DATA GENERAL
CORPORATION
Southboro,
Massachusetts 01772
(617) 485-9100

PROGRAM

DISK OPERATING SYSTEM
USER'S MANUAL

ABSTRACT

Data General's Disk Operating System can be used with any Nova-line computer of 12K or larger memory, having any combination of fixed or moving head disks. DOS provides comprehensive file system capabilities, allowing the user to edit, compile, assemble, execute, debug, save, protect, and delete files.

093-000048-05

INTRODUCTION

Data General's DOS is a versatile, sophisticated operating system of a design comparable to those used with the largest current computer configurations. It can be used with any NOVA-line computer of 12K or larger memory configuration having any combination of fixed or moving head disks.

DOS provides comprehensive file system capabilities, allowing the user to edit, compile, assemble, execute, debug, save and delete files. Complete file protection is provided using a number of system defined attributes. File directories are maintained on a fixed head disk and disk pack basis, where each disk pack can be removed from the system. All peripheral devices are named and treated as files, providing complete device independence by symbolic name. All I/O including file I/O is buffered, interrupt driven.

Two modes of program communication are provided. The first is interactive teletype communication made possible by an executable system program, the Command Line Interpreter (CLI). The second mode enables the user to communicate directly with the system using a series of command words recognized by the assembler and forming an integral part of his executable program. These command words are interpreted by DOS at run time.

A complete line of system software is available for use under DOS. This includes a relocatable assembler, relocatable loader, text editor, octal editor, Extended ALGOL 60, a superset of FORTRAN IV, a library file editor, and a symbolic debugger. In addition, the use of interpretive system calls enables the user to write his own special-purpose software while utilizing all the file capabilities and peripheral device support of DOS.

TABLE OF CONTENTS

CHAPTER 3 - COMMAND LINE INTERPRETER (CLI) (Continued)

CHAPTER 4 - PROGRAM MODE OF SYSTEM COMMUNICATION

CHAPTER 4 - PROGRAM MODE OF SYSTEM COMMUNICATION

CHAPTER 5 - MULTIPLE FILE DEVICES

CHAPTER 1

FILES AND DIRECTORIES

DEFINITION OF A FILE

The term *file* applies to any collection of information. Typical examples are:

Source program file
Relocatable binary file
Listing file
Core image file (save file)

With the exception of the core image file, these files should be familiar to NOVA users and most programmers. The source program is input to the assembler, which produces as output a relocatable binary file. The relocatable binary file is input to the relocatable loader, which loads and relocates the program at absolute locations, producing a core image file, also called a save file. It is called a core image file because it is stored on disk word for word as it will be loaded in memory. In addition to loading, there are other means by which a user can produce a core image file, and these will be described in later chapters.

FILE NAMES

All files and devices are accessible by *filename* . The basic file name is a string of alphanumeric characters and the character $. A file name can contain any number of characters, but the system considers only the first 10 significant.

File Name Extensions

An extension can be appended to a file name. An extension is a string of alphanumeric characters and the character $. The extension can be any number of characters but the system considers only the first two significant. A period (.) separates the extension from the file name. An example of a file name with an extension is:

FOO.PS

The Command Line Interpreter, described in Chapter 3, often appends extensions to the name of a file, indicating the type of information it contains and distinguishing it from other types of files resulting from the same source file. For example, if a source file is named A.SR, the names of the different types of files produced from the source file might be:

File Name Extensions (Continued)

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

The user must, in some instances, give the extension for his file name in a command. Usually, however, the particular command uses a search algorithm that will locate the file with the correct extension. (See Chapter 3, File Name Searches)

There are instances when the user may want to append his own extension to a file name. This is permissible, but the user should avoid conflicts with system extensions. For example, a user should not name a source file A.SV because of the confusion it might cause with save files having the SV extension.

Special File Names

Conditional access devices are given special file names, which begin with $ for uniqueness. File names of these devices are:

$TTI	-	teletype keyboard input*
$TTR	-	teletype reader input
$TTO	-	teletype printer output**
$TTP	-	teletype punch output **
$PTR	-	paper tape reader input
$PTP	-	paper tape punch output
$CDR	-	card reader input
$LPT	-	line printer output
$PLT	-	incremental plotter output

* Input devices other than the teletypewriter keyboard automatically provide end-of-file when input ceases for a device-specified time. On TTI input, however, the user indicates an end-of-file by pressing the CTRL and Z keys.

** $TTP has the characteristic: "requires leader/trailer"; $TTO does not. Otherwise, there is no difference between the devices.

Special File Names (Continued)

The user can, if he wishes, assign names beginning with $ to files other than devices.

A few examples will indicate how input/output operations are facilitated by the convention of equating a file and a device. The command:

XFER

is used to transfer the contents of a file from one file to another file. There are two arguments:

XFER *sourcefile destinationfile*

If the user types

XFER $PTR A↵

the contents of the paper tape mounted in the paper tape reader are transferred to a file named A. (The symbol ↵ represents a carriage return.) If the user types

XFER P $PTP↵

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

Device Prefixes

File names may be prefixed by a device specifier. The device specifier is a two-letter mnemonic, followed by a unit number, followed by a colon that separates the specifier from the file name. For example:

DKØ:FOO.SV

specifies save file FOO on fixed head disk unit Ø, and:

DP2:TEST.SR

specifies assembler source program TEST.SR on moving head disk pack unit 2.

A device prefix is used to reference a file that is not in the default directory but is in the directory of a device that is part of the system configuration. File directories and the devices that maintain directories of files are explained in the section immediately following, "File Directories".

FILE DIRECTORIES

Contents of the Directory

Information required about files on a given device is kept in the file directory of the device. The information includes the file name, the length in bytes of the file, and the file attributes.

Since all file names on a given storage device are contained in a single directory, each file name must be unique. An attempt to add a file name to the directory when the same file name already exists causes an error indication.

A file may have a byte length from 0 up to a maximum of 33,423,360 bytes.

File attributes are characteristics of files that can be set and changed by the user. These are:

P - Permanent file, which cannot be deleted or renamed.

S - Save file (core image).

W - Write - protected file, which cannot be written.

R - Read - protected file, which cannot be read.

A - Attribute - protected file. The attributes of such a file cannot be changed. *

The LIST command, described in Chapter 3, allows the user to obtain information from the file directory about one or more files on the device.

The Disk Operating System contains a number of permanent and attribute-protected system files, for example, the $TTI. The user should be careful not to place overly restrictive attributes on his own files unless necessary. Note, for example, that a file with the attributes AP cannot be deleted by the user in any way.

* The A attribute can be set by a user only in program mode of system communication. Other attributes can be set either in program or CLI mode.

Directory Devices

It is possible to configure DOS with up to four similar moving head disk units and up to eight fixed head disk units. All fixed head disk units are logically combined into a single unit for user reference purposes. The device specifier for the fixed head disk:

DKØ (fixed head disk, control Ø)

refers to a single directory and storage area with up to two million 16-bit words.

Moving head units are organized with a separate file directory per unit. DOS may be configured with up to four such units with device specifiers:

DPØ, DP1, DP2, and DP3

Each disk pack unit has a distinct file directory and free storage map. Any file on such a device is completely contained on that device. Precise configuration is determined via the SYSGEN program (see Appendix B).

Default Directory Device

The default directory device is the current device within the system to which all file name references are directed in the absence of a device specifier, either DKØ or DPn, prefixed to the file name.

Master Storage Device

The master storage device can be designated at SYSGEN time to be any legitimate device specifier within the system. The master storage device is used for two purposes:

1. It becomes the default directory device at system initialization and after a bootstrap.

2. It is used for temporary storage area for pushing the current address space when the .EXEC monitor command is executed either by the CLI or by a user program. Since the master device is used during this swapping operation, it should be selected during SYSGEN to be the fastest access device available.

Bootstrap Device

It is possible to generate (via SYSGEN) a DOS system which is bootstrappable from either the fixed head disk (DKØ) or from the moving head disk unit zero (DPØ). Thus, once a system is generated it is bootstrappable from only one type of device. If a user desires a configuration with both types of storage media, it is possible to generate two systems identical in all respects but for bootstrap device type.

Removable Media

Since individual removable disk units contain only complete files and file directory information, they may be removed from the system without affecting the file contents. New packs are introduced to the system with the INIT/F CLI command. This causes rudimentary directory information to be written to the disk in anticipation of file creation. The same command, without the /F switch is used to reintroduce a pack with valuable file contents to the system.

Command Summary

The following is a list of the pertinent CLI commands used to manage multiple directory devices.

Command		Description
INIT [/F]	*device_specifier*	Prepare device for system use.
RELEASE	*device_specifier*	Remove device from system.
DIR	*device_specifier*	Change default directory device.

System Installation

The system saved file (SYS. SV) produced during system generation (see Appendix B) must be installed via the INSTALL command before bootstrapping can take place. When this command is invoked, DOS copies the bootstrap program from the system saved file to logical address zero of the default directory device. When the bootstrap program begins execution, it locates the remainder of the system file and loads the entire system into core. If a system has never been INSTALLed, bootstrapping is not possible.

Further System Information

This chapter attempts to summarize some of the features of directory devices under DOS and to define some of the terms applicable to directories and files that are used throughout the manual. The user should read the entire manual carefully before attempting to generate, install, bootstrap, and use DOS.

CHAPTER 2

TELETYPE BREAKS

There are two possible program breaks that can be generated at the teletypewriter.

Pressing CTRL and A on the teletypewriter causes an immediate interrupt regardless of present program status. This is a trouble break used, for example, when necessary to interrupt a long $TTO output. The word INT is typed by the Command Line Interpreter upon recognition of a CTRL A break.

Pressing CTRL and C on the teletypewriter causes an eventual interrupt of a program and a file to be created and written as an image of core at the time of interrupt. The word BREAK is typed by the Command Line Interpreter upon recognition of this interrupt, and the name of the save file will be BREAK.SV. The termination of execution depends upon the state of the user program. If the program is not within the system (i.e., not executing one of the monitor calls described in Chapter 4) the interrupt will occur immediately. If the program is executing within the system, the interrupt will not occur until the monitor has satisfied the user request. Under no circumstances is the CTRL C ever transmitted to a user program reading from the $TTI. The implications of this are as follows:

> If the user program is in the process of reading from the $TTR, a break should not be attempted until the reader has stopped. Depressing CTRL C while the reader is active causes garbled character transmission.
>
> If the user program is in the process of reading from the $TTI, the read request must be satisfied before the break will occur. Specifically, if a character has been requested (see .GCHAR, Chapter 4) another character in addition to CTRL C must be input since the CTRL C is transparent to the input request. If a read line has been requested (see .RDL, Chapter 4) a carriage return or a form feed must be sent. If a read sequential has been requested (see .RDS, Chapter 4) the sequential character count must be satisfied before the break will occur.
>
> If the user program to be interrupted is not issuing reads from the $TTR or $TTI, the break will occur upon system completion of the call.

See Appendix A, Relocatable Loader Section, for a discussion of how the user can service CTRL A and CTRL C teletype breaks.

CHAPTER 3

COMMAND LINE INTERPRETER (CLI)

CLI DEFINITION

The Command Line Interpreter (CLI) is a system program that accepts command lines from the teletypewriter and translates the input as commands to the operating system. The CLI is basically a string handler that acts as an interface between the user at the teletypewriter and the system. In addition, the CLI performs certain file housekeeping chores for the user.

The system restores the CLI to core whenever the system is idle - after initialization, after a bootstrap, after a teletype break, after execution of a program, etc.

READY MESSAGE

The CLI indicates to the user that the system is idle and the CLI is ready to accept commands by typing a ready message on the teletypewriter. The message consists of R followed by a carriage return.

CLI ACTIVATION

The user activates CLI responses to a command by typing a line and pressing the RETURN key or the CTRL L (form feed) keys. The CLI will not respond until RETURN or CTRL L is pressed. (RETURN and CTRL L are interchangeable; use of RETURN in the remainder of the text means either line terminator.)

CLI RESPONSE TO COMMAND LINES

The CLI itself executes certain system commands such as CREATE and RENAME. More complex commands cause the CLI to build a file containing an edited version of the command line and load the program named in the command line for execution. When execution is finished, control is returned to the CLI.

SYMBOLS AND CONVENTIONS USED IN COMMAND LINE SYNTAX

Symbol	Usage	Example
↵	Represents pressing RETURN key, causing termination of the command line input and activation of the CLI.	CREATE A B ↵
↓	Represents pressing CTRL L keys (form feed). which acts in the same way as the RETURN key.	CREATE A ↓
\	Represents pressing SHIFT L keys, which causes deletion of the entire line. \ ↵ are echoed on the teletypewriter.	CCRREAGE \
←	Represents pressing RUBOUT key, which causes erasure of the previous character. ← is echoed on the teletypewriter.	CC←REAG←TE
, (space)	Arguments are separated by commas or spaces. Extra commas and spaces have the same effect as a single symbol.	DELETE A B ↵ DELETE A, B ↵ DELETE A B ↵ DELETE A, ,B ↵
/	Right slash indicates that the character immediately following is to be interpreted as a switch.	LIST/A ↵
;	Command delimiter in a command line. Two or more commands may appear on a line separated by semicolons, none are executed until RETURN is pressed.	CREATE A;LIST ↵
↑	The next RETURN is ignored as a command terminator. ↑ must appear as the character before the carriage return.	RENAME A ALPHA ↑↵ B BETA ↵
*	Can match any character in a file name or its extension or any set of characters in constituting a file name or its extension, according to rules described later.	DELETE FOO. * ↵ LIST T* ↵ CHATR FOO *W ↵
. ↵	Complements the ready message switch. All ready messages are suppressed until the next occurrence of a . ↵ command.	. ↵ CREATE A ↵ DELETE B ↵

SYMBOLS AND CONVENTIONS USED IN COMMAND LINE SYNTAX (Continued)

Symbol	Usage	Example
@	Change of CLI input command stream.	ASM @FOO@ ↵

NOTE: Use of these symbols and conventions is described in greater detail in sections following.

CLI COMMANDS AVAILABLE TO USERS

The library of CLI commands available to users provides for complete file maintenance and an interface to standard system software. CLI commands are listed below.

ALGOL - Compile an ALGOL source file.

APPEND - Append one, two or more files to produce a single file.

ASM - Assemble a program.

BLDR - Load absolute binary tape with binary block loader (stand-alone operation).

BPUNCH - Punch a file or files in binary on the high speed punch.

CHATR - Change the attributes of an existing file.

CLG - Compile, load, and execute FORTRAN programs.

CONT - Resume execution of a save file interrupted by a CTRL C break.

CREATE - Create a file or series of files.

DEB - Read in a program and transfer to the symbolic debugger instead of resuming execution.

DELETE - Delete a file or series of files.

DIR - Change the current default directory device specification.

DISK - Obtain a list of the number of blocks used and the number of blocks still available on the default device.

DUMP - Dump files. The dump includes directory information for each file, which enables later reloading.

EDIT - Bring in the Text Editor to build or edit source files.

FORT - Compile and assemble a FORTRAN source file.

CLI COMMANDS AVAILABLE TO USERS (Continued)

INIT	-	Initialize a directory device or magnetic tape.
INSTALL	-	Specify system saved file for use in bootstrapping DOS.
LFE	-	Update DOS library files.
LIST	-	List names of files in the default file directory with their length in bytes and attributes.
LOAD	-	Reload dumped files.
MKABS	-	Make an absolute binary file from a core image file.
MKSAVE	-	Make a core image file from an absolute binary file.
OEDIT	-	Bring in octal editor to examine and modify locations in octal.
PRINT	-	Print a file or files on the line printer.
PUNCH	-	Copy a file or files in ASCII mode to the high speed punch.
RELEASE	-	Prevent further I/O access to a directory device or rewind magnetic tape.
RENAME	-	Change the name of a file.
RLDR	-	Load a core image from a series of relocatable files.
SAVE	-	Save a core image as a file.
TYPE	-	Copy a file or files in ASCII mode to the teletypewriter.
XFER	-	Transfer contents of a file to another file.

COMMAND LINES

A command line can consist of one or more commands followed by RETURN. A basic command line has one command.

COMMAND LINES (Continued)

Basic Command Line

The basic command line is simply a list of one or more file names. Except for a number of simple commands that the CLI executes directly, the first file name in the command line is the name of the program to be loaded into core by the CLI for execution. Thus, some of the commands listed on the previous page are names of save files. If, for example, the user types the command line

```
ASM $PTR ↵
```

the CLI builds a file, called COM.CM, containing the edited command line, and loads the save file that has the file name ASM.SV for execution.

Any additional file names besides the program name are used as arguments. In the example, $PTR is the file name of the paper tape reader from which a file is to be assembled.

User action and CLI response are the same when a user wants to execute one of his own programs. For example, if a user has a save file named A.SV and he types

```
A ↵
```

the CLI builds a file containing the command line and calls the operating system to load the save file named A.SV for execution.

Stacking Commands on a Command Line

A command line is executed by the CLI when the user presses the RETURN key or the CTRL L keys on the teletypewriter.

A number of commands may be stacked on a given line for execution. They are separated by semicolons. For example:

```
CREATE A; LIST A;DISK;DELETE B ↵
```

The four commands are executed when the user presses RETURN. The CLI indicates execution of each command with the appropriate information, if any. At the completion of the entire command line, the CLI will prompt the user again with a ready message. For example, the previous command line might cause the response:

```
A.          0                    ← response to LIST A
LEFT: 56, USED:  200             ← response to DISK
R                                ← command line completed
```

COMMAND LINES (Continued)

Long Command Lines

There is no limit (other than memory capacity) to any command line. The user can type a command line that is longer than the ASR33 line length by typing the symbol ↑ in the command line immediately before pressing the RETURN key. The up arrow causes the carriage return to be ignored. For example:

```
CREATE A B C; LIST; DISK; APPEND NEW.SR ↑↵
GAMMA.SR DELTA.SR ↵
```

is executed as if the following had been typed:

```
CREATE A B C; LIST; DISK; APPEND NEW.SR GAMMA.SR DELTA.SR ↵
```

In the previous example, the second line starts a new argument. Note that when a RETURN is ignored, there is no delimiter between the last character on one line and the first character on the next line. Therefore, in the example the blank argument delimiter has been inserted before the up arrow.

The user can, of course, break an argument or command word into two lines:

```
CREATE A B C; LIST; DISK; APPEND NEW.SR, GAM ↑↵
MA.SR,DELTA.SR ↵
```

is equivalent to:

```
CREATE A B C; LIST; DISK;  APPEND NEW.SR,GAMMA.SR,DELTA.SR ↵
```

Suppression of Ready Messages

The user can suppress typing of ready messages by using the symbol period (.) as a command. For example:

```
. ↵                                    ← suppress prompt
CREATE A; LIST A; DISK; DELETE B ↵
A.          0                          ← response to LIST A
LEFT: 56, USED: 200                    ← response to DISK
                                       ← no ready message at completion
                                         of command line
```

To restore typing of ready messages, the user issues a second command:

```
. ↵
```

COMMAND LINES (Continued)

For example:

```
. ↵                                    ← turn prompt off
CREATE A; LIST A; DISK ↵
A.          0
LEFT: 56,  USED: 200
. ↵                                    ← turn prompt on
DELETE C ↵
R                                      ← response to DELETE C
```

SWITCHES

Commands and their arguments may be modified by a series of switches pertaining to the command or argument. A switch is indicated by a right slash (/) followed immediately by either a letter or a decimal digit.

Numeric Switches

Numeric switches specify the number of times the previous argument is to be repeated in the command line. For example:

```
RLDR $PTR/6 ↵
```

indicates that six relocatable binary tapes are to be loaded from the paper tape reader.

Numeric switches are cumulative. The following commands are equivalent:

```
RLDR $PTR/1/0/3/2 ↵

RLDR $PTR/6 ↵
```

The digit 1 in a numeric switch is the same as no switch. The following commands are equivalent:

```
RLDR $PTR ↵

RLDR $PTR/1 ↵
```

The digit 0 has no effect upon the number of times a file name is repeated if it appears in a list of numeric options. For example, the following commands are equivalent:

```
RLDR $PTR/6 ↵

RLDR $PTR/1/0/2/3 ↵

RLDR $PTR/1/2/3/0/0 ↵
```

SWITCHES (Continued)

Numeric Switches (Continued)

However, when used alone, the 0 switch has the same effect as 1. For example, the following are equivalent commands:

```
RLDR $PTR/1 ↵

RLDR $PTR/0 ↵

RLDR $PTR ↵
```

The user should note the effect of applying a numeric switch to a CLI command. For example, the following are equivalent:

```
DELETE/2 ↵

DELETE DELETE ↵
```

The command could be used if the user has a file named DELETE that he wishes to delete.

Letter Switches

Letter switches have distinct meanings that depend upon the command or argument with which they are associated. The detailed descriptions of each CLI command indicate the meanings of each letter switch that can be used in the command.

A letter switch that follows a command word is a global switch and applies to all arguments of the command line. A switch that follows an argument is a local switch and applies only to the particular argument. For example, the assembly command ASM has both a local and global switch, L (listing file). The command:

```
ASM A B ↵
```

causes files A and B to be assembled but, by default, no listing is produced. The command:

```
ASM/L A B ↵
```

causes files A and B to be assembled, and a listing file named A. LS to be produced. The command:

```
ASM A B $LPT/L ↵
```

causes files A and B to be assembled and a listing of the assembly to be output to the line printer.

SWITCHES (Continued)

Effect of Switches on Command Lines

A switch affects a command line as if the switch were a comma or a space. For example, the following commands are equivalent:

```
ASM/L A B ↵
ASM /LA B ↵
ASM/LA B ↵
```

Thus, the switch delimits the command word ASM from the argument A.

If a character other than a number or letter follows the right slash, the slash acts merely as a delimiter. For example,

```
ASM/ L A B ↵
```

The slash is ignored because it is followed by a space. The command will cause the assembly of files L, A, and B. If there is no source file named L, an error message will result.

ASTERISK (*) CONVENTION

When referencing the default directory, an asterisk can be used to represent any given character in a file name. For example, the command:

```
DELETE A**M ↵
```

will cause all four-letter file names without extensions that begin with A and end with M to be deleted. For example, files that have names like the following would be deleted:

```
ATOM
ADAM
A22M
A$RM
```

The command:

```
LIST B* ↵
```

would cause a list to be typed, giving all two-character files beginning with the letter B and having no extension.

Asterisk (*) Convention (Continued)

A single asterisk can be used to represent the entire file name or extension and thus represent a number of characters. For example, the command:

```
DELETE *.LS↵
```

would cause deletion of every file on the default directory with the extension LS. The files might be:

```
A.LS
OMEGA.LS
TESTPROG.LS
ATOM.LS
```

The command:

```
LIST *↵
```

would cause a list of every file in the directory that does not have an extension, and the command:

```
LIST *.*↵
```

would cause a list of every file, whether or not it has an extension.

It is possible to delete all files that are not protected with the single command:

```
DELETE *.*↵
```

Note, however, that it is not possible to delete all single-letter files, only, since the command:

```
DELETE *↵
```

is interpreted to mean 'delete all files without extensions.'

A device specifier cannot be used with the * convention. An attempt to give a command such as:

```
LIST DP1: *.*↵
```

causes the default directory to be searched for the name DP1: *.* which is never found.

The last part of this chapter contains writeups on each of the CLI commands. Each writeup indicates whether or not the asterisk convention can be used in file names.

INDIRECT (@) CONVENTION

Paired @ signs around a file name are understood to represent the contents of the file rather than the file name itself.

Suppose a user regularly concludes each teletypewriter session by deleting listing files, checking the list of non-permanent files, and determining how much space he has left on disk. The command line for this would be:

```
DELETE *.LS; LIST; DISK ↵
```

These commands could be written into a file called END in the following way:

```
XFER /A $TTI END ↵
DELETE *.LS; LIST; DISK ↵
R
```

Transfer commands in ASCII from TTI to file END. (User terminates input with CTRL Z; CLI types out R.)

Then the command:

```
@END@ ↵
```

is equivalent to typing the three commands.

As another example, suppose the user has five source programs called PART1, PART2, PART3, PART4 and PART5. He can then use the XFER command as shown above to build a file called TEST, containing the ASCII line:

```
PART1 PART2 PART3 PART4 PART5
```

If he issues the command:

```
ASM @TEST@ ↵
```

the five files are assembled.

The contents of a file on disk may, in turn, point to another file. As a simple example, suppose:

```
file A contains L@B@
file B contains I@C@
file C contains ST
```

then the command:

```
@A@ ↵
```

is equivalent to the command:

```
LIST ↵
```

Only four files (including the teletypewriter) may be open at any one time. In the example above, the maximum number of files (A, B, C and teletypewriter) are open. Suppose the contents of the three files were:

```
file A contains L
file B contains I
file C contains ST
```

INDIRECT (@) CONVENTION (Continued)

Then the command:

```
@A@@B@@C@ )
```

is equivalent to:

```
LIST )
```

and only one file plus the teletypewriter is open at a given time.

PARENTHESIZED FILE NAME CONVENTION

CLI commands can be repeated with each of several file names or strings of file names by first separating the file names or strings of file names by commas and then enclosing all the file names within parentheses. Since the CLI command words are themselves file names, a series of commands can be made to use a common argument or string of arguments by placing the commands within parentheses. For example:

```
(INIT, DIR) DP0 )
```

is equivalent to:

```
INIT DP0 )
DIR  DP0 )
```

and

```
ASM (A, B, C)  $LPT/L )
```

is equivalent to:

```
ASM A  $LPT/L )
ASM B  $LPT/L )
ASM C  $LPT/L )
```

Each repetition of the ASM command generates a relocatable binary file (A.RB, B.RB, and C.RB respectively). This differs from the command:

```
ASM A B C $LPT/L )
```

which generates a single relocatable binary file named A.RB from the single assembly of source files A, B, and C.

A string of file names that are not separated by commas is treated as an entity, i.e.,

```
ASM (A  B,  C, D E F) $LPT/L )
```

is equivalent to:

```
ASM  A  B  $LPT/L )
ASM  C  $LPT/L )
ASM  D  E  F  $LPT/L )
```

where the three relocatable binary files generated by the command are A.RB, C.RB, and D.RB.

PARENTHESIZED FILE NAME CONVENTION

Use of commas with no file names within parentheses is permitted. For example:

```
LIST ( , , , ) ↵
```

is equivalent to:

```
LIST ↵
LIST ↵
LIST ↵
LIST ↵
```

producing four lists of all non-permanent files in the default directory.

Only one set of parenthesized file names may appear within a given command line. Use of the indirect (@) convention is illegal when using the parenthesized file name convention. Command terminators (carriage return, end-of-file) cannot be used within parentheses.

In using the parentheses convention, care must be taken not to overwrite disk files created by repetitive commands. For example:

```
FORT ($CDR, $CDR) $LPT/L ↵
```

When the command is executed, two separate compilations would be listed on the line printer, but only the second program input via the card reader would exist as a relocatable binary file on disk. The previously created disk file, $CDR.RB would be overwritten by the second file having the same name.

FILE NAME SEARCHES

The file directory may contain a number of entries having the same file name but different extensions, such as:

```
A.SR
A.RB
A.SV
A.32
A.XX
```

File name arguments to most commands must specify both the file name and extension. Certain commands, however, accept a file name without extension and search on both the file name with appropriate extension appended and on the file name without an extension. Similarly, some commands append an appropriate extension to a specified output file name. For example:

```
ASM A ↵
```

causes a search for a file named A.SR. If A.SR is found, it is the source file for assembly. Otherwise a search is made for A. If A is found, it is the source file for assembly. The output file when the ASM A↵ command is executed is a relocatable binary file named A.RB.

If the user types:

RLDR A ↵

A search is first made for A.RB and if not found, for A. The CLI creates an output file for the relocatable loader called A.SV, the extension used to signify a save file.

The commands SAVE and MKSAVE also have a save file as output. In both cases, the CLI adds the extension SV to the name of the output file. If the user attempts to substitute his own extension, it will be ignored. For example:

SAVE A.XX ↵

causes a core image to be stored as a save file on disk. The name of the file will be A.SV. The extension XX is ignored.

The command MKABS has as input a save file. For example, if the user types:

MKABS A $PTP ↵

a search is made for A.SV and, if not found, for A.

To execute the file A.SV, the user types

A ↵

The CLI calls the operating system to load into memory the file called A.SV and transfer control to its starting address. In this special case of loading a save file, the only search made is for the file name with the SV extension.

Most other commands require appropriate file name extensions to be given explicitly. If the user types:

DELETE A ↵

only the file named A will be deleted. Files such as:

A.SR
A.SV
A.RB
A.33

would not be deleted.

If the user types the command:

RLDR A FOO/S ↵

the /S switch indicates that the user wants the save file output of the loader to be named FOO.SV.

If a user gives his own extensions to file names, such as A.33, such files must be referenced with their file name and extension.

MESSAGES CONCERNING I/O

Some commands require manual operation of an I/O device. If the user issues such a command, he will receive a message prompting him on the proper action. For example, if the user issues the command:

```
XFER/A  $PTR  A.SR ↵
```

which requests that a source file be transfered from the paper tape reader to a disk file named A.SR, the system replies:

```
LOAD $PTR, STRIKE ANY KEY.
```

The user can then load the paper tape reader and strike any key on the teletypewriter. The key struck to start the device is not echoed on the teletypewriter.

When a series of files are to be transferred, assembled, or loaded from a device requiring manual intervention for each file, the message will be issued the appropriate number of times. For example, if the user issues the command:

```
APPEND NEWFILE  $PTR/2 ↵
```

which requests that a file called NEWFILE be created from two files input from the paper tape reader, the following responses will occur:

```
LOAD $PTR, STRIKE ANY KEY.
LOAD $PTR, STRIKE ANY KEY.
```

The second message is typed out after the first file has been transferred.

ERROR MESSAGES

When the user issues a command that contains an error, an appropriate error message will be typed out.

When a user gives a command that is legal for some arguments and illegal for others, an error message is issued for each of the illegal arguments. The correct portions of the command are executed. For example,

```
R
CREATE A B C D ↵                    create four empty files
R
XFER $PTR A ↵                       transfer file from PTR to A
R
CREATE A E ↵                        illegal argument A; legal argument E
ERROR: FILE ALREADY EXISTS, NAME: A
R
LIST E ↵
E.          Ø                       E was created
R
```

When the CLI cannot respond to a user command, an error message does not necessarily result. For example, if the user requests list information on a non-existant file, the CLI responds to the LIST command with a ready message only.

ERROR MESSAGES (Continued)

The error messages appropriate to each command are listed in the detailed descriptions of each command. In general, error messages are quite explicit, giving the user the sufficient information to correct his error easily. A few samples are shown:

```
CREATE A #A *A ↓
ERROR: ILLEGAL FILE NAME, NAME: #A
ERROR: ILLEGAL FILE NAME, NAME: *A
R

XFER FOO $PTR ↓
ERROR: FILE WRITE PROTECTED, FILE: $PTR
R

CREATE TEST:CHATR TEST W ↓
R
XFER SYS.DR TEST ↓
ERROR: FILE WRITE PROTECTED, FILE: TEST
R

CHATR $LPT Ø ↓
ERROR: UNABLE TO CHANGE MODE, FILE: $LPT
R

MKSAVE $PTR CLI.SV ↓
LOAD $PTR, STRIKE ANY KEY.
IO ERROR: DISK SPACE EXHAUSTED
R

XFER NONFILE NEWFILE ↓
ERROR: FILE DOES NOT EXIST, NAME: NONFILE
R
```

CLI COMMANDS

Following are definitions and descriptions of each of the CLI commands. The commands are listed in alphabetical order.

Name: ALGOL

Format: ALGOL *inputfilename* [*outputfilename*]

Purpose: To compile an ALGOL source file. Output may be a relocatable binary file, an intermediate source file, a listing file, or combinations of all three. The command name, ALGOL, must be used in compiling ALGOL source programs; the name, ALGOL, cannot be changed.

Switches: By default, execution of the command produces an intermediate source file, *inputfilename*.SR (compiler output), and a relocatable binary file *inputfilename*.RB (assembler output). However, once assembly has been successfully completed, the intermediate source file is deleted. No listing is produced by the default command.

Global:
/A - Assembly is suppressed.
/B - Brief listing (compiler source program input only).
/E - Error messages from compiler are suppressed at the $TTO. (Assembler error messages are not suppressed).
/L - Listing produced to *inputfilename*.LS.
/N - No relocatable binary file is produced.
/S - Save the intermediate source output file.

Local:
/B - Relocatable binary output directed to given file name.
/L - Listing output directed to given file name.
/S - Intermediate source output directed to given file name.

Asterisk: Not permitted.

Errors: See NOVA ALGOL Reference Manual, 093-000052.

Extensions: On input search for *inputfilename*.AL. In not found, search for *inputfilename*. On output, produce *inputfilename*.RB by default and other files with .LS or .SR extensions as determined by switches.

Examples: ALGOL MAIN ↵

Produce relocatable binary file, MAIN.RB. No listing is produced.

ALGOL /E/B SUBR $LPT/L ↵

Produce relocatable binary file, SUBR.RB with a brief (ALGOL source) listing to the line printer. Suppress compiler error messages.

Name: ALGOL (Continued)

Examples: ALGOL/A $PTP/S↵

Do not invoke an assembly phase. Punch intermediate source output on high speed punch.

Name: APPEND

Format: APPEND *newfilename oldfilename*$_1$... *oldfilename*$_n$

Purpose: To create a new file, consisting of a concatenation of one or more old files in the order in which their names are listed as arguments. The old files are not changed by the command.

Switches: None.

Asterisk: Not permitted.

Errors:

FILE ALREADY EXISTS.	(newfilename)
FILE DOES NOT EXIST.	(oldfilename)
NOT ENOUGH ARGUMENTS.	
DISK SPACE EXHAUSTED.	
FILE WRITE PROTECTED.	(newfilename)

Examples: APPEND COM.SR COM1 COM2 COM3 COM4 ↵

causes creation of the file COM.SR containing the contents of files COM1, COM2, COM3, and COM4 in that order.

APPEND DP1:ALL.LB A.LB B.LB DPØ:C.LB ↵

causes creation of the file ALL.LB on disk pack unit 1 containing the contents of files A.LB and B.LB from the default directory and C.LB from disk pack unit Ø.

Name: ASM

Format: ASM $filename_1 \ldots filename_n$

Purpose: To assemble one or more source files. Output may be a relocatable binary file, a listing file, or both. The command name, ASM, must be used in assembling programs; the name, ASM, cannot be changed.

Switches:

Global: By default, output of an assembly is a relocatable binary file (no listing file).

/L - listing file is produced.
/N - no relocatable binary file is produced.
/U - user symbols are appended to the relocatable binary output.
/E - error printouts on the TTO are suppressed unless there is no listing file for the current pass.
/S - skip pass 2. A BREAK is signaled after pass 1 permitting the user to save a version of the assembler that contains his own permanent symbols.
/T - symbol table list is not produced as part of the listing. (Used when a listing is requested, which produces a symbol table by default.)
/X - produces cross referencing of symbol table. Symbol table output will contain page number - line number pairs for the symbol definition as well as every reference to the symbol within the assembly.

Local:
/B - relocatable binary output directed to the given file name.
/L - listing output directed to the given file name.
/S - skip this file on pass 2 of assembly. (This switch should be used only if the file does not assemble any storage words.)
/N - no listing of this file. (Used, when a listing is requested, to list a selected number of files to be assembled.)

Asterisk: Not permitted.

Errors:
NO SOURCE FILE SPECIFIED.
ILLEGAL FILE NAME.
FILE DOES NOT EXIST. (input file)
FILE ALREADY EXISTS. (output file)
FILE WRITE PROTECTED. (output file)
FILE READ PROTECTED. (input file)
SWITCH ERROR. (listing and binary files cannot be same)

Name: ASM (Continued)

Extensions: On input, search for *filename*.SR. If not found, and the *filename* did not have an extension, search for *filename*.

On output, produce *filename*.RB for relocatable binary and *filename*.LS for listing (global L switch), where *filename* will be the name portion of the first source file specified without a /S, /L, or /B local switch given.

Examples: ASM Z ↵

causes assembly of source file Z, producing a relocatable binary file called Z.RB.

ASM/N/L A ↵

causes assembly of file A, producing as output a listing file called A.LS.

ASM A B $PTP/B C D $PTR $TTR DKØ:E $LPT/L ↵

causes assembly of files A, B, C, and D from the default directory, a tape mounted in the paper tape reader, a tape mounted in the teletypewriter tape reader, and E from fixed head disk unit Ø. A binary relocatable file is punched to the paper tape punch. A listing file is printed on the line printer.

```
ASM /S/N ICODES ↵        ← no output. Automatic BREAK after pass 1.
BREAK
R
SAVE ASM ↵               ← User can save the assembler with the
                           user's permanent symbols.
```

Name: ASM (Continued)

Examples: (Continued)

```
ASM /X EXAMP $LPT/L↵          ← assemble EXAMP with cross
                                referenced symbol table output
                                listing to line printer.
```

← Program listing

```
 ASM1  EXAMP
01
02
03                          .TITLE  EXAMPLE
04
05                          .EXTN   SBROU
06                          .NREL
07
08                  ; SAMPLE CROSS-REFERENCE PROGRAM
09
10  000000 012427 START:    LDA     1,C3
11  000001 040424           STA     0,CNT
12  000002 012425           LDA     1,MAGIC
13  000003 040423           STA     0,BITS
14  000004 125014           INC#    1,1,SZR
15  000005 000414           JMP     LOOP
16  000006 000417           ISZ     CNT
17  000007 000417           ISZ     BITS
18  000010 000411           JMP     OUT
19
20  000011 004421 LOOP:     JSR     @.SUBR
21  000012 101014           MOV#    0,0,SZR
22  000013 000405           JMP     OUT
23  000014 021000           LDA     0,A0,2
24  000015 025377           LDA     1,A1,2
25  000016 004413           JSR     @.SUBR
26  000017 051376           STA     2,A2,2
27  000020 041000           STA     0,A0,2
28  000021 004411 OUT:      JSR     @.SUBR
29  000022 000002           .SYSTM
30  000023 000000           .RTN
31  000024 000000           JMP     .
32
33  000025      CNT:        .BLK    1
34  000026      BITS:       .BLK    1
35
36  000027 000003 C3:       3
37  000030 001013 MAGIC:    1013
38
39  000031 177777 .SUBR:    SBROU
40
41  000000      A0=         0
42  177777      A1=         -1
43  177776      A2=         -2
44
45                          .END
```

Name: ASM (Continued)

Examples: (Continued)

```
 1992  EXAMP

A0      000000  1/23  1/27  1/41
A1      177777  1/24  1/42
A2      177776  1/26  1/43
BITS    000251  1/13  1/17  1/34
C3      000027  1/10  1/36
CNT     000025  1/11  1/16  1/33
LOOP    000011  1/15  1/20
MAGIC   000341  1/12  1/37
OUT     000021  1/18  1/22  1/28
SBROU   000031  1/39
START   000000  1/10
.SUBR   000031  1/20  1/25  1/28  1/39
```

Cross-reference table.

Cross-referencing is accomplished by outputting symbol table and symbol referencing information to temporary disk files during assembly. A separate save file, XREF.SV, is called by the assembler to output the cross reference list. Note that all pages and lines of the assembler's listing are numbered for this purpose.

Name: BLDR

Format: BLDR *devicename*

Purpose: To load an absolute binary tape with the binary block loader, using either the high speed paper tape reader or the teletype reader. This command implies a transition from DOS mode to stand-alone mode. The loading will overwrite part of core containing DOS, so that after completion of the stand-alone job, DOS must be bootstrapped.

Switches: None.

Asterisk: Not permitted.

Errors: ILLEGAL DEVICE NAME.

Examples: BLDR $PTR ↵

BLDR $TTR ↵

The examples give the only acceptable command lines.

Name: BPUNCH

Format: BPUNCH $filename_1$ [$filename_2$... $filename_n$]

Purpose: To punch a given file or files in binary on the high speed punch. The command is the equivalent of a series of XFER commands:

XFER $filename_1$ $PTP; ...;XFER $filename_n$ $PTP ↵

The files may come from any device.

Switches: None.

Asterisk: Not permitted.

Errors:

ILLEGAL FILE NAME.	(source)
FILE DOESN'T EXIST.	(source)
FILE READ PROTECTED.	(source)

Example: BPUNCH FOO.RB ALPHA.RB BETA.RB ↵

BPUNCH $PTR ↵

BPUNCH DP2: MYFILE.SR ↵

Name: CHATR

Format: CHATR *filename*$_1$ *attributes*$_1$ ... *filename*$_n$ *attributes*$_n$

Purpose: To change, add, or delete file attributes of a given file. All current attributes of the file are replaced by those given in the attributes argument.

Switches: None.

Asterisk: Permitted in attributes argument only.

Attributes:
P - Permanent file. Cannot be deleted or renamed.
S - Save file.
W - Write - protected file. Cannot be altered.
R - Read-protected file. Cannot be accessed for reading.
Ø - No attributes.
* - Represents current file attributes.

When several attributes are specified for a given file name, they must be given as a single argument. Attributes may be listed in any order in the argument.

Errors:
FILE DOES NOT EXIST.
ILLEGAL FILE NAME.
NOT ENOUGH ARGUMENTS.
UNABLE TO CHANGE MODE. (attribute-protected.)
ILLEGAL ATTRIBUTE. (for example, G)

Examples: CHATR A WP ↵

causes file A to be write-protected and permanent.

CHATR A Ø B R ↵

deletes all attributes of A and causes B to be read-protected.

CHATR A.SV SW ↵

causes A.SV to be write-protected save file. If A.SV had had other previous attributes, these would be deleted.

CHATR A.SV *W ↵

causes A.SV to be write-protected save file. Any previous attributes would also be retained (*).

Name: CLG

Format: CLG *filename*$_1$ [*filename*$_2$ ... *filename*$_n$]

Purpose: To compile, load, and execute one or more FORTRAN source files. Output includes one or more intermediate source files, one or more relocatable binary files, and an executable save file. The save file is created by the relocatable loader using the relocatable binary files and the FORTRAN libraries, which must have been merged into a single library file named FORT.LB.

CLG differs from the FORT command (page 3-36), which can produce a relocatable binary file, but cannot produce a save file and execute it. In addition, CLG can treat source input files individually, where some require loading, others assembly and loading, and still others compilation as well.

Switches: If a listing device is specified by a local switch but no global listing switches are given, listings of each FORTRAN compilation, each assembly, and the loader map are output to the specified listing file.

Global:
- /B - Brief listing (compiler source program input only).
- /M - Loader map is suppressed. All compiler and assembler source programs are listed.
- /E - Error messages from the compiler are suppressed at the $TTO. Assembler messages are not suppressed.

Local:
- /L - Listing output directed to the given file name.
- /A - Assemble this file only; do not compile.
- /N - Load this file only, do not compile or assemble.

Asterisk: Not permitted.

Errors: See the FORTRAN IV Reference Manual (093-000053), and the ASM and RLDR commands (pages 3-20 and 3-56 of this manual).

Extensions: On input, search for *filename*.FR; if not found, search for *filename*. If /A is specified, search for *filename*.SR. If not found, search for *filename*. If /N is specified, search for *filename*.RB; if not found, search for *filename*.

On output, produce temporary assembler source files, *filename*$_i$.SR (i = 1 ... n). Produce relocatable loader input files, *filename*$_i$.RB (i = 1 ... n). Produce save file *filename*.SV.

Name: CLG (Continued)

Examples: CLG/B MAIN $LPT/L ↵

Compile MAIN.FR (or MAIN), producing MAIN.SR, with the listing to the $LPT. Assemble MAIN.SR, producing MAIN.RB and delete MAIN.SR. Load MAIN.RB and FORT.LB, producing MAIN.SV. Execute MAIN.SV.

CLG/M/E PROG1 PROG2 PROG3/A PROG4/N MTØ:1/L ↵

Compile PROG1.FR (or PROG1), producing PROG1.SR. Assemble PROG1.SR, producing PROG1.RB and delete PROG1.SR. Compile PROG2.FR (or PROG2), producing PROG2.SR. Assemble PROG2.SR, producing PROG2.RB and delete PROG2.SR. Assemble PROG3.SR (or PROG3), producing PROG3.RB. Listings from each compilation and assembly are appended to file 1 on magnetic tape unit Ø. Load PROG1.RB, PROG2.RB, PROG3.RB, PROG4.RB (or PROG4), and FORT.LB, producing PROG1.SV with no loader map. Execute PROG1.SV.

CLG A B C ↵

Compile A.FR (or A), producing A.SR. Assemble A.SR, producing A.RB and delete A.SR. Compile B.FR (or B) producing B.SR. Assemble B.SR, producing B.RB and delete B.SR. Compile C.FR (or C), producing C.SR. Assemble C.SR, producing C.RB and delete C.SR. Load A.RB, B.RB, C.RB and FORT.LB, producing A.SV. Execute A.SV.

Name: CONT

Format: CONT *filename*

Purpose: To resume the execution of a save file which was interrupted by a CTRL C break.

Switches: None.

Asterisk: Not permitted.

Errors:

FILE DOES NOT EXIST.	(*filename*.SV)
NOT ENOUGH ARGUMENTS.	(no *filename*)
NO CONTINUATION ADDRESS.	(*filename*.SV was never interrupted and saved)

Extensions: If the argument *filename* was not SAVEd and no further CTRL C commands have been issued, the filename is BREAK.SV by default. Only the most recent core image saved by CTRL C is named BREAK.SV. If the filename was SAVEd, the filename extension .SV will be unconditionally appended to the filename, e.g.,

CONT EXAMP) or CONT EXAMP.SV)

would cause the save file execution to be resumed at the point where it was interrupted.

Examples:

```
.)
SAVE EXAMP↵          ← Name core image EXAMP.SV and
.                      resume EXAMP's execution.
.
.
CONT EXAMP)
```

Name: CREATE

Format: CREATE *filename*$_1$ [*filename*$_2$ ... *filename*$_n$]

Purpose: To add an entry to the default file directory. The entry specifies a file of zero length and no attributes.

Switches: None

Asterisk: Not permitted.

Errors: ILLEGAL FILE NAME.
FILE ALREADY EXISTS.

Examples: CREATE ALPHA ↵

Creates a file name, ALPHA, in the default directory.

CREATE TEST TEST1 DPØ:TEST2 ↵

Creates three file names, TEST, and TEST1 in the default directory and TEST2 in the directory of the pack on disk pack unit Ø.

Name: DEB

Format: DEB *filename*

Purpose: To debug a program about to be executed. The symbolic debugger, Debug III, must have been loaded as part of the program save file, as described under the RLDR command.

When debugging, memory can be examined, break points set, the program run, etc. After making any necessary changes in the program, the user can save the current core image of the program by issuing a break (CTRL C) and saving the core image under some file name, as described under the SAVE command. The program can then be resumed in the debugger at a later time.

Switches: None.

Asterisk: Not permitted.

Errors:
ILLEGAL FILE NAME.
ILLEGAL START ADDRESS. (Debugger not loaded with the program).
NOT A SAVED FILE.
FILE DOES NOT EXIST.

Examples:

```
DEB A ↵                          ← Debug A.SV.
1004/  ADD 0 2   ADD 1 2 ↵       ← Change program.
BREAK                            ← CTRL C issued.
SAVE A ↵                         ← Changed version (current core
                                   image) saved.
A ↵                              ← New attempt to execute.
BREAK                            ← CTRL C issued.
SAVE CORE$A ↵                    ← Current core image saved.
ASM FOO ↵                        ← Assembly command and other
  .                                commands.
  .
  .
DEB CORE$A ↵                     ← Restore CORE$A in the debugger.
```

Name: DELETE

Format: DELETE *filename*$_1$ [*filename*$_2$ ... *filename*$_n$]

Purpose: To delete the files having the names given in the argument list from the default directory. No filename may be preceded by a device specifier.

Switches:

Global: /V - verify deletion with a list of names of deleted files.

Local: None.

Asterisk: Permitted.

Errors: FILE DOES NOT EXIST.
ILLEGAL FILE NAME.
NO FILES MATCH SPECIFIER. (When using asterisk convention).

Examples: DELETE ALPHA BETA GAMMA ↵

deletes the files named ALPHA, BETA, and GAMMA.

DELETE *.LS ↵

deletes all files having the extension LS.

DELETE LIMIT.* ↵

deletes all files having the name LIMIT and any extension (including null).

```
DELETE /V *.LS ↵
DELETED A.LS
DELETED COM.LS
DELETED MAP.LS
R

DELETE *.QQ ↵
NO FILES MATCH SPECIFIER: *.QQ
R
```

Name: DIR

Format: DIR *device_specifier*

Purpose: To change the current default directory device. At system initialization (See Appendix B), a default directory device is established. The DIR command permits another device to be substituted as the default directory device.

Switches: None.

Asterisk: Not permitted.

Errors: DEVICE NOT IN SYSTEM.

Example: DIR DPØ↵

Change all default file name references to the moving head disk unit number Ø.

Name: DISK

Format: DISK

Purpose: To obtain a count of the number of blocks used and the number of blocks still available on the default directory device.

Switches: None.

Asterisk: Not permitted.

Errors: None.

Examples:
```
DISK ↵
LEFT: 90, USED: 166
R
```

The message indicates that 90 out of 256 blocks on the disk are still available for use.

Name: DUMP

Format: DUMP *outputfilename* [*filename*$_1$ ... *filename*$_n$]

Purpose: To dump a given file or files to a given file or device. The directory information for each file -- name, length, and attributes -- is written as a header to each dumped file. If no file names are given, all non-permanent files are dumped. If file names are given, no name can be preceded by a device specifier.

Switches:

Global: /A - all files, permanent as well as non-permanent, are to be dumped.
/V - verify dump with a list of names of dumped files.

Local: None.

Asterisk: Permitted.

Errors:

FILE ALREADY EXISTS.	(output file)
FILE WRITE PROTECTED.	(output file)
ILLEGAL FILE NAME.	(input file)
FILE READ PROTECTED.	(input file)
FILE DOES NOT EXIST.	(input file)
DISK DATA ERROR.	(input file)
DISK SPACE EXHAUSTED.	

Examples:

```
DUMP $PTP FOO.SV ↵
```

causes file FOO.SV to be punched out with a header for later reloading.

```
DUMP/A $PTP *.SV ↵
```

causes all permanent and non-permanent files with the extension SV to be punched out.

```
DUMP/V DUMPFI *.SV ↵
EDIT.SV
ASM.SV
RLDR.SV
```

causes all non-permanent files with the extension SV to be written to the file DUMPFI and a list of files dumped to be given.

Name: EDIT

Format: EDIT

Purpose: To invoke the text editor to build a new source file or edit existing source files. The NOVA Editor is described in DGC Document 093-000018. Appendix A summarizes the program.

Switches: None.

Asterisk: Not permitted.

Errors: FILE DOES NOT EXIST. (EDIT. SV)

Example:

```
EDIT ↵
*                    ← Response of Editor indicating the program
                         is ready to accept commands.
.
.                    ← User issues editing commands.
.
H$$ ↵                ← User terminates editing by pressing
R                        the H key followed by two ESC
                         keys. Return is made to the CLI.
```

Name: FORT

Format: FORT *inputfilename* [*outputfilename*]

Purpose: To compile a FORTRAN source file. Output may be a relocatable binary file, an intermediate source file, a listing file, or combinations of all three. The command name, FORT, must be used in compiling FORTRAN source programs; the name, FORT, cannot be changed.

By default, execution of the command produces an intermediate source file, *inputfilename*.SR (output of compilation) and a relocatable binary file, *inputfilename*.RB (output of assembly). However, once assembly has been successfully completed, the intermediate source file is deleted. No listing is produced by the default command.

Switches:

Global:
- /A - Assembly is suppressed.
- /B - Brief listing (compiler source program input only).
- /E - Error messages from compiler are suppressed at the $TTO. (Assembler error messages are not suppressed.)
- /F - FORTRAN variable names and statement numbers are equivalenced to symbols acceptable to the assembler.
- /L - Listing produced to *inputfilename*.LS.
- /N - No relocatable binary file is produced.
- /S - Save the intermediate source output file.
- /X - Compile statements with X in column 1.

Local:
- /B - Relocatable binary output directed to given file name.
- /L - Listing output directed to given file name.
- /S - Intermediate source output directed to given file name.

Asterisk: Not permitted.

Errors: See FORTRAN IV Reference Manual, 093-000053.

Extensions: On input, search for *filename*.FR. If not found, search for *filename*. On output, produce *filename*.RB by default and other output files with .LS or .SR extensions as described under switches and examples.

Name: FORT (Continued)

Examples:

FORT/L MAIN↵

produce relocatable binary file MAIN.RB with both a compiler and an assembler listing to file, MAIN.LS.

FORT /N DP1:TABLE $LPT/L INTAB/S↵

compile the file TABLE from disk pack unit 1 and produce compiler source and assembly listing on the LPT and intermediate source output file, INTAB, to the default directory. Do not produce a relocatable binary file from the assembly.

FORT/A/L/S TABLE↵

produce and save intermediate source file TABLE.SR and listing file TABLE.LS containing compiler source input listing. Assembly is suppressed. (Note that /A implies /B).

Name: INIT

Format: INIT *device_specifier*

Purpose: To initialize a directory device or magnetic tape unit. Until the device is released (RELEASE command) all files on the initialized device are now available to the system software.

Switches:

Global: By default, when a directory device is initialized, the current directory of the device is found and read into system core, allowing access to all files on the device.

/F - full initialization. Clears all previous files and information from the specified device and writes a new file directory and free storage map on the device.

Local: None.

Asterisk: Not permitted.

Errors: ILLEGAL COMMAND FOR DEVICE (attempt to INIT the only disk device in the system)

DEVICE NOT IN SYSTEM

Example INIT DP3 ↵

Initialize the disk pack on unit number 3.

INIT /F MT1 ↵

Magnetic tape initialization means rewind the tape. Full (/F switch) initialization of MT1 causes the tape on drive MT1 to be rewound and two EOF's written on the tape.

Name: INSTALL

Format: INSTALL *filename*

Purpose: To specify a new system save file for use when bootstrapping DOS from the current default directory device. *filename* becomes the DOS core image that will be bootstrapped from the default directory device. The system to be installed must have specified the current default device as its bootstrap device.

Switches: None.

Asterisk: Not permitted.

Extensions: Any filename and extension may be used, but common practice is to give a recognizable system name with the SV extension.

Errors: ILLEGAL COMMAND FOR DEVICE (the bootstrap program of *filename* does not correspond to the default directory device.)

Example: INSTALL SYS.SV ↵

Name: LFE

Format:

LFE A *inputmaster* arg_1 [... arg_n]

LFE A/M $inputmaster_1$ [... $inputmaster_n$]

LFE D *inputmaster outputmaster/O* arg_1 [...arg_n]

LFE I *inputmaster outputmaster/O* arg_1 [...arg_n]

LFE M *outputmaster/O* $inputmaster_1$ [...$inputmaster_n$]

LFE N *outputmaster/O* arg_1 [...arg_n]

LFE R *inputmaster outputmaster/O* arg_1 arg_2 [... arg_{n-1} arg_n]

LFE T $inputmaster_1$ [*listing-device*/L] [$inputmaster_2$...]

LFE X *inputmaster* arg_1 [... arg_n]

Purpose: To update and interpret library files, which are sets of relocatable binary files having special starting and ending blocks and which are usually designated by the extension . LB.

In the format, A, D, I, M, N, R, and X and keys designating LFE functions; *inputmaster* and *outputmaster* represent library files; and *args* represent logical records on the library files or relocatable binary files.

Appendix A contains more detailed information on library files, the LFE, and on the use of and output from each of the functions than is given here.

Action taken by the LFE depends upon the function given in the command:

A - analyze - Analyze global declarations of *inputmaster*; or of a series of *inputmasters*, or of logical records specified from one *inputmaster*. Output is a listing with symbols, symbol type, and flags; no new output library file is created.

D - delete - Delete logical records, specified by *args* from *inputmaster*, producing *outputmaster*.

Name: LFE (Continued)

Purpose: (Continued)

I - insert - Insert relocatable binary files, merging with logical records of *inputmaster* in the manner described under Switches.

M - merge - Merge library files (*inputmaster*) into a single library file named *outputmaster*.

N - new - Create new library file, *outputmaster* from one or more relocatable binary files given by *args*.

R - replace - Replace logical records in *inputmaster* by relocatable binary files, producing *outputmaster*. *Args* are paired with the first being the logical record and the second the relocatable binary file that replaces the logical record.

T - titles - Output to the listing device (or to the teletype by default) the titles of logical records on *inputmaster* and on any optional additional library files given by $inputmaster_2$ ...

X - extract - Extract from library file, *inputmaster* one or more relocatable binary files given by *args*. Output is one or more relocatable binary files named *args*.

Switches:

Global: /M multiple input library files. The switch modifies the A function (not the filename LFE) and causes all library file names following, except the listing file,to be analyzed as one library.

Local: /A Insert after. The switch modifies a logical record in an I function command line. Arguments following the switch are inserted after the logical record whose name precedes the switch. When neither a /A or /B switch is given, inserts are made at the beginning of the new library file.

Name: LFE (Continued)

Switches:

Local:

/B - Insert before. The switch modifies a logical record in an I function command line. Arguments following the switch are inserted before the logical record whose name precedes the switch. When neither a /A or /B switch is given, inserts are made at the beginning of the new library file.

/L - listing file. The switch modifies the name of a file to be used as listing output in the A function command line. (The TTO is used by default.)

/O - output library file. The switch always modifies *outputmaster* in D, I, M, N, and R functions.

Asterisk: Not permitted.

Errors:

Fatal Errors

NOT ENOUGH ARGUMENTS
UNEXPECTED ARGUMENT AT OR FOLLOWING: *string*
INVALID SWITCH FOR: *string*
NOT A LFE COMMAND: *key*
TOO MANY ARGUMENTS
ILLEGAL HEADER IN INPUT LIBRARY
CHECKSUM ERROR IN UPDATE FILE: *filename*
CHECKSUM ERROR IN LOGICAL RECORD: *recordname*
ILLEGAL BLOCK IN UPDATE FILE: *filename*
ILLEGAL BLOCK IN LOGICAL RECORD: *recordname*
FILE DOES NOT EXIST, FILE: *libraryfilename*

Non-Fatal Errors

UPDATE FILE MATCHES INPUTMASTER: *filename*
FILE DOES NOT EXIST, FILE: *updatefilename*
LOGICAL RECORD NOT FOUND - *recordname*
DEFAULT OUTPUT IN FILE - *filename*
FILE ALREADY EXISTS - *filename*

(See the LFE section of Appendix A for additional information on meanings of error messages.)

Extensions: If the .LB extension for *inputmaster* or the .RB extension for an update file are not given in the command, LFE searches for *inputmaster* or *arg* respectively. If not found, LFE searches for *inputmaster*.LB or *arg* .RB respectively.

Name: LFE (Continued)

Examples:

LFE N $PTP/O A.RB C.RB ↵

Create a library file, output to the punch from two disk disk files, A.RB and C.RB.

LFE R MATH.LB $PTP/O ATAN $PTR TAN/2↵

Output a new library file to the PTP, replacing ATAN on input file MATH.LB by a file on the PTR and replacing TAN on the input file by disk file TAN or, if not found, TAN.RB.

LFE A/M MATH1.LB $PTR $LPT/L MATH2.LB↵

Analyze library file MATH1.LB, $PTR, and MATH2.LB as one library and list results on the line printer.

LFE D $TTR UTIL/O MOVE LDBYT STBYT MULT↵

Delete logical records MOVE, LDBYT, STBYT, and MULT from $TTR and produce library file UTIL.

Name: LIST

Format: LIST [$filename_1$ ... $filename_n$]

Purpose: To list directory information from the default directory about one or more files, consisting of file name, byte count, and attributes. If LIST has no arguments, all non-permanent files are listed. If filenames are given, no filename can be preceded by a device specifier.

Switches:

Global:

/A - all files, permanent as well as non-permanent, are listed.

/B - brief list, giving file name but not byte count and attributes.

/L - listing printed to line printer ($LPT).

Local: None.

Asterisk: Permitted.

Errors: None.

Examples:

```
LIST ↵                                   ← lists all non-permanent files.
ACT.SV        1002   S
COM.SV        2345   S
COM.          40
B.            0
R

LIST/A ↵                                 ← lists all files
SYS.DR        512        APW
$TTI.         0          APW
$TTO.         0          RAP
$TTR.         0          APW
$PTR.         0          APW
$PTP.         0          RAP
$LPT.         0          RAP
CLI.SV        7354       SP
ACT.SV        1002       S
COM.SV        2345       S
COM.          40
B.            0
R
```

Name: LIST continued

Examples:

```
LIST/A *.SV ↵                          ← lists all .SV files
CLI.SV          7354      SP
ACT.SV          1002      S
COM.SV          2345      S
R

LIST/B ↵                               ← lists all non-permanent files without
ACT.SV                                   giving their attributes or byte counts.
COM.SV
COM.
B.
R
```

Name: LOAD

Format: LOAD *inputfilename* [*filename*$_1$ ... *filename*$_n$]

Purpose: To reload onto disk from a given file or device a previously dumped file or files. If no filenames are given, all non-permanent files on the input file are reloaded. LOAD can be used only to load previously dumped files (DUMP command). These files must be nonexistent prior to the LOAD command. If filenames are given, no name can be preceded by a device specifier.

Switches:

Global: /I - ignore checksum errors.
/V - verify the load with a list of names of loaded files.
/A - all including permanent files.

Local: None.

Asterisk: Permitted.

Errors:

FILE DOES NOT EXIST.	(input file)
FILE READ PROTECTED.	(input file)
FILE ALREADY EXISTS.	(output file)
ILLEGAL FILE NAME.	(input file)
DISK SPACE EXHAUSTED.	

Examples:

```
LOAD $PTR ↵
```

causes whatever previously dumped non-permanent files are in the paper tape reader to be reconstructed on disk under the same names. File name, length, and attributes are entered in the file directory.

```
LOAD /V $PTR *.SV ↵
LOAD $PTR, STRIKE ANY KEY.
EDIT.SV
ASM.SV
```

causes loading of all files with the extension .SV and a list of the files loaded.

Name: MKABS

Format: MKABS *save_filename absolute_binary_filename*

Purpose: To make an absolute binary file from core image (save) file.

MKABS gives users the facility of converting files that are executable under the operating system into absolute binary files that are executable, for example, on another machine without DOS.

Switches:

Global: /Z - save file starts at location zero. (See RLDR switches.)

Local: /S - starting address switch. An octal argument followed by /S will output an absolute binary start block with the address specified by the argument.

Asterisk: Not permitted.

Errors:

NOT ENOUGH ARGUMENTS.	
FILE DOES NOT EXIST.	(save file)
FILE ALREADY EXISTS.	(absolute binary file)
ILLEGAL FILE NAME.	
TOO MANY ARGUMENTS.	
DISK SPACE EXHAUSTED.	

Extensions: Search for *save_filename*.SV. If not found, search for *save_filename*.

Examples: MKABS FOO $PTP↵

punches an absolute binary file to the paper tape punch from file FOO.SV or, if not found, from FOO.

MKABS FOO $PTP 1000/S↵

punches an absolute binary file with a start block specifying 1000 as the starting address.

Name: MKSAVE

Format: MKSAVE *absolute_binary_filename save_filename*

Purpose: To create a core image (save) file from an absolute binary file.

Switches:

Global: /Z - create save file beginning at location Ø rather than 16_8.

Local: None.

Asterisk: Not permitted.

Errors:

PHASE ERROR.	(addresses not all in ascending order within the binary file)
NOT ENOUGH ARGUMENTS.	
ILLEGAL FILE NAME.	
FILE DOES NOT EXIST.	(absolute binary file)
FILE ALREADY EXISTS.	(save file)
DISK SPACE EXHAUSTED.	

Extensions: On output, produces *save_filename*.SV, regardless of the extension specified by the save file argument.

Example: MKSAVE $PTR DKØ:A↵

Causes creation of a core image file on fixed head disk unit Ø called A.SV, with the S attribute, from the absolute binary file loaded in the paper tape reader.

Name: OEDIT

Format: OEDIT *filename*

Purpose: To invoke the octal editor in order to examine and modify in octal any location in any type file. See Appendix A for a detailed description of the octal editor.

Switches: None.

Asterisk: Not permitted.

Errors: NO FILENAME SPECIFIED.
INPUT FILE DOES NOT EXIST.

Extensions: The octal editor searches for whatever file name and extension are given.

Example:

```
OEDIT FOO.SV↵
                          ← If OEDIT finds FOO.SV, the
                            editor gives a carriage return/
                            line feed.
14/ 016762                ← User proceeds with editing as
                            described in Appendix A.
.
.
.
HOME                      ← To return to the CLI, user
R                           types H. OEDIT echoes OME,
                            and user is at command level.
```

Name: PRINT

Format: PRINT *filename*$_1$ [*filename*$_2$ ... *filename*$_n$]

Purpose: To print a given file or files on the line printer. The command is the equivalent of a series of XFER commands:

XFER/A *filename*$_1$ $LPT; ... ; XFER/A *filename*$_n$ $LPT ↵

The source files may come from any device.

Switches: None.

Asterisk: Not permitted.

Errors:

ILLEGAL FILE NAME.	
FILE DOES NOT EXIST.	(source)
FILE READ PROTECTED.	(source)
LINE LIMIT EXCEEDED.	(source)
PARITY ERROR.	(source)

Example: PRINT FOO.SR DP2:COM.SR EXT.SR $PTR ↵

Name: PUNCH

Format: PUNCH *filename*$_1$ [*filename*$_2$ ... *filename*$_n$]

Purpose: To copy a given file or files to the high speed punch. The command is the equivalent of a series of XFER commands:

XFER/A *filename*$_1$ $PTP; ...; XFER/A *filename*$_n$ $PTP ↵

The source files may come from any device.

Switches: None.

Asterisk: Not permitted.

Errors:

ILLEGAL FILE NAME.	
FILE DOES NOT EXIST.	(source)
FILE READ PROTECTED.	(source)
LINE LIMIT EXCEEDED.	(source)
PARITY ERROR.	(source)

Example: PUNCH DKØ:ALPHA.SR BETA.SR $TTR ↵

Name: RELEASE

Format: RELEASE *device_specifier*

Purpose: To prevent further I/O access to a directory device or to rewind a magnetic tape unit. The command must be issued before a disk pack can be physically removed from a removable disk unit. No further access to the disk device is permitted unless an INIT command is executed.

Switches: None.

Asterisk: Not permitted.

Errors: ILLEGAL COMMAND FOR DEVICE.
DEVICE NOT IN SYSTEM.

Example: RELEASE DP1↵

The command permits the disk pack to be removed from moving head disk unit 1.

RELEASE MTØ↵

MTØ will be rewound.

Name: RENAME

Format: RENAME *oldname*$_1$ *newname*$_1$ [... *oldname*$_n$ *newname*$_n$]

Purpose: To change the current name of a file or files.

Switches: None.

Asterisk: Not permitted.

Errors:
PERMANENT FILE (old file)
ILLEGAL FILE NAME.
FILE ALREADY EXISTS (new file)
FILE DOES NOT EXIST.
FILES EXIST ON DIFFERENT DIRECTORIES.

Example:
```
DELETE Q.SV↵
R
RENAME QTEST.SV Q.SV↵
R
```

The commands above replace the old version of Q.SV with a new version, one previously named QTEST.SV.

```
RENAME DKØ:A1 DKØ:A  B1  B↵
```

Rename file A1 to A on fixed head disk unit Ø. Rename file B1 to B on the default directory.

Name: RLDR

Format: RLDR $filename_1$ [$filename_2$ ... $filename_n$]

Purpose: To create a save file from the loading of relocatable binary files and library files, which are collections of relocatable binary files. The command name, RLDR, must be used in loading relocatable binary files; the name RLDR cannot be changed.

Switches:

Global: The default conditions are:

1. No listing of the core map is produced.
2. The symbol table is built in memory but is not transferred to the save file after the relocatable binary files have been loaded.
3. The debugger is not loaded with the user files.

The switches that change the default conditions are:

/A produce an alphabetical and numerical core map. (The local /L switch must also be given to produce a core map.)

/D load symbolic debugger from SYS.LB. This causes the symbol table to be transferred to the save file after loading and also forces a system library search as in the /L global switch.

/L search SYS.LB after loading all user specified files.

/S leave symbol table at high end of memory if it is to be transferred to the save file. Normally, the symbol table would be transferred to locations just above all loaded programs in the save file.

/Z start save file at location zero. (A save file produced using the /Z switch cannot be executed properly under DOS. Its primary purpose is to enable loading of routines that use page zero locations 0-15. The save file can then be output using MKABS/Z to produce an absolute binary that can be read in stand-alone using the binary loader.)

Local: The default conditions are:

1. The first file name in the argument list becomes the name of the save file.
2. No user symbols are loaded.
3. Loading of NREL code proceeds into ascending contiguous locations beginning at 1000_8.
4. A listing of the core map is not produced.

Name: RLDR (Continued)

Switches: (Continued)

Local: The switches that change the default conditions are:

/L listing of the core map is produced to the file or device whose name precedes the switch. The listing is numeric unless the global /A switch is also set.

/N. NMAX, the starting address for loading a file, is forced to an absolute address given by the octal number preceding the switch. This becomes the starting address of the file whose name follows the switch. The absolute address must be higher than the current value of NMAX.

/S the save file is given the name that precedes the switch.

/U user symbols are loaded for the file whose name precedes the switch.

Asterisk: Not permitted.

Errors: ILLEGAL FILE NAME.
FILE ALREADY EXISTS.
FILE DOES NOT EXIST. (input file)
NO SOURCE FILE SPECIFIED.

Extensions: A search is made for each input file with the name *filename*.RB. If not found, then a search is made for *filename*. Names of library files must have the .LB extension.

The output save file will be $filename_1$.SV by default or will be the name preceding the /S switch with the .SV extension appended.

Examples: RLDR A B C DP2:D ↵

causes files A, B, and C from the default directory and D from disk pack unit 2 to be loaded to produce save file A.SV on the default directory.

RLDR A/S $PTR ↵

causes the file in the paper tape reader to be loaded and produce a save file A.SV.

RLDR/D MAIN SUB FORT1.LB FORT2.LB FORT3.LB FORT4.LB ↵

causes the assembled FORTRAN main program and subroutine, the FORTRAN libraries, and the debugger to be loaded as save file MAIN.SV.

RLDR $LPT/L D/S A 4400/N B ↵

causes A and B to be loaded as save file D.SV. Loading of B starts at 4400_8. A numeric core map is printed on the line printer.

Name: SAVE

Format: SAVE *filename*

Purpose: To create a save file from the file named BREAK.SV on the default directory device. The SAVE is commonly used to save the core image of a program interrupted by a CTRL C break. The SAVE command causes the most recent core image saved under the name BREAK.SV to be given a new name by deleting *filename*.SV (if it exists) and renaming the BREAK.SV to *filename*.SV. *filename* cannot be preceded by a device specifier.

Switches: None.

Asterisk: Not permitted.

Errors:
ILLEGAL FILE NAME.
DISK SPACE EXHAUSTED.
FILE DOES NOT EXIST. (BREAK.SV)

Extensions: Output always has the SV extension. If the filename argument already has an extension, the extension will be ignored, e.g. either

SAVE GAMMA↵ or SAVE GAMMA.YY↵

would produce the save file GAMMA.SV.

Examples:

```
.↵
DEB ALPHA ↵          ← enter debugger to correct location PP
PP/LDA 2,@ØLDA 2, @Ø,3
BREAK                ← exit from debugger
SAVE ALPHA ↵         ← save core image as ALPHA.SV
```

Name: TYPE

Format: TYPE *filename*$_1$ [*filename*$_2$ ... *filename*$_n$]

Purpose: To copy a given file or files to the teletypewriter. The command is the eqivalent of a series of XFER commands.

XFER/A *filename*$_1$ $TTO; ... ; XFER/A *filename*$_n$ $TTO ↵

The source files may come from any device.

Switches: None.

Asterisk: Not permitted.

Errors:

ILLEGAL FILE NAME.	
FILE DOESN'T EXIST.	(source)
FILE READ PROTECTED.	(source)
LINE LIMIT EXCEEDED.	(source)
PARITY ERROR.	(source)

Example: TYPE A.SR B.SR $PTR DP1:XX.SR ↵

Name: XFER

Format: XFER *sourcefile destinationfile*

Purpose: To transfer a file to another file.

Switches

Global: By default, files are transferred sequentially without alteration. There is one switch:

/A - ASCII transfer. Transfer the file line by line taking appropriate read/write action, such as inserting line feeds after each carriage return when transfer is from disk to line printer.

Local: None

Asterisk: Not permitted.

Errors:

ILLEGAL FILE NAME.	
FILE DOESN'T EXIST.	(source)
FILE READ PROTECTED.	(source)
LINE LIMIT EXCEEDED.	(ASCII source)
PARITY ERROR.	(ASCII source)
UNABLE TO WRITE FILE.	(destination)
FILE WRITE PROTECTED.	(destination)
NOT ENOUGH ARGUMENTS.	
DISK SPACE EXHAUSTED.	

Examples: XFER $PTR Q ↵

causes the file in the paper tape reader to be transferred to a disk file named Q.

XFER/A ALPHA.SR $LPT↵

causes ALPHA.SR to be printed on the line printer.

XFER $PTR $PTP ↵

causes another tape to be punched, identical to the one read from the paper tape reader.

XFER DPØ:MYFILE DP1:MYFILE ↵

transfers MYFILE from disk pack unit Ø to disk pack unit 1.

CHAPTER 4

PROGRAM MODE OF SYSTEM COMMUNICATION

SYSTEM COMMAND WORDS

The user communicates with the disk operating system (DOS) using system command words assembled into his program. System command words and the mnemonic .SYSTM that must precede the command word are recognized as legal mnemonics by the DOS assembler. Appearance of the mnemonic

```
.SYSTM
```

in a program results in the assembling of JSR @2 instruction which allows system communication through the main system entry address stored in page zero. The system command word must be assembled as the word following the .SYSTM.

Once system action is complete, normal return is made to the second instruction after the system command word. If an exceptional condition is detected, return is made to the first instruction following the system command word.

The general form of a system call in a program is

```
.SYSTM
command
exceptional return ;STATUS IN AC2
normal return      ;AC'S preserved or information returned as
                    specified for the particular command.
```

COMMAND WORD FORMAT

There are two basic command word formats:

command n and *command*

where: *n* is a digit (0-7)* representing an I/O channel number. The channel number indicates a logical link to an "opened" file.

---

* Any system command requiring a channel number *n* need not specify this number in the command itself. By specifying octal 77 as the channel number in the instruction, the system will use instead the number passed in AC2. For example, the following instructions specify a write to channel 3:

```
        LDA     2,C3
        .SYSTM
        .WRS    CPU
        JSR     EOF
        :
        :
C3:     3
```

COMMAND WORD FORMAT (Continued)

When no I/O channel is needed in command execution, the command word appears alone in the instruction. If the command requires arguments, these are passed in the accumulators.

STATUS ON RETURN FROM SYSTEM

Status of the accumulators upon return from the system is as follows:

If the system returns no information as a result of the call, the carry and all accumulators except AC3 will be preserved.

AC2 is used when an exceptional return is made to return a numeric error code. Error codes are listed by number at the end of this chapter and the applicable codes are listed for each command.

AC3 is destroyed by .SYSTM (as it is a JSR). On return from the system, however, AC3 is loaded from the contents of memory location 00016. This location is defined as a permanent symbol by the DOS assembler and has the name USP (User Stack Pointer). A convenient method of saving AC3 is to store it in location 00016 before issuing the .SYSTM.

LIST OF COMMAND WORDS

The command word mnemonics are:

.CREAT	Create a file
.DELET	Delete a file.
.RENAM	Rename a file.
.CHATR	Change file attributes.
.GTATR	Get file and device attributes.
.OPEN	Open a file.
.APPEND	Open a file for appending.
.CLOSE	Close a file.
.RESET	Close all open files.
.RDS	Read sequential characters.
.RDL	Read sequential line.
.RDR	Read random.
.WRS	Write sequential characters.
.WRL	Write sequential line.
.WRR	Write random.

LIST OF COMMAND WORDS (Continued)

.GCHAR	Read a character from TTI.
.PCHAR	Write a character to the TTO.
.MEM	Determine available memory space.
.MEMI	Allocate an increment of memory.
.BREAK	Save the current state of memory in save file format.
.EXEC	Execute a save file overlay.
.RTN	Return to the previously overlayed program at the normal return point.
.ERTN	Return to the previously overlayed program at the exceptional return point.
.INIT	Initialize a directory device.
.DIR	Change the current default directory device.
.RLSE	Release a directory device, preventing further file access.
.INST	Install a new DOS system from the default directory device.

DIRECTORY DEVICE MONITOR COMMANDS

DOS incorporates the ability to manage multiple directory devices simultaneously. The precise system configuration is specified via the SYSGEN program.

Directory devices are specified within the system by a three-character code, the first two characters of which specify device types and the third the unit number. For example,

DKØ

indicates fixed head disk (DK), unit Ø, while:

DP3

indicates moving head disk pack (DP), unit 3.

Initialize a Directory Device (.INIT)

A directory device is initialized via the following monitor command:

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

Initialize a Directory Device (.INIT) (Continued)

On entry to the system, ACØ contains a byte pointer* to a directory device specifier character string terminated by a null byte. If AC1 contains 177777, a full initialization of the device results: a virgin file directory and free storage map are constructed and written on the device. All previous files and other information are lost.

If AC1 does not contain 177777 when .INIT is invoked, a partial initialization of the device results. The current device file directory is located on the device and read into the system core area thus allowing subsequent file access to the device. All files on the device are now available to the system software.

The following error conditions might arise during the execution of the .INIT monitor command. When such a condition is encountered, the error return to the user program is taken with an error code in AC2.

AC2	Mnemonic	Meaning
2	ERICM	Illegal command for device.
36	ERDNM	Device not in system.

Changing a Default Directory Reference (.DIR)

All file name arguments to monitor commands may contain an optional device specifier. Those that do not are taken to be files on the current default directory device. At system initialization time, and after a bootstrap, the default device is set to be the master device (see SYSGEN, Appendix B). The current default device can be changed via the following monitor command:

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

On entry to the system ACØ contains the byte pointer to a directory device specifier character string terminated by a null byte. If a normal return to the user program is taken, the default directory device has been changed as specified. If the error return is taken, AC2 contains an error code indicating an abnormal condition; the default directory device has not been altered. The following error codes are possible:

* A byte pointer contains the word address in bits 0-14, which contain or will receive the byte. Bit 15 specifies which half (0 left, 1 right); note that this is the reverse of the byte pointer as specified in "How To Use the NOVA Computers." To use the subroutine shown on Page 2-21 of the NOVA manual, change the MOV Ø,Ø,SZC instruction to a MOV Ø,Ø,SNC.

Changing Default Directory References (Continued)

AC2	Mnemonic	Meaning
36	ERDNM	Device not in system.

Release a Device to Prevent Further File Access (.RLSE)

In order to prevent further I/O activity on a directory device, the following monitor command is provided.

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

On entry to the system, ACØ contains a byte pointer to a directory device specifier. If the normal return to the user program is taken, it is guaranteed that 1) all I/O activity to and from the device has subsided and 2) no further access will be permitted without the execution of the .INIT monitor command. In the case of a removable media directory device, .RLSE must be issued before the pack can be physically removed from the unit. (This is normally accomplished using the RELEASE CLI console command).

If the error return to the user program is taken, AC2 will contain an error code designating an abnormal condition.

The possible error codes are:

AC2	Mnemonic	Meaning
2	ERICM	Illegal command for device.
36	ERDNM	Device not in system.

Install a Bootstrap System (.INST)

DOS allows a user program to install a new DOS system under program control. When the installation is complete, the new system designated can be bootstrapped into operation in place of the current running system.

A file, previously opened on channel *n*, becomes the DOS core image that will be bootstrapped from the default directory device. The command to install the system is given on the following page.

Install a Bootstrap System (Continued)

```
.SYSTM
.INST n
error return
normal return
```

There is one possible error return to AC2 resulting from the command:

AC2	Mnemonic	Meaning
3	ERICD	Illegal bootstrap program for device.

FILE MAINTENANCE COMMANDS

File maintenance commands are used to enter file names into the file directory and perform file maintenance. All file maintenance commands require the file names to be specified by means of a byte pointer to the file name. The file name is stored as a character string. The string must consist of characters packed left to right (.TXTM 1) with the high order bit of each byte equal to 0. The string must have a terminating byte containing one of the following characters: null (000), form feed (014), carriage return (015), or space (040).

The extension of a file name (if any) is separated by the character ".". For example, the word at label "BPTR" contains a byte pointer to a properly specified file name, "MYFILE.SR".

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

Create a File (.CREAT)

This command causes an entry for the file name to be made in the system file directory. ACØ must contain a byte pointer to the file name. The format of the .CREAT command is:

```
.SYSTM
.CREAT
error return
normal return
```

Create a File (Continued)

Possible errors resulting from a .CREAT command are:

AC2	Mnemonic	Meaning
1	ERFNM	Illegal file name.
11	ERCRE	Attempt to create an existent file.

Delete a File (.DELET)

This command causes a file and its entry in the system file directory to be deleted. ACØ must contain a byte pointer to the file name. The format of the .DELET command is:

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

Possible errors resulting from a .DELET command are:

AC2	Mnemonics	Meaning
1	ERFNM	Illegal file name.
12	ERDLE	Attempt to delete a non-existent file.
13	ERDE1	Attempt to delete a permanent file.

Rename a File (RENAM)

This command causes the name of a file to be changed. ACØ must contain a byte pointer to the current name of the file. AC1 must contain a byte pointer to the new name. The format of the .RENAM command is:

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

Upon a normal return, the old name no longer appears in the file directory.

Possible errors resulting from a .RENAM command are:

Rename a File (Continued)

AC2	Mnemonic	Meaning
1	ERFNM	Illegal file name.
11	ERCRE	Attempt to create an existent name. (AC1).
12	ERDLE	Attempt to rename a non-existent file. (ACØ).
13	ERDE1	Attempt to rename a permanent file. (ACØ).
35	ERDIR	Files specified on different directories.

FILE ATTRIBUTE COMMANDS

File attribute commands allow the user to determine the current attributes of a file or device and to change the file attributes if desired. The bit settings of ACØ and AC1 determine the file attributes and device attributes respectively.

Change File Attributes (.CHATR)

This command causes the attributes of a file to be changed in accordance with the contents of ACØ. To change the attributes of a file, a file must be opened (see .OPEN). The number of the channel is given in the system command. The format of the .CHATR command is:

```
.SYSTM
.CHATR n
error return
normal return
```

When the .CHATR command is given, ACØ must contain an attribute word having the appropriate bit set for every attribute desired. The bit/attribute correspondence used in setting the contents of ACØ is given below in the table:

Bit	Mnemonic	Meaning
1BØ	ATRP	Read-protected file. Cannot be read.
1B1	ATCHA	Attribute-protected file. Attributes cannot be changed.
1B2	ATSAV	Save file (core image file).
1B14	ATPER	Permanent file. Cannot be deleted or renamed.
1B15	ATWP	Write-protected file. Cannot be written.

Possible errors resulting from a .CHATR command are:

AC2	Mnemonic	Meaning
0	ERFNO	Illegal channel number.
14	ERCHA	Illegal attempt to change file attributes.
15	ERFOP	Attempt to change attributes of an unopened file.

Get File Attributes (.GTATR)

This command obtains the attributes of a file or device characteristics. To obtain attributes, the file must be opened (see .OPEN). The number of the channel is given in the system command. The format of the .GTATR command is:

```
.SYSTM
.GTATR  n
error return
normal return
```

Upon return, AC0 contains the file attributes. The bit positions used to specify the file attributes were given with the .CHATR command. AC1 contains the device attributes of the file (e.g., $PTR). The bit/attribute correspondence used in interpreting the bit configuration returned in AC1 is shown below:

Bit	Mnemonic	Meaning
1B0	DCDIR	Directory device.
1B1	DCC80	Card input (80-column) device.
1B2	DCLTU	Device changing lower case ASCII to upper case.
1B3	DCFFO	Device requiring form feeds on opening.
1B4	DCFWD	Full word device (reads or writes more than a byte).
1B6	DCLAC	Output device requiring line feeds after carriage returns.
1B7	DCPCK	Input device requiring a parity check; output device requiring parity to be computed.
1B8	DCRAT	Output device requiring a rubout after every tab.
1B9	DCNAF	Output device requiring nulls after every form feed.
1B10	DCKEY	A keyboard input device.
1B11	DCTO	A keyboard output device.
1B12	DCCNF	Output device without form feed hardware.
1B13	DCIDI	Device requiring operator intervention.
1B14	DCCGN	Output device without tabbing hardware.
1B15	DCCPO	Output device requiring leader/trailer.

Possible errors resulting from a .GTATR command are:

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

INPUT/OUTPUT COMMANDS

All I/O is handled by system I/O commands. These commands require a channel number (0-7) to be given in the second field of the command word. A channel is initially linked to a particular file or device by means of the .OPEN (or. APPEND) command. Thereafter, all commands pertaining to that file merely require a channel number. The system provides three basic modes for reading and writing files.

The first mode is *line* mode where data read or written is assumed to consist of ASCII character strings terminated by either carriage returns or form feeds. In this mode, the system handles all device dependent editing at the device driver level. For example, line feeds are ignored on paper tape input devices and supplied after carriage returns to all paper tape output devices. Further, reading and writing never require byte counts, since reading continues until a carriage return is read and writing proceeds until a carriage return is written. The line mode commands are .RDL and .WRL.

The second mode is unedited *sequential* mode. In this mode, data is transmitted exactly as read from the file or device. No assumption is made by the system as to the nature of this information. Thus, this mode would always be used for processing *binary* files. This mode requires the user program to specify specific byte counts necessary to satisfy a particular read or write request. The sequential mode commands are .RDS and .WRS.

The third mode is available for processing files stored on devices capable of random access, e.g. any disk file. This mode provides for *random access* to files by means of record number. The random access mode commands are .RDR and .WRR.

The association of a file and a channel number can be broken by using the .CLOSE command. All currently open files can be closed using the .RESET command.

Open a File (.OPEN)

Before other I/O commands can be used, a file must be linked to a channel number. A byte pointer must be passed in AC0, pointing to the file name.

A "characteristic inhibit" mask must be passed in AC1. For every bit set in this word, the corresponding device characteristic (as defined on the previous page) is inhibited. The characteristics will be inhibited for the duration of the .OPEN. For example, if the user has an ASCII tape without parity to be read from the paper tape reader, he may inhibit parity checking by the following:

Open a File (Continued)

```
        LDA     0,READR
        LDA     1,MASK
        .SYSTM
        .OPEN   3

READR:  .+1*2
        .TXT    *$PTR*
MASK:   DCPCK                   ;PARITY CHARACTERISTIC
```

In general, the user will wish to leave all characteristics as defined by the system.

```
        SUB     1,1
```

before the [.SYSTEM / .OPEN *n*]

The format of the .OPEN command is:

```
        .SYSTEM
        .OPEN n                 ;OPEN CHANNEL n
        error return
        normal return
```

If the file opened requires leader, it will be output on the .OPEN. If the file opened requires intervention, the message:

LOAD *filename* ,STRIKE ANY KEY.

will be typed. Possible errors resulting from a .OPEN command are:

AC2	Mnemonic	Meaning
0	ERFNO	Illegal channel number.
1	ERFNM	Illegal file name.
12	ERDLE	File does not exist.
21	ERUFT	Attempt to use a channel already in use.
31	ERSEL	Unit not properly selected.

Open a File for Appending (.APPEND)

An alternate system call for opening a file is implemented that is identical to .OPEN in every respect except that it enables the writing of a file that already exists. Specifically, it opens the file for appending. The format of the call is:

Open a File for Appending (Continued)

```
.SYSTM
.APPEND  n
error return
normal return
```

As with .OPEN, ACØ must contain a byte pointer to the file name, and AC1 must contain the characteristic disable mask. For peripheral devices, such as the line printer, the call is in every respect identical to .OPEN. For a directory device, such as the disk pack, the file is opened and any writes to that file are appended to it and its length extended. For a device such as magnetic tape, a file is opened and read until end of file is encountered, and writing takes place from that point.

This command provides a convenient feature for subsystems running under DOS to append to the same output file.

Possible errors resulting from a .APPEND command are:

AC2	Mnemonic	Meaning
Ø	ERFNO	Illegal channel number.
21	ERUFT	Attempt to use channel already in use.

Close a File (.CLOSE)

After use, files must be closed to insure the updating of directory information. The channel number is then available for other I/O. The format of the .CLOSE command is:

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

If the file closed requires trailer, it will be output on the .CLOSE.

Possible errors resulting from a .CLOSE command are:

AC2	Mnemonic	Meaning
Ø	ERFNO	Illegal channel number.
15	ERFOP	Attempt to reference a channel not in use.

Close all Files (.RESET)

This command causes all currently open files to be closed. The format of the .RESET command is:

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

The error return from this command is never taken.

Read a Line (RDL)

This command causes an ASCII line, written with even parity, to be read. ACØ must contain a byte pointer to the starting byte address within the user area into which the line will be read.

Reading will terminate normally after transmitting either a carriage return or a form feed to the user. Reading will terminate abnormally after transmission of 132 (decimal) characters without detecting a carriage return or a form feed, upon detection of a parity error, or upon end of file. In all cases, the byte count read will be returned in AC1. If the read is terminated because of a parity error, the character having incorrect parity will be stored (high order bit zero) as the last character read. The byte pointer to the character can always be computed as:

C(ACØ)*+C(AC1) -1

The format of the .RDL command is:

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

Possible errors resulting from a .RDL command are:

AC2	Mnemonic	Meaning
0	ERFNO	Illegal channel number.
3	ERICD	Illegal command for device.
6	EREOF	End of file.
7	ERRPR	Attempt to read a read protected file.
15	ERFOP	Attempt to reference a file not opened.
22	ERLLI	Line limit (132 characters) exceeded.
24	ERPAR	Parity error.
26	ERMEM	Attempt to allocate more memory than is available.
30	ERFIL	File read error.

* C(α) means "contents of (α)"

Read Sequential (RDS)

Sequential mode transmits data exactly as read from the file. AC0 must contain a byte pointer to the starting byte address within the user area into which the data will be read and AC1 must contain the number of bytes to be read. The format of the .RDS command is:

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

Possible errors resulting from a .RDS command are:

AC0	Mnemonic	Meaning
0	ERFNO	Illegal channel number.
3	ERICD	Illegal command for device.
6	EREOF	End of file.
7	ERRPR	Attempt to read a read protected file.
15	ERFOP	Attempt to reference a file not opened.
26	ERMEM	Attempt to allocate more memory than is available.
30	ERFIL	File read error.

Upon an end of file, the partial count read will be returned in AC1.

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

When using $CDR (card reader) as an input device, the end of file condition on a .RDL will occur only if a special end of file code is detected in column 1 of a card. This code is at 12,11,0,1 multipunch. It can be punched on a 029 keypunch by multipunching "A", "0", and "-".

Note also that a Hollerith to ASCII translation only occurs if a .RDL has been requested. The translation assumes 029 keypunch codes. A table of Hollerith - ASCII translation is given on page 4-16.

If .RDS is given, the card is read in image binary. In this case, an even byte count must be specified, since two bytes are required to store each card column. If an odd number of bytes is requested, status ERICD, Illegal Command for Device, is returned in AC2. Each two bytes will be used to store a single column. The packing is shown on the following page.

Byte	← 1 →								← 2 →					
Column Number	12	11	0	1	2	3	4	5	6	7	8	9		
Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	Ø	Ø	Ø	Ø	d	d	d	d	d	d	d	d	d	d	d	d

The "d's" will be 1 for every column punched.

Hollerith				ASCII		Hollerith				ASCII	
12	11	0	1-9	Character	Octal	12	11	0	1-9	Character	Octal
0	0	0	0	space	040	0	0	0	8,1	form feed	014
0	0	0	1	1	061	0	0	0	8,2	:	072
0	0	0	2	2	062	0	0	0	8,3	#	043
0	0	0	3	3	063	0	0	0	8,4	@	100
0	0	0	4	4	064	0	0	0	8,5	'	047
0	0	0	5	5	065	0	0	0	8,6	=	075
0	0	0	6	6	066	0	0	0	8,7	"	042
0	0	0	7	7	067						
0	0	0	8	8	070	0	0	1	8,1	tab	011
0	0	0	9	9	071	0	0	1	8,2	]	135
						0	0	1	8,3	,	054
						0	0	1	8,4	%	045
0	0	1	0	0	060	0	0	1	8,5	←	137
0	0	1	1	/	057	0	0	1	8,6	>	076
0	0	1	2	S	123	0	0	1	8,7	?	077
0	0	1	3	T	124						
0	0	1	4	U	125	0	1	0	8,2	!	041
0	0	1	5	V	126	0	1	0	8,3	$	044
0	0	1	6	W	127	0	1	0	8,4	*	052
0	0	1	7	X	130	0	1	0	8,5	)	051
0	0	1	8	Y	131	0	1	0	8,6	;	073
0	0	1	9	Z	132	0	1	0	8,7	\	134
0	1	0	0	-	055	1	0	0	8,2	[	133
0	1	0	1	J	112	1	0	0	8,3	.	056
0	1	0	2	K	113	1	0	0	8,4	<	074
0	1	0	3	L	114	1	0	0	8,5	(	050
0	1	0	4	M	115	1	0	0	8,6	+	053
0	1	0	5	N	116	1	0	0	8,7	↑	136
0	1	0	6	O	117						
0	1	0	7	P	120						
0	1	0	8	Q	121						
0	1	1	9	R	122						
1	0	0	0	&	046						
1	0	0	1	A	101						
1	0	0	2	B	102						
1	0	0	3	C	103						
1	0	0	4	D	104						
1	0	0	5	E	105						
1	0	0	6	F	106						
1	0	0	7	G	107						
1	0	0	8	H	110						
1	0	0	9	I	111						

Hollerith-ASCII Translation Table

Read Random (.RDR)

Random access files are assumed to consist of fixed length, 64-word records. These records are numbered sequentially from Ø. No EOF is ever given on a read or write random. A read random of a record number never written will result in a 64-word record of zeroes. The length of a random access file is computed by the system as:

(highest record number written + 1) * 128_{10} bytes

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

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

Possible errors resulting from the .RDR command are:

AC2	Mnemonic	Meaning
0	ERFNO	Illegal channel number.
3	ERICD	Illegal command for device.
7	ERRPR	Attempt to read a read protected file.
15	ERFOP	Attempt to reference a file not opened.
26	ERMEM	Attempt to allocate more memory than is available.
30	ERFIL	File read error.

Write a Line (.WRL)

This command presumes an ASCII file. ACØ must contain a byte pointer to the starting byte address within the user area from which characters will be read.

Writing will terminate normally upon writing of a null, a carriage return or a form feed, and abnormally after transmission of 132 (decimal) characters without detection of a carriage return, a null, or a form feed. In either case, AC1 will contain, upon termination, the number of bytes read from the user area to complete the request. The termination of a write line on a null allows for formatting output without forcing a carriage return.

Write a Line (Continued)

The format of the .WRL command is:

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

Possible errors resulting from the .WRL command are:

AC2	Mnemonic	Meaning
0	ERFNO	Illegal channel number.
3	ERICD	Illegal command for device.
5	ERWRO	Attempt to write an existent file. (Detected upon the first attempted write.)
10	ERWPR	Attempt to write a write protected file.
15	ERFOP	Attempt to reference a file not opened.
22	ERLLI	Line limit (132 characters) exceeded.
27	ERSPC	Out of disk space.

Write Sequential (.WRS)

This command transmits data exactly as read from the user area. AC$\emptyset$ must contain a byte pointer to the starting byte address of the data within the user area and AC1 must contain the number of bytes to be written. The format of the .WRS command is:

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

Possible errors resulting from a .WRS command are:

AC2	Mnemonic	Meaning
0	ERFNO	Illegal channel number.
3	ERICD	Illegal command for device.
5	ERWRO	Attempt to write an existent file. (Detected upon the first attempted write).
10	ERWPR	Attempt to write a write-protected file.
15	ERFOP	Attempt to reference a file not opened.
27	ERSPC	Out of disk space.

Write Random (.WRR) (Continued)

ACØ must contain the source record core address and AC1 must contain the number of the record to be written. Sixty-four words will be written, starting from the address specified in ACØ. The format of the .WRR command is:

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

Possible errors resulting from a .WRR command are:

AC2	Mnemonic	Meaning
0	ERFNO	Illegal channel number.
3	ERICD	Illegal command for device.
10	ERWPR	Attempt to write a write protected file.
15	ERFOP	Attempt to reference a file not opened.
27	ERSPC	Out of disk space.

TELETYPEWRITER COMMANDS

Buffered transfer of single characters between the teletypewriter and ACØ is handled by the commands, .GCHAR and .PCHAR. No channel number is required for these commands, and the teletype is always available to them without requiring the .OPEN command.

Get a Character (.GCHAR)

This command returns a character typed from the teletypewriter in ACØ. The character is right-adjusted in ACØ with bits 0-8 cleared. No channel is required; the TTI is always used as input for this command. The format of the .GCHAR command is:

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

No error return is possible from this command; if no character is currently in the TTI input buffer, the system waits.

Put a Character (.PCHAR)

This command transmits a character in ACØ, bits 9-15, to the teletypewriter. No channel is required; the TTO is always used as output for this command. The format of the .PCHAR command is:

Put a Character (Continued)

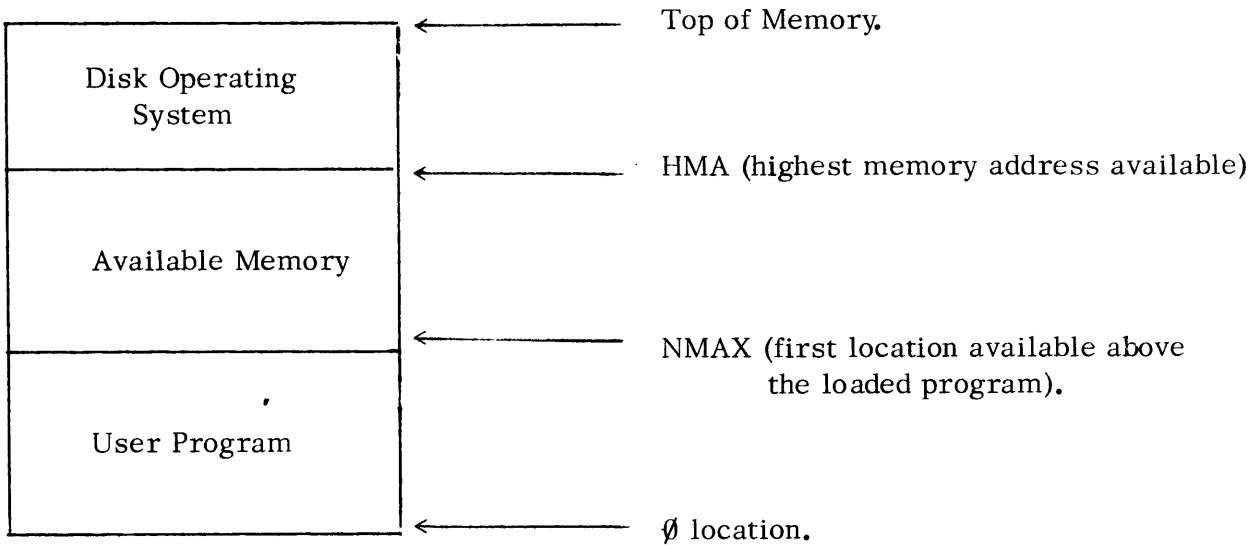
```
.SYSTM
.PCHAR
error return
normal return
```

No error return is possible from this command.

MEMORY COMMANDS

Appendix A shows in detail how memory is allocated using the Disk Operating System. The following is a simplified diagram.

The Disk Operating System resides in upper memory. User programs are loaded in lower memory. Memory then looks, essentially, as follows:



The highest memory address available (HMA) is usually the first word below the Disk Operating System. If a user symbol table has been loaded at the high end of user memory, the high memory address will be the first word below the user symbol table. The latter will occur when the user specifically requests that the relocatable loader leave the table in upper memory. (The loader, by default, moves the symbol table down so that the bottom of the table coincides with the first location not loaded into by the program).

The .MEM command returns both the current value of NMAX and HMA. The .MEMI command allows the user to adjust the value of NMAX.

Determine Available Memory (.MEM)

This command returns the current value of NMAX in AC1 and the value of HMA in ACØ. HMA may represent either the bottom of DOS or the end of the user symbol table. A SUB 1, 0 instruction determines the limit of memory available to the user program. The format of the .MEM command is:

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

There are no error returns from this command.

Change NMAX(.MEMI)

This command allows the user to increase or decrease the value of NMAX. The increment or decrement (in two's complement) is passed in ACØ. The command causes the value of NMAX to be updated in the User Status Table* and the new NMAX to be returned in AC1. The format of the .MEMI command is:

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

NMAX will not be changed if the new value of NMAX would be higher than the lowest address of the Disk Operating System. No check is made as to whether or not the user decreases NMAX below its original value (as determined at relocatable load time) nor, if his symbol table resides in upper memory, whether he increases NMAX above the bottom of his symbol table.

Whenever a user program requires memory space above the loaded program, a .MEMI should be executed first to allocate the number of words needed. The value of NMAX is used by the operating system to determine the extent of memory to be saved should a program be suspended. If temporary storage is being used without having updated NMAX, the program may be suspended with insufficient information for continuation. This is explained further in the discussion of Program Overlays.

There is one error resulting from a .MEMI command:

AC2	Mnemonic	Meaning
26	ERMEM	Attempt to allocate more memory than available. (Attempted overlap with DOS).

* See Appendix A, Relocatable Loader Section.

PROGRAM OVERLAY COMMANDS

Program Overlays

Any program executing under the operating system can suspend its own execution and invoke another program. Every program requested for execution must exist as a save file on disk.

The program that is suspended is stored temporarily on disk; its User Status Table is used to save its AC's, Carry, and the PC (Program Counter) at the time of its suspension. This information enables the program to be resumed upon termination of the program overlay being requested.

The "calling" program will be referred to as executing at a "higher level" in the system. The "called" program, or program overlay, will be referred to as running at a "lower level". These terms are relative, since the called program may in turn invoke another program overlay for execution, and therefore, become the calling program.

Upon program suspension, the current core image will be saved up to the higher of NMAX or SST (start of the user symbol table). It is very important for a program using temporary storage above its original value of NMAX at load time to have the system allocate memory increments (see .MEMI) before this space is used. If this is not done and the program invokes another program, the calling program's suspended memory state will not all be saved in its entirety. Even if the program executing does not call another program, a teletype BREAK may force suspension. In order to insure the ability to restart a suspended routine, NMAX must correctly reflect the core in use at the time.

The Command Line Interpreter is merely one program executable under the operating system. Its only special property is that it executes at the "highest" level in the system. This will be called "level zero".

The operating system provides for up to five levels of program overlays. This implies a program invoked by the CLI (causing the CLI to be overlayed) can in turn invoke a third program (causing the second program to be overlayed) ; the third program can invoke a fourth, and the fourth can invoke a final program. The system will reject all further overlay attempts. Normally, the system programs supported by the CLI (e.g., Text Editor, Assembler, Relocatable Loader) execute at level one.

Read in a Save File Overlay (.EXEC)

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

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

AC0 must contain a byte pointer to the program save file name. AC1 must contain an appropriate starting address code. Three possible starting addresses are allowed: the program starting address (USTSA), the Debug III starting address (USTDA), or the previously saved program counter (USTPC).* 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 codes permissible in AC1 are:

Code	Meaning
0	Starting address
1	Debugger address
2	Program counter

If the code is not one of these three or the address required is not given in the User Status Table, ERADR status is returned. This can occur if:

A. No starting address was specified for the save file and code 0 is given.

B. The debugger was not loaded as part of the save file and code 1 is given.

C. The save file was not the result of a BREAK (CLI) or .BREAK and code 2 is given.

Error returns possible before the overlay has been read into memory are:

AC2	Mnemonic	Meaning
1	ERFNM	Illegal file name.
4	ERSV1	File requires "S"ave attribute.
12	ERDLE	File does not exist.
25	ERCM3	Trying to push too many levels.
26	ERMEM	Attempt to allocate more memory than is available.
32	ERADR	Illegal starting address.

* See Appendix A, Relocatable Loader section, for descriptions of USTSA, USTDA, and USTPC.

Return from Overlay (.RTN)

Upon successful completion of a program invoked by a .EXEC, this command causes return to the "calling" program at its normal return point. The format of the .RTN command is:

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

Note that the usual "normal return" is impossible, since, if execution of the return is successful, the calling program is restored to memory. The one possible error return is:

AC2	Mnemonic	Meaning
23	ERRTN	Attempt to restore a non-existent image.

This error return is only possible if the program at execution level zero issues a .RTN. Since the CLI executes at level zero, system and user programs should never obtain this error indicator.

Return to the calling program preserves, as normal, AC0, AC1, AC2, and Carry. AC3 will contain the contents of the calling program's User Stack Pointer, C(USP).

Return from Overlay with Exceptional Status (.ERTN)

A called program can return exceptional status information to the calling program with this command. The format of the .ERTN command is:

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

This call is identical to .RTN in every respect except that return is made to the error return of the calling program and AC2, upon return, contains the called program's AC2 instead of the calling program's AC2. A single word of status can, therefore, be returned. If the program issuing a .ERTN had been executing at level one (and is returning, therefore, to the CLI), the CLI will output an appropriate message concerning the status code in AC2. If the code is recognized as a monitor exceptional status code, a text message is printed. The code ERDLE (12_8) for example, would cause the message:

```
FILE DOES NOT EXIST
```

to be typed out. If the code is greater than any system codes, the message:

UNKNOWN ERROR CODE *n*

will be typed out, where *n* is the numeric code in octal. This feature is useful for returning status unrelated to the operating system but directly related to the user program that was running.

Saving Current State of Memory (.BREAK)

DOS provides a system call for conveniently saving the state of memory in save file format. The format of call is

```
.SYSTM
.BREAK
error return
normal return
```

This call causes the operating system to save the current state of memory from location SCSTR (the start of save files) to the higher of NMAX or the start of the symbol table, SST. The file name used is BREAK.SV; any previous file BREAK.SV is deleted by the command. The device used is the current default directory device. File BREAK.SV is in every respect an executable save file. Note that when using the CLI command SAVE, the CLI merely renames BREAK.SV to be the file name specified by the command.

One error message is possible:

AC2	Mnemonic	Meaning
27	ERSPC	Out of disk space.

ERROR MESSAGES

CODE	MNEMONIC	MEANING	APPLICABLE COMMANDS
0	ERFNO	ILLEGAL CHANNEL NUMBER	.CHATR .OPEN .RDS .WRS .GTATR .CLOSE .RDL .WRL .APPEND .RDR .WRR
1	ERFNM	ILLEGAL FILE NAME	.CREAT .RENAM .EXEC .DELET .OPEN
2	ERICM	ILLEGAL SYSTEM COMMAND	.INST .RLSE .INIT
3	ERICD	ILLEGAL COMMAND FOR DEVICE	.RDR .RDS .WRS .WRR .RDL .WRL
4	ERSV1	FILE REQUIRES THE "SAVE" ATTRIBUTE	.EXEC
5	ERWRO	ATTEMPT TO WRITE AN EXISTENT FILE	.WRS .WRL
6	EREOF	END OF FILE	.RDS .RDL
7	ERRPR	ATTEMPT TO READ A READ PROTECTED FILE	.RDS .RDL .RDR
10	ERWPR	ATTEMPT TO WRITE A WRITE PROTECTED FILE	.WRS .WRL .WRR
11	ERCRE	ATTEMPT TO CREATE AN EXISTENT FILE	.CREAT .RENAM .OPEN
12	ERDLE	ATTEMPT TO REFERENCE A NON-EXISTENT FILE	.DELET .RENAM .EXEC .OPEN
13	ERDE1	ATTEMPT TO ALTER A PERMANENT FILE	.DELET .RENAM
14	ERCHA	ILLEGAL ATTEMPT TO CHANGE FILE ATTRIBUTES	.CHATR
15	ERFOP	ATTEMPT TO REFERENCE A FILE NOT OPENED.	.CHATR .RDS .WRL .GTATR .RDL .WRS .CLOSE .RDR .WRR

ERROR MESSAGES (Continued)

CODE	MNEMONIC	MEANING	APPLICABLE COMMANDS
21	ERUFT	ATTEMPT TO USE A CHANNEL ALREADY IN USE	.OPEN .APPEND
22	ERLLI	LINE LIMIT EXCEEDED ON READ OR WRITE LINE	.RDL .WRL
23	ERRTN	ATTEMPT TO RESTORE A NON-EXISTENT IMAGE	.RTN .ERTN
24	ERPAR	PARITY ERROR ON READ LINE	.RDL
25	ERCM3	TRYING TO PUSH TOO MANY LEVELS	.EXEC
26	ERMEM	ATTEMPT TO ALLOCATE MORE MEMORY THAN AVAILABLE	.RDL .RDS .MEMI .RDR
27	ERSPC	OUT OF DISK SPACE	.WRS .WRL .BREAK .WRR
30	ERFIL	FILE READ ERROR	.RDS .RDL .RDR
31	ERSEL	UNIT NOT PROPERLY SELECTED	.OPEN
32	ERADR	ILLEGAL STARTING ADDRESS	.EXEC
35	ERDIR	FILES SPECIFIED ON DIFFERENT DEVICES.	.RENAM
36	ERDNM	ILLEGAL DEVICE NAME	.INIT .RLSE

CHAPTER 5

MULTIPLE FILE DEVICES

DEVICES PROVIDING MULTIPLE FILE ACCESS

At present there are three possible types of devices on which system and user files can be stored that the Disk Operating System can readily access for multiple reading and writing. These are:

1. Fixed head disk - Usually one per system configuration, having the mnemonic:

   DKØ

   If there were more than one fixed head disk controller, the second controller would be designated DK1.

2. Disk pack of the movable head disk - There can be up to four disk packs per disk controller and they are designated:

   DPØ, DP1, DP2, DP3

3. Magnetic tape units - Up to eight magnetic tape drives are permitted per system, and they are designated:

   MTØ, MT1, . . . MT7

DETERMINING SYSTEM DEVICE CONFIGURATION

As described in Appendix B, once a bootstrap system has been loaded during system generation, the program SYSGEN queries the user as to the system configuration. Responses to SYSGEN (in addition to determining storage, I/O devices, etc.) will determine:

1. Which device will be the master storage device (fixed head disk or a given disk pack unit.)

2. How many disk pack units, if any, and how many magnetic tape drives, if any, are in the system.

DIRECTORY DEVICES

Directory devices are those devices that have their own file directory, containing the names, attributes, and byte counts of all files stored on the device. Only disk

DIRECTORY DEVICES (Continued)

packs and the fixed head disk can maintain such a file directory. See Chapter 1 for a discussion of disk file directories.

Files are stored on magnetic tape by file number, the number indicating the order in which they were written onto the tape. DOS accesses files on magnetic tape by their file number, not by referencing a file name in a directory.

MAGNETIC TAPE

DOS has access to files on magnetic tape and the DOS system will support up to eight magnetic tape drives in any combination of 7 and 9 track units. Reading and writing is at high density (800 bpi). If the unit specified is selected to low density or is not on-line, the message:

UNIT IMPROPERLY SELECTED

will be given.

If the control detects a parity error during reading, the message:

PARITY ERROR: FILE MTn:dd $\quad \emptyset \leq n \leq 7; \quad \emptyset \leq dd \leq 99$

will be given and two results are possible:

1. If a dump file was being LOADed, execution will terminate.

2. If a file was being XFERed, execution will continue; however, the first 128 words of the erroneous record will be lost.

7-Track Units

Data recorded on 7-track units is necessarily encoded. This is accomplished in the following manner;

Original Data Word	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Encoded Data Words	x	x	x	x	0	1	2	3	x	x	x	x	4	5	6	7
	x	x	x	x	8	9	10	11	x	x	x	x	12	13	14	15

Every data word is written on a 7-track unit as two encoded data words.

Number of Tape Drives in System

During system generation (Appendix B), the SYSGEN program will query the user as to how many magnetic tape drives are in the system configuration. That number of drives

Number of Tape Drives in System (Continued)

may then be referenced by device mnemonic. DOS can directly access files on magnetic tape by file number.

Initializing a Tape Drive

Initializing a tape drive causes the tape on that drive to be rewound. Full initialization (/F switch) will cause the tape to be rewound, and a dummy record and two EOF's will then be written.

Releasing a Tape Drive

To rewind a tape drive, the RELEASE command can be given.

Division of Magnetic Tape into Files and Records

Logically, magnetic tape is divided into files, which are placed on tape in numeric order, beginning with file 0. Up to 100 files may be written to a given tape, i.e., files 0-99.

Physically, magnetic tape is divided into magnetic tape records. These records contain 377_8 words, terminated by a word containing the number of the file and a word containing the number of the record.

Referencing a File on Magnetic Tape

A given file is referenced in a command by a tape drive specifier followed by file number. Either a one-digit or two-digit number may be used to reference the first ten file numbers, e.g.,

MT1:04 and MT1:4 are equivalent.

Both the tape drive specifier and the file number must be given. The file number must be in the range 0-99, and the tape drive number must be in the range 0-7. Otherwise; the error message:

ILLEGAL FILE NAME

will be output. If the user referenced a file on tape by name rather than number, e.g.,

MT0:XX

the system gives the error message:

ILLEGAL FILE NAME

Some examples of references to files on tape are:

DUMP MT0:0↵	Dump all non-permanent files onto tape, providing a magnetic tape backup system.
DELETE *.*↵	Delete all non-permanent files from disk.
LOAD MT0:0↵	Reload the files onto disk from tape.

ASM A B MTØ:1/B↵

Assemble files A and B and write the relocatable binary to tape drive Ø, file 1.

RLDR A.SV/S MT1:2 MT1:3 MT1:4↵

Load relocatable binary files 2, 3, and 4 from drive 1, producing save file A.SV.

XFER MTØ:2 FOO↵
XFER MTØ:2 MT1:5↵

Transfer a file from tape to disk or from tape to tape.

XFER FORTEST.FR MTØ:Ø↵

Place FORTRAN source file on tape.

FORT MTØ:Ø↵

Compile from tape, producing a relocatable binary file called Ø.RB

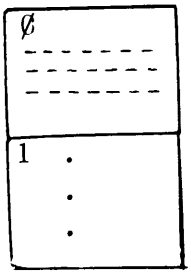
Note that when a file on tape is assembled or compiled onto disk or disk pack as in the example above, the name of the relocatable binary file on disk will become the file number with the extension RB.

Writing Files to Magnetic Tape

Files must be placed on magnetic tape in numeric order. For example, suppose the user transfers a file to tape that has just been initialized:

XFER FILEØ MTØ:Ø↵

FILEØ will be the first file on the tape. The tape on drive Ø will now contain the following:



First file, containing contents of FILEØ.

Once a file is written, the file number of the next file is assigned. File 1 is a null file.

An attempt to place a new file on the tape above with one of the following commands:

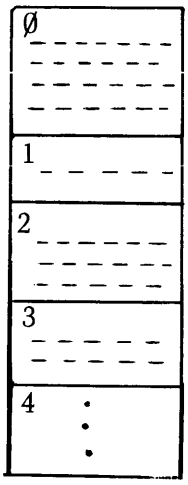
```
XFER FILEX MTØ:2↙
XFER FILEY MTØ:4↙
```

where only file Ø has been written to tape.

will result in an error message:

```
ILLEGAL FILE NAME
```

It is possible to overwrite a magnetic tape file. For example, assume a tape on drive Ø contains four files:

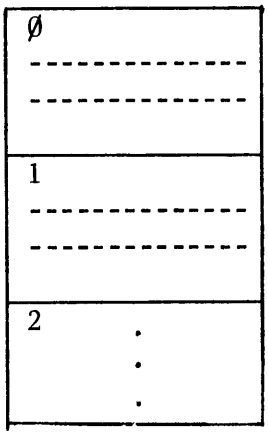


Null file.

The command:

```
XFER MYFILE MTØ:1↙
```

will cause the contents of MYFILE to overwrite the tape beginning at the file 1 position. When a tape file is written in this manner, all subsequent files on tape are lost. In the example, the tape will contain:



← Null file

CHAPTER 6

USER SERVICED INTERRUPTS

A facility has been provided for the user to service his own interrupts using DOS. The procedure and restrictions are outlined below.

Two parameters are defined mnemonically by the Disk Operating System parameter tape 090-000176. The first, USTIS, defines a displacement into the User Status Table where the user must store the address of his own interrupt service routine. The second, UMSK, contains the address of a DOS subroutine that properly maintains the interrupt mask word and enables interrupts for the user.

If the user does not change the word at location USTIS, a PANIC with code 210 is given when an interrupt is detected that is not recognized by DOS. However, if the user initializes the word at USTIS to contain an address within his own program, control is transferred to that address via JSR when an interrupt not recognized by DOS is detected.

When the user receives control, interrupts are OFF and AC0 contains the device code of the interrupting device. The user may examine this code or, alternatively, skip on the appropriate DONE flip flops of the devices from which he expects interrupts. It is the user's responsibility to save AC3 and to return to this address after completion of his service routine, but all other accumulators and carry are saved for him by the system.

If the user can complete his servicing of his device with interrupts disabled and thus is not concerned about interrupts from other devices during servicing, he does not need to make use of the subroutine provided for maintaining the current mask word. In general, however, the user should use this subroutine to mask out his device and all lower priority devices and to turn on interrupts again. To use this routine, the user must supply in AC0 a bit corresponding to the interrupt disable mask bit for every device, including his own, that he considers of lower priority. Note that as a minimum the user must set a bit in AC0 corresponding to his own device interrupt disable flip flop. The calling sequence for this routine is simply

```
JSR @UMSK
```

The system will maintain the mask word properly, turn on interrupts, and return control to the user with AC0 containing the new mask word.

CHAPTER 7

PANICS

There are a number of hardware malfunctions that may cause the system to "PANIC". Should a PANIC occur, the contents of the accumulators will be printed on the TTY, followed by a PANIC code. The output will appear as follows:

```
00015 177777 000011 037500 000210
ACØ   AC1    AC2    AC3    PANIC CODE
```

The PANIC codes are:

210 - Unknown interrupt. Offending device code in ACØ.

220 - System stack overflow.

230 - Repeated critical disk write errors.

240 - Repeated critical read errors.

250 - Repeated critical disk read or write errors.

260 - Runaway tape reader. (An NIOC to an input device did not stop its forward motion).

270 - Fatal magnetic tape hardware status. ACØ contains the magnetic tape controller status.

APPENDIX A

DOS SYSTEM PROGRAMS

Programs supported under DOS are:

Text Editor	Document #093-000018
Relocatable Assembler	Documents #093-000002, #093-000040, and "How to Use the NOVA Computers"
Relocatable Loader	Document #093-000039
Debug III	Document #093-000044
Octal Editor	Document #093-000048 (this manual)
FORTRAN IV	Document #093-000053
Extended ALGOL	Document #093-000052
Library File Editor	Document #093-000048 (this manual)
CLI	Document #093-000048 (this manual)
CLG	Document #093-000048 (this manual)
BLDR (Binary Loader)	Document #093-000048 (this manual)

Certain minor changes were made to the Text Editor, Assembler, Relocatable Loader and Debug III for use under DOS. These changes are outlined in this Appendix. The Library File Editor and the Octal Editor are described in detail here. In addition, information on loading the FORTRAN IV and Extended ALGOL compilers under DOS is included.

TEXT EDITOR

The Text Editor is supplied as one file of a dumped tape, 088-000001, and is named EDIT.SV. To use the Text Editor, the user must create a save file from the tape by mounting the tape in an appropriate input device, such as the $PTR, and giving the LOAD command:

LOAD $PTR↵

The CLI responds with a ready message (R) when the disk file has been created.

Once a disk file of EDIT.SV exists, the command

EDIT↵

brings the Text Editor into core. The Text Editor gives the prompt:

*

The user should reply to the prompt with DOS Text Editor commands that specify input and output file names:

GR*inputfilename*$$	←get for reading (input) the file of the name given. *inputfilename* must be the name of an existing file.
GW*outputfilename*$$	←get for writing (output) the file of the name given. *outputfilename* cannot yet exist.

Upon completion of editing an input file to produce a new output file, the command:

GC$$

can be used to close the output file. A new output file can then be edited by specifying the GW command. If the user attempts to open an additional file for writing without closing the current file, the following error message is given:

OUTPUT FILE ALREADY ACTIVE

The GC command does not force writing of the last output page. The user must be careful to issue a P (or E) command for the last edited page before issuing the GC command.

To return to CLI level, the user issues an H command. Thus, to force writing of the last output page, close out the file, and return to command level the sequence of commands would be:

EGCH$$

The meaning of the Y and P commands is basically unchanged by the change in I/O:

Y$$	- read a page from the input file. (As before, a page is a stream of characters terminated by a Form Feed.)
P$$	- write a page to the output file.

A CTRL C break will cause a return to the CLI and should not normally be used while editing. A CTRL A break can be issued to terminate Editor input/output. The Editor will remain in core and issue an asterisk (*). If there is file I/O in progress at the time of issuing a CTRL A, both input and output files will be closed.

The DOS Text Editor commands are given in the following list. The format shows individual commands terminated by striking two ESC keys ($$), which causes the editor to execute the command. Usually, the programmer will issue a string of editing commands before initiating execution by striking $$; for example:

BSGETC:$-1L2T$$

Note that of the special CTRL commands, Only CTRL A should normally be used.

Command	and Format	Meaning
A	A$$	Append a page to the edit buffer.
B	B$$	Move character pointer to beginning of buffer.
C	C$string_1$string_2$$	Search buffer for $string_1$ and change to $string_2$.
D	nD$$	Delete n characters starting at character pointer.
E	E$$	Output buffer and remainder of input file.
F	F$$ nF$$	Punch a form feed. Punch n inches of leader.
GC	GC$$	Close the output file.
GR	GRinputfilename$$	Get for reading (input) file of name given and close any previous input file.
GW	GWoutputfilename$$	Get for writing (output) the file of name given.
H	H$$	Return to CLI
I	Istring$$ nI$$	Insert string at character pointer position. Mask n to 7 bits and insert character at character pointer.
↑I	(tab) Istring$$	Tabulate and insert string

TEXT EDITOR (Continued)

Command	Format	Meaning
J	nJ$$	Jump character pointer n lines from beginning of buffer.
K	nK$$	Delete (kill) n lines starting from current position.
L	nL$$	Reset character pointer n lines from current position.
M	nM$$	Move character pointer n characters from current position.
N	Nstring$$	Search for string. If not found, output, read, and continue search through input.
P	P$$	Write a page to the output file.
	nP$$	Write n lines to the output file starting at character pointer.
PW	PW$$	Punch entire edit buffer without a form feed.
	nPW$$	Punch n lines starting at character pointer without a form feed.
Q	Qstring$	Search for string. If not found, read and continue search through input.
R	R$$	Output entire edit buffer and read in next page.
	nR$$	Perform R n times.
S	Sstring$	Search edit buffer for string.
T	T$$	Type out entire edit buffer.
	nT$$	Type out n lines from character pointer.
X	nX$$	Execute a macro n times.
XD	XD$$	Delete a macro definition.
XM	XMcommandstring$$	Define a macro command.
Y	Y$$	Read (yank) a page into the edit buffer.
Z	Z$$	Reset character pointer to end of edit buffer.

Command	Format	Meaning
=	=$$	Type the number of characters in the edit buffer.
:	:$$	Type the number of lines in the edit buffer.
.	.$$	Type the number of the line containing the character to which the character pointer points.
↑A	CTRL A$$	Terminate input/output.
↑C	CTRL C$$	Erase command string or halt execution of command string. *
↑T	CTRL T$$	Reset input buffer and stop input device. *

RUBOUT is used to erase the last character typed in a command string. Each erased character is echoed on the teletypewriter, e.g.:

BSLAB: LDA 1,0LL0,1 ADL :BQ: LDA 1,0$L$$

- BSLAB: LDA 1,0$L — original line as typed
- L$0,1 ADL — characters erased by RUBOUTs
- :BQ: LDA 1,0$L$$ — retyped command

The line would then read:

BSLAQ: LDA 1,0$L$$

* Not used in DOS.

RELOCATABLE ASSEMBLER

The relocatable assembler for DOS is supplied to the user as one file of dumped tape, 088-000001, and is named ASM.SV. To use the Relocatable Assembler, the user must create a save file from the tape by mounting the tape in an appropriate input device, such as the $PTR, and giving the LOAD command:

```
LOAD        $PTR ↵
```

The CLI responds with a ready message (R) when the disk file has been created.

Once the disk file of ASM.SV exists, the command ASM with appropriate arguments brings the Relocatable Assembler into core to assemble source files given in the command line. The ASM command line is described in Chapter 3.

The ASM command line is used to build a command file (COM.CM as described in Appendix D). When a command line begins with the file name ASM, the CLI sorts the command line, and creates COM.CM in the format shown below:

ASM

global switches	binary filename (output)	local switches	listing filename (output)	local switches	source filename$_1$ (input)	local switches	...	source filename$_n$ (input)	local switches

RELOCATABLE LOADER

The DOS Relocatable Loader is supplied to the user as one file of dumped tape, 088-000002, and is named RLDR.SV. To use the Relocatable Loader, the user must create a save file from the tape by mounting the tape in an appropriate input device and giving a LOAD command, for example:

```
LOAD        $PTR ↵
```

The CLI responds with a ready message (R) when the disk file has been created.

Once a disk file of RLDR.SV exists, the command line RLDR, followed by appropriate arguments and a carriage return, brings the Relocatable Loader into core to load the relocatable binary files given in the arguments of the command line.

The RLDR command line is used to build a command file (COM.CM as described in Appendix D). When a command line begins with the file name RLDR, the CLI sorts the command line and creates COM.CM in the format given below:

RLDR

global switches	map filename (output)	local switches	save filename (output)	local switches	relocatable binary filename$_1$ (input)	local switches	...	relocatable binary filename$_n$ (input)	local switches

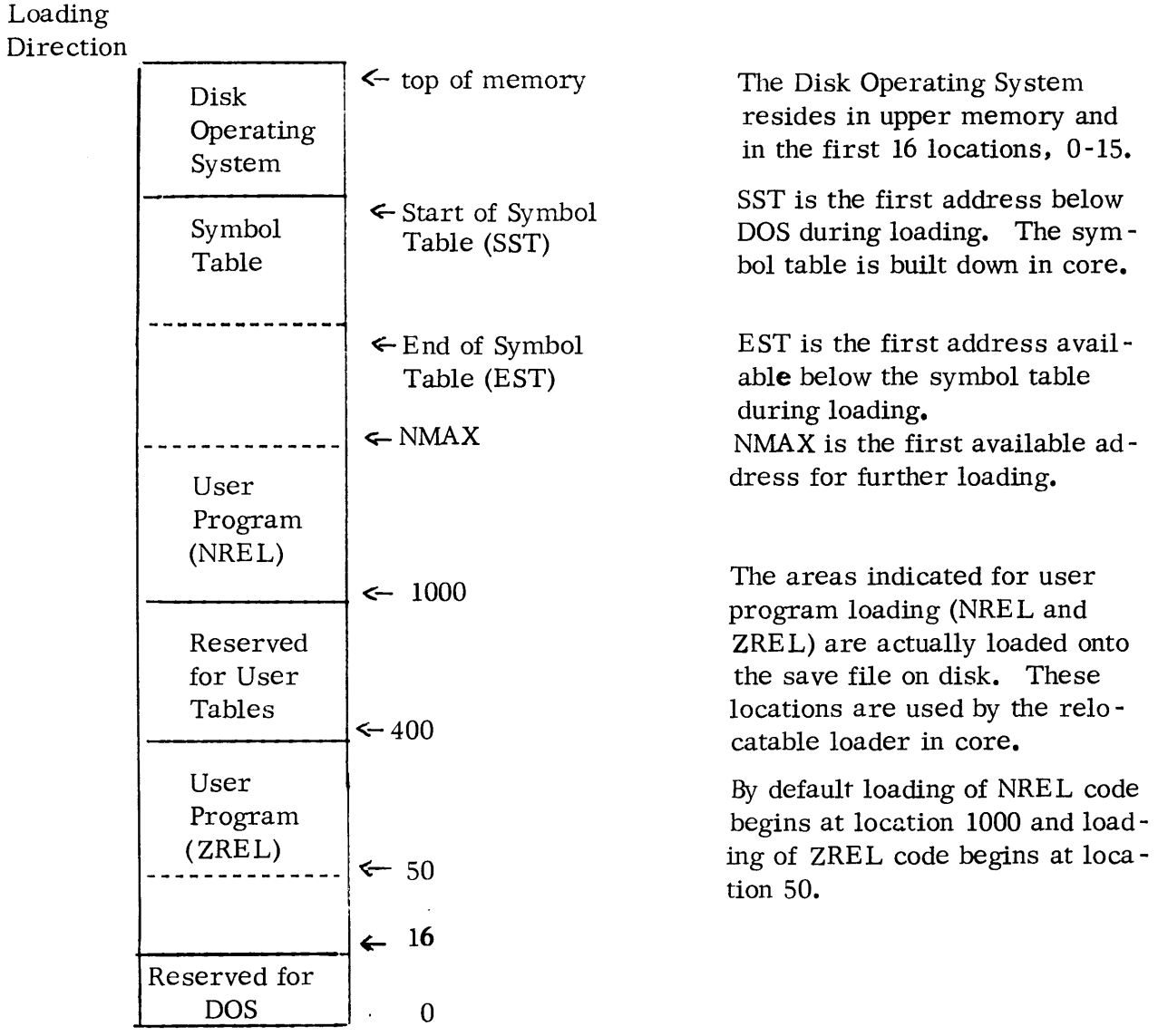
If loading is successful, the output of the Relocatable Loader is a save file, and return is made to the CLI. If loading is not successful, the loader produces an explicit error message and return is made to the CLI. For example, a load overwrite will result in the message:

```
LOAD OVERWRITE          001700
**FATAL LOAD ERROR**
```

The loader begins loading page zero relocatable data at location 50* and normal relocatable data at location 1000. The loader will not load into locations 0-15 nor 400-777. Locations 400-777 (Page 1) are reserved by the loader for necessary DOS status information described later.

* Locations 16-47 can be used by using the . LOC pseudo-op at assembly time.

Actual loading of programs is to the save file. The symbol table however is built in core. The following diagram shows the loading process in progress as if the loaded programs were core-resident.



Dotted lines indicate adjustable locations. While 50_8 is the default starting address for loading of ZREL code, at assembly time the user can change this starting location using the .LOC pseudo-op; by this means locations 16-47 may be used. The first 16 locations cannot be overwritten.

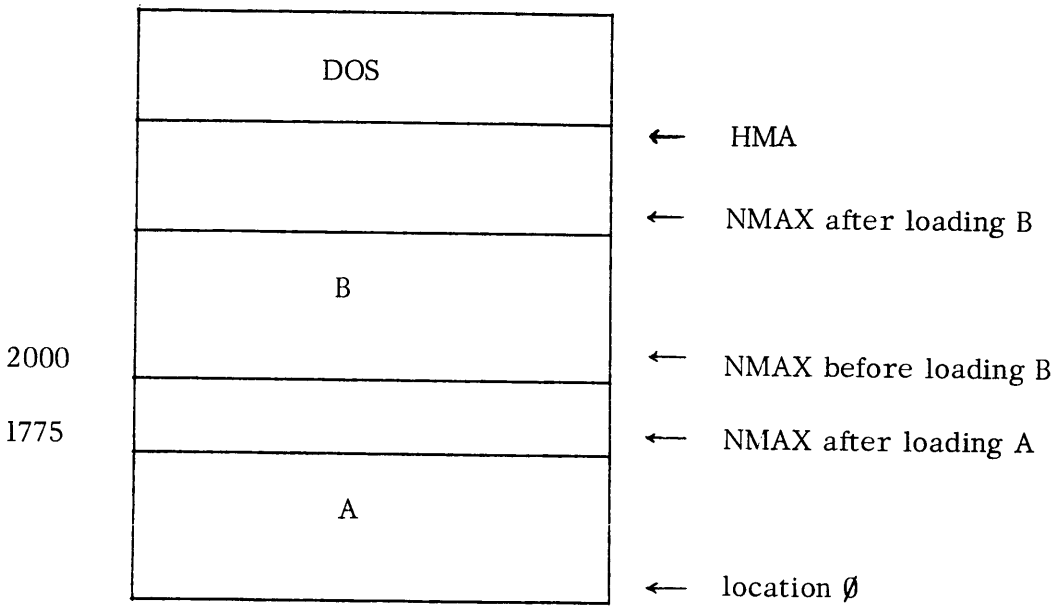
NMAX is adjusted upward as programs are loaded. The user can force NMAX to a given absolute location for the start of loading of a given binary program using the local /N switch, as described in the section following.

User Adjustment of NMAX

When loading a number of programs, the user can adjust the value of NMAX. The loader will accept any value of NMAX that is not less than its current value. The value can be adjusted by a local option as shown below:

RLDR A 2000/N B↵

where: 2000/N is a local option giving an adjusted NMAX (2000_8) at which to begin loading the next program, B.



Page One - User Status Table

Page one is reserved for tables needed by the operating system to load and run programs. The first of these tables is the User Status Table (UST). The User Status Table starts at location 400 and is used for both loader-generated information and run-time information. A template of the UST is shown on the following page.

RELOCATABLE LOADER (Continued)

Page One - User Status Table (Continued)

```
                        ;USER STATUS TABLE (UST) TEMPLATE

UST=    400             ;START OF USER STATUS AREA

USTPC=  0               ;PROGRAM COUNTER (LEAVE AT DISPLACEMENT 0)
USTZM=  1               ;ZMAX
USTSS =  2              ;START OF SYMBOL TABLE
USTES =  3              ;END OF SYMBOL TABLE
USTNM=  4               ;NMAX
USTSA =  5              ;STARTING ADDRESS
USTDA=  6               ;DEBUGGER ADDRESS
USTHU=  7               ;HIGHEST ADDRESS USED BY LOAD MODULE
USTCS=10                ;FORTRAN COMMON AREA SIZE
USTIT=  11              ;INTERRUPT ADDRESS
USTBR=  12              ;BREAK ADDRESS
USTIN=  13              ;INITIAL START OF NREL CODE
   :                    ;SPARE WORDS
   :
USTAØ =  23             ;SAVE STORAGE FOR ACØ
USTA1 =  24             ;AC1
USTA2 =  25             ;AC2
USTA3 =  26             ;AC3
USTCY =27               ;CARRY
```

Location 400, USTPC, is the program counter. The loader initializes this word to -1, indicating that the program has never run.

Location 401, USTZM, points to the first available location in page zero for page zero relocatable code.

Location 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 down at the termination of loading so that the last location in the symbol table coincides with the value of NMAX after all programs are loaded. USTSS, USTES, and NMAX are updated. If the user requests that the symbol table remain in upper core (/S switch on a RLDR command) locations 402, 403 and NMAX remain true and are not updated at the end of loading. If the debugger has not been loaded, locations 402 and 403 are set to zero.

Location 404, USTNM, contains NMAX, the pointer to the first free location for further loading. At load time, the user can set the value of NMAX for a given program by a /N local switch. At the termination of the load, NMAX reflects the first free address for allocation of temporary storage at run time.

Page One - User Status Table (Continued)

Location 405, USTSA, points to the starting address of the save file. The operating system must have this value in order to start a program. The user provides this value by terminating one of his programs at assembly time with:

```
. END adr
```

If no starting address has been specified, the loader stores a -1 in location 405 and issues a warning message, as follows:

NO STARTING ADDRESS SPECIFIED FOR LOAD MODULE.

Location 406, USTDA, points to the starting address of the debugger. If the debugger is not loaded, the loader stores -1 in USTDA. The debugger is loaded if the RLDR command has a /D global switch which requests loading of the debugger from the library file SYS. LB.

The debugger is normally loaded after all other relocatable binary programs in the RLDR command line are loaded. The user can control when the debugger is loaded by placing the file name SYS.LB in the command line at the point where the debugger is to be loaded, e.g.,

```
RLDR/D A SYS.LB B ↵
```

The debugger is loaded immediately after relocatable binary file A in the example.

A symbol table for a user program is appended as part of the save file only if the debugger has been loaded.

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. It is used to reset USTNM whenever a program is started by the system.

Location 410, USTCS, contains the size of the FORTRAN unlabeled COMMON area, used when the binary relocatable programs being loaded were generated by the FORTRAN compiler.

Location 411, USTIT, is the interrupt address (CTRL A). At the termination of loading, this address is set to -1. If unchanged at run time an unconditional return to the CLI occurs when a CTRL A interrupt occurs. The user core image is not saved. The user program can set USTIT at execution time to an address to which control will be tranferred if a CTRL A interrupt occurs. The AC's will be undefined.

RELOCATABLE LOADER (Continued)

Page One - User Status Table (Continued)

Location 412, USTBR, is the break address (CTRL C). At the termination of loading, the address is set to -1. If unchanged at run time, whenever a CTRL C break occurs the core image will be written to the file BREAK.SV on the default directory device and return made to the CLI. Alternatively, the user program can set USTBR to an address to which control will be transferred if a CTRL C break occurs. The AC's will be undefined.

Location 413, USTIN, contains the address of the start of normally relocatable code (NREL), which is 1000_8 for the Disk Operating System. (NREL starts at 440_8 in the Stand-alone Operating System.)

Locations 423-427 are the status storage area for the accumulators and carry, used by the operating system at run time.

Above the UST are the User File Pointer Table and the User File Tables. There are eight User File Tables, used to maintain file information for each of the eight I/O channels in the system. The channel number is used by the system to index the User File Pointer Table and, in turn, locate the User File Table being used for that channel.

Symbol Table Adjustment

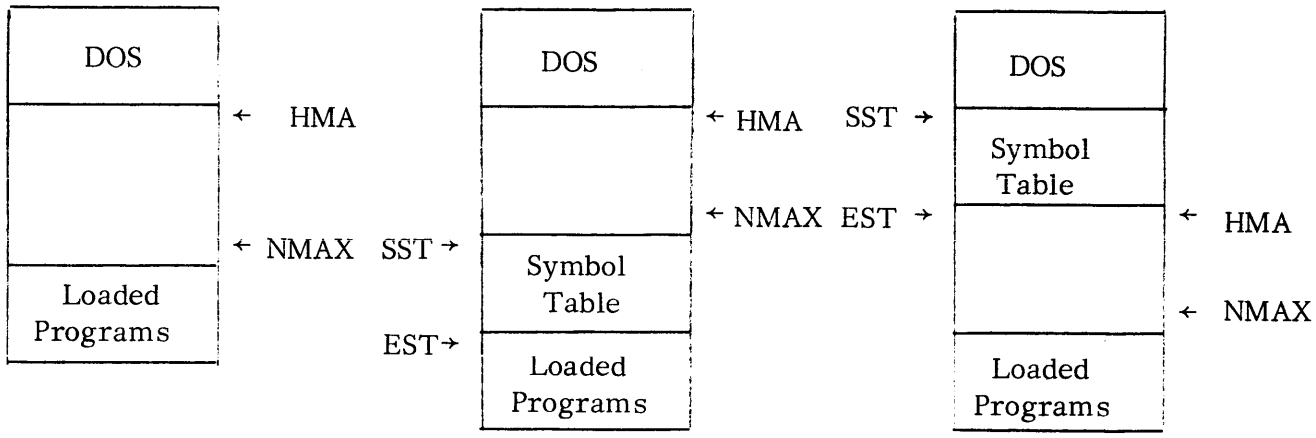
A symbol table is appended as part of the loader's save file output only if the debugger is one of the programs loaded. The debugger is loaded if the RLDR command has a /D switch which requests loading of the debugger from the library file SYS.LB.

When loading is complete, the symbol table is, by default, appended so that the last location in the symbol table coincides with the old value of NMAX. NMAX is then adjusted to be the first location above the symbol table.

The user can request that the symbol table be left in upper core by a /S global switch after the RLDR command. In this case NMAX remains as the first word available above the loaded programs.

On the following page are three representations of core at completion of loading. The first shows programs loaded without a symbol table. The second shows programs loaded with default symbol table adjustment; the third shows programs loaded with the symbol table remaining in upper core. In each case, HMA represents the highest available user memory address.

Symbol Table Adjustment (Continued)



RLDR A B C ↵

(No Debugger. No Symbol Table)

RLDR/D A B C ↵

(Debugger loaded. Symbol Table moved to default locations.)

RLDR/S/D A B C ↵

(Debugger loaded Symbol Table not moved.)

Size of the Save File

Since relocatable binaries are loaded directly to a save file on disk, it is possible to create a save file that is too large to execute within the core limitations of the machine that performed the loading. While there is no direct method by which the user can avoid this possibility, there is a method by which he can determine whether his save file will run in available core once it is loaded.

Refer to the diagrams at the top of the page. If the user appends the D and S switches to RLDR, the symbol table will be fixed at the absolute locations it occupies in high core at the termination of loading. The symbol table will be appended to the save file at these locations. If the symbol table is successfully appended to the save file without a resulting fatal error, the user is assured that the loaded programs will fit into his current core configuration.

Should the current NMAX of the save file be higher than the last location in the symbol table when an attempt is made to append the symbol table, the fatal error message:

SYMBOL TABLE TOO LARGE FOR CORE STORAGE

will be given. Note that the occurrence of the error does not necessarily mean that the loaded programs cannot fit in current core, since the debugger was loaded and the symbol table requires space. However, absence of the error message insures that the save file will fit into current core.

DEBUG III

The symbolic debugger, Debug III, is supported under the operating system. Debug III is supplied as a file within a library, SYS.LB. The library is, in turn, supplied as one file of dumped tape 088-000002. Debug III is loaded when the global switch /D is given in the RLDR command. Debug III is described in document 093-000044; the modifications made for running under DOS are:

1. The starting address of Debug III is stored in the User Status Table (location 406).

2. The punch commands:

   $F
   n$F

   $E
   adr$E

   $adr_1 < adr_2$P

   are not implemented.

3. The interrupt and TTI register commands:

   $I
   $T

   are not implemented.

4. The meaning of $R (with no address argument) is changed to be:

   Run from the starting address in the User Status Table (USTSA).

5. The debugger recognizes all system commands (.CREAT, .GCHAR, .RDL, .WRS, etc.) Since all I/O is handled by the system, the debugger does not recognize I/O instruction mnemonics (NIOS, DOAS, etc.)

6. The eight breakpoint locations are declared as zero-relocatable locations.

EXTENDED ALGOL

The Extended ALGOL compiler is supplied to the user as two dumped tapes, 088-000006 and 088-000007. Tape 088-000006 contains files AL1.SV, ALGOL.SV, and LIBRARY. Tape 088-000007 contains AL2.SV. To use the compiler the user must create save files from the tapes, using the LOAD command.

The ALGOL library is supplied as four library tapes. They must be loaded onto disk using the XFER command. The tapes are:

ALGOL1.LB	-	099-000012	
ALGOL2.LB	-	099-000013	
ALGOL3.LB	-	099-000014	
		099-000008	Software multiply/divide option.
ALGOL4.LB	-	099-000011	Hardware multiply/divide option for NOVA.
		099-000009	Hardware multiply/divide option for NOVA 1200, NOVA 800, or SUPERNOVA.

(The fourth library tape is selected from the above according to the system configuration as indicated.)

The ALGOL command line is used to build a command file (COM.CM as described in Appendix D.) When a command line begins with the file name ALGOL, the CLI sorts the command line and creates COM.CM in the format given below:

ALGOL

global switches	listing file (output)	local switches	assembly source file (output)	local switches	compiler source file (input)	local switches

When the main program and any subprograms have been compiled, they are loaded using the relocatable loader command RLDR. The libraries must be loaded with the programs. A sequence of commands for compilation, loading and executing a main program with two subroutines might be:

```
ALGOL MAIN ↵
ALGOL SUB1 ↵
ALGOL SUB2 ↵
RLDR MAIN SUB1 SUB2 @LIBRARY@ ↵
MAIN ↵
```

EXTENDED ALGOL (Continued)

Either the symbolic debugger or the special Extended ALGOL debugging program TRACE can be used to debug ALGOL programs. To use the symbolic debugger, load it with the relocatable binary programs using the global switch /D:

```
RLDR /D MAIN SUB1 SUB2 @LIBRARY@ ↵
```

TRACE is supplied as dumped tape 088-000009. It must be loaded using the LOAD command. A call to TRACE can be programmed in the ALGOL source program as described in Appendix D of the NOVA ALGOL Reference Manual, 093-000052, or if a run-time error occurs and return is made to the operating system, the file can be brought in by the command:

```
TRACE↵
```

The NOVA ALGOL Reference Manual presents step-by-step debugging procedures for Extended ALGOL using both Debug III and TRACE.

FORTRAN IV

The FORTRAN IV compiler is supplied to the user as two dumped tapes: 088-000005 and 088-000014. Before invoking the compiler, the user must create save files from the tapes using the LOAD command. After the compiler is loaded, the FORTRAN library tapes must be transferred to disk using the XFER command. The library tapes are:

FORT1.LB	-	099-000005	
FORT2.LB	-	099-000006	
FORT3.LB	-	099-000007	
FORT4.LB	-	099-000008	Software multiply/divide option.
		099-000011	Hardware multiply/divide option for Nova.
		099-000009	Hardware multiply/divide option for Nova 1200/Nova 800/Supernova.

Once the compiler and library are loaded onto disk, the FORTRAN IV compiler can be invoked with a FORT command followed by appropriate arguments, as described for the command in Chapter 3. The FORT command line is used to build a command file (COM.CM as described in Appendix D). When a command line begins with the file name FORT, the CLI sorts the command line and creates COM.CM in the format given below:

FORT

global switches	listing file name (output)	local switches	assembly source file name (output)	local switches	compiler source file name (input)	local switches

Each FORTRAN main program, external subroutine, or external function is separately compiled. When the main program and its external subroutines and functions have been successfully compiled (and assembled), the programs are loaded using the RLDR command. The FORTRAN libraries must be loaded with the programs. A series of commands for compiling, loading, and running a FORTRAN program in this manner is shown below:

```
FORT MAIN ↵
FORT XSUB1 ↵
FORT XFUN ↵

RLDR/D MAIN XSUB1 XFUN FORT1.LB FORT2.LB FORT3.LB FORT4.LB ↵
MAIN ↵
```

Merging FORTRAN Libraries

The M(erge) function of the LFE can be used to combine library files into a single library file. The name of the single library file should be FORT.LB, which is the

Merging FORTRAN Libraries (Continued)

name required by the relocatable loader when using the CLG command (see next section). The command to merge the FORTRAN libraries is:

```
LFE  M  FORT.LB/O  FORT1.LB  FORT2.LB  FORT3.LB  FORT4.LB)
```

It is then possible to load a FORTRAN program, as shown in the following example:

```
RLDR  MAIN1  FORT.LB)
```

Use of the CLG Command

The FORT command compiles and assembles a source program or programs, which must then be loaded (RLDR command). The saved file produced can then be executed by invoking the saved file by name.

The CLG command (compile, load, and go) permits the user to invoke whatever system programs are needed to compile, assemble, load, and then execute the saved file produced. The files that are arguments to the command may be FORTRAN source files (.FR), assembly source (.SR), or relocatable binary files (.RB). The command will bring in whatever systems programs are required to create a save file from the input files and will then execute the saved file created.

The following constraints apply to use of CLG:

1. CLG is supplied as a file on FORTRAN system dumped tape, 088-000014, which is loaded as part of the FORTRAN compiler.
2. All other systems programs needed must have been loaded onto disk. These include, besides the FORTRAN compiler, the Assembler (for any needed compilations or assemblies), and the Relocatable Loader.
3. The FORTRAN libraries must have been previously merged into the FORT.LB file as described in the last section.

Symbolic Debugging

The symbolic debugger can be used for run-time debugging of FORTRAN programs. The symbolic debugger can be loaded with the program using the global switch /D with the RLDR command.

LIBRARY FILE EDITOR (LFE)

The LFE is supplied as a dumped tape, 088-000008, and has the name LFE.SV. To use the LFE, the user must create a save file from the tape using the LOAD command, e.g.,

LOAD $PTR ↵

The LFE command line is used to build a command file (COM.CM as described in Appendix D.) When a command line begins with the file name LFE, the CLI sorts the command line and creates COM.CM in the format given below.

LFE

global switches	outputmaster (output)	local switches	listing file name (output)	local switches	key (input)	local switches	arguments (input)	local switches

The Library File Editor provides a means of updating and interpreting library files. A library file is comprised of a set of relocatable binary files (produced by the Extended Assembler) that is denoted by special beginning and ending blocks. For example, DOS Dump Tape #088-000012 contains the CLI library:

LIBRARY START BLOCK
cliprog$_1$.RB
.
.
.
cliprog$_n$.RB
LIBRARY END BLOCK

where each cliprog$_i$.RB represents one of a set of relocatable binary programs.

Library tapes are supplied with the DOS system and with subsystems such as ALGOL and FORTRAN.

The LFE allows the user to analyze the contents of a library file, to list titles in a library file, to merge libraries, to update libraries, and to create his own library files, selected from the contents of system library files or written by the user. The LFE is of special importance in ordering and reordering of relocatable programs in a library file, since the order in which the programs appear determines which programs will be

loaded. This is because of the mechanism employed by the relocatable loader when it operates on library files.

Selection of any program for loading is triggered by the occurrence of a global entry within the program that resolves an external declaration within a previously loaded program. This means that if program A on library file 1 has been loaded and contains a call to program B on library file 1, then B must be located physically after A on the file in order to be loaded. If there are no unresolved external symbols defined as entries in the relocatable binary program, and thus no calls to the program, the program is not loaded. See the Relocatable Loader Manual, #093-000039, "Special Load Modes" section for an additional description of selective loading of library routines.

In some cases it may be necessary to provide two or more copies of a given program on a library file to insure proper referencing. For example:

Program A calls → B calls → C calls → A

(Assume that C follows A in the library file.) If a previously loaded program has called A, then A, B and C are loaded via the standard mechanism. However, if a previously loaded program has called B, then only programs B and C would be loaded. For this case, a second copy of program A should be placed after program C.

The LFE allows the user to list global declarations of the library file (analyze function) to determine whether the programs on the file are the proper selection and in the correct order for his purposes. Other functions of the LFE provide for insertion, deletion, replacement and extraction of programs, merging files or creating a new library file.

The LFE is implemented as a DOS save file and is called by a CLI command under DOS. The command name, LFE is followed by a letter key determining the function to be performed, that is:

Key	Function
A	Analyze
D	Delete
I	Insert
M	Merge
N	New
R	Replace
T	Titles
X	Extract

The key is followed by a series of arguments and local switches. In general, the LFE operates on an input master library file (inputmaster) and relocatable binary update files to produce an output master library file (outputmaster).

In the function descriptions following, those relocatable binary programs that are within a library file are referred to as logical records. This distinguishes them from relocatable binary update files, which are referenced by file name in the LFE command line. A logical record is identified by its five-character title, which occurs within every relocatable binary as a title block.

Function Rules

The following general rules apply to LFE command lines and to the functions involved:

1. Only one function may be performed per command line.

2. An inputmaster library file and an update file cannot reside on the same device, for example, the $PTR.

3. An inputmaster is searched for in the DOS file directory as inputmaster.LB; if not found, a search is made for inputmaster.

4. An update file is searched for in the DOS file directory as filename.RB (or . LB when using A, M, or T function keys); if not found, a search is made for filename.

5. All references to logical records are satisfied by the first matching five-character title of a logical record in the library file. Therefore, it is strongly recommended that each logical record on a file have a unique title.

6. Extracted (X function key) logical records are named title.RB, where title is the five-character record title.

Analyze (A) Function

Format: LFE A/M inputmaster$_1$ [inputmaster$_2$ ... inputmaster$_n$]
LFE A inputmaster$_1$ [arg$_1$ ... arg$_n$]

Purpose: The A function itemizes the global declarations of a library file or of specific logical records within a library file. The output is a list of global declarations, giving the symbol, symbol type, flags, and titles where appropriate.

The symbol types are:

T - Title
EN - Entry Normal
ED - Entry Displacement
N - External Normal
D - External Displacement

Each entry (EN or ED) is followed by titles of records where referenced. Each external (N or D) is followed by the title of the record in which is it defined.

The symbol flags are:

M - Multiply defined entries.
U - Undefined (external normal or external displacement for which no entry has been defined).
P - Phase error (external normal or external displacement for which an entry has been defined before the external declaration).

Optional arguments (arg$_1$ ... arg$_n$) following inputmaster are the specific logical records of the file to be analyzed. If no arguments are given, all records are analyzed. Two or more inputmasters may appear in a command line if the /M global switch is used. They will be analyzed as a single file.

Switches:

Global: /M - Multiple input library files. The switch modifies the function key A and causes all file names following, with the exception of any listing file, to be analyzed as one library.

Local: /L - Listing file. By default, output of analyze is listed on the teletype. The switch causes the file preceding to be used for listing output.

Examples: LFE A $PTR $LPT/L↵

The inputmaster file is $PTR. The library file in the $PTR is analyzed, and the results are printed on the line printer.

Analyze (A) Function (Continued)

Examples:

LFE A/M MATH1. LB MATH2. LB $LPT/L↵

The library files MATH1. LB and MATH2. LB are analyzed as one library and the results are printed to the $LPT.

LFE A MATH. LB↵

The input master file is MATH. LB. All of the logical records in this library file are analyzed and the results are printed at the $TTO (default listing device.)

LFE A MATH. LB SIN COS TAN $LPT/L ↵

The input master file is MATH. LB. The logical records SIN, COS, and TAN are analyzed and the results are printed to the line printer, $LPT.

Analyze (A) Function (Continued)

Output: Sample output of analyze is:

```
LFE A XMODEM

   T  WRCH
   ED .WRCH
U  D  .LDBT
   D  .COUT       COUT
   N  FRET        FLINK
   N  FSAV        FLINK
PAGE ZERO RELOCATABLE DATA = 000001
NORMAL RELOCATABLE DATA = 000015

   T  COUT
   ED .COUT       WRCH
   ED .CIN
   N  FQRET       FLINK
   N  FSAV        FLINK
PAGE ZERO RELOCATABLE DATA = 000002
NORMAL RELOCATABLE DATA = 000023

   T  FLINK
   EN .OFLO
   EN FCALL
   EN FRCAL
   EN FSAV        WRCH     COUT
   EN FRET        WRCH
   EN FQRET       COUT
   ED .FCAL
   ED .FRET
   ED .FSAV
U  D  AFSE
U  D  .RTE0
U  N  .I
PAGE ZERO RELOCATABLE DATA = 000005
NORMAL RELOCATABLE DATA = 000137

TOTAL ZREL COUNT: 000010
TOTAL NREL COUNT : 000177
- - - - - - - - - - - - - - - - - -

R
```

Delete (D) Function

Format: LFE D inputmaster outputmaster/O $arg_1 \ldots arg_n$

Purpose: The D function produces an output master, deleting specified logical records ($arg_1 \ldots arg_n$) from the input master.

Switches:

/O Output master library file. The switch must always modify the name of the output library file, which can appear anywhere within the command line.

Example:

```
LFE D $TTR UTIL. LB/O MOVE LDBYT STBYT DIVI↑↵
      MULT COMP↵
```

The input master file is $TTR.
The output master file is UTIL. LB.
The logical records deleted from the input master are:

MOVE
LDBYT
STBYT
DIVI
MULT
COMP

Insert (I) Function

Format: LFE I inputmaster outputmaster/O $arg_1 \ldots arg_n$

Purpose: The I function permits a merger of update files and logical records on an input master library file to produce an output master library file.

By default, update files in the order listed in the command will be inserted before the first logical record in the input master. To insert an update file or files before or after a given logical record, use the /A or /B switches as described below. A given logical record may appear only once in a command.

No local symbols present in the update files are transferred to the output master.

Switches:

/A Insert after. The switch appears after a logical record name in the argument list of the command line. Arguments following the switch are inserted after the logical record whose name precedes the switch.

/B Insert before. The switch appears after a logical record name in the argument list of the command line. Arguments following the switch are inserted before the logical record whose name precedes the switch.

/O Output master library file. The switch must always modify the name of the output library file.

Examples:

```
LFE I $PTR MATH.LB/O A.RB B.RB SINE/A C.RB D.RB
B COS/A X.RB Y.RB Z.RB
```

inputmaster is $PTR. outputmaster is MATH.LB. Files A.RB and B.RB are inserted at the beginning of the output master. Files C.RB and D.RB are inserted after the program SINE in the output master. Files X.RB, Y.RB, and Z.RB are inserted after the program COS in the output master. (Note that SINE need not precede COS on the input master.)

Merge (M) Function

Format: LFE M outputmaster/O $inputmaster_1$ [$inputmaster_2$ ... $inputmaster_n$]

Purpose: The M function produces an output master that contains as a single library file one, two or more library files (inputmasters).

Switches: /O Output master library file. The switch always modifies the outputmaster file name.

Examples:

LFE M FORT. LB/O FORT1. LB FORT2. LB FORT3. LB FORT4. LB↵

The four FORTRAN library files are merged into a single FORTRAN library file called FORT. LB.

New (N) Function

Format: LFE N outputmaster/O arg_1 [arg_2 ... arg_n]

Purpose: The N function creates a new library file named outputmaster from one or more relocatable binary files.

Switches: /O Output master library file. The switch always modifies the outputmaster file name.

Example:

LFE N $PTP/O $PTR/9/9/1 A.RB C.RB↵

The outputmaster is a file punched to the $PTP. The update relocatable binary files that comprise the outputmaster are 19 files taken from the $PTR followed by files A.RB and C.RB from the default directory device.

Replace (R) Function

Format: LFE R inputmaster outputmaster/O arg_1 arg_2 [... arg_{n-1} arg_n]

Purpose: The R function processes an output master, replacing logical records in the input master with relocatable binary update files.

All arguments are paired as follows:

arg_{i-1} (1, 3, 5...n-1) = Logical record (program title)

arg_i (2, 4, 6...n) = Update file name

No local symbols present in the update files are transferred to the output master.

Switches: /O Output master library file. The switch always modifies the outputmaster file name.

Example:

```
LFE R MATH. LB $PTP/O ATAN $PTR TAN TAN. RB HSINE↑↵
      $PTR ACOS X. RB↵
```

The input master file is MATH. LB.
The output master file is $PTP.
Logical record ATAN is replaced by a file mounted in the paper tape reader, $PTR.
Logical record TAN is replaced by file TAN. RB.
Logical record HSINE is replaced by the file mounted in the paper tape reader, $PTR.
Logical record ACOS is replaced by file X. RB.

Note that all these replacements will be made regardless of the order of the specified logical records on the input master.

Titles (T) Function

Format: LFE T inputmaster [listing-device/L] [arg_1 ... arg_n]

Purpose: The T function outputs to the listing device (teletype by default) the titles of logical records on inputmaster and on any optional additional library files given by the arguments, arg_1 ... arg_n.

Switches: /L indicates the listing device. The listing device argument may appear anywhere in the command line after the function key T.

Example: LFE T $LPT/L $PTR F1.LB $TTR↵

The inputmaster library file is $PTR. Additional library files are F1.LB and $TTR. Titles are listed on the line printer.

Extract (X) Function

Format: LFE X inputmaster arg_1 [arg_2 ... arg_n]

Purpose: The X function permits one or more logical records on library file inputmaster to be extracted as separate relocatable binary files. The relocatable binary files will have the filenames of the logical records to be extracted.

Switches: None.

Example: LFE X MATH. LB SINE COSINE TAN↵

inputmaster MATH. LB is searched and the logical records SINE, COSINE and TAN are extracted, creating relocatable binary files SINE. RB, COSIN. RB, and TAN. RB.

LFE Error Messages

The following messages result from encountering fatal errors in the LFE command line. A return to the CLI without processing any files will result.

NOT ENOUGH ARGUMENTS

For example, unpaired arguments to the replace (R) function.

UNEXPECTED ARGUMENT AT OR FOLLOWING: *string*

For example, *filename* /A followed by *filename* /A for an insert(I) function.

When there is no *string* following the colon in the error message, the message indicates the error occurred at the end of the command line.

INVALID SWITCH FOR: *string*

For example, a switch other than /M in the analyze (A) function will cause the following message: ILLEGAL SWITCH FOR: A

NOT A LFE COMMAND: *key*

A function key that is not recognized by the LFE; currently, any letter key other than A, D, I, M, N, R, T, or X causes this error.

TOO MANY ARGUMENTS

The argument string is too long for the allocated storage (currently, 500 characters.)

ILLEGAL HEADER IN INPUT LIBRARY.

No header or an incorrect header block in the library file.

The following messages result from fatal errors encountered while processing files. When these errors occur, the output file will be terminated with a library end block before returning to the CLI.

CHECKSUM ERROR IN UPDATE FILE: *filename*

Typically, the message indicates a bad record within *filename*.

CHECKSUM ERROR IN LOGICAL RECORD: *recordname*

Very likely the message indicates a bad record. If the checksum occurs within a title block itself, *recordname* will be the name of the previous logical record. If no previous record exists, *recordname* will be the name of the library itself.

ILLEGAL BLOCK UPDATE FILE: *filename*

For example, if a source file is specified as input instead of a binary file, illegal blocks will be encountered.

ILLEGAL BLOCK IN LOGICAL RECORD: *recordname*

A bad block within a logical record will produce this message. If the expected title is missing, the record name will be the name of the previous logical record within the library.

The following message indicates a fatal error detected by the 'system' rather than LFE:

FILE DOES NOT EXIST, FILE: *filename*

filename indicates a library file. The error occurs when no inputmaster is found for the command. The error can occur on command lines having functions other than new (N).

Other fatal errors from the 'system' will refer to the LFE.SV file.

The following messages result from non-fatal errors. Processing will continue as indicated for each error.

FILE DOES NOT EXIST, FILE: *filename*

An update file cannot be found. Search is made for *filename* and *filename*.RB When not found, the file is omitted in processing.

LOGICAL RECORD NOT FOUND - *recordname*

The input master does not contain *recordname*. The record (and any corresponding argument) are passed in processing.

DEFAULT OUTPUT IN FILE - *filename*

The output master was expected and not found. *filename* is used instead as the output file.

FILE ALREADY EXISTS - *filename*

On an extract (X) function, there is already a file on the output device with the same name as the logical record to be extracted. The logical record is omitted in processing.

UPDATE FILE MATCHES INPUT MASTER: *filename*

The result is non-fatal as long as there exists at least one valid update file argument, In this case, the matching update file is ignored.

OCTAL EDITOR

The Octal Editor under DOS enables the user to examine and modify, either in octal or in ASCII notation, any location in any type file. The program is supplied as a dumped tape, 088-000013, and is called OEDIT.SV. Before OEDIT is invoked, a disk file must be created from the tape using the LOAD command. The program is then invoked using the command:

OEDIT *filename*

where *filename* is any file that exists under DOS. If no filename is given, the following message will be printed:

ERROR: NO FILENAME SPECIFIED

If the file specified cannot be found in a DOS system directory, the following message is printed:

ERROR: INPUT FILE DOES NOT EXIST.

If a filename is successfully found, OEDIT will type a carriage return, line feed and is ready to accept input commands. When first invoked, OEDIT is in octal mode.

Octal and ASCII Modes

OEDIT is by default in octal mode. Contents of locations are printed as six octal digits and modifications are made in octal. To switch to ASCII mode, the user types the letter

A

When the user types A (whether at the beginning of a line or after examining and perhaps modifying a register), the octal editor will generate a carriage return, line feed and is ready to respond to a new request in ASCII. Contents of a location are printed in ASCII as two characters. Transparent characters (carriage return, line feed, etc.) are printed in octal enclosed in angle brackets.

To return to octal mode, the user types the letter

O

The octal editor will generate a carriage return, line feed and is ready to respond to a request in octal.

Opening and Examining a Location

Every word within a file can be examined by using a word address relative to the beginning of the file. For example, the first two bytes of any file can be examined using the word address Ø, the second two bytes by using the word address 1, and so forth.

Opening and Examining a Location (Continued)

To examine a location, it is necessary to type the word address, followed immediately by a slash:

```
17/
```

The octal editor will respond in octal mode by printing the contents as six octal digits or in ASCII mode by printing the contents as two characters.

```
17/ 045070
      or
17/ %8
```

The location may be modified at this time or may be closed. If the location is to be closed without modification, the user types one of the delimiters: carriage return (↵), line feed (↓), or up arrow (↑). The three delimiters perform the following functions:

Delimiter	Meaning	Example
↵ Carriage Return	Close the current register	17/ 045070↵
↓ Line Feed	Close current register and open and print contents of next register.	17/ %8↓ 000018 DC
↑ Up Arrow	Close current register and open and print contents of preceding register.	17/ 045070↑ 000016 000023

While a given location is open, it is possible to print the contents in the other mode, whether ASCII or octal, without changing permanently to the other mode. The conventions are:

= Equals sign	Retype the contents of the current location in numeric form. The register remains open until closed by one of the delimiters.	17/ %8 =045070
' Apostrophe	Retype the contents of the current location in ASCII form. The register remains open until closed by one of the delimiters.	17/ 045070 ' %8

Modifying a Location

When a location has been opened and its contents examined, it may be modified by typing the new contents before closing the location. In octal, for example:

```
17/ 045070 177↵          - the new contents will be 000177
```

In ASCII, the new contents must be preceded by a quotation mark ("). If two characters are given, the beginning quotation mark is sufficient to indicate the word in ASCII. If a single character is given, it must be followed by a quotation mark and will be left justified in the word. To enter a quotation mark, the user types two successive quotation marks.

```
17/ %8 "%9 ↓
000018 *D "D"↵
```

Expressions using octal numbers and + and − may be used in writing the new contents for a location:

```
17/ 000177 20-3↵
```

Locations

The user can give word addresses relative to the beginning of the file as previously indicated. He can also use octal expressions containing + and - to denote the desired location:

```
17+5/
```

Since OEDIT is often used to make simple changes to executable save files, it should be noted that a save file's relative word address Ø is really absolute location 16 under DOS. For example, to examine absolute location 406 of a save file, the following command should be given:

```
406-16/
```

Typing Errors

If illegal delimiters or illegal characters for the given mode are encountered, OEDIT will respond with

```
?
```

and a carriage return, line feed. If a mistake is made while typing a line, RUBOUT can be depressed, and a new command can be typed.

Return to CLI Level

To return to the CLI from OEDIT, the user types

H

The editor will echo OME on the same line. A R(eady) message is then issued, indicating the user is at CLI level.

Sample OEDIT Commands

Following is a sample of commands using OEDIT. Note that locations 4, 7, and 11 have been modified by the time return is made to the CLI.

```
OEDIT MYFILE↵

2/ 177777↓
000003 126440↑
000002 177777↵
0/ 001456↵
4/ 136112 136113↵
6/ 000177↓
000007 000377 177↑
000006 000177↑
000005 000030↑
000004 136113↵
12/ 000000↵
11/ 035612 35617↵
24/ 044045 'H%↵
A
20/ TX↓
000021 T<011>↓
000022 *D↓
000023 FG↓
000024 LA =046101 O
17/ 051105↵
HOME
R
```

BINARY LOADER

A version of the stand-alone loader for loading absolute binary tapes is supplied as a file of dumped tape 088-000013. The tape must be loaded onto disk using the LOAD command, for example:

```
LOAD $PTR ↵
```

The saved file created is BLDR.SV, and the binary loader is invoked with the BLDR command. The BLDR command must have as an argument either the $TTR or the $PTR, where the input device is loaded with an absolute binary tape.

BLDR is supplied as a convenience to users who may need to run in stand-alone mode. If the DOS configuration is used in stand-alone, the user should note that the binary loader will overwrite a portion of DOS. After stand-alone operations have been completed, the user must bootstrap the DOS system.

APPENDIX B

GENERATING AND RESTORING THE SYSTEM

The procedures necessary to produce a Disk Operating System tailored to the user's precise hardware configuration are described in this appendix. These procedures include loading a rudimentary DOS (12K bootstrap DOS tape) into core, loading the system dumped tapes including SYSGEN using the rudimentary DOS, configuring and generating the full system, preparing the configured DOS for bootstrapping, and then bootstrapping in the system.

These procedures are used to generate DOS for the first time, which includes configuring the system and INSTALLing it. However, when it is necessary to restore a system that was previously generated, only part of the procedures described may be used. For example, it may be necessary simply to bootstrap the system from the default directory device (page B-7); or the user may load a previously written paper tape of the system and then bootstrap from the default directory device (page B-7); or the user may read in and bootstrap from a magnetic tape containing the system (page B-8).

When generating DOS for the first time, either a magnetic tape or paper tape containing the configured system is always written and then loaded. This tape provides system backup. Users having both fixed and movable head disks may also wish to generate two system tapes -- one bootstrappable from the fixed head disk and the other bootstrappable from moving head disk.

TAPES AND HARDWARE FOR SYSTEM GENERATION

The 12K bootstrap DOS tape assumes a hardware configuration of at least the following:

12K core

128K fixed head disk (091-000027), or
type 4047 or 4057 moving head disk (091-000053), or
type 4048 moving head disk (091-000058).

Teletype

A high speed paper tape reader, high speed paper tape punch, and an 80-column line printer are all optional equipment.

TAPES AND HARDWARE FOR SYSTEM GENERATION (Continued)

DGC Tape Number	Description
091-000027	12K bootstrap DOS (fixed head disk)
091-000053	12K bootstrap DOS (4047 and 4057 moving head disk)
091-000058	12K bootstrap DOS (4048 moving head disk)
095-000071	2314 Disk Pack formatter
095-000072	2311 Disk Pack formatter
088-000003	Dump tape SYSGEN.SV
088-000002	Dump tape RLDR.SV, SYS.LB
088-000012	Dump tape CLI.LB (CLI library)
088-000016	Dump tape SYS0.LB (first system library)
088-000010	Dump tape SYS1A.LB (second system library)
088-000011	Dump tape SYS1B.LB (second system library)

System Library Tapes

Tape 088-000011 is provided for systems having magnetic tape hardware, and tape 088-000010 is provided for all other systems.

Disk Pack Formatter Tape

If the system is configured with either a type 4048 or type 4057 moving head disk, pack, it is necessary to format the disk pack using the appropriate formatter program before the SYSGEN procedure is begun. In general, all disk packs that are to be used in the system must be formatted prior to their use. Note that it is not necessary to format the disk cartridges used in a type 4047 disk drive.

The disk pack formatter programs are stand-alone programs which do not require DOS for operation. The appropriate formatter programs and their associated manuals are listed below:

Disk Pack Drive	Formatter Program	Manual
Type 4048	095-000072	096-000039
Type 4057	095-000071	096-000038

PREPARING FOR SYSTEM GENERATION

Before the DOS system can be generated, the following steps must be taken:

1. Load the absolute binary tape containing the appropriate bootstrap DOS:

   091-000027 or

   091-000053 or

   091-000058

   using the absolute binary loader (091-000004). After the tape has been read into core, the binary loader will halt. There now exists a rudimentary system that can be used to generate a full system tailored to the correct hardware configuration.

2. Using the CPU switches, examine the contents of location 42. Enter the contents of location 42 into the data switches. Press RESET, then START. The rudimentary CLI will respond:

   R

   signifying that the bootstrap is in operation.

3. Type the following command to the CLI:

   LOAD/V {$PTR / $TTR} ↵

   The CLI will respond:

   LOAD {$PTR / $TTR} , STRIKE ANY KEY.

   Load the paper tape reader or teletype reader with tape 088-000003 and strike any console key. The tape will be read into core and

   SYSGEN.SV

   will be printed at the teletype, verifying that the tape has been LOADed.

PREPARING FOR SYSTEM GENERATION (Continued)

4. Continue in like manner until all the dumped tapes of the system have been loaded. Use the /V global switch to verify the printed file names of the tapes. Check the verification typeout against the list following to insure that there are no tape identification errors:

SYSGEN.SV
RLDR.SV
SYS.LB
CLI.LB
SYS0.LB
SYS1A.LB or
SYS1B.LB

GENERATING THE SYSTEM

The SYSGEN program (tape 088-000003) contains the means of tailoring the DOS system to the user's hardware configuration. When the system tapes have been loaded and verified, type the command:

SYSGEN ↵

to invoke the SYSGEN program. SYSGEN then interrogates the user regarding his particular hardware configuration. The dialogue is shown below. The SYSGEN queries are given in capital letters as they appear on the teletype. The possible user responses are discussed following each query.

ENTER CORE STORAGE (IN THOUSANDS OF WORDS)

The user may respond with any number from 12 (12K) to 32 (32K) in increments of 2 (2K).

RESPOND "1" (YES) OR "0" (NO) REGARDING SYSTEM CONFIGURATION DSK?

The query asks whether a fixed had disk is to be part of the system. If the user responds 0, SYSGEN will go on to the next question; if the user responds 1, SYSGEN will query:

ENTER DISK STORAGE (IN THOUSANDS OF WORDS)

The user may respond with any number up to and including the full capacity of the fixed head disk in the system. The usual response is the full capacity of the fixed head disk.

DKP?

The query asks whether moving head disks are part of the configuration. The response is the same as for the fixed head disk: 0 for NO causes SYSGEN to proceed to the next device inquiry; 1 for YES will cause SYSGEN to ask for further moving head disk configuration information:

ENTER NUMBER OF UNITS

The user may specify 1, 2, 3, or 4 moving head units. (Note that the 4047B is considered to be two units.)

ENTER NUMBER OF SECTORS/TRACK

The user may specify 6 or 12 sectors per track depending on his type of unit. The user specifies 6 for the 4048 unit or 12 for either the 4047 or 4057 units.

ENTER NUMBER OF HEADS

The user answers with 2, 10, or 20 depending on his type of unit. The response is 2 for the 4047 unit, 10 for the 4048 unit, or 20 for the 4057 unit.

If the user has responded 1 to both the DSK and DKP queries, SYSGEN will ask the following:

ENTER MASTER DEVICE

The user must then decide which device, fixed head disk or disk pack, should be used for temporary storage space. Normally, the fastest access time device (DSK) should be specified.

ENTER BOOTSTRAP DEVICE

The user must decide the type of device to be used for the system bootstrap operation. A response of DKP or DSK is acceptable.

MTA?

The system queries the user about magnetic tape. A response of 0 will cause SYSGEN to proceed to the next device inquiry. If the magnetic tape drive is to be included, a response of 1 causes SYSGEN to query:

ENTER NUMBER OF UNITS

A number of units from 1 to 8 may be specified.

PTR?

The user responds 1 if a paper tape reader is part of the configuration; the user responds 0 for NO.

PTP?

The user responds 1 if a paper tape punch is part of the configuration; the user responds 0 for NO.

LPT?

The user responds 0 if there is no line printer; a response of 1 causes SYSGEN to query as to the line printer column count.

ENTER COLUMN SIZE

80 or 132 are acceptable responses.

CDR?

The user responds 1 (YES) if there is a card reader; otherwise the user responds 0.

PLT?

The user responds 1 (YES) if there is an incremental plotter; otherwise the user responds 0.

On the basis of user responses to SYSGEN, SYSGEN creates a relocatable binary system file SYS000.

CREATING A PRELIMINARY SAVE FILE OF THE SYSTEM

When the user has correctly responded to the system configuration queries, he is now ready to create a preliminary system save file from the relocatable binary file. To do so, he issues the following command:

RLDR/Z SYS000 CLI.LB @NREL@ SYS0.LB {SYS1B.LB / SYS1A.LB} {$TTO/L / $LPT/L} ↵

CREATING A PRELIMINARY SAVE FILE OF THE SYSTEM (Continued)

SYS000.SV will be the name of the saved file on disk containing the system. The user will give the argument SYS1B.LB if he has magnetic tape hardware; otherwise, the argument SYS1A.LB is used. The loader will print the storage map on the teletypewriter ($TTO argument); if the system configuration includes a line printer, the user can replace the argument with $LPT/L and the loader will list the storage map on the line printer. The file NREL is produced by SYSGEN and forces the system to be as high in memory as possible.

After the loader prints the storage map, the CLI will print R on the teletype.

TRANSFERRING THE SYSTEM FILE TO TAPE AND LOADING THE SYSTEM

The preliminary file, SYS000.SV, containing the system must now be transferred to paper tape in absolute binary format or to magnetic tape.

Transferring to Tape and Loading (Paper Tape)

SYS000.SV must be transferred to paper tape so that it can be loaded (as was the rudimentary system) with the absolute binary loader.

The user issues the command:

MKABS/Z SYS000.SV {$PTP / $TTP} ↵

If available, the paper tape punch should be used. The absolute binary image of the new DOS system will be punched. The paper tape constitutes not only a means of generating the system initially but also a system backup.

Load the punched system tape using the absolute binary loader. After the system tape is loaded, the binary loader will halt. As in Step 2, page B-3, examine the contents of location 42, enter the contents into the data switches, and press RESET and START. The CLI will respond with:

R

The system as now loaded is used to load the final version of the system into a system file, ready it for bootstrapping, and bootstrap the final version of the system.

Preparation for Bootstrapping from Fixed Head Disk or Disk Pack

After the paper tape of the system has been loaded and the CLI gives the R response, the user must take the following steps in preparation for bootstrapping:

1. Check to insure that the default directory device is the device that was specified for bootstrapping when the system was configured.

Preparation for Bootstrapping from Fixed Head Disk or Disk Pack (Continued)

2. Load the system tape into the paper tape reader or teletype reader and type the command:

   MKSAVE/Z{$TTR / $PTR}SYS.SV ↵

   The DOS system tape will be read and the system file, SYS.SV will be on the default directory (bootstrap) device.

3. Issue the command:

   CHATR SYS.SV SP↵

   This protects the system file from accidental deletion.

4. Issue the command:

   INSTALL SYS.SV ↵

   This causes DOS to copy the bootstrap program from SYS.SV to logical address zero of the default directory device. When the bootstrap program begins, it locates the remainder of the system file and loads the entire system into core. Unless the boostrap is copied to the default directory device using INSTALL, the system cannot be bootstrapped.

Transferring to Magnetic Tape in Preparation for Tape Bootstrapping

The system saved file SYS000.SV can be transferred to magnetic tape, provided that the user has a configuration that includes 9-track magnetic tape. * Users having a configuration that includes 9-track magnetic tape receive a tape bootstrap program in dump format (088-000015) called TBOOT.SV . This tape may be loaded any time after the rudimentary system bootstrap has been loaded, using the command:

LOAD {$PTR / $TTR} ↵

For example, TBOOT.SV can be loaded after the absolute binary tape has been read in, along with other dumped system tapes, or after system configuration.

Once SYS000.SV has been created, the user takes the following steps:

1. Select an unused tape, mount it on the tape drive, and issue the command:

* 7-track magnetic tape cannot be used.

Transferring to Magnetic Tape in Preparation for Tape Bootstrapping (Continued)

INIT/F MTn ↵

where: n is the unit number selected by the magnetic tape adapter.

2. The tape bootstrap file is then transferred to the magnetic tape unit n by the command:

XFER TBOOT.SV MTn:0 ↵

This stores the tape bootstrap on magnetic tape. The tape bootstrap program must be the first file (0) on the magnetic tape. The DOS system must immediately follow the tape bootstrap program.

3. To transfer the system file, the user issues the command:

XFER SYS000.SV MTn:1 ↵

This tape, as written, can be used to initialize and bootstrap DOS. It can also be preserved as a backup copy of the system.

If the user wishes, additional files may be written to the magnetic tape containing the system. The user must, however, give such files a number other than 0 or 1. For example, the user may at some time write the DOS assembler to the system tape using the command:

DUMP MT0:2 ASM.SV ↵

BOOTSTRAPPING

Bootstrapping from the Default Directory Device (DSK or DPK)

When SYS.SV has been INSTALLed, the system can be bootstrapped directly from the default device (and only from this device). The bootstrapping procedure varies slightly with the machine configuration.

Before bootstrapping from a moving head disk, it is necessary to insure that the read/write heads are physically positioned in cylinder zero. With a type 4047 disk drive, this is accomplished by momentarily depressing the LOAD/RUN switch to the LOAD position and then depressing the switch to the RUN position. The heads are properly positioned and the bootstrap operation can proceed when the READY light comes on. With a type 4048 or 4057 disk drive, depress the POWER switch and allow approximately one or two seconds to elapse before depressing the POWER switch again to restore power to the drive. The bootstrap procedure can be continued when the READY light comes on.

Bootstrapping from the Default Directory Device (DSK or DPK) (Continued)

Nova or Nova 1200/800 Series with No Program Load Option

1. Enter in location 376:

   601nn

   where: nn = 20 when bootstrapping from fixed head disk.
   = 33 when bootstrapping from moving head disk.

2. Enter 377 in location 377.

3. Press RESET, then press START.

4. The message: DOS REV nn will be printed on the teletype, where nn represents the current system revision level.

5. Press CONTINUE. The system will respond with R and the system is ready for use.

Nova 1200/800 Series with Program Load Option

1. Set bit 0 in the data switches to a 1.

2. Enter in bits 10-15 of the data switches the disk device code:

   20 for bootstrapping from fixed head disk.
   33 for bootstrapping from moving head disk.

3. Press RESET, then press PROGRAM LOAD.

4. The message: DOS REV nn will be printed on the teletype, where nn represents the current system revision level.

5. Press CONTINUE. The system will respond

   R

   and the system is ready for use.

Supernova

1. Enter in the data switches, bits 10-15, the disk device code:

   20 for bootstrapping from fixed head disk.
   33 for bootstrapping from moving head disk.

2. Press RESET, then press CHANNEL START.

3. The message: DOS REV nn will be printed on the teletype, where

Bootstrapping from the Default Directory Device (DSK or DPK) (Continued)

Supernova (Continued)

nn represents the current system revision level.

4. Press CONTINUE. The system will respond

R

and the system is ready for use.

Magnetic Tape Bootstrapping

The system that was written to magnetic tape must be bootstrapped from magnetic tape unit 0; no other unit number can be used. To bootstrap from magnetic tape, take the following steps:

1. Mount the tape on unit 0.

2. Position the tape to the load point. The bootstrap cannot be performed correctly unless the tape is positioned to the load point.

3. Place the unit on-line.

4. Take the following action, depending upon the machine configuration:

   a. Nova or Nova 1200/800 Series without Program Load

      Deposit 60122 (NIOS MTA) in location 376; deposit 377 in location 377. Press RESET and then press START.

   b. Nova 1200/800 Series with Program Load

      Set the data switches to 100022. Press PROGRAM LOAD.

   c. Supernova

      Enter 22 in bits 10-15 of the data switches. Press RESET and then press CHANNEL START.

5. The tape bootstrap program will be read into core and will type the following message on the teletypewriter:

   FULL(0) OR PARTIAL(1) ?

Magnetic Tape Bootstrapping (Continued)

The user responds by striking 0 or 1, indicating a full or partial initialization procedure. If the system has just been generated, he must perform a full initialization. If the system has been running and the user wishes to maintain his directory and file system information, he can perform a partial initialization.

6. When the user responds to the query, the remainder of the system is read into core, initialization performed, and control transferred to the CLI which will transmit the prompt character (R). The tape is then rewound.

If the user is bringing up his system for the first time, he may wish to take the steps that prepare the system for disk bootstrapping. This will insure that he will be able to restore the system by bootstrapping from disk if necessary. The steps are those previously described, i.e.,

1. Insure that the default directory device and the bootstrap device are the same.

2. Issue the commands:

```
XFER MT0:1 SYS.SV )
CHATR SYS.SV SP )
INSTALL SYS.SV )
```

These commands enable the user to bootstrap DOS from whichever bootstrap device (DSK or DPK) was specified at SYSGEN time.

APPENDIX C

SYSTEM AND USER PARAMETER TAPES

Supplied with DOS are two parameter tapes in ASCII, the User Parameter Tape, 090-000090, and the System Parameter Tape, 090-000176. Listings of the User and System Parameters follow. The System Parameters begin on page C-9.

```
.MAIN

        ; DEFINE THE SYSTEM STACK DISPLACEMENTS

177771  .DUSR SSLGT=    -7      ; VARIABLE LENGTH OF CALLING'S FRAME
177772  .DUSR SSOSP=    -6      ; PREVIOUS STACK POINTER
177773  .DUSR SSRTN=    -5      ; RETURN ADDRESS OF CALLING PROGRAM
177774  .DUSR SSEAD=    -4      ; ENTRY ADDRESS OF CALLED ROUTINE
177775  .DUSR SSCRY=    -3      ; CARRY
177776  .DUSR SSAC0=    -2      ; SAVE STORAGE FOR CALLING'S ACCUMULATOR
177777  .DUSR SSAC1=    -1
000000  .DUSR SSAC2=    0       ; (DON'T MODIFY THIS DISPLACEMENT!!)
```

```
.MAIN

        ;
        ; UFT ENTRY
        ;

000000 .DUSR UFTFN=0            ;FILE NAME
000005 .DUSR UFTEX=5            ;EXTENSION
000006 .DUSR UFTAT=6            ;FILE ATTRIBUTES
000007 .DUSR UFTBK=7            ;NUMBER OF LAST BLOCK IN FILE
000010 .DUSR UFTBC=10           ;NUMBER OF BYTES IN LAST BLOCK
000011 .DUSR UFTAD=11           ;DEVICE ADDRESS OF FIRST BLOCK (0 UNASSI
000012 .DUSR UFTDL=12           ;DCT LINK

000013 .DUSR UFTDC=13           ;DCT ADDRESS
000014 .DUSR UFTUN=14           ;UNIT NUMBER
000015 .DUSR UFTCA=15           ;CURRENT BLOCK ADDRESS
000016 .DUSR UFTCB=16           ;CURRENT BLOCK NUMBER
000017 .DUSR UFTST=17           ;FILE STATUS
000020 .DUSR UFTNA=20           ;NEXT BLOCK ADDRESS
000021 .DUSR UFTLA=21           ;LAST BLOCK ADDRESS
000022 .DUSR UFTDR=22           ;SYS.DR DCB ADDRESS
000023 .DUSR UFTFA=23           ;FIRST ADDRESS

000024 .DUSR UFTBN=24           ;CURRENT FILE BLOCK NUMBER
000025 .DUSR UFTBP=25           ;CURRENT FILE BLOCK BYTE POINTER
000026 .DUSR UFTCH=26           ;DEVICE CHARACTERISTICS
                                ;(LEAVE "UFTCH" AS LAST WORD!)

000027 .DUSR UFTEL=UFTCH-UFTFN+1        ;UFT ENTRY LENGTH
000013 .DUSR UFDEL=UFTDL-UFTFN+1        ;UFD ENTRY LENGTH

        ;
        ; SYSTEM FILE ENTRY
        ;

177773 .DUSR SFKEY=-5           ;KEY
177774 .DUSR SFLK=-4            ;MAP.DR LINK (=1 IF NOT DSK DVC)
177775 .DUSR SFNX=-3            ;NEXT ENTRY IN CHAIN
177776 .DUSR SFBK=-2            ;NUMBER OF LAST BLOCK IN FILE
177777 .DUSR SFBC=-1            ;BYTE IN LAST BLOCK
000000 .DUSR SFDCB=0            ;DCB ENTRY

177773 .DUSR UDBAT=UFTAT-UFTDC  ;NEGATIVE DISP. TO ATTRIBUTES
177776 .DUSR UDBAD=UFTAD-UFTDC  ;NEGAVIE DISP. TO FIRST ADDRESS
177774 .DUSR UDBBK=UFTBK-UFTDC  ;NEGATIVE DISP. TO LAST BLOCK
000011 .DUSR UDBBN=UFTBN-UFTDC  ;POSITIVE DISP. TO CURRENT BLOCK
```

```
        ; FILE ATTRIBUTES
        ;

100000 .DUSR ATRP=100000          ;READ PROTECTED
040000 .DUSR ATCHA=100001         ;CHANGE ATTRIBUTE PROTECTED
020000 .DUSR ATSAV=100002         ;SAVED FILE
004002 .DUSR ATPER=100014         ;PERMANENT FILE
000001 .DUSR ATWP=100015          ;WRITE PROTECTED

        ;
        ; FILE STATUS
        ;

000001 .DUSR STER=100015          ;ERROR DETECTED
000002 .DUSR STIOP=100014         ;I/O IN PROGRESS
000004 .DUSR STFWR=100013         ;FIRST WRITE FLAG
040000 .DUSR STINI=100001         ;NO INIT BIT
100000 .DUSR STCMK=100000 ;SET = READ (FILIO)
                                  ;(INIT/RELEASE SWTCH FOR SYS.DR DCB)

        ;
        ; BUFFER STATUS
        ;

000001 .DUSR QTMOD=100015         ;HAS BEEN MODIFIED
000002 .DUSR QTER=100014          ;ERROR DETECTED
000010 .DUSR QTIOP=100012         ;I/O IN PROGRESS
000020 .DUSR QTLCK=100011         ;BUFFER LOCKED
000040 .DUSR QTCMD=100010         ;COMMAND = 1 = READ, 0 = WRITE
000100 .DUSR QTEMD=100009         ;ERROR MODE (MAG TAPE)
000200 .DUSR QTIND=100008         ;INDIRECT DRIVER MODE SW.


        ;
        ; SYSTEM CONSTANTS.
        ;

000377 .DUSR SCWPB=255.           ;WORDS PER BLOCK
000004 .DUSR SCLLG=132.           ;MAX LINE LENGTH
000030 .DUSR SCAMX=24.            ;MAX ARGUMENT LENGTH IN BYTES
000005 .DUSR SCFNL=UFTEX-UFTFN+1        ;FILE NAME LENGTH
000012 .DUSR SCMER=10.            ;MAX ERROR RETRY COUNT
000016 .DUSR SCSTR=16             ;SAVE FILE STARTING ADDRESS
177660 .DUSR SCTIM=-60.           ;RINGIO 1 MS. LOOP TIME (SN)
000001 .DUSR SCSYS=1              ;DEVICE ADDRESS FOR SYS.DR
000002 .DUSR SCMAP=2              ;DEVICE ADDRESS FOR MAP.DIR
000003 .DUSR SCSVB=SCMAP+1        ;4 CONTIGUOUS BLOCKS FOR CORE IMAGES
000004 .DUSR SCSNO=4              ;NUMBER OF LEVELS
000005 .DUSR SCEXT=UFTEX-UFTFN    ;EXTENSION OFFSET IN NAME AREA
000100 .DUSR SCRRL=64.            ;WORDS PER RANDOM RECORD
100000 .DUSR SFINT=100000         ;INTERRUPT FLAG
000004 .DUSR SFCRD=100013         ;CRITICAL READ ERROR
000002 .DUSR SFPRD=100014         ;PANIC ON READ ERROR
000001 .DUSR SFBRK=100015         ;BREAK FLAG
000040 .DUSR CADZ=40              ;CA LOCATION IN BOOTSTRAP
000041 .DUSR LADZ=CADZ+1          ;LA LOCATION IN BOOTSTRAP
000042 .DUSR SCFUL=LADZ+1
000043 .DUSR SCPAR=SCFUL+1
000044 .DUSR SCKEY=SCPAR+1
```

```
; DEFINE THE EXCEPTIONAL STATUS 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 ERUFT=    21      ; ATTEMPT TO USE A UFT ALREADY IN USE
.DUSR ERLLI=    22      ; LINE LIMIT EXCEEDED 0
.DUSR ERRTN=    23      ; ATTEMPT TO RESTORE A NON-EXISTENT IMAG
.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 ERRO=     33      ; ATTEMPT TO READ INTO SYSTEM AREA
.DUSR ERDIR=    35      ; FILES SPECIFIED ON DIFF. DIRECTORIES
.DUSR ERDNM=    36      ; ILLEGAL DEVICE NAME
```

```
.TITLE
        ; CLI ERROR CODES

        .DUSR CNEAR=100          ;NOT ENOUGH ARGUMENTS
        .DUSR CILAT=101          ;ILLEGAL ATTRIBUTE
        .DUSR CNDBD=102          ;NO DEBUG ADDRESS
        .DUSR CNCTD=103          ;NO CONTINUATION ADDRESS
        .DUSR CNSAD=104          ;NO STARTING ADDRESS
        .DUSR CCKER=105          ;CHECKSUM ERROR
        .DUSR CNSFS=106          ;NO SOURCE FILE SPECIFIED
        .DUSR CNACM=107          ;NOT A COMMAND
        .DUSR CILBK=110          ;ILLEGAL BLOCK TYPE
        .DUSR CSPER=111          ;NO FILES MATCH SPECIFIER
        .DUSR CPHER=112          ;PHASE ERROR
        .DUSR CTMAR=113          ;TOO MANY ARGUMENTS

        ; DEFINE THE PANICS

        .DUSR   PNOP=    010     ; NOP MAGIC
        .DUSR   POFFS=   1011    ; OFFSET

        .DUSR   PNCUI=   21*POFFS+PNOP   ; UNKNOWN INTERRUPT
                                         ; DEVICE CODE IN AC0
        .DUSR   PNCSO=   22*POFFS+PNOP   ; SYSTEM STACK OVERFLOW
        .DUSR   PNCDW=   23*POFFS+PNOP   ; CRITICAL DISK WRITE ERRORS
        .DUSR   PNCDR=   24*POFFS+PNOP   ; CRITICAL DISK READ  ERRORS
        .DUSR   PNCDE=   25*POFFS+PNOP   ; CRITICAL DISK READ/WRITE ERROR
        .DUSR   PNCRR=   26*POFFS+PNOP   ; RUNAWAY READER
        .DUSR   PNCMT=   27*POFFS+PNOP   ; MTA CONTROLER ERROR
```

```
.MAIN

                        ; DEFINE THE CHARACTERISTICS

000001  .DUSR    DCCPO=  1B15    ; DEVICE REQUIRING LEADER/TRAILER
000002  .DUSR    DCCGN=  1B14    ; GRAPHICAL OUTPUT DEVICE WITHOUT TABBIN
                                ; HARDWARE
000004  .DUSR    DCIDI=  1B13    ; INPUT DEVICE REQUIRING OPERATOR INTERV
000010  .DUSR    DCCNF=  1B12    ; OUTPUT DEVICE WITHOUT FORM FEED HARDWA
000020  .DUSR    DCTO=   1B11    ; TELETYPE OUTPUT DEVICE
000040  .DUSR    DCKEY=  1B10    ; KEYBOARD DEVICE
000100  .DUSR    DCNAF=  1B9     ; OUTPUT DEVICE REQUIRING NULLS AFTER FO
000200  .DUSR    DCRAT=  1B08    ; RUBOUTS AFTER TABS REQUIRED
000400  .DUSR    DCPCK=  1B07    ; DEVICE REQUIRING PARITY CHECK
001000  .DUSR    DCLAC=  1B06    ; REQUIRES LINE FEEDS AFTER CARRIAGE RTN
004000  .DUSR    DCFWD=  1B04    ; FULL WORD DEVICE (ANYTHING GREATER THA
010000  .DUSR    DCFFO=  1B03    ; FORM FEEDS ON OPEN
020000  .DUSR    DCLTU=  1B02    ; CHANGE LOWER CASE ASCII TO UPPER
040000  .DUSR    DCC80=  1B01    ;READ 80 COLUMS
100000  .DUSR    DCDIR=  1B00    ; DIRECTORY DEVICE
```

```
.MAIN

        ; USER STATUS TABLE (UST) TEMPLATE

000400  .DUSR    UST=     400       ; START OF USER STATUS AREA

000000  .DUSR    USTPC=   0         ; PROGRAM COUNTER (LEAVE AT DISPLACEMENT
000001  .DUSR    USTZM=   1         ; ZMAX
000002  .DUSR    USTSS=   2         ; START OF SYMBOL TABLE
000003  .DUSR    USTES=   3         ; END OF SYMBOL TABLE
000004  .DUSR    USTNM=   4         ; NMAX
000005  .DUSR    USTSA=   5         ; STARTING ADDRESS
000006  .DUSR    USTDA=   6         ; DEBUGGER ADDRESS
000007  .DUSR    USTHU=   7         ; HIGHEST ADDRESS USED
000010  .DUSR    USTCS=   10        ; FORTRAN COMMON AREA SIZE
000011  .DUSR    USTIT=   11        ; INTERRUPT ADDRESS
000012  .DUSR    USTBR=   12        ; BREAK ADDRESS
000013  .DUSR    USTIN=   13        ; INITIAL START OF NREL CODE
000014  .DUSR    USTIS=   14        ;INTERRUPT SERVICE WORD
000015  .DUSR    USTWA=   15        ; I/O WAIT RETURN
000016  .DUSR    USTRS=   16        ; I/O COMPLETION RESTORE
                                   ; DEFINE 4 SPARE WORDS
000023  .DUSR    USTA0=   23        ; SAVE STORAGE FOR AC0
000024  .DUSR    USTA1=   24        ; AC1
000025  .DUSR    USTA2=   25        ; AC2
000026  .DUSR    USTA3=   26        ; AC3
000027  .DUSR    USTCY=   27        ; CARRY

000030  .DUSR    USTEL=   30        ; ENTRY LENGTH
000001  .DUSR    USTEC=   1         ; ENTRY COUNT

000010  .DUSR  MXFNO=10            ;MAX NUMBER OF FILE TABLES
000430  .DUSR  UFPT=UST+USTEL      ;USER FILE POINTER TABLE
000010  .DUSR  UFTEC=MXFNO         ;ENTRY COUNT
000440  .DUSR  UFT=UFPT+UFTEC      ;UFT'S
```

```
.MAIN

        ;
        ; SYSTEM PARAMETERS.
        ;

        ; LINKAGE

        .DUSR SAVE = JSR@ 3          ;CALL TO SAVE REGISTERS
        .DUSR RTRN = JSR@ 4          ;CALL TO RESTORE REGISTERS
        .DUSR RTLOC=0                ;RETURN LOCATION
        .DUSR AC0=1                  ;AC0
        .DUSR AC1=2                  ;AC1
        .DUSR AC2=3                  ;AC2
        .DUSR TMP=4                  ;FIRST TEMPORARY
        .DUSR MXTMP=TMP+10
        .DUSR SP=-1                  ;CURRENT STACK POINTER
        .DUSR SLGT=MXTMP-SP+1        ;STACK FRAME LENGTH
        .DUSR OSP=-SLGT+SP           ;LAST FRAME POINTER
        .DUSR NSP=SLGT+SP                  ;NEXT FRAME POINTER
        .DUSR OAC0=AC0-SLGT          ;OLD AC0
        .DUSR OAC1=AC1-SLGT          ;OLD AC1
        .DUSR OAC2=AC2-SLGT          ;OLD AC2
        .DUSR ORTN=RTLOC-SLGT        ;RETURN LOCATION.
        .DUSR NFRAM=22               ;NUMBER OF SYSTEM STACK FRAMES

        ;
        ; MISC.
        ;

        .DUSR RLOC = 6               ;PAGE ZERO TEMP.
        .DUSR CSP = 10               ;STACK POINTER
        .DUSR .PNIC = 7              ;PANIC
        .DUSR UMSK=14                ;USER MASKING ROUTINE

        ;
        ; BUFFER ENTRY
        ;

        .DUSR BQTLA=-11              ;TIME LAST ASSIGNED ( 0 = USE ME FIRST)
        .DUSR BQDST=-10              ;DEVICE STATUS WORD
        .DUSR BQDCB=-7               ;DCB ADDRESS
        .DUSR BQERC=-6               ;ERROR COUNT
        .DUSR BQST=-5                ;STATUS WORD
        .DUSR BQDCT=-4               ; DCT ADDRESS
        .DUSR BQUN=-3                ;UNIT NUMBER
        .DUSR BQCA=-2                ;CURRENT BLOCK DEVICE ADDRESS
        .DUSR BQNXT=-1               ;LINK TO NEXT BUFFER
        .DUSR BQBF=0                 ;START OF DATA
        .DUSR BQNXL=377              ;LINK WORD/FILE NUMBER
        .DUSR BQXTA=400              ;XTRA WORD

        .DUSR BQEL = BQXTA-BQTLA+1
```

```
;
; DEVICE CONTROL BLOCK
;

.MAIN
          .DUSR DCBDC=0          ;DCT ADDRESS
          .DUSR DCBUN=1          ;UNIT NUMBER
          .DUSR DCBCA=2          ;CURRENT BLOCK DEVICE ADDRESS
          .DUSR DCBCB=3          ;CURRENT BLOCK NUMBER
          .DUSR DCBST=4          ;STATUS
          .DUSR DCBNA=5          ;NEXT ADDRESS
          .DUSR DCBLA=6          ;LAST ADDRESS
          .DUSR DCBUR=7          ;SYS.DR DCB POINTER
          .DUSR DCBFA=10         ;FIRST ADDRESS

          .DUSR SFEL=DCBFA-SFKEY+1          ;ENTRY ELNGTH



        ; DEVICE CONTROL TABLE (DCT) LAYOUT

        ; COMMON TO ALL DEVICES

          .DUSR DCTCD=    0       ; DEVICE CODE
          .DUSR DCTMS=    1       ; MASK OF LOWER PRIORITY DEVICES
          .DUSR DCTCH=    2       ; DEVICE CHARACTERISTICS
          .DUSR DCTLK=    3       ; LINK TO NEXT DCT
                                  ; (-1 TERMINATES THE CHAIN)
          .DUSR DCTIS=    4       ; INTERRUPT SERVICE ROUTINE ADDRESS
          .DUSR DCTCM=    5       ; COMMAND ENABLE BIT WORD

                ; DEFINE THE COMMAND BITS
                .DUSR   OF=     1015    ; OPEN FILE
                .DUSR   CF=     1014    ; CLOSE FILE
                .DUSR   RS=     1013    ; READ SEQUENTIAL
                .DUSR   RL=     1012    ; READ LINE
                .DUSR   RR=     1011    ; READ RANDOM
                .DUSR   WS=     1010    ; WRITE SEQUENTIAL
                .DUSR   WL=     1009    ; WRITE LINE
                .DUSR   WR=     1008    ; WRITE RANDOM
                .DUSR   OA=     1007    ;OPEN FOR APPEND

          .DUSR DCTDT=    6       ; COMMAND DISPATCH TABLE ADDRESS WORD
```

```
.MAIN


        ! COMMON TO DEDICATED DEVICES (I.E. SINGLE USER/SINGLE BUFFER)

000007  .DUSR DCTST=     7       !ADDRESS OF DEVICE START ROUTINE
000010  .DUSR DCTSP=     10      !ADDRESS OF DEVICE STOP ROUTINE
000011  .DUSR DCTFL=     11      !FLAGS

                !DEFINE THE FLAGS
000001          .DUSR DCACT=1B15        !ACTIVE FLAG
000004          .DUSR DCACP=1B13        !ACCEPT CHARACTER FLAG
000010          .DUSR DCKMD=1B12        !TTY KEYBOARD MODE FLAG

000012  .DUSR DCTBS=     12      ! BUFFER SIZE ( BYTES )
000013  .DUSR DCTBF=     13      ! BUFFER FIRST ADDRESS (BYTE )
000014  .DUSR DCTBL=     14      ! BUFFER LAST  ADDRESS
000015  .DUSR DCTIP=     15      ! BUFFER INPUT POINTER (BYTE )
000016  .DUSR DCTOP=     16      ! BUFFER OUTPUT POINTER
000017  .DUSR DCTCN=     17      ! COUNT OF ACTIVE DATA
000020  .DUSR DCTTO=     20      ! TIMEOUT WORD (ALL INPUT DEVICES)
000020  .DUSR DCTCC=     20      ! COLUMN COUNTER (ALL OUTPUT DEVICES)
000021  .DUSR DCTRC=     21      ! RESTART CONSTANT (ALL INPUT DEVICES)
000021  .DUSR DCTLC=     21      ! LINE COUNTER (ALL OUTPUT DEVICES)
000022  .DUSR DCTS0=     22      ! DEVICE SPECIAL WORD 0
000023  .DUSR DCTS1=     23      ! DEVICE SPECIAL WORD 1
000022  .DUSR DCTTR=     DCTS0   ! TRANSLATION ROUTINE ADDRESS

        ! COMMON TO BLOCK TRANSFER DEVICES

000007  .DUSR DCTRDB=7          !READ A BLOCK
000010  .DUSR DCTPRD=10         !PREREAD NEXT BLOCK
000011  .DUSR DCSTI=11          !START INPUT
000012  .DUSR DCSTO=12          !START OUTPUT
000013  .DUSR DCCRQ=13          !CURRENT REQUEST BUFFER POINTER
000014  .DUSR DCDCL=14          !DCT LINK
000015  .DUSR DCTRL=15          !READ LAST BLOCK
000016  .DUSR DCTRN=16          !READ NEXT BLOCK
000017  .DUSR DCTIN=17          !DEVICE INITIALIZATION
000020  .DUSR DCTRS=20          !DEVICE RELEASE
000021  .DUSR DCNBK=21          !NUMBER OF BLOCKS ON DEVICE
```

```
.MAIN

                ; DEFINE THE CHARACTERISTICS

        .DUSR   DCCPO=  1B15    ; DEVICE REQUIRING LEADER/TRAILER
        .DUSR   DCCGN=  1B14    ; GRAPHICAL OUTPUT DEVICE WITHOUT TABBIN
                                ; HARDWARE
        .DUSR   DCIDI=  1B13    ; INPUT DEVICE REQUIRING OPERATOR INTERV
        .DUSR   DCCNF=  1B12    ; OUTPUT DEVICE WITHOUT FORM FEED HARDWA
        .DUSR   DCTO=   1B11    ; TELETYPE OUTPUT DEVICE
        .DUSR   DCKEY=  1B10    ; KEYBOARD DEVICE
        .DUSR   DCNAF=  1B9     ; OUTPUT DEVICE REQUIRING NULLS AFTER FO
        .DUSR   DCRAT=  1B8     ; RUBOUTS AFTER TABS REQUIRED
        .DUSR   DCPCK=  1B7     ; DEVICE REQUIRING PARITY CHECK
        .DUSR   DCLAC=  1B6     ; REQUIRES LINE FEEDS AFTER CARRIAGE RTN
        .DUSR   DCFWD=  1B4     ; FULL WORD DEVICE (ANYTHING GREATER THA
        .DUSR   DCDIR=  1B0     ; DIRECTORY DEVICE


.MAIN

        ; DEFINE THE DEVICE MASK BITS

        .DUSR   MSTTO=  1B15    ; TTO
        .DUSR   MSTTI=  1B14    ; TTI
        .DUSR   MSPTP=  1B13    ; PTP, RTC
        .DUSR   MSLPT=  1B12    ; LPT, PLT
        .DUSR   MSPTR=  1B11    ; PTR
        .DUSR   MSCDR=  1B10    ; CDR, MTA
        .DUSR   MSDSK=  1B9     ; DSK
        .DUSR   MSDKP=  1B7     ; DKP
```

APPENDIX D

CLI INTERPRETATION OF USER COMMANDS

The action taken by the CLI upon reading a command line is sufficiently flexible so that users can, if they wish, design programs to perform system command functions.

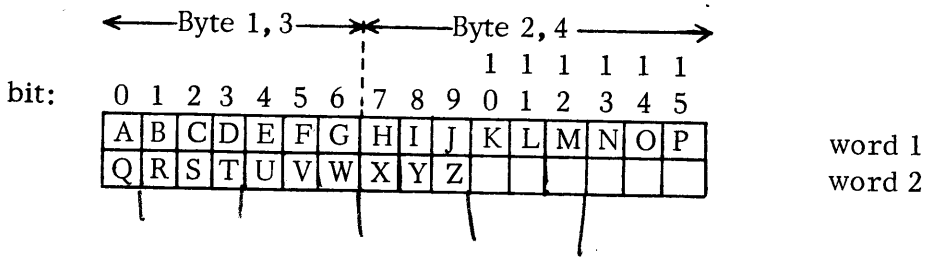
When the CLI reads a command line and does not recognize the first file name, the CLI always builds a command file before the save file of that name is loaded. The command file reflects an edited version of the command line.

For example, suppose the user issues the command line:

FOO↵

The CLI does not recognize FOO as a known command word. It builds a command file with the byte organization shown below:

Byte	Contents	
0	F	Each character of the file name occupies a byte. The file name is terminated by a null byte.
1	O	
2	O	
3	null	
4		Four bytes (2 words) are set aside for global switches of FOO. Each letter switch sets a bit. A sets bit Ø of the first word, etc., as shown in the switch/bit correspondence diagram below.
5		
6		
7		
8	377 →	The command file is always terminated by a 377_8.



Note that the CLI does not attempt any interpretation of switches in building the command file. The CLI simply sets the appropriate bit.

Additional file name arguments and local switches are handled in the same way when the CLI builds the command file. Suppose the user types the command:

FOO/B AA ZZ/X MUMB↵

The CLI would then build the following command file:

Byte	Contents	
0	F	Command file name FOO, terminated by null byte.
1	O	
2	O	
3	null	
4	1	Global switches of FOO. Bit 1 (switch B) set ON.
5		
6		
7		
8	A	Argument AA, terminated by null byte.
9	A	
10	null	
11		Four bytes set aside for local switches of AA. None set.
12		
13		
14		
15	Z	Argument ZZ, terminated by null byte.
16	Z	
17	null	
18		Local switches of ZZ. Bit 23 (switch X) set ON.
19		
20	1	
21		
22	M	Argument MUMB, terminated by null byte.
23	U	
24	M	
25	B	
26	null	
27		Local switches of MUMB. None set.
28		
29		
30		
31	377 →	Command file terminator.

Since the CLI does not interpret switches, the user can set up program interpretation of such switches. This gives the user an added means of passing information to a program to be executed, since he can use switches as well as arguments.

The command file is always named:

COM.CM

and is created on the default directory device.

A read line from a disk file will terminate on a null (as well as carriage return and form feed.) This is quite useful in reading COM.CM arguments. The following example illustrates how a user could read the first argument of the command file as well as its global switches.

```
        LDA     0,CFILE         ;COM.CM POINTER
        .SYSTM
        .OPEN   3               ;OPEN ON CHANNEL 3
        JSR     EROR            ; ? ?
        LDA     0,ARG1          ;FIRST ARGUMENT POINTER
        .SYSTM                  ;READ IT (THE NULL
        .RDL    3               ;TERMINATOR IS ALSO
        JSR     EROR            ;TRANSFERRED)
        LDA     0,GLOB          ;POINTER FOR GLOBAL
        LDA     1,C4            ;SWITCHES
        .SYSTM                  ;READ FOUR BYTES
        .RDS    3
        JSR     EROR
        .
        .
        .
C4:     4
GLOB:   2*.GLOB
ARG1:   2*.ARG1
CFILE:  2*.CFIL

.GLOB:  .BLK    2
.ARG1:  .BLK    10
.CFIL:  .TXT    *COM.CM*
```

APPENDIX E

ADDING DEVICES TO THE SYSTEM

The I/O devices that are included as part of the DOS system are listed in Chapter 1. The user can, however, add device drivers to DOS enabling the use of additional devices. All changes to DOS to incorporate another device should be made at the source level. This Appendix describes briefly the required changes to add a device. However, to understand fully the process of adding a device driver, the user must make a source listing of those DOS programs described in this document and study the entries made in tables for system devices. The tapes required are available from DGC by ordering model number 3040 (without magnetic tape) or 3157 (with magnetic tape).

During full intialization of the system, names of all peripheral devices in the system are added to the system file directory, SYS.DR. The device name will be entered only if the user has forced the driver to be loaded. The general procedure for adding a device driver is as follows:

1. Add an entry to the SITAB table in DVINIT to enable entry of a device name in SYS.DR.

2. Declare the Device Control Table (DCT) link and the Device Control Table address as .EXTNs in DVINIT, forcing the driver to be loaded at SYSGEN time.

3. Add an entry to the Device Control Table Pointer Table (DCTT) in TABLES, determining the priority of device service at interrupt time.

4. Add an entry in TABLES for the DCT link word, insuring that it is assigned after the last system defined link.

5. Define and supply all DCT information and routines required. Use the Library File Editor to add this relocatable module to the system libraries.

6. Perform a SYSGEN using the updated libraries.

Sections following describe each of these steps.

CREATING A DEVICE ENTRY IN SITAB

SITAB is a table in DVINIT (System Device Initialization). SITAB consists of a series of three-word entries, and the table is terminated by a word of zeroes. Each entry in SITAB consists of:

WORD 1 - A byte pointer to the device system name, packed left to right.

WORD 2 - The address of the word in TABLES that contains the Device Control Table (DCT) link for the device. *

WORD 3 - File attributes of the device. (All system devices are declared ATCHA and ATPER, "attribute protected" and permanent".) The possible attributes are:

Mnemonic	Bit Position	Meaning
ATRP	1BØ	Read protected device.
ATCHA	1B1	Attribute protected device.
ATPER	1B14	Permanent device.
ATWP	1B15	Write protected device.

An example of an entry in SITAB is shown below for the high speed paper tape reader:

```
                2*SPTR                          ;NAME BYTE POINTER
                PTRL                            ;POINTER TO DCT LINK
        ATPER+ATWP+ATCHA                        ;ATTRIBUTES
                .
                .
                .
SPTR:           .TXT    *$PTR*
```

DECLARING THE DCT ADDRESS

The last relocatable binary file of the system library, called TABLES, allocates storage for all system tables not residing in page one. One of these tables is the Device Control Table Pointer Table (DCTT). This table must contain an entry for every device control table within the system. The form of each entry is:

```
.dvdDCT:        dvd  DCT
```

where *dvd* represents the DGC device mnemonic for each particular device's control table. The label must be declared as an entry, while the control table address must be declared as a normal external. When adding an entry, the user

* DCT links must be assigned beginning with the last link defined by the System +1.

DECLARING THE DCT ADDRESS (Continued)

can select any three-letter device mnemonic not used for a DGC device.

A DCT link equivalence must also be defined and declared as an entry. The link is used to index the table, DCTT.

DEFINING AND SUPPLYING DCT INFORMATION

Each device defined must specify its own Device Control Table. Each table consists of 20 words, described below.

DEVICE CONTROL TABLE (DCT) LAYOUT

Word 1: Device code.

Word 2: Mask word.

Clear a bit for every priority considered higher than the priority of this device. The devices corresponding to the priority bits that are left cleared will be permitted to interrupt the current device.

Word 3: Device characteristic word. A list of device characteristics is given in the table on the next page.

Word 4: Link to the next DCT. (Allocate as .BLK 1)

Word 5: Address of the device interrupt service routine.

This word is initialized by DVINIT.

Word 6: Command enable bit word.

A series of two-letter mnemonics, added together, indicating the operations the device can perform.

Mnemonic	Order	Function
OF	1	open file (always necessary)
CF	2	close file (always necessary)
RS	3	read sequential
RL	4	read line
RR	5	read random
WS	6	write sequential
WL	7	write line
WR	8	write random

DEVICE CHARACTERISTICS

MNEMONIC	BIT POSITION	MEANING
DCCPO	1B15	Device requiring leader/trailer
DCCGN	1B14	Device requiring tab simulation.
DCIDI	1B13	Device requiring operator intervention.
DCCNF	1B12	Device requiring form feed simulation.
DCTO	1B11	Teletype output device.
DCKEY	1B10	Keyboard input device (uncontrollable)
DCNAF	1B09	Device requiring nulls after form feeds.
DCRAT	1B08	Device requiring rubouts after tabs.
DCPCK	1B07	Device requiring even parity check on input, even parity computation on output.
DCLAC	1B06	Device requiring line feeds after carriage returns.
DCFWD	1B04	Full word device (anything greater than a byte).
DCFFO	1B03	Form feeds sent on .OPEN
DCLTU	1B02	Convert lower to upper case ASCII.
DCC80	1B01	Read 80 columns on input if set, 72 if reset. Send 80 characters on output, 72 if reset.
DCDIR	1B00	Directory Device.

Word 7: Address of the device command dispatch table.

One entry is required for every bit specified in Word 6. Further, the table order must correspond exactly to the order of the functions given under Word 6. For example, if Word 6 appeared as follows:

OF+CF+RR+WL

the dispatch table must look as follows:

```
DTAB:   OFILE       ;OPEN FILE ROUTINE
        CFILE       ;CLOSE FILE ROUTINE
        RNDOM       ;READ RANDOM ROUTINE
        LINE        ;WRITE LINE ROUTINE
```

Word 8: Address of device start routine.

Device Start Routine Specification

Input devices: Activate the device and return.

Output devices: Character passed in AC0.

Activate the device. If the device will not interrupt as a result of this action, return to the normal return point. Otherwise, bump RLOC for a return to normal return + 1.

For example:

```
LPTST:  STA     3, RLOC     ;SAVE RETURN LOCATION
        DOAS    0, LPT      ;KICK PRINTER
        SKPBZ   LPT         ;WILL IT INTERRUPT?
        ISZ     RLOC        ;YES, BUMP RETURN
        LDA     3, CSP      ;STACK POINTER IN AC3
        JMP     @RLOC       ;RETURN
```

Word 9: Address of device clear routine.

Device Clear Routine Specification

Clear the device and return.

Word 10: Flag word used by RINGIO (allocate as .BLK 1)

Word 11: Buffer size in bytes.

Word 12: Buffer starting byte address.

Word 13: Buffer ending byte address + 1.

Words 14-16: Variable words used by RINGIO (Allocate as .BLK 3)

Word 17: Input devices: EOF timeout constant.

A parameter "SCTIM" is defined on the user parameter tape, (090-000090), which corresponds to a time of 1 millisecond on the Supernova SC. If the device requires six milliseconds to timeout, the word can be allocated as

6*SCTIM

Output devices: column counter (allocate as .BLK 1).

Word 18: Input devices: restart constant

Output devices: not used.

Words 19, on Spare words, which may be used for any special purpose device temporaries or constants.

SUBROUTINE LINKAGE

Subroutine linkage among all system subprograms is implemented within the module GSUB. Before attempting to interface a driver to the system, the user should be familiar with the subroutine linkage facilities. Adhering to these conventions will enable pure, reentrant routines to be written with little effort.

On all system I/O commands, the DCT address of the device requested will be passed in AC2. The source or destination byte pointer for .RDL, .RDS, .WRL, .WRS is passed in ACØ. The record number for .RDR, .WRR is passed in AC1. On .RDS and .WRS, the byte count is passed in AC1. These parameters are specifically those required by the generalized RINGIO routines (see next section), and in many cases these RING I/O routines alone suffice.

GENERALIZED RING I/O ROUTINES

The Ring I/O Module (RINGIO) provides a number of useful, general purpose, reentrant routines for handling byte I/O from any device, input or output, using the program interrupt facility. The basic ring buffer philosophy is to maintain two pointers, a current input pointer and a current output pointer. The input pointer indicates the first free byte slot in the buffer; the output pointer, if not equal to the input pointer, indicates the next byte available for output. These terms are relative. An input device "inputs" to the buffer at interrupt time and "outputs" from the buffer at program base level. An output device "inputs" to the buffer at program base level and "outputs" from the buffer at interrupt time. (See diagram on the next page.)

A brief description of the major routines and their calling sequences will be given below. More detailed information can be obtained by scanning the listing of RINGIO. It is important to note that although buffer input/output is in byte increments, devices transmitting larger data widths can use the same basic scheme. The card reader, for example, inputs its full word by calling for two consecutive byte inputs.

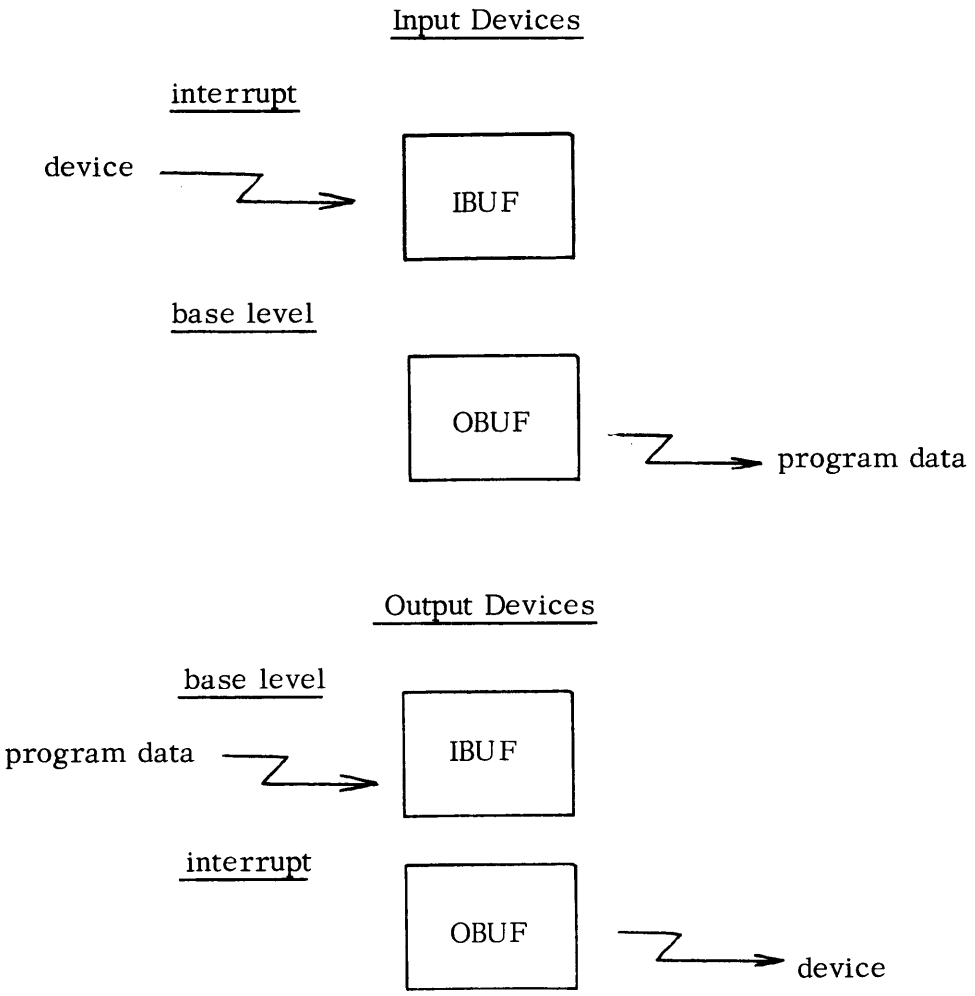
Generalized Open Routine

Input:	ACØ	- file name byte pointer
	AC2	- UFT address
Calling Sequence	JSR OPN	
	error return	
	normal return	

This routine clears the device and initializes its device control table. This implies

Generalized Open Routine (Continued)

the DCT has provided all necessary ring buffer information as well as the four words of variable storage (Words 10, 14-16).



Generalized Close Routine

Input: AC2 - UFT address

Calling Sequence: JSR CLSO | CLSI (close output or close input)
error return
normal return

CLSO should be used only to close output devices. It waits until all output has settled, clears the column counter, clears the device, and initializes the DCT.

CLSI should be used to close input devices. It merely clears the device and initializes the DCT.

Generalized Read Sequential

Input: DCT address in AC2
Destination byte pointer in AC0.
Destination byte count in AC1.

Calling Sequence:

JSR RDS
error return
normal return

The device will be read, a byte at a time, until the byte count requested is satisfied. The error return is taken if:

1. End of file occurs on device. The partial count read is returned in AC1.

2. The device has the "full word" characteristic, and an odd number of bytes is requested.

Generalized Write Sequential

Input: DCT address in AC2.
Source data byte pointer in AC0.
Source data byte count in AC1.

Calling Sequence:

JSR WRS
error return
normal return

Generalized Write Sequential (Continued)

The data will be read and transmitted to the device until the byte count is expired. Neither read nor write sequential alters the data in any manner. This mode is, therefore, the standard mode for "binary" transfers.

Generalized Read Line

Input: DCT address in AC2.
Destination byte pointer in ACØ.

Calling Sequence:

```
JSR  RDL
error return
normal return
```

Output: Byte count read in AC1.

This routine is used to transmit ASCII data and terminate after transmission of a carriage return or a form feed. All bytes transmitted are masked to seven bits. Nulls, line feeds, and rubouts are unconditionally ignored. The error return is taken and a system error code returned in AC2 for the following:

1. An end of file.
2. A parity error (the last character transmitted.)
3. An excessive line length (132 characters) without an appropriate terminator.

Generalized Write Line

Input: DCT address in AC2.
Source data byte pointer in ACØ.

Calling Sequence:

```
JSR  WRL
error return
normal return
```

Output: Byte count read from data area in AC1.

This routine will transmit ASCII data to the appropriate device and terminate after transmitting either a carriage return or a form feed. Termination will also occur on a null but without transmitting it. Checks are made of the device characteristics to determine whether to perform :

Generalized Write Line (Continued)

1. Parity on output.
2. Nulls after form feeds.
3. Line feeds after carriage returns.
4. Tab simulation (every 8 columns).
5. Rubouts after tabs.

The error return is taken after 132 bytes have been transmitted without detection of a terminator.

Input to Ring Buffer

Input: Character in ACØ (left byte ignored)
DCT address in AC2.

Calling Sequence:

```
    JSR  IBUF
return  - buffer full
return  - buffer not full
return  - buffer became full (active flag cleared)
```

The byte is input to the current slot in the ring buffer and all bookkeeping in the DCT maintained. Note that for output devices the active flag should be "set" again if the third return is taken.

Output from Buffer

Input: DCT address in AC2.

Calling Sequence:

```
    JSR  OBUF
return  - buffer not empty
return  - buffer empty (active flag cleared)
```

Output: Character returned in AC0 (if successful) with bits 0-7 cleared.

A byte is grabbed (if possible)from the buffer and all ring buffer bookkeeping in the DCT maintained.

UPDATING THE SYSTEM LIBRARY

The system library is ordered as shown in the table following. To add a driver, it must be inserted into the library using the library file editor. It must be placed within the library somewhere after SYSTEM and before TABLES. To insure a driver is loaded, an .EXTN for its device control table should be declared in DVINIT.

SYSTEM LIBRARY ORDER

Relocatable Binary Title	Primary Function
INIT	System full and partial initializations
DVINIT	System device initialization
SYSTEM	System call decoding
FILEIO	Disk File I/O
FILSYS	File system management
GSUB	General purpose subroutines and linkage
PANIC	Panic module
INTD	First level interrupt determination
RINGIO	Ring buffer I/O management
TTYDRV	ASR 33 Teletype driver
PTRDRV	High speed paper tape reader driver
PTPDRV	High speed paper tape punch driver
PLTDRV	Incremental plotter driver
LPTDRV	Line printer driver
CDRDRV	Card reader driver
MTADRV	Magnetic tape driver
DKPDRV	Disk pack driver
DSKDRV	Fixed head disk driver
TABLES	Tables storage allocation

SYSTEM GENERATION

To load the system, determine the additional space necessary to load your driver plus any additional words added to the system. Invoke the SYSGEN save file and answer all queries.

Invoke the RLDR and MKABS commands necessary to load and punch the system, adjusting the value for NMAX down by the additional amount of space the system now requires.

APPENDIX F

SYSTEM TAPES

The following dump tapes will be distributed with all DOS systems.

NUMBER	NAME
088-000003	System Generation (SYSGEN.SV)
088-000002	Relocatable Loader; Debug III (RLDR.SV, SYS.LB)
088-000001	Relocatable Assembler; Editor; Cross Reference (ASM.SV, EDIT.SV, XREF.SV)
088-000013	Octal Editor, Binary Loader (OEDIT.SV, BLDR.SV)
088-000008	Library File Editor (LFE.SV)
088-000012	Command Line Interpreter Library (CLI.LB)
088-000016	System File Ø (SYSØ.LB)
088-000010	System Library 1A (No magnetic tape software) (SYS1A.LB)
or	
088-000011	System Library 1B (Magnetic tape software) (SYS1B.LB)

Three 12K bootstrap systems will be provided in absolute binary format:

NUMBER	NAME
091-000027	Fixed head disk bootstrap.
091-000053	Moving head disk bootstrap (4047 or 4057).
091-000058	Moving head disk bootstrap (4048).

If a 9-track magnetic tape drive is part of the configuration, the tape bootstrap program will be sent.

NUMBER	NAME
088-000015	Tape Bootstrap (TBOOT.SV)

Two parameter tapes in ASCII will be provided:

NUMBER	NAME
090-000090	User parameters.
090-000176	System parameters.

If the configuration is 16K or larger, the following compilers will be sent:

NUMBER	NAME
088-000005	FORTRAN Compiler (FIV.SV)
088-000014	FORTRAN Dispatch (FORT.SV)
088-000006	ALGOL Compiler (AL1.SV, ALGOL.SV LIBRARY)
088-000007	ALGOL Compiler (AL2.SV)

INDEX

cut along dotted line

DATA GENERAL CORPORATION
PROGRAMMING DOCUMENTATION
REMARKS FORM

DOCUMENT TITLE _____

DOCUMENT NUMBER (lower righthand corner of title page) _____

Specific Comments. List specific comments. Reference page numbers when applicable. Label each comment as an addition, deletion, change or error if applicable.

General Comments and Suggestions for Improvement of the Publication.

FROM: Name: _____ Date: _____
Title: _____
Company: _____
Address: _____
_____

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:

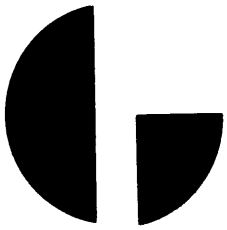
Data General Corporation

Southboro, Massachusetts 01772

ATTENTION: Programming Documentation

FOLD UP SECOND FOLD UP

STAPLE



DATA GENERAL
CORPORATION
Southboro,
Massachusetts 01772
(617) 485-9100