR   SCPAR=SCPSA       ;PARTIAL INIT ADDRESS
.DUSR   SCINT=3           ;FULL/PARTIAL-OVERLAYS INIT ADDRESS
.DUSR   SCCLI=SCINT+1     ;ADDRESS OF END OF CLI
.DUSR   SCZMX=SCCLI+1     ;SQUASHED/UNSQUASHED FLAG
.DUSR   SCCPL=SCZMX+1     ;CURRENT PARTITION LEVEL
.DUSR   SCPBA=SCCPL+1     ;PARTITION BASE ADDRESS (LOW ORDER)
.DUSR   SCOFA=SCPBA+1     ;OVERLAY BASE ADDRESS (LOW ORDER)
.DUSR   SCPB1=SCOFA+1     ;PARTITION BASE ADDRESS (HIGH ORDER)
.DUSR   SCOF1=SCPB1+1     ;OVERLAY BASE ADDRESS (HIGH ORDER)
.DUSR   SCBAS=SCOF1+1     ;BASE OF INFORMATION BLOCK
.DUSR   SCSWC=SCBAS       ;SWITCH FOR SCINS ENTRY
.DUSR   SCIDV=20          ;INITIAL DEVICE CODE

.DUSR   SCALN=0           ;ASCII UNIT NUMBER
.DUSR   SCUN=1            ;UNIT (DEVICE CODE)
.DUSR   SCGC=2            ;ENTRY TO PASS FILENAME
.DUSR   SCNGO=4           ;ENTRY TO ASK FROM CONSOLE
```

```
; SYSTEM ERROR CODES

.DUSR ERFNO=     0        ; ILLEGAL CHANNEL NUMBER
.DUSR ERFNM=     1        ; ILLEGAL FILE NAME
.DUSR ERICM=     2        ; ILLEGAL SYSTEM COMMAND
.DUSR ERICD=     3        ; ILLEGAL COMMAND FOR DEVICE
.DUSR ERSV1=     4        ; NOT A SAVED FILE
.DUSR ERWR0=     5        ; ATTEMPT TO WRITE AN EXISTENT FILE
.DUSR EREOF=     6        ; END OF FILE
.DUSR ERRPR=     7        ; ATTEMPT TO READ A READ PROTECTED FILE
.DUSR ERWPR=     10       ; WRITE PROTECTED FILE
.DUSR ERCRE=     11       ; ATTEMPT TO CREATE AN EXISTENT FILE
.DUSR ERDLE=     12       ; A NON-EXISTENT FILE
.DUSR ERDE1=     13       ; ATTEMPT TO ALTER A PERMANENT FILE
.DUSR ERCHA=     14       ; ATTRIBUTES PROTECTED
.DUSR ERFOP=     15       ; FILE NOT OPENED
.DUSR ERFUE=     16       ; FATAL UTILITY ERROR
.DUSR   EREXQ=   17       ; EXECUTE CLI.CM (NO ERROR)
.DUSR   ERNLL=   20       ; INVISIBLE ERROR CODE
.DUSR ERUFT=     21       ; ATTEMPT TO USE A LFT ALREADY IN USE
.DUSR ERLLI=     22       ; LINE LIMIT EXCEEDED 0
.DUSR ERRTN=     23       ; ATTEMPT TO RESTORE A NON-EXISTENT IMAGE
.DUSR ERPAR=     24       ; PARITY ERROR ON READ LINE
.DUSR ERCM3=     25       ; TRYING TO PUSH TOO MANY LEVELS
.DUSR ERMEM=     26       ; NOT ENUF MEMORY AVAILABLE
.DUSR ERSPC=     27       ; OUT OF FILE SPACE
.DUSR ERFIL=     30       ; FILE READ ERROR
.DUSR ERSEL=     31       ; UNIT NOT PROPERLY SELECTED
.DUSR ERADR=     32       ; ILLEGAL STARTING ADDRESS
.DUSR ERRD=      33       ; ATTEMPT TO READ INTO SYSTEM AREA
.DUSR ERDIO=     34       ; FILE ACCESSIBLE BY DIRECT I/O ONLY
.DUSR ERDIR=     35       ; FILES SPECIFIED ON DIFF. DIRECTORIES
.DUSR ERDNM=     36       ; DEVICE NOT IN SYSTEM
.DUSR EROVN=     37       ; ILLEGAL OVERLAY NUMBER
.DUSR EROVA=     40       ; FILE NOT ACCESSIBLE BY DIRECT I/O
.DUSR ERTIM=     41       ; USER SET TIME ERROR
.DUSR ERNOT=     42       ; OUT OF TCB'S
.DUSR ERXMT=     43       ; SIGNAL TO BUSY ADDR
.DUSR ERSGF=     44       ; FILE ALREADY SQUASHED ERROR
.DUSR ERIBS=     45       ; DEVICE ALREADY IN SYSTEM
.DUSR ERICB=     46       ; INSUFFICENT CONTIGUOUS BLOCKS
.DUSR ERSIM=     47       ; SIMULTANEOUS READ OR WRITE TO MUX LINE
.DUSR ERQTS=     50       ; ERROR IN USER TASK QUEUE TABLE
.DUSR ERNMD=     51       ; NO MORE DCB'S
.DUSR ERIDS=     52       ; ILLEGAL DIRECTORY SPECIFIER
.DUSR ERDSN=     53       ; DIRECTORY SPECIFIER NOT KNOWN
.DUSR ERD2S=     54       ; DIRECTORY IS TOO SMALL
```

```
.DUSR ERDDE=    55      ; DIRECTORY DEPTH EXCEEDED
.DUSR ERDIU=    56      ; DIRECTORY IN USE
.DUSR ERLDE=    57      ; LINK DEPTH EXCEEDED
.DUSR ERFIU=    60      ; FILE IS IN USE
.DUSR ERTID=    61      ; TASK ID ERROR
.DUSR ERCMS=    62      ; COMMON SIZE ERROR
.DUSR ERCUS=    63      ; COMMON USAGE ERROR
.DUSR ERSCP=    64      ; FILE POSITION ERROR
.DUSR ERDCH=    65      ; INSUFFICIENT ROOM IN DATA CHANNEL MAP
.DUSR ERDNI=    66      ; DIRECTORY NOT INITIALIZED
.DUSR ERNDD=    67      ; NO DEFAULT DIRECTORY
.DUSR ERFGE=    70      ; FOREGROUND ALREADYS EXISTS
.DUSR ERMPT=    71      ; ERROR IN PARTITON SET
.DUSR EROPD=    72      ; DIRECTORY IN USE BY OTHER PROGRAM
.DUSR ERUSZ=    73      ; NO ROOM FOR UFTS ON EXEC/EXFG
.DUSR ERMPR=    74      ; ADDR ERROR ON .SYSTM PARAM
.DUSR ERNLE=    75      ; NOT A LINK ENTRY
.DUSR ERNTE=    76      ; CURRENT BG IS NOT CHECKPOINTABLE
.DUSR ERSDE=    77      ; SYS.DR ERROR
.DUSR ERMDE=    100     ; MAP.DR ERROR
.DUSR ERDTO=    101     ; DEVICE TIME OUT
.DUSR ERENA=    102     ; ENTRY NOT ACCESSIBLE VIA LINK
.DUSR ERMCA=    103     ; MCA REQUEST OUTSTANDING
.DUSR ERSRR=    104     ; INCOMPLETE TRANSMISSION CAUSED BY RECIEVER
.DUSR ERSDL=    105     ; SYSTEM DEADLOCK
.DUSR ERCLO=    106     ; I/O TERMINATED BY CHANNEL CLOSE
.DUSR ERSFA=    107     ; SPOOL FILE(S) ACTIVE
.DUSR ERABT=    110     ; TASK NOT FOUND FOR ABORT
.DUSR ERDOP=    111     ; DEVICE PREVIOUSLY OPENED
.DUSR EROVF=    112     ; SYSTEM STACK OVERFLOW
.DUSR ERNMC=    113     ; NO MCA RECEIVE REQUEST OUTSTANDING
.DUSR ERNIR=    114     ; NO INIT/RELEASE ON OPENED DEVICE (MAG TAPE)
.DUSR ERXMZ=    115     ; .XMT & .IXMT MESSAGES MUST BE NON-ZERO
.DUSR ERCANT=   116     ; 'YOU CAN'T DO THAT'
.DUSR ERGOV=    117     ; .TOVLD NOT LOADED FOR QUEUED OVERLAY TASKS
.DUSR EROPM=    120     ; OPERATOR MESSAGE MODULE NOT SYSGENED
.DUSR ERFMT=    121     ; DISK FORMAT ERROR
.DUSR ERBAD=    122     ; DISK HAS INVALID BAD BLOCK TABLE
.DUSR ERBSPC=   123     ; INSUFFICIENT SPACE IN BAD BLOCK POOL (CORE)
.DUSR ERZCB=    124     ; ATTEMPT TO CREATE CONTIG OF ZERO LENGTH
.DUSR ERNSE=    125     ; PROGRAM IS NOT SWAPPABLE
.DUSR ERBLT=    126     ; BLANK TAPE
.DUSR ERRDY=    127     ; LINE NOT READY
.DUSR ERINT=    130     ; CONSOLE INTERRUPT RECEIVED
.DUSR EROVR=    131     ; CHARACTER OVER RUN ERROR
.DUSR ERFRM=    132     ; CHARACTER FRAMING ERROR
.DUSR ERSPT=    133     ; TOO MANY SOFT ERRORS (DOS ONLY)
```

```
; CLI ERROR CODES

.DUSR   CNEAR=  300     ; NOT ENOUGH ARGUMENTS
.DUSR   CILAT=  301     ; ILLEGAL ATTRIBUTE
.DUSR   CNDBD=  302     ; NO DEBUG ADDRESS
.DUSR   CCLTL=  303     ; COMMAND LINE TOO LONG
.DUSR   CNSAD=  304     ; NO STARTING ADDRESS
.DUSR   CCKER=  305     ; CHECKSUM ERROR
.DUSR   CNSFS=  306     ; NO SOURCE FILE SPECIFIED
.DUSR   CNACM=  307     ; NOT A COMMAND
.DUSR   CILBK=  310     ; ILLEGAL BLOCK TYPE
.DUSR   CSPER=  311     ; NO FILES MATCH SPECIFIER
.DUSR   CPHER=  312     ; PHASE ERROR
.DUSR   CTMAR=  313     ; TOO MANY ARGUMENTS
.DUSR   CTMAD=  314     ; TOO MANY ACTIVE DEVICES
.DUSR   CILNA=  315     ; ILLEGAL NUMERIC ARGUMENT
.DUSR   CSFLE=  316     ; FATAL SYSTEM UTILITY ERROR
.DUSR   CILAR=  317     ; ILLEGAL ARGUMENT
.DUSR   CCANT=  320     ; IMPROPER OR MALICIOUS INPUT
.DUSR   CTMLI=  321     ; TOO MANY LEVELS OF INDIRECT FILES
.DUSR   CSYER=  322     ; SYNTAX ERROR
.DUSR   CBKER=  323     ; BRACKET ERROR
.DUSR   CPARE=  324     ; PAREN ERROR
.DUSR   CCART=  325     ; < WITHOUT > OR > WITHOUT <
.DUSR   CCAR1=  326     ; ILLEGAL NESTING OF <> AND ()
.DUSR   CINDE=  327     ; ILLEGAL INDIRECT FILENAME
.DUSR   CPAR1=  330     ; ILLEGAL NESTING OF () AND []
.DUSR   CIVAR=  331     ; ILLEGAL VARIABLE
.DUSR   CILTA=  332     ; ILLEGAL TEXT ARGUMENT
.DUSR   CTATL=  333     ; TEXT ARGUMENT TOO LONG

.DUSR   CCMAX=  CTATL   ; MAX CLI ERROR CODE
.DUSR   ERML=   30.     ; MAXIMUM ERROR MESSAGE LENGTH


; EXCEPTIONAL SYSTEM STATUS CODES

.DUSR   PNMPE=  @1      ; MAP.DR ERROR
.DUSR   PNSDE=  @2      ; SYSTEM DIRECTORY ERROR
.DUSR   PNCSO=  @3      ; SYSTEM STACK FAULT
.DUSR   PNIDA=  @4      ; INCONSISTENT SYSTEM DATA
.DUSR   PNMDD=  @5      ; MASTER DEVICE DATA ERROR
.DUSR   PNMDT=  @6      ; MASTER DEVICE TIME OUT
.DUSR   PNDPE=  @7      ; MOVING HEAD DISK ERROR
.DUSR   PNCLI= @10      ; UNCLEARABLE UNDEFINED INTERRUPT
.DUSR   PNCBK= @12      ; INSUFFICENT CONTIGUOUS BLOCKS TO BUILD
                        ; PUSH SPACE INDICES
.DUSR   PNILL= @11      ; ILLEGAL EXTENDED INSTRUCTION
.DUSR   PNPSH= @13      ; RTN BEYOND TOP OF WORLD
.DUSR   PNIPB= @14      ; INCONSISTENT OR IMPOSSIBLE CONDITION
                        ; RELATED TO DUAL PROCESSORS (IPB)
.DUSR   PNITR= @15      ; INT WORLD TRAPPED
.DUSR   PNERC= @16      ; MULTIBIT MEMORY ERROR
.DUSR   PNPAR= @17      ; MEMORY PARITY ERROR
```

```
;
; USER STATUS TABLE (UST) TEMPLATE
;
.DUSR   UST=    400     ; START OF BACKGROUND USER STATUS AREA

.DUSR   USTP=12         ; PZERO LOC FOR UST POINTER
; NOTE- USTP MUST CORRESPOND TO PARS PZERO ALLOCATIONS

.DUSR   USTPC=  0       ; 0=>BACKGROUND, 1=>FOREGROUND
                        ; (WHEN NOT IN SCHED STATE)
.DUSR   USTZM=  1       ; ZMAX
.DUSR   USTSS=  2       ; START OF SYMBOL TABLE
.DUSR   USTES=  3       ; END OF SYMBOL TABLE
.DUSR   USTNM=  4       ; NMAX
.DUSR   USTSA=  5       ; STARTING ADDRESS
.DUSR   USTDA=  6       ; DEBUGGER ADDRESS
.DUSR   USTHU=  7       ; HIGHEST ADDRESS USED
.DUSR   USTCS=  10      ; FORTRAN COMMON AREA SIZE
.DUSR   USTIT=  11      ; INTERRUPT ADDRESS
.DUSR   USTBR=  12      ; BREAK ADDRESS
.DUSR   USTCH=  13      ; # TASKS (LEFT), # CHANS (RIGHT)
.DUSR   USTCT=  14      ; CURRENTLY ACTIVE TCB
.DUSR   USTAC=  15      ; START OF ACTIVE TCB CHAIN
.DUSR   USTFC=  16      ; START OF FREE TCB CHAIN
.DUSR   USTIN=  17      ; INITIAL START OF NREL
.DUSR   USTCD=  20      ; OVLY DIRECTORY ADDR
.DUSR   USTSV=  21      ; FORTRAN STATE VARIABLE SAVE ROUTINE (OR 0)
.DUSR   USTRV=  22      ; REVISION
                        ;  ENVIRONMENT STATE WORD WHEN EXECUTING
.DUSR   USTIA=  23      ; TCB ADDR OF INT OR BREAK PROC

.DUSR   USTEN=  USTIA   ; LAST ENTRY

.DUSR   UFPT=   30      ; SAVE SOS


; ENVIRONMENT STATUS BITS  (IN USTRV DURING EXECUTION)

.DUSR   ENMAP=  1B0     ;MAPPED MACHINE
.DUSR   ENUEC=  1B2     ;UNMAPPED ECLIPSE
.DUSR   ENMEC=  1B3     ;MAPPED ECLIPSE
.DUSR   ENUNV=  1B4     ;UNMAPPED NOVA
.DUSR   ENMNV=  1B5     ;MAPPED NOVA
.DUSR   ENUN3=  1B6     ;UNMAPPED NOVA 3
.DUSR   ENMN3=  1B7     ;MAPPED NOVA 3
.DUSR   ENUMN=  1B8     ;UNMAPPED MICRO NOVA

.DUSR   ENDOS=  1B11    ;DOS SYSTEM
.DUSR   ENINFO= 1B12    ;INFOS SYSTEM
.DUSR   ENSOS=  1B13    ;STAND ALONE SYSTEM
.DUSR   ENRTOS= 1B14    ;RTOS SYSTEM
.DUSR   ENRDOS= 1B15    ;RDOS SYSTEM
```

```
;
; TASK CONTROL BLOCK (TCB) TEMPLATE
;

.DUSR TPC=      0       ;USER PC (B0-14) + CARRY (B15)
.DUSR TAC0=     1       ;AC0
.DUSR TAC1=     2       ;AC1
.DUSR TAC2=     3       ;AC2
.DUSR TAC3=     4       ;AC3
.DUSR TPRST=    5       ;STATUS BITS (LEFT) + PRIORITY (RIGHT)
.DUSR TSYS=     6       ;SYSTEM CALL WORD
.DUSR TLNK=     7       ;LINK WORD
.DUSR TUSP=     10      ;USP
.DUSR TELN=     11      ;TCB EXTENSION ADDR
.DUSR TID=      12      ;TASK ID
.DUSR TTMP=     13      ;SCHEDULER TEMPORARY
.DUSR TKLAD=    14      ;USER KILL PROC ADDR
.DUSR   TSP=    15      ;STACK POINTER
.DUSR   TFP=    16      ;FRAME POINTER
.DUSR   TSL=    17      ;STACK LIMIT
.DUSR   TSO=    20      ;OVERFLOW ADDR

.DUSR TLN=TKLAD-TPC+1   ;SHORT TCB LENGTH
.DUSR TLNB= TSO-TPC+1   ;LONG TCB LENGTH


; TASK STATUS BITS  (IN TPRST)

.DUSR   TSSYS=  1B0     ;SYSTEM BIT
.DUSR   TSSUSP= 1B1     ;SUSPEND BIT
.DUSR   TSXMT=  1B2     ;XMT/REC AND OVERLAY BIT
.DUSR   TSRDOP= 1B3     ;.TRDOP BIT
.DUSR   TSABT=  1B4     ;ABORT LOCK BIT
.DUSR   TSRSV=  1B5     ;RESERVED
.DUSR   TSUPN=  1B6     ;USER PEND BIT
.DUSR   TSUSR=  1B7     ;USER FLAG BIT
```

```
;
; OVERLAY DIRECTORY
;

.DUSR   OVNDS=  0       ;NUMBER OF NODES

; FOR EACH NODE:

.DUSR   OVRES=  1       ;CURRENT OVLY(B0-7), USE COUNT(B8-15)
.DUSR   OVDIS=  2       ;# OVLYS (B0-7), LOADING BIT (B8),
                        ; SIZE IN BLKS (B9-15)
.DUSR   OVBLK=  3       ;STRT BLK # IN OVLY FILE FOR FIRST OVLY
.DUSR   OVNAD=  4       ;CORE ADDR FOR NODE(B1-15)
                        ; 1B0 FLAGS VIRTUAL NODE



;
; USER TASK QUEUE TABLE
;

.DUSR   QPC=    0       ;STARTING PC
.DUSR   QNUM=   1       ;NUMBER OF TIMES TO EXEC
.DUSR   QTOV=   2       ;OVERLAY
.DUSR   QSH=    3       ;STARTING HOUR
.DUSR   QSMS=   4       ;STARTING SEC IN HOUR
.DUSR   QPRI=   TPRST   ;MUST BE SAME
.DUSR   QRR=    6       ;RERUN TIME INC IN SEC
.DUSR   QTLNK=  TLNK    ;MUST BE SAME
.DUSR   QOCH=   10      ;CHAN OVERLAYS OPEN ON
.DUSR   QCOND=  11      ;TYPE OF LOAD
.DUSR   QAC2=   12      ;WAKEUP AC2
                        ; 1B0= LOADING, 1B15= DEQUE REQ REC
.DUSR   QTLN=   QAC2-QPC+1
.DUSR   QPEX=   QTLN    ;USER TASK Q AREA EXTENSION



;
; USER PROGRAM TABLE FOR OPERATOR COMMUNICATIONS PACKAGE
;

.DUSR   LPN=    0       ;PROGRAM NUMBER
.DUSR   LOV=    1       ;OVERLAY NUMBER OR -1
.DUSR   LCOND=  2       ;CONDITIONAL/UNCONDITIONAL LOAD
.DUSR   LTPR=   3       ;TASK ID (LEFT) + PRIORITY (RIGHT)
.DUSR   LPC=    4       ;PROGRAM COUNTER

.DUSR   LTLN= LPC-LPN+1 ;TABLE LENGTH

.DUSR   LPEX=   LTLN    ;COMMUNICATIONS EXTENSION AREA START
```

```
;
; TUNING FILE DISPLACEMENTS
;

.DUSR   .TUN=0              ;OFFSET TO NUMBER WORD IN PAIR
.DUSR   .TUC=.TUN+1         ;OFFSET TO 1ST COUNT IN PAIR
.DUSR   .TUP=.TUC+2         ;OFFSET TO 2ND COUNT OF PAIR
.DUSR   .TUNX=.TUP+2        ;LENGTH OF COUNT PAIR

.DUSR   .TUNSTK=1                   ;NUMBER STACKS IN SYSTEM
.DUSR   .TUSTK= .TUNSTK+.TUC-.TUN   ;STACK COUNT
.DUSR   .TUPSTK=.TUNSTK+.TUP-.TUN   ;STACK PEND COUNT

.DUSR   .TUNCEL=.TUNSTK+.TUNX   ;NUMBER CELLS IN SYSTEM
.DUSR   .TUCEL= .TUNCEL+.TUC-.TUN   ;CELLS COUNTS
.DUSR   .TUPCEL=.TUNCEL+.TUP-.TUN

.DUSR   .TUNBUF=.TUNCEL+.TUNX   ;BUFFERS, EXCLUDING TUNING BUFFERS
.DUSR   .TUBUF= .TUNBUF+.TUC-.TUN   ;COUNTS
.DUSR   .TUPBUF=.TUNBUF+.TUP-.TUN

.DUSR   .TUNOV= .TUNBUF+.TUNX   ;OVERLAYS
.DUSR   .TUCV=  .TUNOV+.TUC-.TUN
.DUSR   .TUPOV= .TUNOV+.TUP-.TUN

.DUSR   TULEN=.TUNOV+.TUNX
```

End of Appendix

APPENDIX F

BOOTSTRAPPING DOS FROM DISK

FROM A COLD SYSTEM

For a diskette-based system, ensure that the write-protect hole of your system diskette is covered. Turn DP0 ON, and power up all other equipment. Insert the system diskette into slot DP0, and make sure that no other slot has the same number. Note: never change the number of a disk while it is initialized. RELEASE it before you change its number.

For a disk-based system, make sure that a system disk in in DH0 (microNOVA) or DP0 (NOVA); press the LOAD switch to READY, then wait for the READY light.

The following steps apply to the first disk or diskette controller on your system which runs DP0/DP1 or DH0/DH0F on a microNOVA or DP0 through DP3 on NOVAs. If your system has a second controller, and you want to bootstrap from it, use device code 73_8 instead of 33_8. For the third and fourth controllers (microNOVA only), use 30_8 and 70_8, respectively.

For the first diskette controller on a microNOVA with hand-held console, press RESET, CLR D, enter 33; then press PR LOAD. For the first hard-disk controller, press RESET, CLR D, enter 100027, then press PR LOAD.

If you are using the console debug option to the terminal asynchronous controller board then:

- Type in: 100027) for hard disk

  or

  33L) for diskette

If none of the above is present, the program load option to the CPU board must be present.

- Set the jumpers for device code 27 (hard disk) or 33 (diskette) as described in the appropriate SYSGEN chapter.
- Hit the FRONT PANEL ROCKER SWITCH to the 'PL/START' position.

For the second microNOVA hard-disk controller, substitute 67 for 27. For the second, third, or fourth microNOVA diskette controllers, substitute 73, 30 or 70 respectively for 33.

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. It invokes BOOT.SV, which 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 DH0 or 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".

   Note: you cannot bootstrap a system which is in a user directory.

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)? 9 20 78)
TIME (H:M:S)? 14 32 30)
R
```

To sign off the current system, type:

RELEASE master-directory-name)

or

RELEASE %MDIR%)

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.

As described in Chapter 7 and 8, under "Releasing the System", crucial DOS directories may be wrong when you bootstrap DOS after an exceptional status or crash; thus we recommend that you fully initialize (INIT/F) all disks that were initialized when the exceptional status or crash occurred.

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. Run the disk initializer program (DOSINIT.SV) on the system disk.

2 SYS.DR error detected while accessing a directory on the system disk. 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, process the disk 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 by a continually-interrupting device, which, in turn, is often caused by the wrong mask in a user-defined device. Locate and remove the problem device.

4 Inconsistent system data, such as illegal device address or partial overwrite by a user program.

5 System disk data error; run disk reliability test.

6 System disk timeout. Make sure that the system disk is ON, and on-line. If there are no obvious errors, run disk reliability test.

7 Illegal device address on the system disk. This can be caused by misreading of the disk. 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.

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 AC2 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 $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 a backup diskette. 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 ***@***D**AZZZQZ
01040  054532 176660 054524 024466 125224 000420 034012 020742 YZ**YT)@****@*|*
01050  163000 042741 025415 021414 101005 045414 020454 025405 **E***H***K*|,**
01060  106414 000404 025406 041406 045405 034012 025405 044546 ******C*K*@***I*
01070  030436 051405 025377 044017 025414 044537 030431 051414 1*@***H***I*1*@*
01100  060177 024016 045010 024426 044505 020536 040422 040422 **(*J*)*IE|*A*A*
01110  034502 152120 021420 043410 175400 153102 000774 137000 9B*P**@****B****
01120  025776 044511 014467 000410 176440 002466 005731 005756 **II*7*** *@****
01130  000011 000011 001014 015766 000406 011766 030453 004434 ************1***
01140  004404 000763 034450 000535 054455 006447 126400 044450 ****@(**Y**!**I(
01150  006446 024446 034475 137000 025400 006443 024441 010440 *&)&@******@)!*
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 002307 177770 ***************
  :      :      :      :      :      :      :      :      :
```

END OF APPENDIX

APPENDIX H

RDOS - DOS COMPATIBILITY CONSIDERATIONS

Read this appendix before you try to access disks or 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.

All disks and diskettes created under DOS are entirely compatible with RDOS. Disk(ettes) created under RDOS may not be compatible with DOS if certain RDOS features not supported under DOS are used. These features are:

1. DOS cannot access disk partitions. If a disk(ette) to be used under DOS contains partitions, you cannot initialize the partitions.

2. Diskettes used under DOS must not have any blocks in the RDOS bad block table. (Hard disks DO have an RDOS-compatible bad block table.) When a diskette is initialized under DOSINIT (the DOS initializer program), an empty bad block list is created. If DOS finds a non-empty list, it returns error ERBAD. 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 MOVE the files to a good diskette under RDOS. 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 diskettes have 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 ran RDOS DKINIT on the diskette, the diskette cannot be used with DOS. The only remedy is to copy the diskette files using the CLI MOVE command to a diskette that has the correct frame size.

DETAILS

DOS does not support:

- Any disk type other than 6030 and 6038 series diskettes or 6095 and 6045 dual-plotter disk subsystems.

- 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.)

- System 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:

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 three copying commands: the CLI COPY, and the DOSINIT COPY and DUPLICATE.

The DOS initializer, DOSINIT, differs from the RDOS initializer, DKINIT, as described above. You can configure any 4234, 4233 or similar disk as a pseudo-diskette by answering 6030 to the RDOS DKINIT query, but you'll be limited to 608 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 or remove the diskette 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.

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.

If you write on a label after it has been applied, use a felt-tipped pen - never a ball-point pen or pencil.

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 How to Generate Your DOS System (093-000222).

END OF APPENDIX

APPENDIX J

ADVANCED MULTITASK PROGRAMMING

For most multitask application programs, the features described in Chapter 5 will suffice. You need read this appendix only if:

- You want to write your own multitasking primitives (task calls)
- Your tasks require one or more special resources (for example, floating-point hardware), that the system does not provide for in a TCB.

The features described in this appendix can:

- provide more programming flexibility than the standard features alone, without requiring you to modify the task monitor sources; and
- provide this flexibility in a system-independent way.

You can use the calls in this appendix to develop application programs for any system configuration (DOS, RTOS or RDOS, NOVA or microNOVA). All you need do to reconfigure for a different system is load a program (via RLDR) with the appropriate system libraries.

Before you proceed, you should be familiar with the material in Chapter 5.

DEFINITIONS

The following definitions relate to tasks and task states; they apply throughout DOS, RTOS, and RDOS.

General Terms

Task Resources are those storage elements of the computer, such as accumulators and special memory locations, which two or more tasks must share. The task scheduler allows such sharing by ensuring that the proper values for each task's resources appear in the actual storage elements of the computer while the task is executing. When a task is not executing, the current values of its resources are held in its TCB.

Rescheduling is the process of selecting and executing the highest priority ready task. The task scheduler performs rescheduling after each task call, after receiving control from the system following an interrupt, and when a system call completes. You can suppress rescheduling via the .DRSCH or .SINGL task calls, or by entering scheduler state, as described below. If you have not disabled rescheduling, you must assume that it can happen at any time.

A task swap occurs during rescheduling when the task scheduler determines that it should execute a different task from the one which was last executing. If the task which was executing was not terminated (by .KILL, etc.), the scheduler saves the current state of the task's resources in the task's TCB. The scheduler then restores the former state of the new task's resources from its TCB. Then, the scheduler places the new task's TCB in the active TCB chain at the end of its priority class, so that the next time rescheduling occurs the task will be considered for execution only after all other tasks in its class. Finally, the new task receives CPU control and becomes the current task.

CTCB is a location maintained by the scheduler which contains the address of the current task's TCB. If no task is currently active (for example, if all tasks are suspended or rescheduling is occurring), CTCB contains the address of the most recently executing task's TCB, if that task was not terminated. If it was terminated, then CTCB contains 0. Thus CTCB identifies the task to which the current values of task resource storage elements belong; 0 means that these values are no longer valid.

CTCB is a page zero location. You can access it as follows to obtain the TCB address for the current task:

```
.EXTD CTCB
LDA ac, CTCB
```

Location USTCT in the User Status Table (UST) also contains the current TCB address. However, you should use CTCB instead of USTCT.

The hardware stack is an area of memory that you access via special hardware registers. On a NOVA 3 or micro-NOVA computer, these registers are the stack and frame pointers and location 42 (octal), which the system interprets as the stack limit. DOS treats the hardware stack registers as a task resource, thus they are available for use by all tasks.

A reentrant section of code (sequence of instructions) allows another task to enter this code before the original task exits. Code which several tasks can access is reentrant only if each task has its own local storage, which no other task executing the code can access. Giving each task its own stack area and using the stack for local storage is a common way to achieve reentrancy.

State Definitions

User state is the normal state for an application program. This is the state from which system and task calls are made, as described in Chapter 5. Code must be reentrant in user state if more than one task will use it. In this state, task execution is suspended on an interrupt if a higher priority task is ready for execution; it is also suspended on a system or task call. A task in user state can use the User Stack Pointer (USP, location 16 octal) and the hardware stack; it can also examine (but not modify) CTCB and the current TCB. If there are no indicators of other states, the program is in user state.

Singletask state is used occasionally for a critical section of an application program. You enter this state via the .SINGL task call; it prevents other tasks from gaining control. However, interrupts continue to execute. A task can issue system calls from singletask state as well as from user state; it can also issue any task call except .MULTI or one which would kill or suspend itself. If it issues .MULTI, or kills or suspends itself, the program enters user state. Code executed from singletask state need not be reentrant. It can use USP, the hardware stack, CTCB, and the current TCB as it can in user state. If location SM.SW contains a nonzero value, the program is in singletask state.

Scheduler state is the normal state for task call code. An interrupt can cause temporary loss of control, but, unlike user and singletask states, control returns to the point of interruption without rescheduling. Thus, scheduler state ensures that no other task will get control, although interrupts continue. Control does not pass to the scheduler on return from an interrupt, and the active TCB chain is not scanned. In scheduler state, there is no TCB associated with the executing task. Code executed in scheduler state need not be reentrant. It should not use USP or the hardware stack; however it can both read and modify CTCB and the current TCB, subject to restrictions described later. A task is in scheduler state for DOS if location USTPC contains a value other than 0 or 1. For RTOS, location .SYS. is nonzero in scheduler state.

Use interrupt-disabled state to perform critical manipulation of TCB data or the active TCB chain. There is no way for a task in this state to lose control of the CPU, even temporarily.

CODING YOUR OWN TASK CALLS

TCB and Status Bits

Two status bits of word TPRST in a TCB are allocated for your use; you can use them to extend the standard features. Bit TSUPN, the user suspend bit, will prevent a task from running when set. Bit TSUSR, the user status bit, will not affect task readiness but is available for storing an additional piece of task-related information.

Also, word TELN is available for your own use. A typical use for TELN is to store the address of a "TCB extension" in it. This allows you to store as much additional task-related information as you need.

Scheduler Calls

The scheduler calls defined below are, like task calls, external symbols which you must identify as .EXTN in your source program. The relocatable loader (RLDR) resolves them at load time, according to system type.

Enter Scheduler State (EN.SCHED)

To enter scheduler state from user or singletask state:

```
;AC3 not equal to 0 or 1
;
EN.SCHED
;Returns here with all ACs and carry preserved.
```

A task already in scheduler state can safely reissue EN.SCHED, but no change in state will occur.

Task State Save (.TSAVE)

For a task in scheduler state, this call saves the ACs, carry, and program counter in its TCB. The PC saved is the value in bits 1-15 of AC3 at the time of the last EN.SCHED.

```
;ACs, carry, PC to be saved.
.TSAVE
;Returns here with AC0, AC1, and carry unchanged,
;         AC2 = value that was in AC3 at
;         time of last EN.SCHED;
;         AC3 = TCB address.
```

EN.SCHED and .TSAVE are meant to be used together at the start of code which implements a user-designed task call. For a task call with error return, you might use them this way:

```
        .ENT .TASK, T.ASK
        .EXTN EN.SCHED, .TSAVE
        .ZREL
.TASK=  JSR @.
        T.ASK
        .NREL
T.ASK:  INC 3,3             ;Assume normal return.
        EN.SCHED            ;Enter scheduler state.
        .TSAVE              ;Save task state.
          .
          .
```

For a task call without an error return, you would omit the INCrement instruction.

Leave Scheduler State Normally (RE.SCHED)

When you successfully complete the processing for a task call, issue RE.SCHED to exit to the scheduler for rescheduling. Use RE.SCHED in scheduler state.

```
;No input.
RE.SCHED
;No return.
```

Leave Scheduler State Abnormally (ER.SCHED)

When you detect an error during task call processing, place an error code in AC2 and exit to the scheduler via ER.SCHED. This returns control to the location preceding the one specified by TPC, and passes back the error code in AC2. Use ER.SCHED in scheduler state.

```
;AC2 = error code.
ER.SCHED
;No return.
```

Enter Interrupt-Disabled State (INT.DS)

Use INT.DS to enter interrupt-disabled state from scheduler state.

```
;No input.
INT.DS
;Returns here with AC0, AC1, AC2,
;         and carry unchanged.
```

Leave Interrupt-Disabled State (INT.EN)

To leave interrupt-disabled state and return to scheduler state, use INT.EN.

```
;No input.
INT.EN
;Returns here with AC0, AC1, AC2,
;         and carry unchanged.
```

Task ID Search (ID.SRCH)

Use ID.SRCH to search for a task with a given ID. You can issue ID.SRCH in either scheduler or interrupt-disabled state.

```
;AC1, right byte = ID of sought task.
ID.SRCH
;Error return here, AC2 = error code.
;Normal return here, with AC2 = TCB address
;         of sought task.
```

For both returns, AC0 and carry are preserved -- the left byte of AC1 is zeroed and the right is preserved.

HANDLING ADDITIONAL TASK RESOURCES

This section tells you how to manage task resources that are not automatically managed by the system. At certain points in its scheduling process, the scheduler calls out to routines which you may supply to handle your additional task resources. These call-outs are described in the first section, below.

If floating-point hardware and/or a block of contiguous memory locations are among the resources you need, you can simply use a handler supplied in SYS.LB. This is explained in the second section below.

If you want to handle additional task resources while using operator communications, see the final section.

Task Scheduler Call-outs

To use any call-out described below, write, assemble, and load a routine of the appropriate name and function. You must insert the name of the routine in the RLDR command line before RLDR searches SYS.LB (by default, this occurs at the end of the command line). If you do not supply a routine, RLDR will load a dummy routine, which does nothing, from SYS.LB.

Task Initiation Call-out (TSK.X)

This call-out allows you to endow a new task with additional task resources. When the scheduler initiates a task, it first removes a TCB from the free TCB chain. Then it initializes certain parts of the TCB, as described later under "Task Control Block Values". The scheduler then calls out to your TSK.X routine in scheduler state.

Your TSK.X routine can initialize certain parts of the TCB and change the parts the scheduler initialized (subject to the restrictions mentioned in the TCB Values section). On a normal return, the scheduler links the TCB for the new task, as modified by your TSK.X code, into the active TCB chain.

The scheduler transfers control to address TSK.X with the accumulators set up as follows:

AC0	contains the value passed to .TASK in AC2. AC0 is irrelevant if .QTSK initiates the task.
AC1	contains -1 if .TASK initiates the task or the address of the task queue table if .QTSK initiates the task.
AC2	contains the address of the TCB for new task.
AC3	contains the (error) return address.

The routine you supply with entry address TSK.X need not preserve accumulators or carry. If you detect an error, place an error code in AC2 and return control to the location whose address you received in AC3. On a normal return, return control to the location whose address is one greater than the one you received in AC3. For example:

```
          .ENT TSK.X
          .NREL
TSK.X:    STA 3, RTNAD        ;SAVE RETURN.
            .
            .
          COM# 1,1,SZR        ;.TASK OR .QTSK?
          JMP QUE
TSK:        .                 ;HANDLE .TASK CASE.
            .
            .
QUE:                          ;HANDLE .QTSK CASE.
            .
            .
            .
BAD:      LDA 2, CODE         ;ERROR RETURN.
          JMP @RTNAD

GOOD:     ISZ RTNAD           ;NORMAL RETURN.
          JMP @RTNAD

RTNAD:    .BLK 1
```

When you return an error indication, and .TASK is initiating the task, the task will not be initiated, and its TCB will return to the free TCB chain; the error code you place in AC2 will be passed to the task which issued .TASK. When you return an error indication and .QTSK is initiating the task, the system will try again one second later.

Task Termination Call-out (TRL.X)

The TRL.X call out frees a task's additional resources when a task is terminated -- typically those resources you assigned in a TSK.X routine. The scheduler calls this routine in scheduler state whenever a task is being killed, with the task's TCB already unlinked from the active chain but not yet restored to the free chain. The scheduler transfers control to address TRL.X with AC2 set up as follows:

AC2	contains the TCB address of task being killed.
AC3	contains the return address.

The routine you supply with entry address TRL.X need not preserve accumulators or carry. When you have finished your processing, return control to the location whose address you received in AC3. There is no way to signal an error from TRL.X.

Task Swap Call-out (ESV.X)

This call-out allows you to save and restore additional task resources as needed when a task swap occurs. The scheduler calls the routine in scheduler state and transfers control to address ESV.X with the accumulators set up as follows:

AC2	TCB address for task losing control, or 0 if no task is losing control (as described under CTCB, earlier).
CTCB	TCB address of task gaining control.
AC3	return address.

The routine you supply with entry address ESV.X need not preserve accumulators or carry. When you have finished processing, return control to the location whose address you received in AC3. There is no way to signal an error from ESV.X.

A 0 passed to you in AC2 indicates that there is no valid most recently active task whose resources you would need to save. This situation occurs as the default task is initially selected for exectuion. ESV.X will be called with 0 in AC2 and the TCB address for the default task in CTCB. It also occurs after a task is terminated, because the terminated task's resources were freed by TRL.X, and are no longer meaningful to ESV.X.

Additional Resource Handler

The system library (SYS.LB) contains an ESV.X routine, which partly provides for the additional task resources of floating-point hardware and a block of contiguous storage words. To load this module, insert a .EXTN ESV.X in any source module whose name will occur in the RLDR command line before SYS.LB is searched.

For each task that needs access to the floating-point hardware, you must provide a block of words to store the task's values for its floating point state. The size and contents of this block depend on the kind of computer you use. For a NOVA computer, the block has this format:

FPAC	4 words
TEMP	4 words
Status	1 word

To provide for a block of contiguous memory locations as an additional task resource, you must define two symbols with .ENT and give them the following values:

ESV.S	must equal the starting address of the block.
ESV.Z	must equal the number of words in the block.

Also, you will need to provide a block of memory that is ESV.Z words long for each task that is to use this additional resource.

Finally, for each task that will use either the floating-point hardware or contiguous memory locations, you must initialize offset TELN in the TCB, within the TSK.X routine that you supply. The value you place in TELN depends on the task's needs.

- If TELN contains either 0 or 100000 (octal), neither the floating-point resource nor contiguous memory resource will be handled. Thus, since DOS initializes TELN to 0, you need not change it for a task which needs neither resource.

- If the task requires floating-point hardware but not the contiguous memory, set TELN to the indirect address of the appropriate floating-point block described earlier.

- If the task needs the contiguous memory but not the floating-point hardware, set TELN to the (direct) address of a block of words ESV.Z+1 words long, and set the first of these words to either 0 or 100000 (octal). The contiguous memory locations will be saved in the remaining words of this block.

- If the task requires both resources, set TELN as immediately above, but set the first word of the block to the (direct) address of a floating-point save area as described above.

When you initialize TELN, you can also initialize the contents of these save areas as well. The values that your TSK.X routine places in these areas will be the initial values when the task being initiated starts executing.

Restrictions and Warnings

The system-supplied additional-resource handler assumes that TELN is set up properly for either or both of the resources; it does not prevent an unprepared task from using one of these resources inadvertantly. If this occurs, results are unpredictable.

For a task to use these resources, you must set up the task's TELN in your TSK.X routine. You cannot change a task's TELN after the task has been initiated.

Providing Even More Resources

If a task needs resources in addition to floating-point hardware and contiguous memory locations, you can write your own ESV.X routine to handle the extra resources and use the supplied handler as a subroutine. From within your own ESV.X routine, call out to the supplied handler, using the alias ESV.A instead of ESV.X, with the accumulators set up appropriately.

Extending the Task Queue Table

When you issue a .QTSK task call, you must pass in AC2 the address of a task queue table; see Chapter 5 for a description of the format and length of this table. When the scheduler calls out to TSK.X, it passes the queue table address in AC1. Thus, you can append additional information to the queue table (that is, you can supply a longer table), and access this information from within TSK.X.

To do this, define symbol LPN.X with .ENT and set it equal to the number of additional words for the task queue table. On a .QTSK command, these words will be copied, in order, to the end of the associated task queue table, where they will be accessible to TSK.X.

TASK CONTROL BLOCK VALUES

Table J-1 describes the initial values the scheduler assigns to words in a TCB and when these values can be changed during a task's lifetime. A bracketed number indicates a note. Entry Name is the symbol in PARU.SR which represents the offset within the TCB. Initial Contents describes the value placed there by the scheduler and seen on input to TSK.X. In column .TASK? a "Yes" means that TSK.X can set or change the contents of this word if .TASK is initiating the task; "No" means that TSK.X can't change this word. A "Yes" or "No" in column .QTSK means the same thing for .QTSK. In column Later?, a "Yes" means that this word can be changed later in the task's life; "No" means it cannot be changed. In the Initial Contents column, the entry applies to both .TASK and .QTSK, unless there are two entries separated by a slash (/); in this case, the first entry applies to .TASK and the second to .QTSK.

Table J-1. TCB Words and How They Can Be Changed

Name	Initial Contents	.TASK?	.QTSK?	Later?
TPC	B0-14:Start addr;B15:Undefined	Yes	Yes	Yes
TAC0	Undefined / System-maintained	Yes	No	Yes
TAC1	Undefined / System-maintained	Yes	No	Yes
TAC2	AC2 at .TASK/System-maintained	Yes	No	Yes
TAC3	K.ILL[1] / System-maintained [2]	Yes	No	Yes
TPRST	B0-7:0 ; B8-15: Start pri.	Yes	Yes	[3] [4]
TSYS	System-maintained	No	No	No
TLNK	System-maintained	No	No	No
TUSP	Undefined	Yes	Yes	[5]
TELN	0	Yes	Yes	[6]
TID	Task identifier	No	No	No
TTMP	System-maintained	No	No	No
TKLAD	0	Yes	Yes	Yes
TSP	Undefined	Yes	Yes	[5]
TFP	Undefined	Yes	Yes	[5]
TSL	Undefined	Yes	Yes	[5]
TSO	Undefined	Yes	Yes	[5]

Notes:

1. Address K.ILL is the entry for the .KILL task call code. This address is placed in TAC3 so that a task can kill itself by simply returning to the address it receives in AC3.

2. At TSK.X time for a task initiated by Q.TSK, TAC3 does not contain the address K.ILL. However, after TSK.X completes, but before the new task gains control, the scheduler places the address K.ILL in TAC3, so that the task's initial AC3 will be correct (see also note 1).

3. The modification of a task's status bits must be an indivisible operation. The interrupt world can modify the status bits on suspended tasks only. Thus, modifying the status bits of a ready task must be a "task-indivisible" operation, while modification of a suspended task must be an "interrupt-indivisible" operation. Scheduler state and interrupt-disable state both provide task- and interrupt-indivisibility.

4. Don't modify the priority portion of TPRST; use the task call .PRI instead.

5. Because these values are saved and restored only on task swaps (not by .TSAVE, as are the accumulators, for example), it is meaningless to change the values while in scheduler state. Instead, you should change the actual storage locations. Change USP (16 octal) instead of TUSP. On a NOVA 3 or microNOVA computer, change the hardware stack and frame pointers and locations 42 (octal) (stack limit) and 46 (octal) (instruction trap PC).

6. As mentioned earlier, you cannot change word TELN after TSK.X time, if you use the additional resource handler (ESV.X routine) supplied in SYS.LB.

End of Appendix

APPENDIX K

MULTITASK PROGRAMMING EXAMPLES

This appendix contains two examples of multitask assembly language programs.

DUCLK PROGRAM

The first example, Figure K-1, is DUCLK, which creates a second task and uses system call .DUCLK to give the second task control on a regular basis. Real-time clock cycles provide this basis, and you can specify any reasonable number of RTC cycles if you want to type in the program.

DUCLK also shows the transmit (.IXMT) and receive (.REC) mechanism in action; generally, this works the same way for .XMT and .IXMTW as it does here.

When it runs, DUCLK creates a second task, then types the message:

I"M THE MAIN PROG., ABOUT TO WAIT FOR A CHAR.

on the console. Then it defines a user clock and waits for a character to be typed. The second task gets control as the main program waits, and issues .REC, which suspends the second task until it receives a message from the user clock routine. Each time the clock interval expires, the clock routine issues .IXMT to the second task's receive address. This wakes up the second task, which types the message:

I'M TASK: THE CLOCK INTERVAL HAS EXPIRED.

on the console. TASK then loops back to issue .REC again, and, when the interval has expired again, prints the message again. The program defines the interval as 20 seconds, thus for a system with an RTC frequency of 10Hz (normal for NOVA DOS systems), the message appears every two seconds. For a microNOVA internal clock, you might want to specify a value somewhere around 1000.

TASK continues typing the "EXPIRED" message until someone presses a console key. This wakes up the main task, which removes the user clock and returns to the CLI.

EXAMPLE PROGRAM

The second program, EXAMPLE, illustrates the multitask overlay calls and task queuing. See Figure K-2. EXAMPLE has two distinct tasks, AGAIN and READ. AGAIN examines the console for input. It recognizes only two characters, B and C. If AGAIN reads a C, it loads overlay COMP, which types the message "I AM A DATA GENERAL COMPUTER" on the console. If AGAIN reads a B, it returns to the CLI, which kills all tasks and closes all channels.

The second task, READ, resides in overlay READ. It is queued by .QTSK to execute every 3 seconds. It reads the console switches or keys, and, if they have changed, it types the message "CONSOLE SWITCHES CHANGED" on the console.

EXAMPLE creates READ at a higher priority than it has, so that if READ is ready to run (every three seconds), AGAIN will be momentarily unable to return control to the CLI.

Sample console output from EXAMPLE, after someone has changed a console data switch three times, then typed C, is:

```
CONSOLE SWITCHES CHANGED
CONSOLE SWITCHES CHANGED
CONSOLE SWITCHES CHANGED
I AM A DATA GENERAL COMPUTER
```

The following pages show the assembler listings for EXAMPLE, overlay READ, and overlay COMP.

To run EXAMPLE, you must type the EXAMPLE code into a file named EXAMPLE.SR, the READ code onto a file named READ.SR, and the COMP code into file COMP.SR. Then, assemble the three files and load them.

The command line used to assemble EXAMPLE was:

MAC/L (EXAMPLE,READ,COMP))

We used the pseudo-op ".NOLOC 1" before all byte pointers to text strings; this suppressed listing of these, but did not print them. The ASM assembler does not recognize .NOLOC, so if you want to use ASM, omit the ".NOLOC 0" statement as you type in the program.

The load line was:

RLDR 2/K EXAMPLE [READ,COMP])

```
01                          ;This program sets up a timeslicing task which
02                          ; receives control via a user clock routine,
03                          ; defined with system call .DUCLK.
04                          ; The load line is "RLDR 2/K filename".
05
                            .TITL DUCLK
07                          .NREL
08                          .EXTN .TASK, .REC, .IXMT, .UCEX  ; Get task code
09                                                           ;  from SYS.LB.
10          000001          .TXTM 1             ;Pack text bytes left to right.
11
12          000024          INTVL=20.           ;Define number of RTC pulses for
13                                              ; clock routine.  (For microNOVA
14                                              ; internal clock, try "1000.")
15
16                          ; Open console output and create new task.
17
18 00000'020534 INIT:       LDA     0, NTTO ; Get console name.
19 00001'126400              SUB     1, 1    ; Zero AC1 for default disable mask.
20 00002'006017              .SYSTM          ; Call system.
21 00003'014000              .OPEN   0       ; Open console output on channel 0.
22 00004'000447              JMP     ER      ; Process any error.
23
24 00005'024426              LDA     1, .TSK ; Get starting address of new task.
25 00006'102400              SUB     0, 0    ; A task ID of 0, priority of 0.
26 00007'077777              .TASK           ; Create the new task.
27 00010'000443              JMP ER          ; Process error.
28 00011'020445              LDA     0, .MES1 ; Get start address for "WAITING"
29                                             ;  message.
30 00012'006017              .SYSTM          ; Call system.
31 00013'017000              .WRL    0       ; Write "WAITING" message to console.
32 00014'000437              JMP     ER      ; Process error.
33
34                          ; Define user clock and issue GCHAR to wait for
35                          ;  console character.
36
37 00015'020523              LDA     0, .INTVL  ; Get number of RTC cycles for
38                                              ;  clock interval.
39 00016'024425              LDA     1, .RCUT   ; Get address of user clock routine.
40 00017'006017              .SYSTM             ; Call system.
41 00020'021001              .DUCLK             ; Define the user clock - which will
42                                              ;   freeze things and execute clock
43                                              ;   routine after "INTVL" cycles.
44 00021'000432              JMP     ER         ; Process error.
45 00022'006017              .SYSTM             ; Call system.
46 00023'007400              .GCHAR             ; Wait for a console character,
47                                              ;  suspending yourself until you
48                                              ;  receive one.  This gives control
49                                              ;  to the other task.
50 00024'000427              JMP     ER         ; Process error.
51
52                          ;On receipt of console char, wake up, remove clock,
53                          ;   and return to CLI.
54
55 00025'006017              .SYSTM          ; Call system.
56 00026'021002              .RUCLK          ; Remove the user clock.
57 00027'000424              JMP     ER      ; Handle error.
58 00030'006017              .SYSTM          ; System,
59 00031'004400              .RTN            ; Return to the CLI.
60 00032'000400              JMP .           ; Reserved, never taken.
```

Figure K-1. DUCLK Program Listing

```
 0002 DUCLK
01
02                          ;Address of task.
03
04 00033'000034'.TSK:       TASK
05
06                          ; Task code.
07
08 00034'020506 TASK:       LDA     0, .MADDR       ; Get receive address.
09 00035'077777             .REC                    ; Wait for message ,IXMT
10                                                  ;   from clock routine.
11 00036'020451             LDA     0, .MES2        ; When awakened by .IXMT, get
12                                                  ;  addr of "INTERVAL EXPIRED"
13                                                  ;  message.
14 00037'006017             .SYSTM                  ; Call system.
15 00040'017000             .WRL    0               ; Write "EXPIRED" message to
16                                                  ;   console.
17 00041'000412             JMP     ER              ; Process error.
18 00042'000772             JMP     TASK            ; Now do the .REC, wait, and
19                                                  ;   write again.
20                          ; Address of user clock routine.
21
22 00043'000044'.ROUT:      ROUT
23
24                          ; Code for user clock routine.
25
26 00044'054475 ROUT:       STA     3,  RETRN       ; Save return address.
27 00045'020475             LDA     0, .MADDR       ; Get TASK's receive address.
28 00046'024476             LDA     1, C1           ; Get nonzero message.
29 00047'077777             .IXMT                   ; Transmit message, awakening
30                                                  ;  TASK.
31 00050'000403             JMP     ER              ; Process error.
32 00051'034470             LDA     3,  RETRN       ; Restore AC3.
33 00052'077777             .UCEX                   ; Leave user clock routine.
34
35                          ;Error handler.
36
37 00053'006017 ER:         .SYSTM
38 00054'006400             .ERTN           ;Report error through the CLI.
39 00055'000400             JMP             ;Reserved, never taken.
40
41                          ;Text for messages.
42
43                                  ; To suppress text listing for MAC
44                                  ;   assembler, insert ".NOLOC 1" here.
45                                  ;   For ASM, insert nothing.
46
47 00056'000136".MES1:      .+1*2
48 00057'044447             .TXT "I'M THE MAIN PROG., ABOUT TO WAIT FOR A CHAR.<15>"
49 00107'000220".MES2:      .+1*2
50 00110'044447             .TXT "I'M TASK; THE CLOCK INTVL HAS EXPIRED.<15>"
51 00134'000272"NTTO:       .+1*2           ; Byte pointer to console output name.
52 00135'022124             .TXT "$TTO"     ; Console output name.
53       000000             .NOLOC  0       ; Insert this for MAC, not for ASM.
54
55 00140'000024 .INTVL:     INTVL           ; Address which contains the interval.
56
57 00141'000000 RETRN:      0               ; Return address from clock routine.
58
59 00142'000143'.MADDR:     MADDR           ; Address for .IXMT, .REC message.
60 00143'000000 MADDR:      0               ; Contents of message address.


 0003 DUCLK
01
02 00144'000001 C1:         1               ; A nonzero value is all that's
03                                          ;   needed for the message.
04
05                          .END    INIT

**00000 TOTAL ERRORS, 00000 PASS 1 ERRORS
```

Figure K-1. DUCLK Program Listing (Continued)

```
                        .TITL   EXAMPLE
02                      .ENT    SWTCH,AGAIN,ICOMP,IREAD,ERROR
03                      .EXTN   OCOMP,OREAD,COMP,READ
04                      .EXTN   .PRI,.QTSK,.TOVLD
05      000001          .TXTM 1         ;PACK BYTES LEFT TO RIGHT.
06                      .NREL
07
08              ; OPEN CONSOLE OUTPUT AND OVERLAY FILE FOR I/O.
09
10 00000'020440 START:  LDA     0,NTTO  ;SPACE FOR THE CONSOLE OUTPUT FILENAME.
11 00001'006017         .SYSTM          ;SYSTEM,
12 00002'021037         .GCOUT          ;  GET THE CONSOLE OUTPUT NAME.
13 00003'000471           JMP   ERROR   ;   CAPTURE ANY ERROR.
14
15 00004'126400           SUB   1,1     ;SET DEFAULT DEVICE CHARACTERISTIC MASK.
16 00005'006017         .SYSTM          ;OPEN THE CONSOLE OUTPUT FILE
17 00006'014000         .OPEN   0       ;  ON CHANNEL 0.
18 00007'000465           JMP   ERROR   ;  ERROR RETURN.
19
20 00010'020434         LDA     0,OFILE ;GET OVERLAY FILENAME.
21 00011'030476         LDA     2,OCHAN ;GET CHANNEL NUMBER FOR OVERLAY FILE.
22 00012'006017         .SYSTM          ;OPEN OVERLAY FILE ON THE
23 00013'012077         .OVOPN  CPU     ;  SPECIFIED CHANNEL.
24 00014'000460           JMP   ERROR   ;   ERROR.
25
26              ;PROCEED WITH MAIN PROGRAM.
27
28 00015'006017         .SYSTM          ;READ THE INITIAL CONTENTS OF
29 00016'021065         .RDSW           ; THE SWITCHES OR KEYS.
30 00017'000455           JMP   ERROR   ;  ERROR RETURN.
31 00020'040437         STA     0,SWTCH ;SAVE THE INITIAL CONTENTS.
32
33 00021'020432         LDA     0,C40   ;NOW SET YOUR PRIORITY
34 00022'077777         .PRI            ; TO 40.
35
36 00023'030433         LDA     2,RTASK ;SET UP AN OVERLAY TASK TO
37 00024'077777         .QTSK           ; RUN EVERY 3 SECONDS.
38 00025'000447           JMP   ERROR
39
40              ; THIS IS THE MAIN KEYBOARD LISTENER LOOP.
41
42 00026'006017 AGAIN:  .SYSTM          ;WAIT FOR A CHARACTER FROM THE
43 00027'007400         .GCHAR          ; CONSOLE.
44 00030'000444           JMP   ERROR
45 00031'024423         LDA     1,B     ;WAS THE CHARACTER A "B"?
46 00032'122415         SUB#    1,0,SNR ;
47 00033'000436           JMP   BYE     ;  YES, RETURN TO THE CLI.
48 00034'024421         LDA     1,C     ;WAS THE CHARACTER A "C"?
49 00035'122415         SUB#    1,0,SNR ;
50 00036'000422           JMP   GCOMP   ;  YES, GO TO THE 'COMPUTER' OVERLAY.
51 00037'000767         JMP     AGAIN   ;NO, IGNORE CHARACTER, TRY AGAIN.
52
53
54 00040'000102"NTTO:   .+1*2           ;3 WORDS TO
55 00041'000003         .BLK    3       ;HOLD "$TTO".
56
57 00044'000112"OFILE:  .+1*2           ;OVERLAY FILE NAME.
58 00045'042530         .TXT    "EXAMPLE.OL"
59       000000         .NOLOC  0
60
```

Figure K-2. Example Program and Example Listing

```
 0002 EXAMP
01 00053'000040 C40:    40              ;KEYBOARD TASK PRIORITY.
02 00054'000102 B:      "B              ;ASCII "B".
03 00055'000103 C:      "C              ;ASCII "C".
04 00056'000077'RTASK:  QTAB            ;ADDRESS OF THE 'READ' TASK QUEUE TABLE.
05 00057'000001 SWTCH:  .BLK    1       ;STORAGE FOR LAST SWITCH POSITION.
06
07              ; THIS CODE PROCESSES THE "C" CHARACTER.  IT LOADS AN OVERLAY
08              ; AND TRANSFERS TO A ROUTINE IN THE OVERLAY TO PRINT A MESSAGE.
09
10 00060'020410 GCOMP:  LDA     0,ICOMP ;GET  'COMPUTER' OVERLAY NAME.
11 00061'126400          SUB     1,1     ; SPECIFY CONDITIONAL LOADING.
12 00062'030425          LDA     2,OCHAN ; GET OVERLAY FILE CHANNEL NUMBER.
13 00063'077777          .TOVLD          ;HUMBLY REQUEST SYSTEM ACTION.
14 00064'000410            JMP   ERRCR   ;CCCCCPPPPPSSSSS.
15 00065'006402          JSR     @ACOMP  ;EXECUTE THE SUBROUTINE, THEN
16 00066'000740          JMP     AGAIN   ; GO BACK FOR MORE INPUT.
17
18 00067'077777 ACOMP:  COMP            ;SUBROUTINE ADDRESS IN OVERLAY.
19 00070'077777 ICOMP:  OCOMP           ;'COMPUTER' OVERLAY IDENTIFIER.
20
21
22              ; THIS CODE PROCESSES THE "B" CHARACTER.  IT TERMINATES
23              ; THIS PROGRAM AND RETURNS TO THE CLI.
24
25 00071'006017 BYE:    .SYSTM          ;RETURN TO THE RDOS CLI.
26 00072'004400          .RTN           ;
27 00073'000401          JMP     ERRCR  ;RESERVED, NEVER TAKEN.
28
29
30              ; THIS IS THE ERROR HANDLER.
31
32 00074'006017 ERROR:  .SYSTM          ;LET THE CLI TELL US WHAT'S WRONG.
33 00075'006400          .ERTN          ;
34 00076'000776          JMP     ERRCR  ;NEVER TAKEN.
35
36
37              ; THIS IS THE QUEUE TABLE FOR THE 'READ' OVERLAY TASK.
38
39 00077'077777 QTAB:   READ            ;STARTING ADDRESS FOR THE TASK.
40 00100'177777          -1             ;EXECUTE UNLIMITED NUMBER OF TIMES.
41 00101'077777 IREAD:  OREAD           ;OVERLAY IDENTIFIER.
42 00102'177777          -1             ;STARTING HOUR: RIGHT NOW.
43 00103'000001          .BLK    1      ;STARTING SECOND (DOESN'T MATTER HERE).
44 00104'000430          1B7+30         ;TASK ID OF 1, PRIORITY OF 30.
45 00105'000003          3.             ;RERUN EVERY 3 SECONDS.
46 00106'000001          .BLK    1      ;SYSTEM WORD.
47 00107'000001 OCHAN:  1               ;USE CHANNEL 1 FOR THE OVERLAY FILE.
48 00110'000000          0              ;CONDITIONAL OVERLAY LOADING.
49 00111'000001          .BLK    1      ;SYSTEM WORD.
50
51                       .END   START   ;STARTING ADDRESS IS START.

**00000 TOTAL ERRORS, 00000 PASS 1 ERRORS
```

Figure K-2. Example Program and Example Listing (continued)

```
 0001 READ
                          .TITLE  READ
02                        .ENT    READ
03                        .ENTO   OREAD
04                        .EXTN   SWTCH,ERROR,IREAD
05                        .EXTN   .OVKIL
06        000001          .TXTM   1
07                        .NREL
08
09                ; 'READ' OVERLAY - MONITORS THE SWITCHES FOR CHANGE.
10
11 00000'006017 READ:     .SYSTM          ;READ THE CURRENT STATE OF
12 00001'021065           .RDSW           ; THE CONSOLE SWITCHES.
13 00002'002416             JMP   @ERR    ;  ERROR RETURN.
14 00003'026413           LDA     1,@SWT  ;GET OLD STATE OF SWITCHES.
15 00004'042412           STA     0,@SWT  ;STORE NEW STATE.
16 00005'106415           SUB#    0,1,SNR ;COMPARE.
17 00006'000405             JMP   NEXT    ;  THEY'RE THE SAME--EXIT.
18
19 00007'020412           LDA     0,MESS  ;THEY DIFFER-- GET MESSAGE ADDR.
20 00010'006017           .SYSTM          ;WRITE MESSAGE
21 00011'017000           .WRL    0       ;TO CONSOLE OUTPUT.
22 00012'002406             JMP   @ERR    ;  ERROR RETURN.
23
24 00013'022404 NEXT:     LDA     0,@ORD  ;GET THE OVERLAY IDENTIFIER.
25 00014'077777           .OVKIL          ;RELEASE OVERLAY AND KILL TASK.
26 00015'002403             JMP   @ERR
27
28 00016'077777 SWT:      SWTCH           ;SWITCH STATE SAVED IN EXAMPLE.SR.
29 00017'077777 ORD:      IREAD           ;OVERLAY IDENTIFIER.
30 00020'077777 ERR:      ERROR           ;ERROR HANDLER.
31
32 00021'000044"MESS:     .+1*2
33 00022'041517           .TXT "CONSOLE SWITCHES HAVE CHANGED.<15>"
34        000000          .NOLOC  0
35
36                        .END

**00000 TOTAL ERRORS, 00000 PASS 1 ERRORS
```

Figure K-2. Example Program (continued) and Read Overlay Listing

```
 0001 COMP
                          .TITLE  COMP
02                        .ENT    COMP
03                        .ENTO   OCCMP
04                        .EXTN   ERROR,ICOMP
05                        .EXTN   .OVEX
06        000001          .TXTM   1
07                        .NREL
08
09                ; 'COMPUTER' OVERLAY - PRINT MESSAGE AND RETURN.
10
11 00000'054016 COMP:     STA     3,LSP   ;FOR RE-ENTRANCY..
12 00001'020412           LDA     0,CMESS ;GET MESSAGE ADDR.
13 00002'006017           .SYSTM          ;WRITE IT
14 00003'017000           .WRL    0       ;TO THE CONSOLE.
15 00004'002406             JMP   @ERR    ;ERROR RETURN.
16
17 00005'022404           LDA     0,@OCP  ;GET THE OVERLAY IDENTIFIER.
18 00006'030016           LDA     2,LSP   ; AND THE RETURN ADDRESS
19 00007'077777           .OVEX           ; TO EXIT AND RELEASE THIS OVERLAY.
20 00010'002402             JMP   @ERR
21
22 00011'077777 OCP:      ICOMP           ;OVERLAY IDENTIFIER.
23 00012'077777 ERR:      ERROR           ;ERROR HANDLER.
24 00013'000030"CMESS:    .+1*2
25 00014'044440           .TXT    "I AM A DATA GENERAL COMPUTER.<15>"
26        000000          .NULOC  0
27
28                        .END

**00000 TOTAL ERRORS, 00000 PASS 1 ERRORS
```

Figure K-2. Example Program (continued) and COMP Overlay Listing

END OF APPENDIX

INDEX

NOTE: The letter "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 usually system or task calls (e.g., .ABORT); if they lack a period, they are 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