Prime Computer User Guide

MAN-1675

For Disk and Virtual Memory Operating Systems

REV. 7.Ø

356. PAGES.

MAY. 1975.

PRIME COMPUTER, INC., 145 PENNSYLVANIA AVENUE, FRAMINGHAM, MA. 01701

OBSOLETE

MAN1675

DISK AND VIRTUAL MEMORY

OPERATING SYSTEMS

USER GUIDE

Revision A

May 1975

PRIME COMPUTER, INC.

[145 Pennsylvania Ave., Framingham, Mass., 01701]

Prime Computer, Incorporated

145 Pennsylvania Avenue

Framingham, Massachusetts 01701

Performance characteristics are
subject to change without notice.

CONTENTS

CONTENTS

CONTENTS

CONTENTS

MAN 1675

CONTENTS

CONTENTS

CONTENTS

CONTENTS

CONTENTS

TABLES

LIST OF ILLUSTRATIONS

FOREWORD

This manual provides operating and programming information for the Prime Disk Operating System (DOS) and the Prime Virtual Memory Disk Operating System (DOS/VM). The version of these systems described herein are each implemented on master disk as Revision 7 (Rev. 7.0).

Users must be familiar with FORTRAN or Prime macro-assembly language programming and also familiar with operation of the Prime CPU Control Panel as described in the operator's guide.

Information is organized as follows:

Section 1	General information on DOS, DOS system configuration, and DOS relationships to other operating system.
Section 2	Definition of a DOS file, DOS file types, an overview of the DOS file system and file structures. This section includes a primer on files and is further supported with detailed information.
Section 3	Information on bootstrapping; installing and copying master disk; initializing and running DOS for the first time; DOS backup; DOS shutdown procedures.
Section 4	DOS and DOS/VM user commands. First, an overview of the commands and their functions arranged in a logical sequence (i.e., the widely used to the less used). Next, detailed descriptions of commands normally used for programming development and productions, arranged alphabetically.
Section 5	Description of subroutines available to the user for file system and terminal I/O.
Section 6	Introduces the Virtual Memory Disk Operating System (DOS/VM), describes system configuration, and gives detailed information on DOS/VM startup and shutdown. This section also describes commands normally issued from the DOS/VM supervisor terminal by a system operator.
Section 7	Describes how input/output is virtualized on DOS/VM.

APPENDICES

Appendix A	Describes the format of DOS file record headers and the physical organization of DOS files; also describes UFD format and the format of the DSKRAT.
Appendix B	Describes the BOOTSTRAPS available for hardware configurations.
Appendix C	Gives specific examples that show the user how to create Segment Directories and DAM files.
Appendix D	Discusses Data Base Management (DBM) and those features of the DOS (DOS/VM) file system that support DBM.
Appendix E	Is entitled "All About FIXRAT". It describes FIXRAT in detail for both old and new users. It also lists all FIXRAT messages.
Appendix F	Is a discussion of the File Utility FUTIL and the file manipulation command available when operating under control of that file utility. This appendix also lists messages that may occur while using FUTIL.
Appendix G	Is a summary of the FORTRAN libraries and the Input Output Control System (IOCS) library. Some of this information may be obsolete and will be replaced by the library manual.
Appendix H	Is a definitive example of use of the DOS file system, particularly if the user wishes to learn how to use the subroutines SEARCH, PRWFIL, and ATTACH.
Appendix I	Describes the contents of ERRVEC, the system error vector, for both error return and normal return.
Appendix J	Lists the DOS (DOS/VM) error messages and value of the disk status word.
Appendix K	Describes disk hardware operation for typical moving head disk drives connected to DOS or DOS/VM.
Appendix L	Lists the Prime ASCII character sets.
Appendix M	Is a summary of the DOS and DOS/VM commands and their formats. The last page of this appendix lists obsolete commands for old users who may be interested.

RELATED PUBLICATIONS

The following Prime documents should be available for reference:

Title	Manual No.
Prime CPU Operator's Guide (Console and peripheral device operation)	MAN1672
Prime CPU System Reference Manual (instruction set, addressing modes, input/output programming)	MAN1671
Macro Assembler Language Reference Manual	MAN1673
Program Development Software Manual (Editor, Loader, TAP, etc.)	MAN1879
Library Subroutine Manual*	-----
FORTRAN IV Language Reference Manual	MAN1674
Magnetic Tape Controller User Guide	MAN1940
System Option Controller User Guide	MAN1944
System Terminals User Guide	MAN1941
Disk Controller User Guide	MAN1948

*Currently undergoing development

SYMBOLS AND ABBREVIATIONS

Symbols and abbreviations and special characters used frequently in the rest of this handbook are defined below:

Symbol	Meaning
Number representations:	
1000	1000 decimal
'1000	1000 octal
$1000	1000 hexadecimal
Teletype functions:	
CR	Carriage Return
LF	Line Feed
\	Backslash (upper case L) used as tab character (Editors only)
"	Delete character (cancels last typed character); do not use in DOS command strings. However, this delete character may be used in DOS/VM lines.
?	Kill character (deletes all characters in current line).
,:;,+-	In the editor, ED, separates multiple commands on a line.
↑ or ^	Signal escape (see Editor in Program Development Systems Manual).

Symbol	Meaning
Miscellaneous:	
SA	Starting address of program of memory block.
EA	Ending address of program or memory block.
[ ]	Brackets enclose optional parameters in command strings.
Underlining	Indicates user input in examples of user/DOS dialogues.
␣	Spaces (in command strings)
b	Blanks or space characters (in Hollerith or ASCII strings).
	Abbreviations
Altrtn	Alternate return program step in case of I/O errors, missing EOF, etc.
Ba	Buffer Address (IOCS).
CPU	Central Processor Unit (the Prime computer proper as opposed to peripheral devices or main memory).
DSKRAT	Disk Record Availability Table
Filename	A DOS filename (in the current UFD, unless otherwise specified).
Funit	DOS File unit (1-16)
Ldisk	Logical disk unit number as assigned by STARTUP command.

Symbol	Meaning
Lunit	Logical device number, (1-15) as used in FORTRAN READ and WRITE statements. (Same as IOCS logical device number.)
MFD	Master File Directory
Password	A DOS password
Punit	IOCS Physical device number (1-15)
UFD	User File Directory
Ufd	In FORTRAN calling sequence: pointer to a UFD name (Hollerith expression or 3-word array)

Filename Conventions

B←XXXX	Binary (Object) file
L←XXXX	LISTING file
C←XXXX	Command file
XXXXXX	Source file
*XXXXX	SAVED (Executable) file
S←XXXX	Segment Directory
U←XXXX	User File Directory (UFD)

SECTION 1

INTRODUCTION

SCOPE OF DOS AND DOS/VM

DOS is the Disk Operating System for the entire Prime family of computers. It is a memory-resident operating system that provides a complete working environment for the user's software development process and for user program development and production use of the various Prime disk options.

DOS/VM has the same capabilities as DOS; and in addition, allows a sharing of the computer resources among a community of up to 31 simultaneous users and a variety of peripheral devices. DOS/VM also gives each user a virtual memory environment.

DOS may function in any of the possible Prime computer system configurations. DOS allows direct memory addressing of up to 64K. It operates under control of a Prime 100, 200, or 300 central processor with or without available options. A broad range of disks are supported by DOS. Up to four disk units, each with a capacity of 30 million words, can be attached to a disk controller (type 4001/4002, which handles up to four disk pack units as well as one fixed head disk (either 128K or 256K word capacity). Alternatively, mass storage (disk) configurations supported by DOS include moving head cartridge disks providing 1-.5-, 3.0- and 6.0-million word capacities, and high-speed fixed-head disks storing 128K, 256K or 512K words (using a Type 4000 Disk Controller). Finally, diskette drives (floppy disks) are supported by DOS and DOS/VM via a diskette controller. All disk units are supported interchangeably by DOS and other Prime system software. DOS configurations may also include a high-speed paper tape reader for system generation. The DOS System Terminal is either any Teletype (or compatible terminal) or a CRT type terminal attached to the system option controller running at 110 baud. Peripherals that are supported by IOCS running under DOS control include: up to four 7- or 9-track magnetic tape transports on one controller, card reader (one per system), character printer (one per system, connected to the system option controller), line printer (one per system), and paper tape reader/punch (one per system). For further details about DOS configuration, refer to Section 3.

The DOS/VM operating system requires a Prime 300 system with a minimum of: 32K of high-speed memory, disk system terminal, and 1 to 31 user terminals. DOS/VM fully supports virtual memory and up to 256K of real memory. For details of the DOS/VM configurations, refer to Section 6.

The minimum configuration upon which DOS operates is a Prime computer with a Teletype for a System Terminal, 16K of memory, and mass storage consisting of diskettes; DOS is upward compatible and it operates on any Prime computer system configuration that is more sophisticated than the minimum.

DOS FEATURES

1. DOS operates in several environments. DOS, when run as the chief operating system of the computer system, functions as a batch operating system, providing automatic job and data stream routing; by storing command sequences on disk. In addition, the Prime RTOS (Real Time Operating System) and the DOS/VM (Virtual Memory Operating System) are started from DOS. Users of these systems must know at least how to start up DOS and then start up their systems from DOS as well as how to shut down DOS. Furthermore, once RTOS or DOS/VM is running, DOS can be run as a background job in RTOS, or DOS can be started up from DOS/VM. The former is a fairly common and useful practice; the latter is a bit esoteric, but it has been done.

2. The fundamental unit with which most DOS commands and concepts deal is the file. Each disk is organized into a system of files, permitting the user to reference programs and data by file name only. Consequently, there is no need for the user to identify specific physical records or to have knowledge of the format of the disk. An overview of files and the associated file system is presented in Section 2.

3. DOS provides an interactive command language for summoning programs and manipulating the file system. The command language interfaces the user to DOS by simple commands entered at the terminal. The same command functions may also be performed by programs, reducing the amount of operator involvement. Software written for stand-alone execution may be run under DOS with no changes. Section 4 describes DOS commands.

4. All standard Prime software is available under DOS and makes use of its command structure and file-handling abilities.

DOS/VM FEATURES

DOS/VM includes all the features of DOS and in addition allows sharing of the computer resources in a virtual memory environment.

SECTION 2

FILE STRUCTURES

CONCEPTS

The following paragraphs define terms used in describing a disk-based operating system (DOS or DOS/VM).

File

A file is a named set of information organized and stored on a magnetic disk in such a way that a computer program can use the information.

For Prime DOS, a file consists of a list of 16-bit binary words; a binary word is the smallest item of information that can be moved to or from a file at one time.

File Access

The process of moving information from a file stored on disk to a location in high-speed memory is called reading from a file. Moving information from a location in high-speed memory to a file stored on disk is called writing to a file.

File Creation

Files may be made through the use of a DOS system editor at a terminal (Teletype or CRT keyboard type); they may be made by copying information stored on paper tape, punched cards, etc.; or they may be generated by computer programs.

Some Typical File Content

1. Lists of employee names, addresses, salaries, etc. stored as files for payroll and bookkeeping programs to use.

2. Computer programs coded in languages that may be read by humans and stored as files in order that other programs may be used to translate the human-readable program into a program that is both meaningful to a computer and can be run on a computer.

Why a File System?

The purpose of having a file system is to simplify the manipulation of large quantities of data using the computer. The major goals of the file system are listed below:

1. File creation without manual pre-allocation of the storage medium
2. Ability to reference a file by name
3. Clustering like files together.

The first goal is implemented by keeping a file on each disk that lists the available space for the disk. Since the whole process is automatic, the average user does not need to concern himself with this process, other than to know that it works.

Referencing files by name means the desired file may be selected for operation by giving the system an array of alphanumeric characters. The file system does this by having a file that is used as a directory; it contains the names of other files and their locations on the disk. The systen can find this Master File Directory (MFD) because its name is always the same and its location is always the same.

The third goal is achieved in two ways. The first is to have many file directories; this allows like files to have their names and locations saved in the file directory file. The second way is by nesting file directories. This means some of the filenames saved in a file directory can be the names of other file directories. Thus, files may be classified to an arbitrary extent.

A side-effect of clustering files in a file directory (files whose names are stored in a file directory are often said to be "in" the directory) is that some degree of access protection can be built in by associating a password with each file directory. In order to examine the files in a directory, the user must first supply the password for that directory.

Summary

A file directory is a file that contains the names of other files on the disk and the location on the disk of these files. A file directory may contain the names of other file directories. In order to access files stored in a directory, the password for that directory must be given.

Using the File System

To access files, the user must be attached to some file directory. This means the file system has been supplied with the proper file directory name and password, and it has found and saved the name and location of the file directory. It can therefore find and operate-on all files contained in that file directory.

The major operations on files are: initialization for access (open); access; shutdown and resource deallocation (close); and deletion.

Opening a File

A file may be opened for reading only, for writing only, or for both reading and writing. If a file is opened for reading only, it may be read; but it cannot be changed.

The operation of opening a file does the following:

1. searches the file directory to see if the filename requested is there;
2. sets up tables and initializes buffers in the operating system;
3. defines a pseudonym for the file. This pseudonym is called the file unit number and is the only name used for transfer of data to and from the file.

If a file is opened for writing only, or for reading and writing, it may be changed; and if the filename is not found in the directory, the filename is added to the file directory, and a new file is created. When a new file is created at the time of opening, no information is contained in the file.

Using an Open File

Once a file has been opened, a file pointer is associated with the file. The file pointer indicates the next binary word to be accessed. To understand how the file pointer works, imagine that the words in a file are serially numbered from 0. The file pointer is then the number of the next word to be accessed in the file.

Access

On an open file, information may be read from the file starting at the file pointer into high speed memory or information may be written to the file starting at the file pointer.

Access and File Pointer

When a file is accessed, the file pointer is incremented once for each binary word accessed.

Position

The file pointer may also be moved backward and forward within a file without moving any data. This is called positioning a file. The value of a file pointer is called the position of the file. Positioning a file to its beginning is often called rewinding a file.

Truncation

It is possible to shorten a file by truncating it. When a file is truncated, the part of the file that is at or beyond the file pointer is eliminated from the file. If the file pointer is positioned at the beginning of the file, all of the information in the file is removed but the filename remains in the file directory.

Closing a File

A file that has been opened may be closed. The file unit number (pseudonym) and the corresponding table areas in the operating system are "cleaned up" and released for reuse.

Deleting a File

The filename of a file that is deleted is removed from the file directory, and all of the disk memory that the file occupied is released for use by other files.

Concept Conclusions

The fundamental operations possible using the DOS file system have now been described. The following paragraphs re-examine files and file directories in more detail and introduce the idea of how files are put together, and how they are related to directories.

Physical Disk Consideration

A disk storage medium is composed of many separate blocks of data recording space (disk records or sectors). How these blocks are put together to make a file can affect the efficiency of positioning by several orders of magnitude. Because of this, the file system has two different ways of linking physical disk records together to form a file. One way, SAM (Sequential Access Method), is stored more compactly on the disk and requires less computer fast memory for efficient operation, but is much slower for repeated random positioning over a file. The other way, DAM (Direct Access Method), is quick for positioning over a file, but requires more disk space and more fast memory. SAM and DAM files are functionally equivalent in all other respects.

More on File Directories

File directories were previously described as files containing the names and locations of other files on the disk. This kind of file directory is referred to elsewhere in the documentation as a User File Directory (UFD). The file system supports a second kind of file directory called a segment directory. Segment directories differ from UFDs in one fundamental respect: they contain file locations but not file names. As far as the file system is concerned, the files in a segment directory have no names. This means that the file system user is charged with all of the bookkeeping involved in the use of a segment directory.

Segment Directory Use

Each binary word in a segment directory is assumed to hold a legitimate file location on the disk. The segment directory file is opened for reading/writing on a unit of the user's choice. The segment directory file is then positioned to the word containing the location of the desired file.

If the desired file is a UFD, the user may attach to it by passing the file unit number to the file system in place of a file name. Similarly, the desired file may be opened, closed, deleted or truncated by giving the file unit number of the segment directory file rather than filename.

FILE SYSTEM

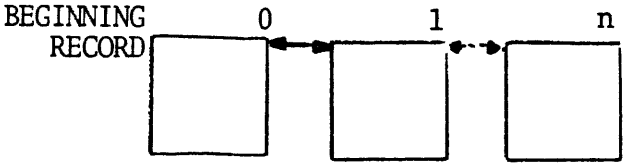
The DOS (and also DOS/VM) file system consists of a hierarchy of files and file directories. There are two types of files and types of directories. These are described in this section.

TYPES OF FILES

The two types of files are: Sequential Access Method files (SAM files) and Direct Access Method files (DAM files). The structural differences between these two file types are transparent to the user.

SAM Files

A SAM file is the basic way of structuring disk records into an ordered set; i.e., a threaded list of physical disk records. The following example shows this structure:



SAM File Structure

In DOS, a SAM file consists of a collection of disk records chained together by forward and backward pointers to and from each record (see Appendix A). Further, each record in a SAM file (or any file) contains a pointer to the beginning record address (BRA) of the file. The file system maintains the record headers and is responsible for the structure of the records on the disk.

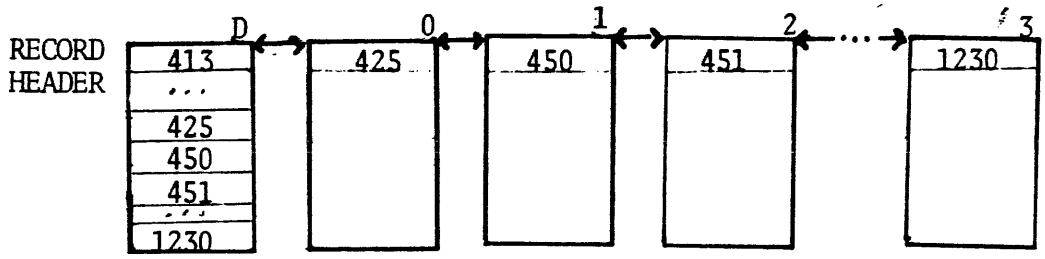
Figure 2-1 shows an example of how SAM files may be related within a DOS file system hierarchy.

DAM Files

A DAM file is a direct access file. DAM file organization uses the SAM file method of making an ordered set; but for purposes of rapidly accessing the i'th data record, a special trick is used:

> Physical disk record 0 of a DAM file is reserved for use by the system. No user data is ever written in this record.

The first disk record (logical record 0) to contain user-written data is the second record of the threaded list of disk records. The first disk record 0 contains pointers to the second, third, ... i'th 440th disk record of the file as shown in the following example:



DAM File Structure

Figure 2-1 shows a typical relationship of DAM files within a DOS file hierarchy.

Note that a DAM file can continue to grow beyond 440 records. In this case, the records beyond the 440th will be threaded and referenced as if they were records in a SAM file. For example, to access the 445th record of a DAM file, the file system would go to the 440th record directly and seek through the remaining five records sequentially. For an example of how to create a DAM file, refer to Appendix H.

File Records

All files on Prime DOS disks are stored in fixed length 448-word records, chained together by forward and backward pointers. The first eight words of a record is the record header. Specific content of record headers is discussed in Appendix A. After the record header, all remaining words within the record may be used to store ASCII character pairs or 16-bit words.

File Contents

A file is a series of records of the type described above, with the distinction that the first record in such a chain is reached from a pointer within a User File Directory or an entry in a Segment Directory.

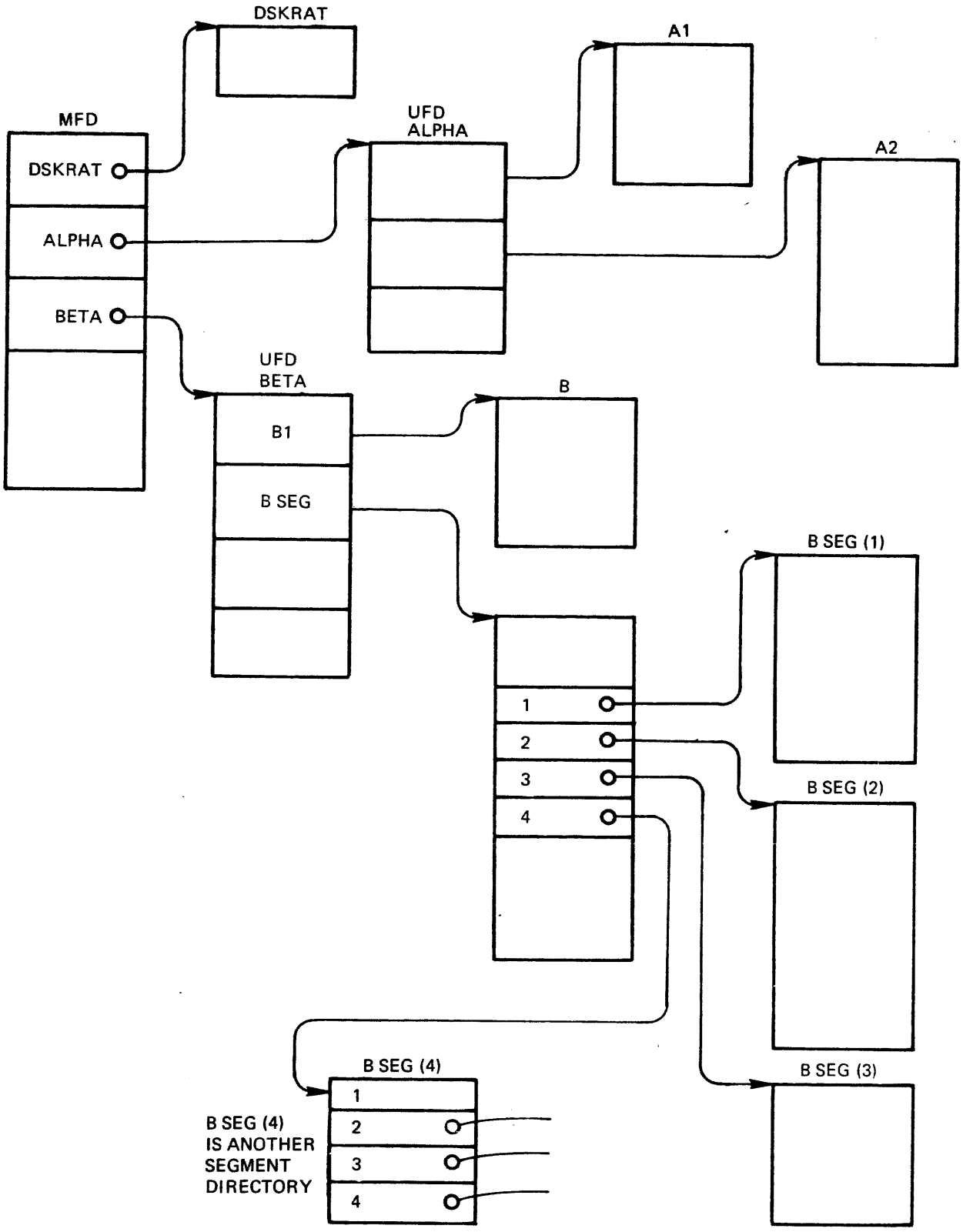


Figure 2-1. Hypothetical DOS File Hierarchy with SAM and DAM File Structures

Every file contains a series of 16-bit words. The format depends on the type of data in the file and how they were originally entered into the file system. The following types of files are in general use in DOS Systems:

Line Image	ASCII character text, packed two characters per word, as entered from a terminal or from the Prime card reader, paper tape readers, etc.
Line Image Compressed	Same as above, but successive spaces are replaced by a relative horizontal tab character followed by a space count, and lines are terminated by a LINE FEED character.
Object Format (Relocatable Binary)	Block-format object code as generated by the Macro Assembler and FORTRAN Compiler for processing by Linking Loader.
Saved Memory Image	Header block followed by a direct transcription of high-speed memory between limits Starting Address (SA) and Ending Address (EA).
Directories (UFD and Segment)	See Appendix A for Format details.

DIRECTORIES

Directories are specialized files that contain entries that point to files or, in some cases, other directories. Directories are the nodes in the file system tree structure hierarchy, whereas files are the branches. Figure 2-1 illustrates this concept. The types of directory are UFD's and Segment Directory.

Segment Directories may be organized as SAM files or DAM files, depending on the kind of file structure the user wishes to build.

MFD and UFD

Each disk pack (or device, in the case of non-removable media) has one User File Directory called a Master File Directory (MFD) that contains an entry for each User File Directory (UFD) in the MFD. In turn, each UFD contains an entry for every file or directory file in that directory. UFD's and MFD's are accessed in the same way as other files.

Master File Directory (MFD)

Each disk unit contains one MFD file as an index to the first physical record of each UFD in the MFD. The MFD has the same format as any UFD. The first record of the MFD begins at physical record one of the disk. Figure 2-1 shows a chain of pointers extending from the MFD to UFD and Segment Directories, and to a DAM or SAM file.

User File Directory (UFD)

A User File Directory is a file that links DOS filenames to the physical record of a file.

A UFD, in the format shown in Appendix A, is associated with each system user. The UFD header consists of a word count followed by the password. After the header, the UFD contains an entry for every file or directory named by the user. Each entry consists of a filename and a word that contains the address of the first physical record of the file (called the beginning record address or BRA). Currently, each UFD can contain up to 72 entries. The first physical record of each UFD is accessed from a pointer (entry) in a UFD or is accessed from an entry in a Segment Directory. Details of the contents of the UFD Header and entries are given in Appendix A.

Segment Directory

A segment directory is formatted just like a UFD except that instead of several words per entry, there is only one: the pointer to the beginning record of a file. For information on how to create a segment directory, refer to Appendix C.

Disk Record Availability Table (DSKRAT)

DOS maintains a file, formerly always named DSKRAT, that stores the status of every physical record on the disk. The name of this file may also be the name of the disk (which is referred to as the Packname). Each record is represented by a single binary bit; a "1" means the record is available, and a "0" means it is in use. On a typical DOS disk, the DSKRAT file is allocated one record. The DSKRAT file is maintained as a file on the disk, starting at physical record 2. The format of DSKRAT is shown in Appendix A.

Disk Organization

DOS supports the use of all the Prime disk options. (Refer to Section 1.) DOS also permits the user to write a programming package that supports blocked, random access device (ISAM). Multiple disks are organized so that every fixed disk and every removable disk is a self-consistent volume with its own bootstrap, DSKRAT and MFD.

Logical record zero is cylinder zero, head zero, sector zero on all options except the dual (fixed and removable) device which has two logical zeros -- one on a fixed disk (head 2) and one on the removable disk (head 0).

File Units

When a disk file is made active for reading or writing, to hold one or two disk records at a time, it must be assigned one buffer in high-speed memory if the file is a SAM file, or two buffers if the file is a DAM file. The buffer, plus associated pointers and status indicators, serves as an access port for the exchange of data between the disk file and the active program. A user generally is concerned with file units; he is not aware of a buffer, except when DOS runs out of memory or overwrites a user program. One file at a time can be assigned to each unit; therefore, a maximum of 15 files can be active (open) at any one time. However, DAM files use two buffers, so that a maximum of 12 DAM files may be open at any one time. The files may be open on several different logical disk units at once. Under DOS/VM, no space is used in the user's address space for file buffers.

Opening, Closing File Units

Refer to the discussion on file units, buffers, and opening files in the first part of this section.

Various ways are provided to associate a specific filename (Filename) to one of the memory buffers (Funit numbers). One method is the OPEN command, for example:

OPEN Filename Funit Status

Filename is the name of a file listed in the UFD to which the user is currently attached; Funit is a DOS file unit number ('1-'17), and Status is 1 for reading, 2 for writing, etc. Note, the character ' is used to denote an octal number. (For full information, see Section 4 "OPEN".) In response to this command, DOS selects an unassigned buffer area, assigns one or two buffers the specified Funit number, and uses it as the data buffer when reading from or writing to the named file. Whether one buffer or two buffers are assigned depends on whether Filename specifies a SAM file or a DAM file. The file is then said to be open. The 448-word memory buffers are allocated downward starting from the beginning of DOS itself. DOS associates a Funit number to the highest unassigned block when a file is opened. From the terminal, the user can open files with the OPEN, BINARY, INPUT and LISTING commands, and close them with the CLOSE command. The command INPUT opens unit 1 for reading (for example, to provide a source input file to the Assembler or Compiler). The BINARY command opens unit 3 for writing (of the object output), and LISTING opens unit 2 for writing (of the listing file). The OPEN command allows a user to assign a file to a unit of his choice and specify the activity - reading, writing, or both. File units 1 to 15 may be specified by the user.

Unit 16 should not be opened by the user; it is used by DOS for reading and writing of system files such as DSKRAT and user file directories. Under DOS/VM, unit 16 may be opened as it is not used by DOS/VM.

When the user is communicating with the file structure through one of the standard Prime translator or utility programs, he refers to files by name only. DOS, or the program itself, handles the details of opening or closing files and assigning file units. For example, the user can enter an external command such as ED FILE1, which loads and starts the text editor and takes care of the details of assigning the file FILE1 to an available unit for reading or writing.

Because open files are subject to alteration (deliberate or accidental), the user must keep files closed except when they are being accessed. The CLOSE ALL command returns all open file units to a closed and initialized state.

File-Handling Subroutines

All file handling is done by a collection of special subroutines, some internal to DOS or DOS/VM, and others available as library routines. These routines are used in common by DOS and all Prime system software for simplified and uniform file handling. In addition, they can be called from FORTRAN or assembly-language user programs. The principal routines are:

ATTACH	Attaches user to a specified UFD or device.
GETERR	Moves n words from the system error vector ERRVEC into a specified array.
GINFO	Moves n words of information about DOS (or DOS/VM) into a special array.
PRWFIL	Reads 16-bit words from a specified file unit to high speed memory and writes 16-bits words from memory to a specified file unit. (For details, see Section 5.)
RESTOR	Restores to memory an executable program previously filed by a SAVE operation.
RESUME	Restores to memory and starts an executable program previously filed by a SAVE command.
SAVE	Saves a section of high-speed memory as a named file. High and low address limits, the start-execution address, and other key parameters are saved with the program.
SEARCH	Assigns a named file to a file unit and opens the file for reading and writing.

The ATTACH, RESTORE, RESUME, and SAVE routines have exactly the same functions as the commands of the same name. These and other file and character handling subroutines are described in detail in Section 5.

All of the file handling subroutines called by the user are loaded with the user's program when the FORTRAN library is loaded. Most of these subroutines are interlude subroutines which issue supervisor calls to DOS or DOS/VM. The appropriate subroutine in DOS then executes the appropriate file operation.

File Handling in User Programs

The subroutines (refer to Section 5) simplify communication between the DOS or DOS/VM file structure and user programs. In FORTRAN programs, for example, the symbolic device unit numbers in formatted READ and WRITE statements can be associated with DOS file units. The following default assignments are set up by the compiler:

FORTRAN Unit (u)	File Unit (Funit)
5	1
6	2
7	3
8	4
9	5
10	6
11	7
12	8
13	9
14	10
15	11
16	12
17	13
18	14
19	15
20	16

Example: to write a record to file unit 1 (FORTRAN unit 5), the user could enter the command OPEN Filename 1 2. The OPEN command associates the file Filename with the file unit 1 and opens the file for writing (code 2). During subsequent execution of a program containing a formatted WRITE statement such as:

```
WRITE (5,10) LINE
```

the contents of array LINE are written as one record to the FORTRAN unit 5 (file unit 1) according to FORMAT statement 10.

At the program level, a Filename and Funit number can be associated by the DOS subroutine SEARCH, that has the form:

```
CALL SEARCH  (2, 6HTEXTbb, 1, $50)
```

to open the file named TEXT on Funit 1 for writing. Besides maintaining the file directories, SEARCH also initializes the DOS data base when a file is opened and updates it when the file is closed.

Users normally call the IOCS subroutine CONTRL to open or close a file read or written by FORTRAN read or write statements. (See the Library manual.) The appropriate call that replaces the call to SEARCH is:

```
CALL CONTRL (2, 6HTEXTbb, 5, $50)
```

Startup

When DOS is loaded and started, it prints the message OK: on the terminal as a cue that it is ready to receive commands. The first command a DOS user enters must be a STARTUP. This command assigns logical unit numbers to the physical disk drives in the particular system. The STARTUP command determines which disk surface is accessed for MFD and the other command functions, and determines the order that DOS searches disk surfaces for UFD's. Use of the STARTUP command is discussed in greater detail in Section 3.

Attaching to a UFD

To access files or use DOS utility functions, the user must be attached to a UFD. Typically, during program development, each user attaches to a UFD reserved for program files with the ATTACH command. For further information, refer to Section 4. Within executable programs, the user can attach to other UFD's, for example, to access data or to call subroutines. At the program level, this is accomplished by the subroutine ATTACH, described previously. For further information on the ATTACH subroutine, refer to Section 5.

DOS/VM File Access Control

DOS/VM gives a user (owner) the ability to open their file directories to other users with restricted rights to the owner's files. Specifically, the "owner" of a file directory can declare, on a per-file-basis, the access rights a "nonowner" has over each of the owners files. These rights are separated into three categories:

- Read Access (includes execute access)
- Write Access (includes over-write and append)
- Delete/Truncate rights

The owner of a UFD can establish two passwords for access to any file in the UFD; the owner password and the nonowner password. The owner password is required by owner to obtain owner privileges. The nonowner password (if any) is required to obtain nonowner privileges.

The command:

```
PASSWD   Owner-Password   Nonowner-Password
```

replaces the existing passwords in the UFD with a new owner-password and a nonowner-password. This command must be given by the owner

while attached to the UFD. A nonowner cannot give this command. The command:

PROTECT Filename Okey Nkey

replaces the existing protection keys on Filename in the current UFD with the owner (Okey) and nonowner (Nkey) protection keys. Valid numbers for these keys are:

0	No access allowed
1	Read access only
2	Write access only
3	Read and Write access
4	Delete/Truncate only
5	Delete/Truncate and Read
6	Delete/Truncate and Write
7	All access allowed (Read/Write/Delete/Truncate)

The owner can restrict his own access to a file by the mechanism. This can be very useful to prevent accidental deletion or overwriting by an owner of an important file. A nonowner cannot give the PROTECT command.

A user obtains owner status to a UFD by attaching to the UFD giving its name and owner password in the ATTACH command. (Refer to Section 4). A user obtains nonowner status to a UFD by giving its name and nonowner password in the ATTACH command.

A user can find out his owner status through the LISTF command. LISTF types the name of the current UFD, its logical device, O, if the user is an owner, or N if the user is a nonowner. LISTF then types the names of all files in the current UFD. An owner can find out the protection keys on all files in the current UFD through use of the FUTIL LISTF command. (Refer to Section 4).

Other Features of File Access

The owner/nonowner status is updated on every ATTACH and separately maintained for the current UFD and home-UFD.

A user's privileges to files under a segment directory are the same as his privileges to the segment directory. Attaching to UFD under a segment directory establishes new privileges for files under it.

The protection keys of a newly created file are:

owner has all rights

nonowner has none

The passwords of a newly created UFD are:

owner password is Blank

nonowner password is Zero (any password will match)

A nonowner cannot create a new file in a UFD, or give the CNAME, PASSWD or PROTECT commands.

Furthermore, a nonowner cannot open his current UFD for reading or writing. (Refer to Section 5).

In the context of file access control, the MFD has all the features of a UFD.

If file access is violated, the error message is:

NO RIGHT

DOS File Access Control

The stand alone DOS operating system does not have file access control over individual files, but it is compatible to a degree with DOS/VM. Under DOS, a user cannot obtain access to a UFD by Attaching using the nonowner password. If the owner password has been given, the Attach is successful but subsequent access to files in the directory is not checked. Files created under DOS are generated with the same protection keys as under DOS/VM. The passwords of a newly created UFD are the same as under DOS/VM.

Commands

DOS commands fall into two major categories: the internal commands (implemented by subroutines that are memory-resident as a part of DOS) and external commands (executed by programs saved as disk files in the command UFD, CMDNC0).

On receiving a command at the system terminal, DOS checks whether it is an internal command, and if so, executes it immediately. Otherwise, DOS looks in the command directory of logical disk unit 0 for a file of that name. If the file is found, DOS RESUMEs the file (loads it into memory and starts execution). All files in the command directory are SAVEd memory image files, ready for execution. Most are set up to return automatically to DOS when their function is complete or errors occur. The command line that caused the execution of the saved program is retained and may be referenced by the program to obtain parameters, options, and filenames. To add new external commands, the user simply files a memory image program (SAVEd file) under the command directory UFD (CMDNC0). Memory image files may also be kept in other directories and executed by the RESUME command.

With the aid of the DOS subroutine PRWFIL, the user can bypass formatted FORTRAN I/O and write directly from memory arrays to disk files, as in:

```
CALL PRWFIL  (1,1 PTEXT, 36, 0, 0)
```

This statement reads 36 words from the file associated with Funit 1 to memory array TEXT, where PTEXT is a pointer to the beginning of array TEXT. PTEXT may be set up by a call to the FORTRAN function LOC. The statement to set up PTEXT would be:

```
PTEXT = LOC (TEXT)
```

File Access Methods

Under DOS, the means of file access is the Sequential Access Method (SAM) or the Direct Access Method (DAM). With both methods, the file appears as a linear array of words indexed by a current position pointer. The user may read or write a number of words beginning at the pointer, which is advanced as the data is transferred. A file I/O module service call (PRWFIL) provides the ability to position the pointer anywhere within an open file. File data can be transferred anywhere in the addressing range (up to the full 64K). When a file is closed and re-opened, the pointer is automatically returned to the beginning of the file. The pointer can be controlled by the FORTRAN REWIND statement, also it can be controlled by PRWFIL positioning.

With the DAM method of access, the file appears to be a linear array of words also, but this method has faster access times in positioning commands. DOS and DOS/VM keep an index that allows for positioning of the first 440 disk records of a file.

DOS/USER INTERACTION

Loading and Initializing DOS

The DOS monitor is a saved-memory-image file under the UFD named DOS. It must be loaded into the high-speed memory with the aid of a bootstrap loading program. The bootstrap is loaded on the devices available under control of CPU microcode. A system with full disk bootstrap microcode can load DOS directly from the master disk through the panel LOAD function. Other configurations may require a key-in loader and paper tape bootstrap. For information on this and other operating procedures, see Section 3.

DOS and DOS/VM internal and external commands are described in Section 4. Prime system programs (compiler, assembler, editor, etc.) requiring detailed operation instructions are described in the pertinent manuals referenced and summarized in part in the appendices in this manual.

Command Files

As an alternative to entering commands one at a time at the terminal, the user can transfer control to a command file by the command: COMINPUT. This command switches command input control from the terminal to the specified file. All subsequent commands are read from the file. One can assign any unit for the COMINPUT file. This means that command files may call other command files. For detailed information on the COMINPUT command, refer to Section 4.

Command files are primarily useful for performing a complicated series of commands repeatedly, such as loading an extensive system in the debugging stages when it is necessary to recompile and reload often. Command files are also useful in system building when many files must be assembled, concatenated, loaded, etc. (for example, configuring an RTOS system or generating library files).

Saving Programs

After compiling or assembling a program and loading the object version along with library routines, the user can save the program development efforts by the SAVE command:

SAVE Filename SA EA PC A B X Keys

This command string assigns a file, Filename, in the current UFD, saves the memory image between limits SA (starting address) and EA (ending address) and enters other parameters into the header block:

PC	Program Counter setting (address at which to start program execution)
A	A Register value
B	B Register value
X	X (Index) Register value
Keys	Status keys (as processed by INK, OTK instructions)

The preferred way to save newly loaded programs is to use the loader's SAVE command. Refer to the Program Development Software User Guide for details.

When a program is restored to operation by a RESTORE or RESUME command, these parameters are retrieved with the file and replaced in the registers from which they were obtained. These are the RVEC parameters, described in more detail in Section 4.

A program saved in the command UFD (CMDNC0, for example) can be invoked by name like any other external DOS command. All standard Prime translator and utility programs are supplied in this form.

File Maintenance (FIXRAT)

To give the user an efficient and thorough way to check the integrity of data on a DOS disk, DOS provides a file maintenance program, FIXRAT, filed under the command directory, CMDNC0. When FIXRAT is invoked as an external command, it checks for self-consistency in the structure of pointers in every record, file, and directory on the disk. If there are breaks in the continuity of double-strung pointers, discrepancies between the DSKRAT file and the reconstructed record availability map, or other error conditions, FIXRAT prints appropriate error messages. FIXRAT asks the user to specify whether or not to take specific steps to repair a damaged file structure on a specified logical disk unit. For details and examples, refer to FIXRAT description in Section 4 and Appendix E.

DOS MEMORY USAGE

DOS occupies approximately nine sectors at the top of the available memory plus a variable number of 448-word file unit buffers. Figure 2-2 shows a typical memory map for a system with 16K of high-speed memory. DOS/VM takes no part of the user's virtual address space.

Floating DOS

Three versions of DOS are supplied in the UFD, DOS. These versions load DOS starting at locations '27000, '47000 and '67000. The bootstrap program selects the version of DOS that is nearest to the top of high-speed memory. The values in Figure 2-2 may be increased accordingly to 20,000 locations and 40,000 to give an approximation of memory allocation for 32K and 64K systems. If desired, a particular DOS may be selected by manually setting the sense switches (refer to "BOOT" in Section 3).

Other Locations

Sector 0 is reserved. Locations 0 through '177 are dedicated to the Prime CPU's register file and the vector locations for interrupt and DMX. Locations '200 and above are used to store the cross-sector indirect address links generated by the loader, but the user may, with caution, use locations in this area.

For 16K configurations, locations '1000 through '17777 may be used without restrictions unless a symbol table is present. The high end limit is usually determined by the start of the loader, which may be memory-resident during loading. However, FORTRAN common may set the upper limit if it extends below '020000.

FORTRAN common overlaps part of the area that can be occupied by DOS file unit buffers. Up to three SAM file units can be open at a time without the risk of writing over part of common.

Default location of FORTRAN common is the top of the loader extending down in memory. There are two implications:

1. common cannot be loaded with 'BLOCK DATA' statements,

2. only three disk units may be open at any one time. (DOS restriction only.)

This problem can be avoided through use of the loader's COMMON command, which permits the moving of common to a user specified location.

If a program is to be debugged with the aid of Trace and Patch (TAP), only two files can be open at a time. However, TAP can relocate itself elsewhere in memory if this is a problem. For information on TAP, refer to the Program Development Software User Guide.

Memory areas occupied by the DOS file unit buffers are listed in the following table.

Table 2-1

Memory Areas and DOS File Units

Number Open File Units	Top of Available Memory		
	16K System	24K System	32K System
0	'26777	'46777	'66777
1	'26077	'46077	'66077
2	'25177	'45177	'65177
3	'24277	'44277	'64277
4	'23377	'43377	'63377
5	'22477	'42477	'62477
6	'21577	'41577	'61577
7	'20677	'40677	'60677
8	'17777	'37777	'57777
9	'17077	'37077	'57077
10	'16177	'36177	'56177
11	'15257	'135277	'55277
12	'14377	'34377	'54377
13	'13477	'33477	'53477
14	'12577	'32577	'52577
15	'11677	'31677	'51677
16	'10777	'30777	'50777

Notes:

1. 448 words for each SAM file open.

2. 896 words for each DAM file open.

3. There is a difference of octal 700 as the number of open file units increases. Users can estimate the correct figures if they know how much memory is available and the number of open file units.

4. The above figures assume only SAM files. Up to date information may be gathered by the use of the STATUS and GINFO commands.

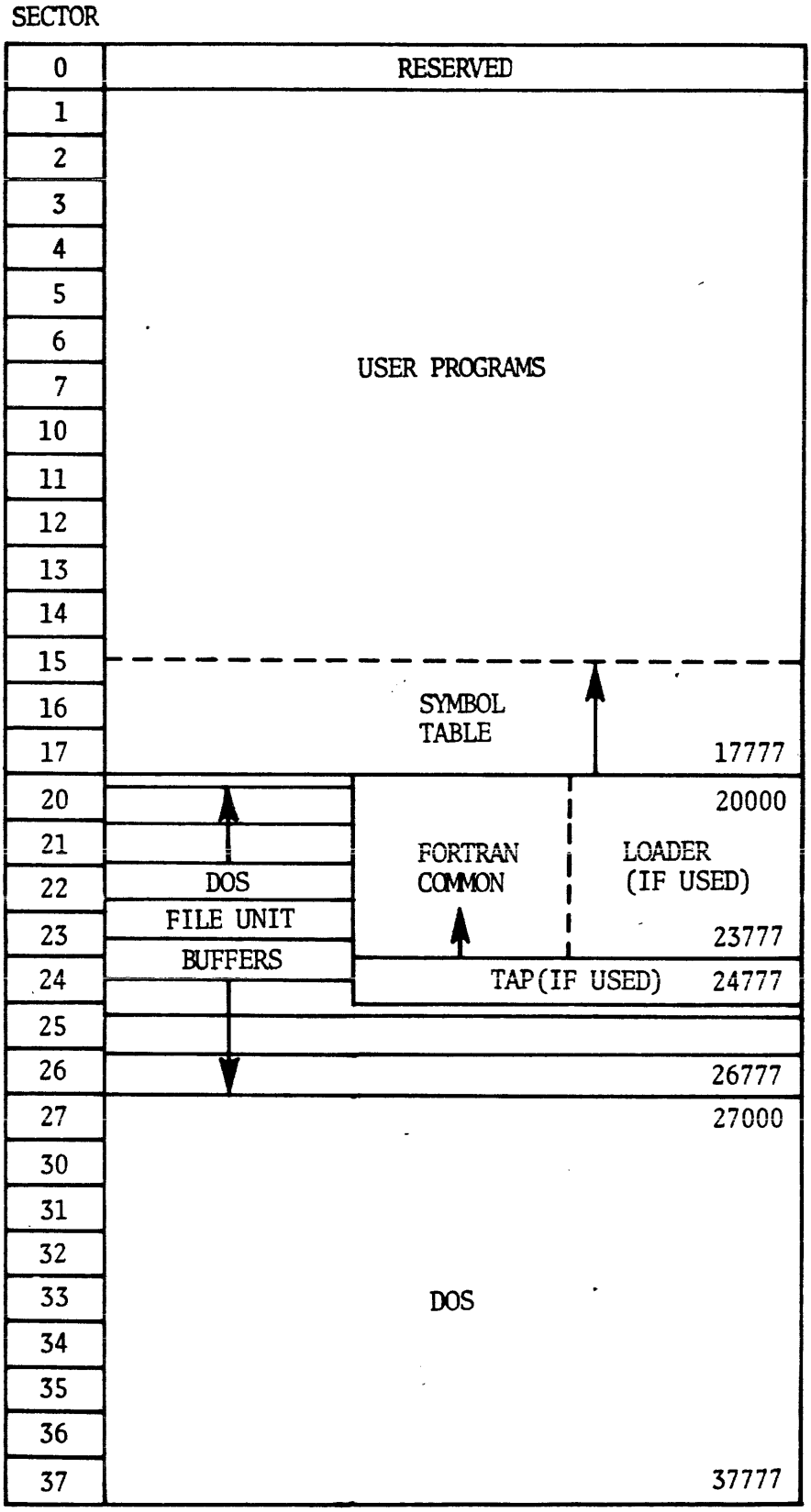


Figure 2-2. Memory Allocation in 16K System

SECTION 3

DOS OPERATION AND SYSTEM MAINTENANCE

This section describes how to load and start DOS, summarizes the essential operator tasks during an operating session, and describes procedures for data backup and system shutdown.

BOOTSTRAPS AND DISK BUILDING

The process of converting a "cold-iron" computer to a useful tool begins with getting that first program into memory (bootstrapping). Then the first program can read other programs and data into memory.

A parallel step in bootstrapping DOS and DOS/VM into memory is the preparation of the mass storage media.

LOADING DOS FROM MASTER DISK

DOS and DOS/VM are usually supplied in the form of a master disk cartridge to be installed as the removable surface of a Pertec (or equivalent) moving-head disk drive. The master disk includes DOS, the command UFD, (CMDNC0), DOS/VM (filed in CMDNC0), and library files (filed under UFD LIB). For information on the library files supplied on the master disk, see Section 5 and the Library Manual.

NOTE: Special instructions accompany versions of DOS supplied on a fixed-head disk drive or other media.

STARTING EQUIPMENT

1. Turn on power to the equipment in the system in the following order:

    a. CPU

    b. Fixed Head Disk Drive (if present)

    c. Moving Head Disk Drive

    d. ASR, high speed tape equipment, etc.

NOTE: Refer to the Prime Operator's Guide for operating instructions for the computer panel, the terminal, and the high-speed paper tape reader/punch. Operating procedures for the various Prime disk drives appear in Appendix K.

2. Install the DOS master disk cartridge in the moving-head disk drive. Press RUN/STOP to start drive. Wait for READY light.

   CAUTION: Place the removable surface in WRITE-PROTECT mode to ensure that accidental operating errors cannot write over the DOS software.

BOOTING DOS

DOS must be transferred from the master disk to CPU memory, where it will take control of subsequent events. In order to do this, the CPU must be loaded ('booted') with the DOS BOOT program. DOS BOOT may be supplied as a self-loading paper tape to be read by the panel LOAD function or a key-in loader. However, if the CPU is equipped with microcode for direct booting of DOS from the disk, no paper tape is needed. The various procedures for booting DOS, according to the types of equipment and LOAD microcode, appear in Appendix B.

BOOT Operation

BOOT performs the following functions:

Cleans up parity, non-destructively, throughout memory.

Sizes available memory.

Requests, from the operator, which device to boot from.

Attaches to the MFD on that device.

Attaches to the UFD, DOS.

Depending upon memory size and/or sense switches, reads *DOS16, *DOS24, or *DOS32 into memory and starts the version of DOS read in.

There are three possible outcomes of a boot operation: (1) a successful boot, in which case DOS takes control; (2) a detected error, in which case the boot returns and again requests, from the operator, which device to boot from; or (3) an undetected error, such as non-existent device; in the latter case, the boot pauses.

When started, the boot prints:

```
PHYSICAL DEV =
```

The operator response must be the physical unit number as defined for the DOS STARTUP command. Possible unit numbers are defined in Table 3-1.

Sense Switches	1-10:	See the following paragraphs
	11-13:	Type = 0 option 4000 MHD (moving head disk)
		1 option 4000 FHD (fixed head disk)
		2 option 4300 (diskette)
		3 option 4002 9 sector MHD
		4 option 4002 64 sector MHD
		5 option 4002 32 sector MHD
		6 option 4001/4002 20 surface MHD
		7 reserved

Sense Switches	14-16:	Unit = physical drive number. For option 4000 and option 4002 MHD's, EVEN units are upper platters and ODD units are lower platters of the drive number in bits 14 and 15.

The version of DOS (*DOS16, *DOS24, *DOS32) that is read in is determined by either memory size or Sense Switches 1, 2, and 3 in the following manner:

If all sense switches are reset, the highest memory DOS that fits in available memory is read. For booting DOS from diskette, the user must specify *DOS16 by setting the sense switches as described below:

If any of these sense switches are set, they are treated as the most significant bits of the high address of memory + 1 as shown in the following example:

Sense Switch Address	DOS
0	highest that will fit
'20000	error
'30000	*DOS16
'40000	*DOS16
'50000	*DOS24
'60000	*DOS24
'70000	*DOS32
'1X0000 (X=don't care)	*DOS32

Once the boot has been successfully brought into memory by the control panel boot, it can be re-executed by pressing MASTER CLEAR and STARTing at '1000. If a status error is detected on the device, BOOT restarts automatically. Both the option 4000 and diskette drivers wait for the device to come ready, but the option 4002 driver treats device not ready as a status error.

Building BOOT

The BOOT program is stored as a normal DOS SAVE file on a normal DOS format record (=0). Consequently, physical record 0 contains first an eight-word DOS record header, second an 8-word DOS SAVE file header, and finally the BOOT program itself. The eight-word DOS record header is eliminated by reading the record starting at '770 but starting execution at '1000; the first word of the eight-word SAVE file header is preserved.

The SAVE file header is as follows:

```
word 0 = start address (SA( (must = '3011)
     1 = end address (EA) (must be correct)
     2 = program counter (PC)
     3 = A-register
     4 = B-register
     5 = X-register
     6 = Keys
     7 = unavailable
    10 = unavailable
```

Since execution starts at '1000, the start address must be '3011 which is also a MJP '1011 (the boot is guaranteed to be executed in 16S mode either by MASTER CLEAR or the control panel boot). The boot executes in sector '1000 and so must be stored there (at '1011), then later moved (by means of PSD or TAP) to '3011 and SAVEd there (the end address must be correct). Because BOOT can never be executed as a user or system terminal command (it cannot execute in sector 3), the PC, A-register, B-register, X-register, and Keys are available as constants to be used by BOOT. They are defined as follows:

```
PC    = '160000     mask for Sense Switches 1, 2 and 3
A     = '110        master clear default control register
B     = '27         SOC master clear default control register 1
X     = '74000      SOC master clear default control register 2
Keys  = '260        ASCII 0
```

Once a BOOT has been placed on a disk, it can be copied to another disk with the following command sequence:

```
A     MFD     XXXXXX    Lunit
RESTORE    BOOT
A     MFD     XXXXXX    Lunit
SAVE    BOOT
```

Since the SAVE parameters can be specified, the A, B, and X registers can be set to other than master clear defaults to allow other types of system terminals. When changing these values, care must be taken not to change any of the others.

The command file C←BOOT on UFD=FILAID produces a file named **BOOT with SAVE parameters defined as above. The following command sequence produces a file (which must be named BOOT) suitable for placing on a disk (User input is underlined).

```
OK:  RESTORE  **BOOT
OK:  PM
SA, EA, P, A, B, X, K=
001011  high  031721  031723  000110  000027  75000
OK:  PSD
$ C 1011  high  3011
$ V 1011  high  3011
$ Q
OK:  SAVE  *BOOT  3011  (high + '2000)  31721  31723  11027  74000
OK:
```

NOTE: high is the EA and varies depending on the revision or the version of BOOT read. It must be correct.

Direct Panel Load of DOS from Disk

Use these procedures if the CPU contains microcode for direct booting from the disk:

1. Set Sense Switches.

   To indicate what disk to boot from, set the rightmost sense switches as follows:

Physical Disk Number (octal)	Sense Switches (octal)
0	XX0004
1	XX0044
10	XX0003
20	XX0006
30	XX0014
31	XX0054
40	XX0013
XXX050	XX0014
XXX051	XX0054
XXX250	XX0034
XXX251	XX0074

NOTES: Physical disk number is associated with disks as shown in Table 3-1. The X's represent don't care octal digits.

A particular DOS (DOS at 16K, 24K, 32K) can be selected by setting the leftmost sense switch as follows:

DOS	Sense Switches
highest DOS that will fit in memory	00XXXX
*DOS16	04XXXX
*DOS24	06XXXX
*DOS32	10XXXX

2. Turn rotary switch to STOP/STEP and press MASTER CLEAR. Turn rotary switch to LOAD and press START. The control panel boot should read in record 0 from the disk containing the DOS BOOT, transfer control to it, and print: PHYSICAL DEVICE = .

3. Refer to the next major topic "LOADING OF DOS".

DOS BOOT Tape, Panel LOAD

Use this procedure if the CPU contains microcode for a panel LOAD from a paper tape device - the ASR Teletype (low-speed reader) or the HSR (high-speed reader).

1. Mount the DOS BOOT tape on the available device (ASR or HSR).

2. Set sense switches as follows:

Device	Sense Switches
ASR	'000001
HSR	'000002

3. Turn rotary switch to STOP/STEP, press MASTER CLEAR, turn to RUN and press START. DOS BOOT tape loads from tape and prints: PHYSICAL DEVICE = .

4. Refer to the next major topic, "LOADING OF DOS".

DOS BOOT Tape, Key-in Loader

1. Prepare the CPU to read self-loading paper tapes by entering the appropriate key-in loader and second-level bootstrap tape as described in the Prime CPU Operator's Guide.

2. Mount the DOS BOOT paper tape on the available device (ASR or HSR).

3. After the second-level bootstrap is loaded, the CPU is ready to read a self-loading tape from the available device. Turn the CPU rotary switch to RUN and press START. The DOS BOOT tape reads into memory and types: PHYSICAL DEVICE =

4. Refer to the next major topic, "LOADING OF DOS".

LOADING OF DOS

In all cases, once DOS has been booted into memory, the system terminal prints:

PHYSICAL DEVICE =

Physical Device codes are assigned as follows:

Device (disk types)	Codes	Controller
Moving head disks	0 - 7	4000
Fixed head disks	10 - 11	4000
Floppy disks	20 - 27	4300
Moving head disks	30 - 37	4002
Fixed head disks	40	4002
Moving head disks (32 Sector/Track)	50 - 57	4002
30 million word disks	5050→5056 (even numbers only)	4001/4002
30 million word disks	5250→5256 (even numbers only)	4001/4002

For further information refer to Table 3-1.

The user types the physical device code at the terminal and presses CARRIAGE RETURN. Then DOS/BOOT loads and starts DOS, which prints a label line such as:

```
DOS REV. 5.0 5/10/74 (AT  070000)
```

When DOS is ready to receive commands from the user it prints:

```
OK:
```

at the system terminal. However, if an error of some kind is discovered, BOOT retypes the message:

```
PHYSICAL DEVICE =
```

Errors that result in this message are the following:

1. disk does not exist on the system or is not turned on, spinning and ready;

2. unrecovered disk error attempting to read *DOS16, *DOS24, or *DOS32 from the disk;

3. disk does not have an MFD, the UFD DOS within the MFD or, the files *DOS16, *DOS24, or *DOS32 in UFD DOS;

4. the MFD, DOS, *DOS16, *DOS24, or *DOS32 has a bad structure.

STARTUP OF DOS

When DOS prints the message OK:, the DOS system is loaded into memory and ready to receive commands from the user. The first command to be entered must be a STARTUP command that connects a physical disk to logical disk unit 0. DOS expects to find all its command files on unit 0. For detailed information on the STARTUP command, refer to Section 4. The usual initial STARTUP is:

```
OK:  STARTUP  0  1
```

This assigns the removable surface of the moving head disk (the master disk pack) as logical unit 0 and the fixed surface as logical 1.

Table 3-1. Physical Device Assignments

Physical Disk Drive Number	Controller Option	Description	Sectors/Track
0	4000	Removable surface of first MH (moving head) disk drive (upper surface)	8
1	4000	Fixed surface of first MH disk drive (lower surface)	8
2	4000	Removable surface of second MH disk drive (upper surface)	8
3	4000	Fixed surface of second MH disk drive (lower surface)	8
4	4000	Removable surface of third MH disk drive (upper surface)	8
5	4000	Fixed surface of third MH disk drive (lower surface)	8
6	4000	Removable surface of fourth MH disk drive (upper surface)	8
7	4000	Fixed surface of fourth MH disk drive (lower surface)	8
10	4000	First fixed head disk drive	8
20	4300	First floppy disk drive	4
21	4300	Second floppy disk drive	4
22	4300	Third floppy disk drive	4
23	4300	Fourth floppy disk drive	4
24	4300	Fifth floppy disk drive	4
25	4300	Sixth floppy disk drive	4
26	4300	Seventh floppy disk drive	4
27	4300	Eighth floppy disk drive	4

Table 3-1. Physical Device Assignments (Cont)

Physical Disk Drive Number	Controller Option	Description	Sectors/Track
30	4002	Removable surface of first MH (moving head) disk drive (upper surface)	8
31	4002	Fixed surface of first MH disk drive (lower surface)	8
32	4002	Removable surface of second MH disk drive (upper surface)	8
33	4002	Fixed surface of second MH disk drive (lower surface)	8
34	4002	Removable surface of third MH disk drive (upper surface)	8
35	4002	Fixed surface of third MH disk drive (lower surface)	8
36	4002	Removable surface of fourth MH disk drive (upper surface)	8
37	4002	Fixed surface of fourth MH disk drive (lower surface)	8
40	4002	Fixed head disk drive	64
50	4002	Removable surface of first MH (moving head) disk drive (upper surface	32
51	4002	Fixed surface of first MH disk drive (lower surface)	32
52	4002	Removable surface of second MH disk drive (upper surface)	32
53	4002	Fixed surface of second MH disk drive (lower surface)	32
54	4002	Removable surface of third MH disk drive (upper surface)	32
55	4002	Fixed surface of third MH disk drive (lower surface)	32

Table 3-1. Physical Device Assignments (Cont)

Physical Disk Drive Number	Controller Option	Description	Sectors/Track
56	4002	Removable surface of fourth MH disk drive (upper surface)	32
57	4002	Fixed surface of fourth MH disk drive (lower surface)	32
5050	4001/4002	First 30-million word moving head disk drive (controller address = 21)	32
5052	4001/4002	Second 30-million word moving head disk drive (controller address = 21)	32
5054	4001/4002	Third 30-million word moving head disk drive (controller address = 21)	32
5056	4001/4002	Fourth 30-million word moving head disk drive (controller address = 21)	32
5250	4001/4002	First 30-million word moving head disk drive (controller address = 23)	32
5252	4001/4002	Second 30-million word moving head disk drive (controller address = 23)	32
5254	4001/4002	Third 30-million word moving head disk drive (controller address = 23)	32
5256	4001/4002	Fourth 30-million word moving head disk drive (controller address = 23)	32

NOTES TO TABLE 3-1:

The logical-to-physical assignment depends on the order in which the physical device numbers are listed as parameters in the STARTUP command. The physical device number specified in the Ldisk0 position is assigned as logical disk unit 0, the physical device number specified in the Ldisk1 position is assigned as logical disk unit 1, and so on. Example:

```
STARTUP  2  3  5  7
```

The physical disks are 2, 3, 5, and 7; where:

physical 2 is logical 0, physical 3 is logical 1, physical 5 is logical 3, and physical 7 is logical 4. The number of parameters in STARTUP indicate to DOS the number of logical drives assigned to the system.

CAUTION: When changing disks, a SHUTDN is required. Otherwise, DOS and DOS/VM will use parameters (such as record availability) applicable to the previous disk and the newly replaced disk with possible disastrous results.

The codes shown in Table 3-1 are used in the STARTUP command (refer to Section 4) and the ASSIGN command (refer to Section 4). The codes are also used by the utility commands FIXRAT, MAKE, and COPY (refer to Section 4).

The physical device codes are the same for three- or six-million word disk drivers connected to the controller.

Data Transfer Between Disks

An 8 sector/track disk pack written on a drive connected to the 4000 controller cannot be read on a drive connected to the 4002 controller and vice-versa, because the method of computing hardware checksum written on the pack is different on the two controllers. A special conversion program to convert packs written on one controller to read on the other controller must be written. A 32-sector/track pack cannot be read or written on a drive connected to the 4000 controller. An attempt to read a 32-sector/track pack using physical device numbers for an 8-sector/track pack will fail. Similarly, an attempt to read an 8-sector/track pack using physical device numbers for a 32-sector/track pack will fail. It is important to keep straight the identification of the disk pack. It is suggested that each pack be labeled with the range of physical device numbers appropriate to the disk pack.

Unlike the other disks, only even numbered physical disk drive numbers are allowed for the 30-million word disk drives. There may be up to four drives connected to a type 4001/4002 controller that is

configured to the system. The default device address for a type 4001/4002 controller is 23, and the disk drive numbers associated with this drive are 5250, 5252, 5254, 5256. Similarly, a 4001 or 4002 controller may have a device address of 21, and disk drive numbers associated with this controller are 5050, 5052, 5054 and 5056. A system configuration could have two type 4001/4002 controllers and up to eight 30-million word disks connected.

Partitioning Disks

A user may partition a 30-million word disk into two or more sub-disks, via use of the MAKE commad (refer to Section 4). Each partition of a disk (sub-disk) is treated by the system commands, DOS, DOS/VM, FIXRAT and COPY, as if it were a physically separate disk. Each partition contains its own MFD, DSKRAT, BOOT, CMDNC0, ETC. A partition is defined by a starting head address, relative to head 0 of the disk, and a number of contiguous heads. The minimum partition contains two heads (i.e., three million words). When a partition of a 30-million word disk is present, the physical disk number varies from those shown in Table 3-1. The number of heads is reflected in the second two digits. Tables 3-2 and 3-3 are useful in constructing partitions. (The X's represent don't care octal digits).

The physical disk number defining a partition on a disk that is partitioned is generated by merging the head offset with the number of heads and with the disk device number. For the purposes of forming a physical disk number for a partitioned disk, the physical disk device numbers are considered to be: 50, 52, 54, and 56 for disks 1 to 4 on the type 4001/4002 controller.

Example: The physical disk number for a disk partition having a head offset of 00 and the number of heads of 50 would be calculated as follows: (assume that the device address of the type 4001/4002 controller is 23). First look up at the appropriate numbers in Tables 3-2 and 3-3, then

head offset + number of heads + physical disk device number

00XXXX + XX52XX + 56 = 005256

As another example, consider a disk split into two partitions with the disk being the first disk on the type 4001/4002 controller with device address 21; the first portion has head offset = 0 heads and number of heads = 10. The physical disk for the first partition is:

00XXXX + XX24XX + 50 = 002450

The second partition has head-offset = 10 heads and number-of-heads = 10. The physical disk for the second partition is:

05XXX + XX24XX + 50 = 05250

Physical Device Numbers Usage

The physical device codes described previously in this section are used in the ASSIGN, CONFIG, DISKS, SHUTDN, STARTUP, STATUS, and UNASSIGN commands. These device codes are also used by the utilities FIXRAT, FUTIL, MAKE and COPY.

Table 3-2. Head Offset Definitions

Offset	Physical Disk Numbers
0 heads	00XXXX
1 head	01XXXX
2 heads	02XXXX
4 heads	03XXXX
6 heads	04XXXX
8 heads	05XXXX
10 heads	06XXXX
12 heads	07XXXX
14 heads	10XXXX
16 heads	11XXXX

Table 3-3. Number of Heads Definition

Number of Heads	Type 4001/4002 Controller Address = 23 Physical Disk Number	Type 4001/4002 Controller Address = 21 Physical Disk Number
2 heads (default)	XX02XX	XX00XX
2 heads (explicit)	XX06XX	XX04XX
4 heads	XX12XX	XX10XX
6 heads	XX16XX	XX14XX
8 heads	XX22XX	XX20XX
10 heads	XX26XX	XX24XX
12 heads	XX32XX	XX30XX
14 heads	XX36XX	XX34XX
16 heads	XX42XX	XX40XX
18 heads	XX46XX	XX44XX
20 heads	XX52XX	XX50XX

STARTUP OF DOS/VM

DOS/VM is started from DOS, once DOS is started and running. For details, refer to Section 6. Note that the response of DOS/VM to a valid command is: OK, ("OK" followed by comma, not a colon). This is one indication which of the operating systems has control.

INITIAL OPERATING SESSION

Attaching to UFD

After a STARTUP, the user must attach to a User File Directory in order to execute DOS commands and create or manipulate files. Each master disk provides several blank UFD's named SPARE1, SPARE2, etc., and the user may attach to any of these with an ATTACH command (Section 4). To determine what spare UFD's are available, ATTACH to the MFD and do a LISTF:

```
OK: A MFD XXXXXX
OK: LISTF

UFD=MFD     0

DSKRAT  MFD     BOOT    CMDNCO  LIB     SRCLIB  DIAG    PMA
FORTRN  LDR     BASIC   FLIB1   FLIB2   FLIB3   FLIB4   FLIB5
FLIB6   LIB7    LIB3    IOCS    AIDS    ED      BINED   T&M
DOS     RTOS    DOSVM   RTOSVM  INDEX   SPARE2  SPARE3  SPARE4
SPARE5  SPARE6  SPARE7
```

Note that the MFD has a password, XXXXXX. This is assigned at the time the master disk is prepared, to discourage casual or inadvertent use of this important directory.

CAUTION: Do not attach to MFD for a program development or normal file handling tasks. Be very careful in entering commands while attached to this UFD. If any of the files in this MFD are damaged, the master disk is spoiled.

DISK BUILDING (COPYING MASTER DISK PACK)

Disk building consists of three phases: format the disk; move run files of DOS (*DOS16, *DOS24, *DOS32 as appropriate) onto the UFD DOS; and move any desired external commands onto the UFD CMDNC0 and/or move libraries onto the UFD LIB.

Format of DOS Disk

If a DOS master disk (or any other DOS disk) is available, it can be COPY'd onto the virgin disk.

If no DOS disk is available or an empty disk is desired, the MAKE program can be run. The COPY and MAKE programs are described in Section 4. When a disk is formatted using MAKE, any needed files are then copied from the master disk onto the new disk. If this new disk is to be boot loaded from, then UFD DOS must contain the files *DOS16, *DOS24 and *DOS32. MAKE ensures that an executable and correct BOOT is written onto record 0 of the disk.

Building a DOS Disk from Paper Tape

If no DOS formatted disks are available, one must be created from Prime-supplied paper tapes. All tapes provided are MDL self-loading tapes and are loaded into memory using the control panel boot. The following procedure must be followed:

1. Load BDOSV2 (DOS bootstrap tape) This loads the loader *DOS16 as well as other necessary modules.

   Start at '6765

   After the header is typed and DOS prompts OK:, type the following commands:

   STARTUP (Pdev) where Pdev is a physical device number.

   ```
   ATTACH  DOS
   SAVE  *DOS16  7000  17777  31000  20000  0  0  2000
   ```

2. Boot the new *DOS16 from the disk using either:

   Control panel boot (Sense Switch 2 set)

   Paper tape DOSBOOT (SLT-start at '1000). Set Sense Switch 2 after the tape has been loaded.

3. Any other command can be added by loading it into memory (control panel boot), starting at '30000 (DOS, ATTACHing to CMDNC0 and SAVEing the command). (Refer to Section 4 for a description of these commands.)

4. Use EDB to read any binary files (e.g., FTNLIB)

5. Use ED to read any source files (e.g., DRATIT)

Once the drive is READY, the user can resume DOS operation. A new STARTUP is required; example:

```
OK: STARTUP 1 0
```

This establishes the fixed surface as logical unit 0; all DOS automatic disk activity supporting the assembler, compiler, editor (etc.) uses logical unit 0.

Creating Additional DOS Disks

Every DOS disk must contain a BOOT, a DSKRAT file, an MFD, the command UFD (e.g., CMDNC0), the command programs FIXRAT, COPY, and the UFD's required by the user. The easiest way to convert a blank disk pack to a DOS disk is to run the MAKE program. Refer to the MAKE command description in Section 4. Another method is to copy the active DOS disk from its present location. (For example, from the fixed surface to the removable surface of a moving head disk drive.) This is done as follows:

```
OK: A CMDNC0

OK: COPY                (Copy operation begins).
```

Any number of DOS-compatible disk packs can be made in this way. Of course, much of the available file space on an original master DOS disk is occupied. To make room, the user can delete UFD's or files as required. Only the DOS disk assigned as logical unit 0 needs to have the full set of DOS command files, UFD's, library, etc. On other disks to be used mainly for user's data or program files, the surplus UFD's, and the files within them, can be deleted. To determine which files will provide the most space, run a FIXRAT to observe the number of disk records occupied by each UFD. To delete UFD's, attach to the MFD and enter DELETE commands; example:

```
OK: A MFD XXXXXX

FUTIL

  > TREDEL LIB

  > TREDEL PMA

  > TREDEL FORTRAN

  > QUIT

OK:
```

Alternatively, if the user needs many UFD's, he could use CNAME to change the UFD names, then attach to the UFD's and delete the files within them using FUTIL. See Section 4.

Entering New UFD's

Another method to coin new UFD names is to attach to the MFD and use the CREATE command. For example:

```
OK: A MFD XXXXXX
OK: CREATE NEWUFD
```

The user must attach to a UFD other than the MFD as soon as possible to reduce the likelihood of spoiling any of the MFD files.

Program Development Using DOS

From this point, the user is free to use DOS and its supporting software to create, assemble or compile, load, save, and execute user programs. The internal and external DOS commands are described in Section 4. The appropriate manuals provide detailed information on the Editors, FORTRAN, BASIC. The Macro Assembler, Loader and other programs are described in the Programs Software Development User Guide.

At the end of this section, is an example of the terminal printout resulting from the development of a simple FORTRAN program. The user may study the example and use its procedures as a guide during initial program development efforts.

Recovering from Errors

If an equipment failure or program error causes the CPU to leave DOS control, it is usually possible to return to DOS by starting the CPU at location '30000, '50000, or '70000, depending on the hardware configuration. See the Operators Guide for instructions to restart at these locations.

Installing New External Commands

The user can install his own custom utility or device control programs to be invoked by external command to DOS. One way this is done is by restoring a program from the user's UFD, and then saving it under the command UFD CMDNCO. Assume for example that the user wants to install a cassette recording and playback monitor program to be invoked by the name CASS:

When DOS prints the message, OK:, the first command to be entered must be a STARTUP command that assigns the disk logical unit 0. DOS expects to find all its command and utility files on unit 0. For detailed information on the STARTUP command, refer to Section 4. The usual initial STARTUP is:

```
OK: STARTUP  0  1
```

This assigns the removable surface of the moving head disk (the master disk pack) as logical unit 0, and the fixed surface as logical unit 1.

```
OK: A USER1
OK: REST CSETV1
OK: PM
SA,EA,P,A,B,X,K=
000100 011100 001000 000000 000000 000000 000000
OK: A CMDNC0
OK: SAVE CASS 100 11100 1000 0 0 0

OK: A USER1
OK: CASS
GO
```

(CASS program begins running.)

In this example, the user restores file CSETV1 from his own UFD (USER1), and does a PM to determine the RVEC parameters (discussed in Section 4). He then attaches to CMDNC0 and saves the program under the name CASS, with the same parameters as the original. Thereafter, when he uses the name CASS as an external command, DOS resumes the saved CASS program.

Another way to install a new external command is:

```
OK: A NEWUFD
OK: FUTIL
>   FROM USER1
>   TO CMDNC0
>   COPY CSETV1  CASS
>   QUIT
OK:
```

BACKUP

General

Each installation can develop its own procedures to save copies of files and disks for backup purposes. The techniques are simple. Individual files can be saved on paper tape through the Text or Binary Editors. DOS disks can be copied to removable disk packs by careful use of the COPY command described in Section 4.

To copy the fixed disk surface to a removable backup pack (not the master disk); first, do a FIXRAT of the fixed surface and do not proceed until an error-free FIXRAT pass is obtained. Then, do the appropriate STARTUP, attach to some UFD, and use the COPY command.

CAUTION

Before entering the COPY command, make sure the FROM surface is in WRITE PROTECT mode.

Use of MAGSAV

If magnetic tape devices are present, files are copied to them by the MAGSAV command. This is the most convenient and simplest method of implementing system backup. Also the tapes produced by MAGSAV can be read back into the system configuration by use of the MAGRST command.

SHUTDOWN

Before terminating an operating session with DOS by loading another operating system or turning off power, enter the following commands:

OK: FIXRAT (This step is optional)

OK: SHUTDN

See Section 4 for details. The SHUTDN command writes to disk DOS data that is buffered in memory.

Changing Disk Packs

To change removable disk packs in the moving-head disk drive, shut down DOS as above. Then power down the disk drive and replace the pack. If DOS/VM is the system in control, SHUT DOWN the physical disk drive with the DOS/VM SHUTDN command then power down the disk drive. Restart the disk drive; when the unit is READY, give a STARTUP command appropriate to the operation with the new pack and resume typing commands.

Using FIXRAT

The external command FIXRAT loads and starts the DOS maintenance program that checks the file integrity on any disk pack. FIXRAT fully supports nested UFD's and nested Segment Directories. Section 4 gives further information on the FIXRAT command, and Appendix E also describes all features of FIXRAT.

FIXRAT must be run whenever there is reason to expect that the file structure is damaged - for example, if a program being debugged runs wild and writes over part of DOS. Until the user gains experience with the system, he should run FIXRAT at the close of every operating session.

The suggested procedure is to maintain a DOS disk pack and to run FIXRAT every morning, and if no error occurs, to copy the disk pack onto a daily backup disk pack. If any files are truncated or deleted, these may be copied from the daily backup disk pack, if they were stored previously on the daily backup disk pack.

Turning Power Off

After a shutdown, the CPU can be used to run other software or power can be turned off. The following power-down order is recommended:

1. Moving-Head Disk
2. ASR, high-speed tape unit, and other peripheral devices
3. PRIME CPU

CAUTION

If you are unfamiliar with the system, do not attempt to restart DOS. Check with someone who knows the system's hardware status, the contents of all disk surfaces, and the correct STARTUP procedure for the particular installation.

A typical procedure to restart DOS after a shutdown is:

1. Turn on power and boot DOS into control as described earlier.

2. Give the appropriate STARTUP command. For example, in a system with a fixed/removable moving head disk drive, the usual startup is STARTUP 1 0. This establishes the fixed surface as the DOS command disk.

3. ATTACH to an authorized UFD and resume operation.

EXAMPLE - INITIALIZING DOS & PROGRAM DEVELOPMENT

The following printout is the actual terminal record of an operating session in which a DOS master disk is installed and copied. DOS is then initialized and used to demonstrate the development of a simple FORTRAN program.

```
PHYSICAL DEV = 0   ← When the master disk has been loaded, the computer has
                     been started, and DOS has been booted. DOS requests the
DOS REV. 5.0 5/10/74 (AT  070000)   physical device number; in
                                    this case, the user types 0.
                  Sign on message from DOS.

OK: STARTUP 0     ← Starts up the top surface of the drive
OK: A CMDNC0      ← Attach to some UFD.
OK: COPY
GO

                  <-------- This copies the master disk contents down
PHYSICAL FROM-TO: 0 1       to the fixed surface.
DONE

OK: STA?SHUTDN    ← Shutdown to show what happens
OK: STARTUP 1 0   ← New startup lets us work from fixed surface
OK: A MFD XXXXXX                during first operating session.
OK: LISTF                       Let's see what UFD names are listed in
                                the MFD.
UFD=MFD      0←Logical device number.

MDDV1   MFD     BOOT    CMDNC0  LIB     SRCLIB  T&M     FLIB1
FLIB2   FLIB3   FLIB4   FLIB5   FLIB6   LIB7    LIB8    IOCS
AIDS    BINED   DOS     RTOS1   RTOS2   RTOS3   RTOS4   INDEX
MATHLB  U-CODE  UII     DBASIC  DVBIN   BASIC   RUNDQV

OK:
 *  CNAME MDDV1 ?
OK: CNAME MDDV1 DOSDEM  ← CNAME to change DSKRAT name; can be used to change
                          name of any file or directory.
OK: CREATE GEORGE
                        } ← Create to create new UFD's; now do a LISTF again
OK: CREATE MIKE           to note changes.

OK: LISTF

UFD=MFD      0

DOSDEM  MFD     BOOT    CMDNC0  LIB     SRCLIB  T&M     FLIB1
FLIB2   FLIB3   FLIB4   FLIB5   FLIB6   LIB7    LIB8    IOCS
AIDS    BINED   DOS     RTOS1   RTOS2   RTOS3   RTOS4   INDEX
MATHLB  U-CODE  UII     DBASIC  DVBIN   BASIC   RUNDQV  GEORGE
MIKE

OK: A MIKE        ← Attach to UFD Mike for program development
OK: LISTF

UFD=MIKE     0 }  Note LISTF of "empty" UFD; now let's enter editor and
                  create a file(2).
```

```
OK: ED
GO
INPUT
```

In response to ED, DOS loads editor and puts user in INPUT mode. Anything typed is stored in editor's buffer as text.

```
C\DUMMY FORTRAN PROGRAM EXAMPLE
\ DIMENSION BUFF(32,32)
\ COMMON BUFF
A=?\ A = 3.
\ B = 4
\ C = SQRT(A**2 + B**2
\ WRITE (1,1000) A,B,C
1000\ FORMAT(HYPOTENUSE OF TRIANGLE WITH SIDES'F8.4,'AND''F8.4,'IS'F8.4
)
\CALL EXTI

\ END
;  ← Note, one way to switch from INPUT to EDIT modes is by typing semicolon.
EDIT
L = 4,P
BOTTOM      Oops.
T,L = 4,P
      B = ← That's better.
C/4/4.     ← Have to put a decimal point after the 4.
BAD C      ← Forgot the closing /.
C/4/4./,P  ← Change made correctly,
      B = 4.←        confirmed by print.
L          ← Remembered that a quote is missing!
P
1000  FORMAT(HYPOTENUSE OF TRIANGLE WITH SIDES'F8.4,'AND''F8.4,'IS'F8.4
C/HY/'HY/P
P
1000  FORMAT('HYPOTENUSE OF TRIANGLE WITH SIDES'F8.4,'AND''F8.4,'IS'F8.4
T,P20  ← To print entire program for cursory inspection.
.NULL.
C     DUMMY FORTRAN PROGRAM EXAMPLE
      DIMENSION BUFF(32,32)
      COMMON BUFF
      A = 3.
      B = 4.
      C = SQRT(A**2 + B**2
      WRITE (1,1000) A,B,C
1000  FORMAT('HYPOTENUSE OF TRIANGLE WITH SIDES'F8.4,'AND''F8.4,'IS'F8.4
)
      CALL EXTI
      END

BOTTOM
```

```
L ),P  ←   Write line looped no continuation; let's try to fix  it
      DIMENSION BUFF(32,32)
L )                                  ←          Located wrong expression, let's
P                                               keep trying.
      WRITE (1,1000) A,B,C           ←
L ),P
)

D
I  ← Use of semicolon to switch from EDIT to INPUT mode
INPUT
\X)   ← Typing double CARRIAGE RETURN also switches from INPUT to EDIT.

EDIT
P
     X)  ← Fixed at last.
FILE TEST   ←              Looks OK.  Let's file it and
                           try compiling it.
OK: FTN TEST
GO
     @WRITE (1,1000) A,B,C
!!****CH****
     @CALL EXTI
!!****HS****

 ** ERRORS (FTN-1082.006).   ←←←←←←          Errors.

OK: ED TEST
GO
EDIT
L B**2
P
      C = SQRT(A**2 + B**2 ←←              Forgot to close the paren
C/B**2/B**2)/                      after the 2.
P
      C = SQRT(A**2 + B**2)  ← That's better.
V   ← Let's switch to editors verify mode.
N
      WRITE (1,1000) A,B,C   ← N command caused next line to be printed
N                                                          automatically.
1000  FORMAT('HYPOTENUSE OF TRIANGLE WITH SIDES'F8.4,'AND''F8.4,'IS'F8.4
C/''/'/ ←← Ye gads!  another error
1000  FORMAT('HYPOTENUSE OF TRIANGLE WITH SIDES'F8.4,'AND'F8.4,'IS'F8.4
FILE TEST  ←← Seems to be fixed, let's file program and recompile it.
```

```
OK: FTN TEST
GO

 NO ERRORS (FTN-1082.006).

OK: LOAD?FILMEM  <<<<<<<<<<<<<<<<<  That time it compiled OK. Let's
GO                                  load it. FILMEM loads unoccupied
                                    memory with zeroes; useful if you want
OK: LOAD                            to make an MDL tape after loading the
GO                                  the program. B<TEST is binary file
$ LO B<TEST  <- Load program.       generated by compiler.
$ LIB  <- Loads library.
$ MA 1  <- No load complete (LC) message; try a load map.
*START  001000    *LOW    000074    *HIGH   010545    *PBRK   010546
*CMLOW  057752    *CMHGH  063752    *SYM    057211    *UII    000005
                  Note, UII should be 000000; anything else means UII
$ LIB UII  <- Load UII                              requirements not met.
$ MA 3  <- Check to see if all subroutines loaded

EXTI    001106**  <- Oops, specified non-existent subroutine.
$ QUIT  <- Let's leave loader

OK: ED TEST  <- Edit to specify correct subroutine
GO
EDIT
y
L EXTI
      CALL EXTI
C/EXTI/EXIT/
      CALL EXIT
FILE  <- Note, no need to specify name if ED was invoked with a Filename argument.

OK: FILMEM  <- Whoops, we have to recompile; no harm done, however
GO

OK: FTN TEST
GO

 NO ERRORS (FTN-1082.006).

OK: FILMEM
GO

OK: LOAD
GO
$ LOAD TEST?LO B<TEST  <<  To load B<test.  This line shows use of ? to
$ LIB                                       cancel incorrect command.
LC  <<  Note, LC Load Complete is spurious unless UII requirements are satisfied.
        Let's check this time by making a full load map.
```

```
$ MAP
*START  001000    *LOW    000074    *HIGH   010545    *PBRK   010546
*CMLOW  057752    *CMHGH  063752    *SYM    057216    *UII    000005
                                    Indicates UII if not 0 → ↑
*BASE   000200  000352  000771  000777

LIST    000001    SQRT    001127    ES21    001221    FSWN    001273
FSWNX   001302    FSIO    001360    FSA1    001714    FSA3    001714
FSA2    001720    FSA5    001720    FSA6    001725    FSIORS  002201
FSCB    002202    FSFLEX  004155    FSER    004323    FSHT    004330
AC1     004410    AC2     004411    AC3     004412    AC4     004413
AC5     004414    RDASC   004415    RDBIN   004421    WRASC   004425
WRBIN   004431    CONTRL  004532    ATTDEV  004603    SETIOS  004632
RATBL   004713    RBTBL   004723    WATBL   004733    WBTBL   004743
CNTBL   004753    LUTBL   004763    PUTBL   004774    ISDASC  005005
ISDVMS  005153    OSDASC  005154    OSDBIN  005346    ISDBIN  005415
OSLASC  005462    ISCASC  006247    IDCASC  006247    ISAASC  006545
ISPASC  006553    OSAASC  006714    OSPASC  006720    ISABIN  007004
ISPBIN  007025    OSABIN  007336    OSPBIN  007350    READ    007624
WRITE   007655    SEARCH  007706    EXIT    007711    OPSCHK  007716
PUTC    007747    T1IN    010005    T1OU    010076    TONL    010112
TNOUA   010116    TNOU    010133    TOOCT   010167    P1IN    010223
P1OU    010244    CSP     010262    CSA     010356    P1OB    010456
P1IB    010462    T1IB    010467    T1OB    010474    FSAT    010501
FSAT1   010503              057752
$ LIB UII ← Load UII package, now we can believe LC message.
LC

$ SA T?SA *TEST← Use of loader SAVE command to save *TEST
$ EX ← To execute *TEST.
HYPOTENUSE OF TRIANGLE WITH SIDES  3.0000AND  4.0000IS  5.0000

                                              ↑
                                    Program output, it works!
                                    Note, at this point, we
                                    have returned to DOS
                                    command level.

OK: R *TEST ↔ Resume *TEST for the fun of it.
GO
HYPOTENUSE OF TRIANGLE WITH SIDES  3.0000AND  4.0000IS  5.0000

OK: PM ↔ Let's do a few more commands, e.g., PM
SA,EA,P,A,B,X,K=
000066 012252 001107 120240 006726 000000 006203
OK: STATUS

UFD=MIKE    0
```

```
BOSLO   FUNITS
067000

LDEV  PDEV
  0     01
  1     00
-

OK: AVAIL
00
000413_←← Note, tells us 413 (octal) records still available on disk (logical Ø).
OK: LISTF

UFD=MIKE     0

B←TEST  TEST     *TEST          Note that compiler generated binary
                                filename B←TEST; Loader generated file *TEST.
OK: SHUTDN ←←
OK:              Shutdown the disk to remove the master disk pack.
```

SECTION 4

COMMANDS

COMMAND STRUCTURE

When properly loaded and started, DOS prompts the operator with the message OK:, and DOS/VM prompts the operator with the message OK,:. This response indicates that the operating system is ready to receive and process a command string. This section defines the form and effect of all legal DOS and DOS/VM commands, both internal (processed by the operating system) or external (executed by system-level programs that are called by DOS or DOS/VM).

All commands consist of a command name and an optional list of arguments typed on a single line and entered by the carriage return key (CR). The operating system analyzes and executes the command, if possible. Blank lines are ignored. Errors in the command string, or caused by the programs that execute external commands, result in an error message.

A series of DOS or DOS/VM commands may be prepared by the Text Editor and stored in a command file for automatic execution under control of the COMINPUT command.

Command Format

The general syntax of a command is:

COMMAND ␣ Name1 ␣ Name2 ␣ Arg1 Arg2 ... Arg9 (CR)

where COMMAND is the command name, usually each Name is a Filename or UFD name (or is a meaningful identifier) and each Arg is an octal argument, or parameter, of up to six octal digits. (If more than six digits are specified, the last six are used.) Up to three names and nine arguments are allowed. Spaces (␣) must be used following the command name and between each Filename or argument. The ellipsis (. . .) indicates that the preceding item can be repeated. The following examples demonstrate the notation used in this section to represent command formats:

RESUME Filename [Pc] [A] [B] [X] [Keys]

In this example, RESUME is the command name. The letter R is underlined as the acceptable abbreviation. The command must specify a Filename, a legal filename existing in the UFD to which the user is currently attached. The remaining items in the command string are the RVEC parameters, described later. Items enclosed in brackets are optional. Parameters that are omitted are assumed to be zero. Parameters are identified by the operating system according to their position in the command string.

An ordinal value followed by a slash and a value can be used to give an octal parameter. For example:

```
R FILENAM 3/1000
```

sets the value of X.

Items enclosed in vertical lines are alternatives, of which one must be chosen, as in

```
        |  TTY      |
COMINP  |  CONTINUE |
        |  PAUSE    |
        |  Filename |
```

Items in all capital letters (e.g., CONTINUE) must be entered literally. Items in initial caps (e.g., Filename) are variables to be assigned values or names by the user.

Levels of Communication

There are two levels of communication between a user at a terminal and DOS/VM (or the user at the terminal in DOS). The user either interacts with the supervisor, or with a program currently being executed under control of the operating system. When interacting with the supervisor, terminal input is interpreted as system commands. This is referred to as command level. If the user is interacting with a program that is running under control of DOS or DOS/VM, terminal input is interpreted as data significant to that program, and it is passed from the supervisor to the running program to be interpreted by that program. (The LINE FEED character is ignored by DOS/VM). In DOS/VM, there is one exception to the interpretation of terminal input in either mode. When the CONTROL-P character is input, it is always interpreted by the supervisor as a QUIT character. Whenever a user program or system command has completed execution, the user returns to command level ready to communicate with DOS/VM. Upon normal completion of a command or program, DOS/VM prints the prompt:

```
OK,
```

or DOS prints the prompt:

```
OK:
```

If an error occurs, the operating system prints an error message and the message:

```
ER!
```

DOS Commands Allowed in DOS/VM

All user commands used in DOS, and described in Section 4, are available for use with DOS/VM. The following commands are not allowed for users at user terminals under DOS/VM: STARTUP and SHUTDN. In addition, the system terminal (operator) commands described in Section 6 are needed, or are useful, for DOS/VM system operation.

Error Correction

Errors typed into the command line may be corrected by using the kill character (?). It deletes everything previously typed on the line; the command must be retyrped in entirety. Do not use the editor's erase character (") to rub out single characters in a command line under DOS (i.e., at DOS command level). Under DOS/VM, both ? and " work in command lines.

DOS and DOS/VM Names

DOS and DOS/VM names, or UFD names, consist of one to six ASCII characters. For compatibility with the command string interpreter and the text editors, the first character must be non-numeric; the others may be any printing character except the question mark (?) or quotation mark ("). Examples:

Legal	Illegal	
CMDNC2	2CMDNC	(Begins with numeral)
LDR$A	LDR A	(Contains space)
TEST1 TEST2	TESTER1 TESTER2	(First six characters not unique)
#%!.)	#%?.)	(Contains question mark)

Disk vs. DOS or DOS/VM Units

DOS file units (1-16) or DOS/VM file units (1-32) referenced by the BINARY, CLOSE, INPUT, LISTING, and OPEN commands are identified by the abbreviation Funit. These are not to be confused with the logical disk units referenced by the abbreviation Ldisk in the ATTACH, BOOT and CREATE commands. Physical disk drives are assigned logical disk unit numbers by the STARTUP command (refer to Table 3-1); thereafter, only the logical unit numbers are meaningful to DOS.

SUMMARY AND INTRODUCTION TO COMMANDS

Internal Commands

Internal commands are executed in the space occupied by DOS or DOS/VM itself, as opposed to the external commands which are external to the operating system and execute in user space. Most internal commands have to do with the file handling and with saving or restoring of filed programs and associated register values. The internal commands are described here in alphabetical order, and also listed in Table 4-1 in an order of most commonly used to least commonly used. This table also indicates those commands that in addition to being useful to users when they are functioning as programmers, are also useful to users when they are functioning as operators. Detailed information about each internal command is given in the last part of this section. The descriptions include all commands, both internal and external, and they are arranged in alphabetical order. The internal commands described are:

ATTACH	LISTF	RESUME
BINARY	LISTING	SAVE
CLOSE	OPEN	SHUTDN
COMINPUT	PM	START
DELETE	PRERR	STARTUP
INPUT	RESTORE	*

Hybrid Commands

DOS recognizes four external commands that are restored into low memory on top of user space. These commands accept an internal command line interpretation, but destroy user memory space. Furthermore, in DOS/VM, they function as internal commands. The file containing hybrid commands, is in CMDNC0 and is named DOSEXT. The versions of DOS all use the same hybrid command file. The hybrid commands are: CNAME, CREATE, PASSWD, and STATUS. The hybrid commands CREATE, PASSWD, CNAME, and STATUS act as internal commands in DOS/VM. (They act as external commands in DOS.) They are listed in Table 4-2 along with the external commands. Detailed descriptions of the hybrid commands are given in the last part of this section, arranged alphabetically along with all the other command descriptions.

External Commands

The external commands are commands to load and start system programs in the command UFD (e.g., CMDNC0). They are external to the operating system and execute in user space (i.e., they may reside in a UFD). In general, these programs include the translators, utilities, and debugging programs used in Prime application program development. However, any type of program can be filed in CMDNC0 and called for thereafter by filename. Some of the external commands, like FUTIL, CRSER and PRSER, control data transfers to or from peripheral devices. The user may want to add programs to CMDNC0 to perform functions peculiar to his system.

Table 4-1. Internal Commands

COMMAND	FUNCTION	FREQUENTLY USED BY OPERATOR
STARTUP	Initialize disk drive configuration	Yes
SHUTDN	Perform tasks necessary to shut down DOS	Yes
ATTACH	Assign user to a UFD	Yes
LISTF	List names of files in current UFD	Yes
RESTOR	Restore a memory image from disk to high-speed memory	
START	Initialize processor keys and start or restart execution of a memory image or user program	Yes
RESUME	Combines operation of RESTORE and START	
PM	Print contents of DOS RVEC vector	
SAVE	Save a memory image by writing it from memory to disk	
DELETE	Delete a file	
COMINP	Execute a file consisting of DOS command lines	
OPEN	Open a file for the specified operation	
CLOSE	Close a file (or all files)	Yes
INPUT	Open ASCII source file for Reading	
LISTING	Open FORTRAN Listing file on Unit 2	
BINARY	Open file on Unit 3 for Writing	
*	Indicates Command Line	
PRERR	Print last error message stored in DOS ERRVEC	

Table 4-2. External and Hybrid Commands

COMMAND	FUNCTION	USED BY OPERATOR
Editors		
ED	Create and Edit a File (Usually Source Programs and Data)	
EDLIN	Simple line editor version of ED (no BOX edit mode)	
EDB	Edit binary file	
Language Processors		
FTN	Invoke the FORTRAN Compiler to compile a program	
PMA	Invoke the Prime Macro Assembler to assemble a program	
MCG	Translates Microcode assembly results into ROM	
BASIC	Invoke BASIC interpretative language	
DBASIC	Invoke double-precision version of BASIC	
LBASIC	Invoke a version of BASIC with matrix and print-using	
BASINP	Read paper tapes containing foreign BASIC programs	
NUMBER	Utility to number BASIC program statements	
Loader and Utilities		
FILMEM	Fills Memory with zeroes	
LOAD	Loads and Starts the Linking Loader (32K DOS)	
LOAD20	Loader for 16K DOS	
HILOAD	Loader for loading programs to run in full 64K	
Run-time Debuggers		
TAP	Load and start interactive debugging program	
PSD	Load and start Prime Symbolic Debugger (32K DOS)	
PSD20	Load and start Prime Symbolic Debugger (16K DOS)	

Table 4-2. (Cont)

COMMAND	FUNCTION	USED BY OPERATOR
File Utilities		
CNAME	Change the name of a file	
SPOOL	Queue a copy of a file for off-line printing on systems high speed printer	
CREATE	Create a new UFD	
CRMPC	Read cards from parallel interface card reader	
CRSER	Read cards from serial interface card reader	
FUTIL	Invokes file utility	
HELP	Print information file in ufd INFO	
PRMPC	Print a file on parallel line printer	
PRSER	Print a file on serial line printer	
STATUS	List current UFD and boundary of DOS buffers	Yes
AVAIL	Print number of available records	
SIZE	Print number of records used by file	
SLIST	Print contents of a file at the users terminal	
SORT	Sort the contents of an ASCII file	
CMPRES	Convert ASCII file to compressed format	
EXPAND	Convert compressed format file to ASCII	
FILVER	Compare two files and verify similarity or print message if dissimilar	
MAGSAV	Save contents of magnetic tape to disk	
MAGRST	Read contents of magnetic tape to disk	
MTDLSK	Read or write disk to magnetic tape	

Table 4-2. (Cont)

COMMAND	FUNCTION
File Access Control	
PASSWD	Assign a password to a UFD or assign "owner" and "other user" passwords to a UFD
PROTEC	Assign access attributes (read, write, delete, execute, etc.) to a file in an owned UFD
RTOS	
RTOSRA	Establish RTOS random access file
RT128F	Write unstructured RTOS records
FILBLK	Read or write 128-word record from memory in a previously created RTOS random access file
Paper Tape Utilities	
PTCPY	Utility program that duplicates and verifies paper tapes
MDL	Memory dump onto paper tape
PTBOOT	Paper tape boot strap
LOADAP	Paper tape linking loader
BPTREP	Paper tape Editor with box editing
PRTED	Paper tape Editor
Operator Utilities	
COPY	Copies from one disk volume to another
FIXRAT	Loads and starts a DOS maintenance program that checks file integrity
MAKE	Creates a DOS disk with a basic DOS structure

Table 4-2. (Cont)

COMMAND	FUNCTION	USED BY OPERATOR
Operator Utilities		
MACHK	Run computer in or out of machine chek mode	Yes
DOSVM	Start DOS/VM from DOS	Yes
Virtual Utility		
VDOS32	Start DOS from DOS/VM	Yes
DOS/VM User Commands		
LOGIN	Allows user to obtain access to DOS/VM (i.e., start user terminal session)	
LOGOUT	Terminates user terminal session	
STATUS	Gives information about the system.*	Yes
PASSWD	Allows assignment of owner and nonowner passwords	
PROTEC	Defines access attributes for a file	
TIME	Prints current value stored in time accounting registers.	
USERS	Prints number of users currently logged-in	
DELAY	Defines a time delay function to be used to delay the printing of a character after a CARRIAGE RETURN	
ASRCWD	Sets the virtual control word as specified	
SVCSW	Controls handling of SVC instructions in the virtual memory environment	
VRTSSW	Allows setting the virtual sense switches	
VDOS32	Invokes a version of DOS that may be run under control of DOS/VM	Yes

* STATUS function in DOS/VM is very similar to its function in DOS. However, STATUS when used in DOS/VM returns additional information. For these reasons, STATUS is redescribed in the following paragraphs.

Table 4-2. (Cont)

COMMAND	FUNCTION	USED BY OPERATOR
ASSIGN	Obtains control over a disk or peripheral device	
UNASSIGN	Releases control over a disk or peripheral device	
MESSAGE	Invokes system message facility	Yes

An example configuration for CMDNC0 is as follows:

```
OK: ATTACH CMDNC0
OK: LISTF

UFD = CMDNC0

FILBLK RTOSRA MCG    NUMBER TAP    RT128F BASINP SIZE   LOADAP CNVT45
FILVER CMPRES EXPAND COPYB  AVAIL  PSD    FILCPY UFDCPY PMA    FTN
PTBOOT COPY   MAGSAV MAGRST LFTN   CRSER  CRMPC  PRMPC  PRSER  BASIC
LBASIC DBASIC SLIST  MTDSK  SPOOL  LOAD   LOAD20 MDL    FILMEM PSD20
ED     PRTED  VDOS32 DOSEXT PTCPY  FUTIL  FIXRAT HILOAD DOSVM  EDB
MACHK  MAKE   SORT
```

Unless otherwise specified, programs invoked by external commands return to command level after they have completed execution.

The external commands are listed along with hybrid commands in Table 4-2 in an order that more or less reflects most common usage to least common usage. Commands of special interest to operators as well as programmers are so noted.

The external commands and the hybrid commands listed alphabetically (as are descriptions of all commands) are arranged in the last part of this section) as follows:

AVAIL	FILVER	NUMBER
BASIC	FIXRAT	PASSWD
BASINP	FTN	PMA
BPTRED(See ED)	FUTIL	PRMPC
CMPRES	HELP	PRSER
CNAME	HILOAD	PSD
COPY	LBASIC	PTBOOT
CREATE	LFTN	PTCPY
CRMPC	LOAD	PTRED(See ED)
CRSER	LOAD20	RTOSRA
DBASIC	MACHK	RT128F
DOSVM	MAGRST	SIZE
ED	MAGSAV	SLIST
EDLIN	MAKE	SORT
EDB	MCG	STATUS
EXPAND	MDL	TAP *
FILCPY	MTDISK	VDOS32
FILMEM		

*In top example.

COMMAND DESCRIPTIONS

All command descriptions (internal, external and hybrid) are arranged in alphabetical order in the following paragraphs. Programs that have operating procedures or an extensive command repertoire may be described in detail in appendices, or in the Program Development Software User Guide (MAN 1879). In the following detailed descriptions of commands, the elements in all capital letters are command names. Elements in initial capital letters are arguments. If an argument is enclosed in square brackets, the argument is optional. The abbreviation of a command name is underlined, and commands unique to DOS/VM are flagged.

ASRCWD ***DOS/VM***

The ASRCWD command allows changing the virtual control word (see I/O virtualization). This control word is used to select one of four devices as effective output and one of four for input. The control word sets the devices output by the OTA 4 instruction and the device input by the INA 4 instruction.

Syntax :

```
| ASRCWD  XXXXXX |
| ASR     XXXXXX |
```

where XXXXXX is an input or output number as specified in Table 4-3.

Device or Port No.	Input (Bits 11, 12)	(Output 13-16)
1	00 User terminal	00 or 10 User terminal
2	01 CENPR (J2)	Octal 4 CENPR (J2)
3	10 CE2PR (J3)	Octal 2 CE2PR (J3)
4	11 CARDR (J4)	Octal 1 CARDR (J4)

Table 4-3. Value for Virtual Control Word and Port Assignment

Example:

```
ASR 10
```

ASSIGN ***DOS/VM***

The ASSIGN command obtains complete control over a disk or a peripheral device (e.g., printer, paper tape reader) from the user terminal.
Syntax:

```
ASSIGN  Device
AS  Device
ASSIGN  Device  [WAIT]
AS  Device
ASSIGN  DISK Number
AS  DISK Number
```

where Device is an available device.

All assignable devices are named as shown in Table 4-4.

```
CARDR - General Card Reader (AMLC Line No. 6)
CENPR - First Centronics printer (System Option Controller port No.2)
CE2PR - Second Centronics printer (System Option Controller port No.3)
CRI   - MPC Parallel Card Reader

DISK  0 - Physical disk 0
DISK  1 - Physical disk 1
DISK  2 - Physical disk 2
DISK  3 - Physical disk 3
DISK  4 - Physical disk 4
...
DISK 57 - Physical disk 57
...
DISK 5256   -  Physical disk 5256
DISK 002452 -  Physical disk partition
MT0  -  Magnetic Tape Unit 0  Dial = 0
MT1  -  Magnetic Tape Unit 1  Dial = 1
MT2  -  Magnetic Tape Unit 2  Dial = 2
MT3  -  Magnetic Tape Unit 3  Dial = 3
PRI  -  MPC Parallel Interface Line Printer
PTR  -  Paper Tape Reader
PUNCH - Paper Tape Punch
```

Table 4-4. Device Names

For disk assignment details, refer to Table 3-1.

A user may only ASSIGN a disk that is not already assigned and appears in the assignable disks table. This table is initially empty, and it is altered from the supervisor terminal using the DISKS command. This restriction provides a degree of system integrity because it prevents users from assigning a disk without the supervisor terminal operator's knowledge; it prevents users from assigning nonexistent disks and partitions; and it prevents users from assigning disks or partitions the operator wishes to reserve for special use.

If the Device is currently assigned to another terminal, the system replies:

```
ER!  DEVICE IN USE
```

unless the optional argument WAIT was supplied. In this case, the ASSIGN command is queued until the device is UNASSIGNED by another user, or until the user presses the QUIT key.

If the user does not ASSIGN a device and attempts to perform I/O to or from the device, the error message;

```
DEVICE NOT ASSIGNED

ER!
```

is printed at the terminal.

In order for a disk to be ASSIGNed to a user, it must not be ASSIGNed to another user, nor may it be a disk specified in a previous STARTUP command, and it must not be the paging disk. To ASSIGN a disk that has been started up by STARTUP, it must first be shut down by the SHUTDN command at the supervisor terminal.

Disks, or devices, ASSIGNed by another user are released when the user invokes the UNASSIGN command and/or when the user invokes the LOGOUT command.

Examples:

```
ASSIGN  CENPR  WAIT
```

assigns the Centronics printer and queues the assignment if the printer is already assigned.

```
AS  PTR
```

assigns the paper tape reader.

```
AS  DISK  0

AS  DISK  54
```

assigns disk drives as defined in Table 3-1.

The maximum number of disk drives that may be ASSIGNed to all users at any one time is 10. If an attempt is made to ASSIGN too many disks, the message:

```
ASSIGN TABLE FULL
```

is printed.

ATTACH Ufd [Password Ldisk Key]

In order to access files, DOS must be attached to some User File Directory. This implies DOS has been supplied with the proper file directory and either the owner or nonowner password, and DOS has found and saved the location of the UFD. After a successful attach, the name, location and owner/nonowner status of the UFD is referred to as the current UFD. As an option, this information may be copied to another place in DOS, referred to as the home UFD. The user obtains owner status if the owner password is specified or nonowner status if the nonowner password is specified. The owner of a file directory can declare on a per-file-basis what rights a nonowner has over the owner's files. The nonowner password may be specified only under DOS/VM. Refer to Section 2 and the commands PASSWD and PROTEC for more information . In attaching to a directory, ATTACH specifies a file directory in the Master File Directory (MFD) on a particular logical disk or a file directory in the current UFD or the home UFD as the directory to be attached. The most common ATTACH command is:

```
ATTACH  Ufd  Passwd
```

The meaning of this command line is: search for UFD in the MFD on all started up logical devices 0, 1, 2 ... n, and attach to the UFD whose Ufd name appears in the MFD of the lowest numbered logical device. Also, the command line indicates attach to Ufd only if Passwd matches the password of UFD Ufd, then set the home UFD to Ufd.

The user may specify the logical disk of the MFD to be searched as in the command:

```
ATTACH  Ufd  Password  Ldisk
```

Ldisk is specified as an octal integer.

Finally, the user may specify a key as in the command:

```
ATTACH  Ufd  Password  Ldisk  Key
```

If Key is 177777, the MFD of the currently attached disk is searched for Ufd. If Key is 100000, all disks are searched in logical order.

The keys are as follows:

Key	Meaning
0	attach to Ufd in MFD on Ldisk; set home UFD
1	attach to Ufd in current UFD; do not set home UFD
2	attach to Ufd in current UFD; set home UFD to current UFD
177777	attach to Ufd in MFD on Ldisk; do not set home UFD

To attach to the home UFD, use ATTACH (blanks)

Example:

```
ATTACH  GOUDY  ABCABC
```

Search for GOUDY in the MFD on all started up disks. Attach to GOUDY on the lowest logical disk where found. Check the password. Set home UFD.

```
ATTACH
```

Attach to home UFD (GOUDY)

```
ATTACH  CARLSO   XXXXX  7
```

Attach to CARLSO. Look for CARLSO with a password of XXXXX in the MFD of logical disk 7. Set home UFD to CARLSO.

Attach is an internal command.

AVAIL

Gives the number of disk records available for use in the specified logical disk (in octal). The format is:

```
AVAIL
AVAIL ZERO
AVAIL ONE
AVAIL TWO
...
AVAIL NINE
AVAIL  Packname
```

If no argument is specified, AVAIL types the number of available records on the logical disk of the current UFD. If Packname is specified instead of ZERO...NINE, the number of available records on the logical disk with DSKRAT name Packname is printed. AVAIL is an external command. Examples:

```
OK:  AVAIL
GO
000113
OK:  AVAIL FOUR
GO
001273
OK:
...
AVAIL  TSDISK
GO
000113
OK,
AVAIL  DUD
GO
001273
OK,
```

BASIC

Loads the Prime BASIC Language interpreter. For further information, refer to the BASIC manual. On the original master disk the version of BASIC that has both the matrix functions and print using functions is named LBASIC. Also, a version called DBASIC, that uses double-precision floating point arithmetic, is available. BASIC is an external command.

BASINP Filename

The BASINP command invokes a program that loads from paper tape a BASIC program that has been written for a computer system other than a Prime computer. Filename is the name of the file into which the contents of the paper tape are to be read. BASINP is an external command.

BINARY Filename

Opens a file for writing on DOS (DOS/VM) file unit 3, usually as a binary output file for use by the compiler or assembler. The file is assigned the name Filename in the current UFD.. Binary is an internal command. This command has the same effect as OPEN Filename 3 2.

> PMA and FTN automatically open a file named B←XXXX as the binary output file (XXXX is the first four letters of the input (source) filename.) A BINARY command is required only if the user wants the output file to have a different name.

CLOSE | [Filename] [Funit] . . [Funit] |
| ALL |

Closes the named files and specified file units. The form C ALL closes all files and units. (In a command file, specify each item to be closed; do not use C ALL or the command file itself will be closed.) CLOSE is an internal command.

The CLOSE ALL command also makes sure that buffers are retrieved properly and resets the state of the file system. If the user is even slightly uncertain about the state of the file system, he should enter a CLOSE ALL. (The STATUS command prints the state of the file system.)

If the file named cannot be found, an error message is printed and the CLOSE command returns to command level.

CMPRES Filename1 [Filename2]

The input ASCII file, Filename1, is translated into the output ASCII file, Filename2, using the relative copy character ('220). The byte following the relative copy character specifies the number of characters to copy from corresponding positions in the preceding line. If Filename2 is omitted, the output replaces Filename1. The amount of space saved is a function of the structure of Filename1. CMPRES handles a line size of up to 720 characters. CMPRES is an external command. Example, contents of typical Filename1 named STEST:

```
C       PROGRAM TO TEST DSQRT
C
        DOUBLE PRECISION A
        READ (0) A
        CALL DSQRT(A)
        WRITE (0) A
        END
```

Command line:

```
CMPRES  STEST  CTEST
```

Contents of Filename2, CTEST

```
C       PROGRAM TO TEST DSQRT
C
        DOUBLE PRECISION A,B
'220'007READ (0) A,B
'220'007CALL DSQRT(A)
'220'007WRITE (0) A
'220'007END
```

To reverse the effect of the CMPRES command, use the EXPAND command.

CNAME Oldname Newname

Changes the name of the file named Oldname to Newname. This command operates within the current UFD. If the user is attached to the MFD, this command can be used to change the name of a UFD. CNAME is a hybrid command. Under DOS/VM, CNAME requires owner status to the UFD. Example:

```
OK: A MFD
OK: CNAME  SPARE2  JHNDOE
```

assigns a new UFD name JHNDOE in the place of the old name SPARE2.

COMINPUT

The available forms of this command are:

```
COMINPUT | TTY          | [Funit]
         | Filename     |
         | CONTINUE     |
         | PAUSE        |
```

CO Filename causes DOS or DOS/VM to read commands from Filename in the current UFD, rather than for the terminal. The file is usually prepared and filed by the text editor (ED). Each line of the file must be a legal command string, one command per line. This type of file is referred to as a command file. Command files may be chained. If the last line in a COMINP command file is of the form:

```
CO  Filenamex
```

the current command file is closed and DOS or DOS/VM reads commands from the new command file Filenamex. This feature allows chaining of command files. The last command in the last command file must be CO TTY to return control to the terminal. Note that "TTY", "CONTINUE" and "PAUSE" are reserved words for DOS (and DOS/VM), and they must not be used for other purposes.

COMINP is an internal command.

DOS or DOS/VM reads commands from the command file, Filename, by opening unit six and reading, the executing, one line at a time. When the command CO TTY is encountered, DOS takes subsequent commands from the terminal. The user must specify a file unit (Funit) for COMINP TTY if not using the default unit. Any error message causes command input to be returned to the terminal. However, the command input file is left open allowing a user to retype the command that caused the error message, then continue reading from the command input file by typing

```
CO  CONTINUE
```

Use of the command CLOSE ALL in a command input file closes the command input unit and causes the message PRWFIL UNIT NOT OPEN to be printed.

The form:

```
COMINP  Filename  Funit
```

has the additional capability of specifying the file unit upon which the command file is to be open. Thus, the user can set up a complex set of interacting command files. Example:

Assume the command file PMLISTF contains the following:

```
PM
LISTF
COMINP PMPM  7
CLOSE  7
PM
COMINP TTY
```

and the command file PMPM contains the following lines:

```
PM
STATUS
PM
COMINP CONTINUE  6
```

Then typing the command line:

```
COMINP  PMLISTF
```

from DOS (DOS/VM) command level causes both command files to be run.

Example: The COMINP command is useful for updating large programs that consist of many files, use several library files, or require special loading procedures. For example, suppose a user with the UFD USER1 has a program consisting of three FORTRAN source files MAIN, SUB1, and SUB2.

This program requires two libraries, GRALIB and FTNLIB. A user makes up the following command input file DPROG:

```
FTN  MAIN
FTN  SUB1
FTN  SUB2
FILMEM
LOAD
 LO  B=MAIN
 LO  B=SUB1
 LO  B=SUB2
 LIB  GRALIB
 LIB  FTNLIB
 MAP
 QUIT

CO  TTY
```

After the programs are corrected and ready to be compiled, the user enters the command CO DPROG. The DPROG file then provides the commands that cause the programs to be compiled, loaded, and a load map printed. DPROG also documents the source files and loading procedure.

The form:

COMINPUT PAUSE

leaves the current command input unit open and returns to command level. Thus, a user can invoke other commands or use COMINP to start another command file on another unit before issuing a COMINP CONTINUE line to continue the original command file.

COPY

COPY is an external command that copies and verifies a disk. COPY copies any disk to any other disk, either under DOS or DOS/VM. Under DOS/VM, both disks must be ASSIGNed before invoking COPY. After the user types COPY at the terminal, the COPY command responds by printing:

PHYSICAL FROM TO SIZE:

The user must specify the disk to be copied from (FROM), and the disk to be copied to (TO), and the size (SIZE) of the disk to be copied. The line of parameters specified are then terminated by the CARRIAGE RETURN (CR). The SIZE parameter is optional and may be omitted. The FROM and TO parameters are physical device numbers separated by a space (ASCII space character) or a hyphen (-). FROM and TO device numbers are listed in Table 3-1.

Possible SIZE numbers are:

Disk Type	SIZE Number	Abbreviation
1.5-million word disk pack	1.5M	1
3-million word disk pack	3.0M	3
30-million word disk pack	30.0M	30
128 thousand word fixed head disk (32 track)	128K	128
136 thousand word diskette	136K	136
256 thousand word fixed head disk (64 track)	256K	256
512 thousand word fixed head disk (128 track)	512K	512
1024 thousand word fixed head disk (256 track)	1024K	1024

If SIZE is omitted, the default size assumed is 1.5M, unless either disk being copied is a diskette. In that case, the diskette size is assumed to be 136K. The SIZE parameter does not need to be given if the disk is 1.5M or a diskette.

If FROM and TO are equal, or if FROM or TO is not a valid physical disk number, or if SIZE is not a valid number or abbreviation, COPY repeats the request message and waits for input. If these parameters are acceptable, COPY inititates the copy operation.

If the 30-million word disk is partitioned, the user may COPY an individual partition of the disk. These other SIZE parameters are as follows;

Partition	Disk Number	SIZE	Abbreviation
2 head (default)	XX025X	3M	3
2 head (explicit)	XX065X	3M	3
4 head	XX125X	6M	6
6 head	XX165X	9M	9
8 head	XX225X	12M	12
10 head	XX265X	15M	15
12 head	XX325X	18M	18
14 head	XX365X	21M	21
16 head	XX425X	24M	24
18 head	XX465X	27M	27
20 head	XX525X	30M	30

Table 4-5. Partitioned Disk SIZE Specification

In Table 4-5, The X's represent octal digits that must be set appropriately. The leftmost X's indicate head offset; and the rightmost X indicates one of the four possible drives connected to the controller. Example:

100252

means a 3M word size partition with a head offset of 10 on the disk that is connected to the second drive (drive 2) connected to the controller.

Note: COPY does not allow rewriting of the same disk. For example:

PHYSICAL FROM TO SIZE: 5-5

is an illegal specification of COPY parameters.

WARNING: A TO disk number must not be a disk connected to DOS by the STARTUP command. It is good practice when running COPY under DOS, to place all active disks in WRITE PROTECT before initiating the COPY command except for the disk to be written to (TO disk). It is good practice when running under DOS/VM, to place all disks assigned to the user terminal at which the COPY command is to be initiated in WRITE PROTEC, except the TO disk, before initiating the COPY.

Copy Method: COPY copies disk records from the FROM disk to the TO disk and when done verifies the copy by reading each record from both disks and performing a word-by-word comparison in memory. During this process, COPY displays the disk record number it is processing in the DATA lights on the processor console panel, bits 2-16. Bit 1 is off during the copy operation and on during the verify operation. When done, COPY prints DONE and returns to DOS or DOS/VM, which prints OK. If any disk read errors occur during the copy, the read is retried nine times. Each error results in an error message of the form:

DISK RD ERROR device number DOS Record-number Status

See Appendix J for explanation of Status.

If the read operation is not successful after ten tries, DOS/VM (DOS) ignores that record and prints the message:

ERROR READING DISK Device-number RECORD Record-number

ERROR IGNORED, COPY CONTINUED

Then, DOS/VM (DOS) continues the copy operation. If any disk write errors occur, COPY retries nine times. Each error results in an error message of the form:

DISK WT ERROR Device-number DOS-Record-number Status

If the write operation is not successful after ten tries, COPY aborts, prints the error message, UNRECOVERED ERROR, and returns to DOS (or DOS/VM). If on either read or write a DISK-NOT-READY status is detected, a single disk error message is printed with the status 177776. The software then retries the read or write, waiting for the disk to become ready. If while verifying the copy, a discrepancy is detected, COPY prints VERIFY ERROR and returns to DOS or DOS/VM.

COPY Success or Abort: If the COPY is successful, the message

DONE

is printed at the terminal (only if both the copy and verify were successful).

Example:

```
OK: COPY
GO

PHYSICAL FROM-TO: 1 0
DONE

OK:
```

CREATE Newufd

Creates a new UFD named Newufd in the current MFD. CREATE is a hybrid command. The passwords of the new UFD are: owner password is Blank, and the nonowner password is Zero (any password will match).

```
OK: A MFD XXXXXX
OK: CREATE BETTY

OK: LISTF

UFD=MFD      Ø

TSDISK  MFD     BOOT    CMDNCØ  PODUSK  JBRWNS  GIBSON  GREATA
BARBOU  STUMPF  GILES   LIB     SPORER  BASIC   DOS     WEYLER
M.JOHN  AROSS   KROY    GRUBIN  DEMO    JSKOL   KAY     CURREV
JCVB    DAVIS   UDIN    BROWN   SEV     LEWIS   PRNGL   BUTTER
BRIGGS  COHEN   GOUDY   DUMAS   BRODIE  CARLSO  PLANIT  WEBB
ETTA    RUNDQV  RUNDQ   BETTY

OK:
```

CRMPC Filename

Reads cards from the parallel interface card reader connected to the MPC controller and loads card image ASCII data into the file Filename. Reading continues until the end of the deck or a $E is read in columns 1-2 of a card. The $E causes a return to DOS or DOS/VM and closes the file. If the reader runs out of cards before a $E card is read, the processor returns to the operating system but the file is not closed. The user can load more cards and enter S (i.e., START) to resume reading cards into the same file. At completion of reading, if there was no $E card; enter CLOSE ALL to close the open file. CRMPC is an external command. Under DOS/VM, CR1 must be assigned before the CRMPC command is given.

CRSER Filename

Reads cards from the serial interface card reader. Reading continues until the end of the deck or a $E is read in columns 1-2 of a card. The $E causes a return to DOS or DOS/VM and closes the file. If the reader runs out of cards before a $E card is read, the processor returns to the operating system but the file is not closed. The user can load more cards and enter S (i.e., START) to resume reading cards into the same file. At completion of reading, if there was no $E card; enter CLOSE ALL to close the open file. Under DOS/VM, CARDR must be assigned before the CRSER command may be given. The CRSER command is an external command.

DBASIC

Loads the Prime BASIC interpretative language version that has double precision arithmetic capabilities. DBASIC is an external command.

DELAY ***DOS/VM***

The DELAY command defines a time function to be used to delay the printing of a character after a line feed (LF) has been output to a terminal. Syntax:

DELAY [Minimum] [Maximum] [Rmargin]

Minimum defines the number of character-times (time it takes the system to type a character on a line) to delay when CR is output at the left margin. Maximum defines the number of character-times to delay when CR is output at the right. Rmargin defines the number of characters required to move to the right margin. If a CR is typed at some point within a line, the time delay is proportional to the number of characters typed. If Rmargin is not specified, 72 is assumed; if Maximum is not specified, 12 is assumed. If the command, DELAY, is given with no parameters, the default values 6, 12 and 72 are assumed; these values are adequate for most 30 cps terminals. Example:

```
DELAY  0  10  100
```

DELETE Filename

Frees the disk storage space used by Filename and removes the name from the current UFD. DELETE is an internal command.

WARNING: Do not delete a directory until all files within the directory have been deleted. Otherwise, available disk storage space is lost until the next time FIXRAT is run. If you wish to delete a directory, use the TREDEL subcommand of the FUTIL command.

ED [Filename]

Loads and starts ED, the most commonly used version of the text editor. If a filename is specified, it is loaded into the editor's text buffer in high-speed memory and the editor is started in EDIT mode. Otherwise, the editor is started in high-speed INPUT mode with an empty text buffer. Files and units are automatically opened and closed. ED is an external command. For details of ED operation, refer to the Program Development Software User Guide (MAN 1879).

If the user accidentally returns control to DOS or DOS/VM, the user can restart the ED without losing any of the text buffer by issuing the command:

START 1000

There are exceptions to this. Refer to Section 2 of the Program Development Software User Guide for details of Recovery Procedures.

EDB Inputfile [Outputfile]

Loads and starts EDB, the binary editor, which prints ENTER and waits for command input. The input and output files may be on disk or paper tape. If paper tape is used for either file, use the filename (PTR). If an output filename is specified, a file of that name is created in the current UFD. If the filename already exists, it is overwritten by the output file. EDB is an external command. For details, see Section 2 of the Program Development Software User Guide.

EXPAND Filename1 [Filename2]

Inverts the operation of CMPRES. If Filename2 is omitted, output is placed in Filename1. EXPAND handles line sizes up to 720 characters. EXPAND is an external command.

FILBLK

Permits reading or writing from high-speed memory to any 128-word record in a previously created RTOS random access files. For a means to create RTOS random access file, refer to "RTOSRA" in this section; and for further details, refer to the RTOS User Guide. FILBLK is an external command.

FILMEM

Fills the memory locations '100 to the top of 32K if under DOS/VM, with zeroes. If running under DOS, FILMEM clears '100 to the top of 32K except for those locations occupied by DOS.

FILVER Filename1 Filename2

Causes Filename1 and Filename2 to be compared for equivalence. If any differences exist, a message is printed indicating failure to verify. If the file Filename1 and Filename2 are exactly the same, a message is printed that confirms successful verification. FILVER is an external command.

FIXRAT [OPTIONS]

Loads and starts a maintenance program that checks the file integrity of any disk pack. Under DOS/VM, the disk to be checked must be ASSIGNed before invoking FIXRAT. FIXRAT is an external command. If OPTIONS is typed, FIXRAT requests printout options; otherwise, it defaults to printing the name and octal number of records used in the MFD and each directory file in the UFD. After the command line is typed, FIXRAT asks the question: FIX DISK?. If the answer is YES followed by a CARRIAGE RETURN (CR), FIXRAT truncates or deletes defective files and generates a corrected DSKRAT file. FIXRAT truncates or deletes files in the MFD as well as files in other directories, FIXRAT then asks the question: PHYSICAL DISK DRIVE = . The user must respond by entering the physical disk drive number in octal on which FIXRAT is to be run followed by a CR. A complete discussion of FIXRAT, along with examples, is given in Appendix E.

FTN Filename [1/A]

Loads the Prime FORTRAN IV Compiler and starts compilation of an ASCII source file, Filename in the current UFD. FTN is an external command. A is the A register setting. If no A register value is specified, a default value is used - typically, 1707. (List errors on terminal, use disk for all input/output.) Other common options are:

1/1777	List errors on Teletype (system terminal)
1/40777	Generate listing file that includes symbolic instructions

Unless it is preceded by BINARY and LISTING commands, the compiler will automatically open unit 3 to write a binary file named B←XXXX, and open unit 2 to write a listing file named L←XXXX, where XXXX is the first four letters of the input filename. The compiler closes any units that it opens. (Units opened by BINARY and LISTING commands are not closed.) The listing file can be printed by using the text editor or the PRMPC, PRSER, or SPOOL commands.

For more information, refer to The Program Development Software User Guide, The Subroutine Library User Guide and the FORTRAN Reference User Guide.

FUTIL

FUTIL invokes a file utility command that provides subsystem commands for the user to copy, delete, and list both files and directories. FUTIL also has an ATTACH command that allows attaching to subdirectories by giving a directory pathname from either the MFD or home UFD to the specified subdirectory. FUTIL allows operations not only with files within UFD's but also files within segment directories. FUTIL may be run from a command file.

For a detailed discussion of subsystem commands available under control of FUTIL, refer to Appendix F.

A summary of FUTIL commands is listed in the following table (command abbreviations are underscored).

Table 4-6. FUTIL Commands

Command Syntax	Command Function
QUIT	return to DOS or DOS/VM command level
FROM Directory Pathname	defines FROM directory
TO Directory Pathname	defines TO directory
ATTACH Directory Pathname	moves the Home UFD to the directory defined by Directory-Pathname
COPY File1 [,File2] [,File3] [File 4]]...	copies a file or files in the FROM directory to the TO directory
COPYSAM File1 [,File2] [,File3] [File4]] ,...	same as COPY, but sets file type of file in TO directory to SAM.
COPYDAM File1 [,File2] [,File3] [File4]] ,...	same as COPY, but sets file type of file in TO directory to DAM.
TRECPY Dir1 [,Dir2] [,Dir3] [,Dir4] ...	copy directory tree specified
UFDCPY	copies all files and directories in the FROM directory to the TO directory
DELETE File1 [,File2] ...	deletes the directory tree specified
TREDEL Dir1 [,Dir2] ...	delete the directory tree specified
UFDDEL	delete all files and directory trees within the FROM directory
LISTF [level] [LISTFIL] [PROTECT] [SIZE] [TYPE]	Lists at the terminal the FROM directory pathname; the TO directory pathname; and all files and directories in the FROM directory

For details on the concepts of pathname and FROM and TO directories, refer to Appendix F.

HILOAD

See LOAD.

INPUT Filename

Opens an ASCII source file on unit 1 for reading by the compiler or assembler. The file is assigned the name Filename in the current UFD. This command has the same effect as OPEN Filename 1 1. (For PMA and FTN, the source filename is usually provided with the command that starts assembly or compilation.) INPUT is an internal command.

LBASIC

Invokes a version of the BASIC interpretive language that contains both MAT functions and PRINT USING functions. For details, refer to the BASIC manual. LBASIC is an external command.

LISTF

Prints the current UFD name, the logical device upon which the UFD resides, and all filenames in the UFD at the terminal. LISTF is an internal command. Attributes of files such as type, size, and protection may be examined using the LISTF subcommand of the FUTIL command.

```
OK: LISTF

UFD=BARBER  1

FDAT     FATI     PMAT     FLN      ARG      B←FDAT   DFAT     B←DFAT
B←FATI   B←ARG    B←FLN    SLITE    N66      N22      DIV      B←DSUB
B←DADD   B←MPY    B←DIV    B←UIIT   UII      MA4      UIIT     NEWMA
P221     MT1      COMIOC   IOCS     B←COMI   B←IOCS   C←PMA    MYPMA
OLAPRN   B←OLAP   PMV2     *UIIT    TRR      P211     F$UII    QNEWMA
MC1      B←FTN2   MYFORT   FTST

OK:
```

For DOS/VM, the LISTF command prints the letter O followed by the device number upon which the UFD resides if the user is an owner or types the letter N followed by the device number upon which the UFD resides if the user is a nonowner. The concept of owner and nonowner is described in Section 2 under the heading "File Access" and is associated with the DOS/VM commands PASSWD and PROTEC.

LISTING Filename

Opens a file for writing on File Unit 2, usually as a listing output file for the compiler or assembler. The file is assigned the name Filename in the current UFD. LISTING is an internal command. This command has the same effect as OPEN Filename 2 2.

NOTE

If no LISTING command has been entered, PMA and FTN automatically open a file named L←XXXX as the listing file (if listing is requested). (XXXX is the first four letters of the source filename.)

LFTN

Invokes a version of the FORTRAN compiler that can perform Sector 0 optimization. The name LFTN is the name of this version on the master disk; it may be changed by the user in his copy of CMDNCO. LFTN is an external command. For further operating details, refer to the description of the FTN command.

LOAD

Loads and starts LOAD, Prime's Linking Loader. LOAD has a command structure and, therefore, a single entry point. LOAD is an external command. For an example of the use of LOAD, refer to Section 3. For a complete discussion of LOAD, refer to the Program Development Software User Guide.

A number of versions of loader are available on the original master disk. The versions of the loader are:

Name	Function
LOAD	(Loader 60000-63777) P-Register = 61000 Normally used with 32K DOS.
LOAD20	(Loader 20000-23777) P-Register = 21000 Normally used with 16K DOS.
LOADAP	Paper tape loader.
HILOAD	(Loader 174000-177777) P-Register = 175000 Normally used to load programs longer than 32K.

Other than the functional and configuration differences noted above, the internal function and user interface is the same. Any version of the loaders may be used on a system configuration equal to or greater than the one specified. If the user chooses, he can rename his particular load command using the CNAME command after deleting the "old LOAD". (e.g., CNAME LOAD40 LOAD). The user is cautioned that if his programs make use of FORTRAN COMMON, then he must be careful to load the correct version of LOAD to avoid a part of the loader being loaded over part of the COMMON area.

LOGIN ***DOS/VM***

LOGIN is the command the user must type at the terminal to obtain access to the DOS/VM system. Syntax:

```
LOGIN  Ufdnam [passwd]  [Device]
```

where Ufdnam must be a valid UFD name on any of the disks available to the system, Passwd is an optional argument that specifies the owner or nonowner password, and Device is an optional argument that specifies logical device numbers to be searched for Ufdnam.

If the UFD has a password, the user must supply it at LOGIN time.

When LOGIN is successful, the user is attached to the UFD specified by Ufdnam. The time-accounting registers for the user are cleared, and some initialization is performed on the users 'virtual machine' (i.e., VRTSSW, ASRCWD, and SVCSW are initialized), then a login message is printed at the terminal and at the supervisor terminal.

The LOGIN command sets the virtual control word to 0 indicating that both input and output are to be from the user terminal. Examples:

```
LOGIN  JHNDOE
```

logs in the user and the UFD, JHNDOE, is attached.

```
LOGIN  JHNDOE  GEMINI
```

logs in the user and attaches the UFD, JHNDOE, if the password GEMINI is correct.

A typical system response to this login at the terminal and at the system supervisor terminal is:

```
JHNDOE(2) LOGGED IN AT 12'39  0304
```

The number in parenthesis is the line number of the user terminal (e.g., in this case, (2)).

The prompt:

```
OK,
```

is printed at the terminal in addition to the login message.

The user may give the ATTACH command, as under DOS. The name given in the argument to the LOGIN command is remembered and printed upon LOGOUT, no matter which UFD is currently attached.

LOGOUT ***DOS/VM***

LOGOUT (or LO) is the last command the user issues when giving up access to the system.

During LOGOUT, all user files are closed, all devices ASSIGNed to the user's terminal are released, the UFD is detached, and a logout message is printed at the user's terminal and at the supervisor terminal. The syntax is:

```
| LOGOUT |
|        |
| LC     |
```

Example:

```
LOGOUT
```

Typical response at the terminal and at the supervisor console:

```
JHNDOE(2) LOGGED OUT AT 13'16 0304
TIME USED = 00'37  03'01  00'54
```

The first number after "TIME USED =" is the connect time in hours and minutes; the second number is CPU time in minutes and seconds, and the third number is paging time in minutes and seconds.

MACHK

Causes the Prime computer that DOS is configured upon to be run in machine check mode. MACHK is an external command. Unless the command specifies otherwise, DOS normally operates out of machine check mode; DOS/VM normally operates in machine check mode.

MAGSAV, MAGRST MAGNETIC TAPE - FILE UTILITIES

MAGSAV and MAGRST are utility programs which move disk files to nine-track magnetic tape and vice versa. The files may be SAM, DAM, segment directories, UFD's, or an entire disk. Whenever a directory is specified, the directory and all components (the subtree) are transferred.

Logical Tapes

A logical tape consists of a header record, a file mark, file records, and two file marks. A logical tape may span multiple physical tapes or a single physical tape may contain multiple logical tapes. The header record contains the tape name, data, and revision number. All tape records are 512 words long, the maximum size permitted by DOS/VM.

Tree Names

A disk file appears on tape as a record containing a tree name followed by as many data records as are required for the file. The tree name contains the path from the file specified by the user to the current file. When an entire disk is saved, all tree names begin in the MFD. For example, an ordinary SAM file might have a tree name of MFD>UFD>JUNK or MFD>UFD>SUBUFD>JUNK.

MAGSAV

Requests information in the following order:

TAPE UNIT:	The proper response is the physical unit number of the disk or the partition.
ENTER LOGICAL TAPE NUMBER:	The response is 1 for the first logical tape, 2 for the second, etc. MAGSAV rewinds the tape, then positions itself correctly. A response of 0 implies the tape is already positioned correctly and MAGSAV takes no action.
TAPE NAME:	Any six-character name.
DATE:	The response format is MMb00bYY where b represents a blank.
REV NO:	An arbitrary number.
NAME:	NAME asks the user what to save. The response is either a file name or one of the alternate action commands: $A, $I, $R, $Q. $A changes the home UFD via an attach, e.g., $A USER3 PASSWD5. $Q and $R each terminate a logical tape and return to the operating system. $R also rewinds the tape. $I causes an index to be printed at the terminal. $I followed by a blank and level number indexes to the level indicated. For example, $I 3, prints an index of the MFD, any UFD's and any Filemanes. NAME: will be typed whenever writing has completed so that further writing may be requested or the current logical tape may be terminated. If the user does not respond correctly to the query NAME, or when the operation specified is complete; MAGSAV again asks NAME:. The user then must give another action command.

To save an entire disk, the user must respond to the query name with MFD. To save a UFD, the user must attach ($A) to the MFD then give the name of the UFD that is to be saved. To save a file with the UFD, the user must attach to the UFD (e.g., $A Ufdnam) and then give the name of the file.

MAGRST

All restore operations take place in the home UFD. MAGRST asks for the tape unit and logical tape number exactly as in MAGSAV. MAGRST then prints the name, date, and revision on the user terminal and asks:

READY TO RESTORE — The responses are YES, NO, PARTIAL (abbreviated Y, NO, PA), or NW level. YES will cause a restore of the entire tape. NO will cause a request for another tape unit and logical tape combination. PARTIAL will permit a restore of part of the tape. NW followed by a level number gives an index of the magnetic tape but does not write it to disk.

TREE NAME: — This is typed when a partial restore is requested. The response is in the form

NAME1>NAME2>.... NAMEn

Any file on the tape whose tree name begins with the sequence entered will be restored. Example - All tree names in a save of the entire disk begin with MFD. The tree name to restore UFD would be MFD>Ufdnam. The tree name to restore a file would be MFD>Ufdnam>Filename

PHYSICAL END OF TAPE

When this condition is encountered in either MAGSAV or MAGRST, a message is logged on the console and a new tape unit is requested. The new unit may be the same as the old unit.

ERRORS

Tape read or write errors are retried five times and are then considered unrecoverable. Both recovered and unrecovered errors are logged. The first record on a tape is not retried.

EXAMPLES OF MAGSAV

```
OK: MAGSAV
GO
TAPE UNIT : 1
ENTER LOGICAL TAPE NUMBER: 0
TAPE NAME: DUD
DATE: 05 23 75
NAME: MFD
NAME: $R
OK:
```

Another example of MAGSAV:

User input is underscored.

```
OK,  STARTUP  0  4
OK,  A MFD  XXXXXX
OK,  MAGSAV
GO
TAPE UNIT: 0
ENTER LOGICAL TAPE NUMBER: 0
TAPE NAME:  MD7V1
DATE:  05 18 75
REV NO: 2
NAME:  $I 3
NAME:  MFD
```

MAGSAV responds to each command input and again asks for name

(During the save, MAGSAV lists directories and files saved for example)

```
MFD
MFD >CMDNC0
MFD >DOS
MFD >CPU
MFD >CPU >OP3FLT
MFD >CPU >P221B
...
MFD >LPRCDR
MFD >LPRCDR >PCRDO3
```

When the listing is complete, MAGSAV again asks:

```
NAME:  R$
```

The MAGSAV rewind command causes the tape to be rewound and exit to be made to command level.

```
OK,
```

EXAMPLE OF MAGRST

```
OK,  MAGRST
GO
TAPE UNIT:  1
ENTER LOGICAL TAPE NUMBER: 1
NAME:  MCARCH
DATE:  05-06-75
REV NO:  1
REEL NO:  1
YOU ARE NOT ATTACHED TO AN MFD
READY TO RESTORE:  NW 3
```

Response to NW 3 is to list the contents of MCARCH as follows:

```
MFD    > CMDNCO
MFD    > DOS
MFD    > CPU
MFD    > CPU    > OP3FLT
...
MFD    > SMLC   > M5374K
INDEX COMPLETE
OK:                Tape is not restored to disk in this case.
```

Another example of MAGRST:

```
OK:  MAGRST
GO
TAPE UNIT:  1
ENTER LOGICAL TAPE NUMBER:  1
NAME:  ADMIN
DATE:  05-23-75
REV NO:  1
REEL NO:  1
READY TO RESTORE:  YES

OK:
```

MAKE

MAKE creates a disk for any disk supported by DOS or DOS/VM. MAKE runs under both DOS and DOS/VM. Under DOS/VM, the disk to be created by the MAKE command must be ASSIGNed before MAKE is invoked. MAKE creates a DOS disk that has the following:

DSKRAT
MFD
BOOT
DOS
CMDNC0

The MAKE program writes the bootstrap (BOOT) into Record 0 of the disk.

To run MAKE, type the command:

MAKE

The response is:

PHYSICAL DISK

The user then must type the number of the physical disk to be created. This disk must not be a disk connected to DOS by the STARTUP command. Possible disk numbers are listed in Table 3-1.

It is recommended that when running MAKE under DOS, all running disks be write protected except the disk to be created by MAKE. Under DOS/VM, it is recommended that only the disk to be created by MAKE be assigned to the terminal.

After the user specifies PHYSICAL DISK, MAKE then types:

RECORDS (OCTAL)

The user types the number of records, in octal, that are to be included in the file structure part of the disk pack. Under most conditions, the entire disk pack is used for the file structure. Possible parameters for RECORDS for using the entire disk are:

Disk	Octal Records
Diskette	460
1.5 million word disk	6260
3.0 million word pack	14540
30 million word disk	176700
128 thousand word fixed head disk (32 track)	400
256 thousand word fixed head disk (64 track)	1000
512 thousand word fixed head disk (128 track)	2000
1025 thousand word fixed head disk (256 track)	4000

If the 30-million word disk is partitioned, other RECORDS parameters may be:

Partition	Disk Number	Records
2 head (default)	XX025X	14540
2 head (explicit	XX065X	14540
4 head	XX125X	31300
6 head	XX165X	46040
8 head	XX225X	62600
10 head	XX265X	77340
12 head	XX325X	114100
14 head	XX365X	130640
16 head	XX425X	145400
18 head	XX465X	162140
20 head	XX525X	176700

Table 4-7. RECORDS Parameters for 30-Million Word Disk

In Table 4-7, the X's represent octal digits that must be set appropriately. (Refer to Table 4-5).

For a 128 thousand word fixed head disk, a diskette or a 1.5 million word disk pack, the user can type carriage return and MAKE defaults to the correct number of records for that disk. MAKE echoes the user input as follows:

DEVICE NUM <number> RECORD COUNT <number> where <number> is a one of the above octal numbers

OK?

If the number is correct, type YES in response to the OK? query followed by CARRIAGE RETURN. If not, type NO followed by CARRIAGE RETURN and MAKE requests the input again. After the number of records are specified, MAKE then asks the question:

VIRGIN DISK?

If the user answers YES followed by CARRIAGE RETURN, MAKE writes the first RECORD-COUNT records of the disk with the first word of each record set to the record address and with a record size of 448 words. This action also writes a valid hardware checksum for each record. If the user answers NO followed by CARRIAGE RETURN, MAKE does not initialize the records. The records need not be initialized if all the records have been initialized by a previous run of MAKE. However, it is strongly recommended that the user answers YES to the VIRGIN DISK ? question at each invocation of MAKE.

MAKE then asks the question:

VERIFY DISK

If the user types YES, MAKE reads every record in the file system part of the disk to verify that each record can be read.

During the reading and the writing of all records, MAKE displays the record number it is processing in the DATA lights. When done, MAKE types DISK CREATED and returns to the operating system, which types OK,. If any disk write errors occur, MAKE retries nine times. Each error results in an error message of the form:

DISK WT ERROR device # DOS record # status

See Appendix J for a description of status.

If the write is not successful after ten tries, MAKE aborts, prints the message, UNRECOVERED ERROR, and returns to the operating system. If a DISK-NOT-READY status is detected, a single disk error message is typed with a status of 177776. The software then retries to write, waiting for the disk to become ready. If the user runs MAKE, the answers YES to VIRGIN DISK ? and VERIFY DISK ?, it is possible to find out immediately if there is any problem in the file structure part of the disk pack.

After MAKE is run, the user must use FUTIL to copy *DOS16, *DOS24, and *DOS32 from UFD DOS on a master disk to UFD DOS on the newly created disk. The BOOT file in the MFD that is read from the disk by the control panel boot expects these files to be in UFD DOS in order to bootload DOS using the newly created disk pack. The user must also use FUTIL to copy DOSEXT from UFD CMDNC0 on a master disk to UFD CMDNC0 on the newly created disk.

It may be desirable to use part of a disk pack for the file structure and part for the paging device under DOS/VM. The user must follow the directions given in Section 6.

MCG Filename

Translates results of microcode assembly into proper code for the ROM simulator. MCG is an external command.

MDL

Punches paper tapes of specified sections of memory in a self loading format that can be read by the panel LOAD operation (or equivalent operation). MDL tapes load into the same memory locations from which they are punched. MDL is an external command, refer to Program Development Software User Guide.

NUMBER

Invokes a utility program that numbers or re-numbers statements in a BASIC program. NUMBER is an external command. For further information, refer to the BASIC manual.

OPEN Filename Funit Key

Opens the specified File Unit (1-16), associates it with the specified Filename, and assigns a Status according to the Key. OPEN is an internal command.

The argument, Key, for OPEN is significant. Key consists of octal values for the type of file, and the action to be taken when the file is opened. The format of Key is as follows;

Bits	Meaning
1-5	New file (file type) key, octal values are:
	000X File is sequential threaded file (SAM)
	200X File is sequential directed file (DAM)
	400X File is a SAM segment directory
	600X File is a DAM segment directory
	1000X File is a UFD
11-16	Action Key, octal values are:
	1 Open for reading
	2 Open for writing
	3 Open for reading and writing

PASSWD Owner-Password [Nonowner-Password]

Under DOS/VM, the PASSWD command replaces any existing passwords in the current UFD with two new passwords. The first is the owner password; the second is the nonowner password. The nonowner password is optional, and if it is not specified, then the PASSWD command functions in the same way as it does under DOS. The PASSWD command must be given by the owner while attached to the UFD. A nonowner cannot give this command. Example:

```
OK, A JHNDOE OLDPW

OK, PASSED US THEM
```

If a nonowner attempted the above PASSWD command, the message:

```
NO RIGHT
```

is printed.

Under DOS, the password command also replaces any existing passwords in the current UFD. However, only one password, the owner password, may be given. Example: OK: ATTACH JHNDOE
PASSWD US

PROTECT ***DOS/VM***

A user (hereafter called the owner) has the ability to open his file directories to other users giving restricted access rights to his files. This declaration of access rights can be made on a per file basis. Access rights to a file are declared and specified through the PASSWD and PROTEC commands. The syntax of this command is:

PROTECT Filename Key1 Key2

Filename is the name of the file to be protected.

Key1 is an integer that specifies the owners access rights to Filename.

Key2 is an integer that specifies the nonowners access rights to Filename.

Possible values and their associated meaning for Key1 and Key2 are:

0 No access of any kind allowed
1 Read only
2 Write only
3 Read and write
4 Delete only
5 Delete, truncate and read
6 Delete, truncate and write
7 All

Example:

OK, PASSWD US THEM
OK, PROTEC MYPROG 7 1
OK, PROTEC OLDLIS 7 7

Gives the owner all access rights to MYPROG and gives nonowners read-only access rights to MYPROG, and gives both owners and nonowners all access rights to the file OLDLIS.

The following brief example is intended to give a user an idea of the use of the PASSWD and PROTEC commands:

```
LOGIN JHNDOE
JHNDOE (2) LOGGED IN AT 10'25 02255

OK, PASSWD JHNDOE US THEM          gives owner password US and
                                   nonowner password THEM to
                                   UFD JHNDOE

OK, LISTF

UFD = JHNDOE  2  0

TIMING FUNCT MNEMOS MYPROG OLD

OK, PROTEC TIMING  7  0            Gives JHNDOE all access
                                   nonowners no access to TIMING

OK, PROTEC MYPROG  7  1            Gives owner access = all
                                   nonowners access = read

OK, PROTEC OLD     7  7            nonowners access = all
                                   owner access = all

LO

JHNDOE (2) LOGGED OUT AT 10'34 02255
TIME USED = 00'03  00'04  00'01

OK, LOGIN MSMYTH

MYSMYTH (2) LOGGED IN AT 11'34 02255

OK, A JHNDOE THEM

OK, LISTF

UFD = JHNDOE  2  N

TIMING FUNCT MNEMOS MYPROG OLD

OK, DELETE TIMING

TIMING NO RIGHT                    MYSMYTH, who is a nonowner, cannot
                                   even read timing since he has no
                                   access.

ER! ED MYPROG

GO                                 MYSMYTH can enter editor and
P2;                                read MYPROG since read access
.NULL.                             has been granted.
C   PROGRAM TO TEST DATA
C   JOHN DOE  02 02 75
```

```
INPUT
C  WITH CHANGES INSERTED BY MSMYTH
;
EDIT

FILE MYPROG
MYPROG NO RIGHT
?
Q  Might as well quit
OK,
ED  OLD
GO
;
INPUT
C CHANGES BY MSMYTH WILL BE RECORDED HERE
;
EDIT
FILE  OLD
OK,
```

MSMYTH attempts to change MYPROG; he seems to have succeeded.

Cannot change file since write access has been denied by JHNDOE

Since all access has been granted for OLD, the changes are made successfully.

PM (POST MORTEM)

Prints contents of the RVEC vector (described later in this section). DOS first prints labels for the items in RVEC, then on the next line prints the values in the same order. PM is an internal command. Example:

```
OK: REST CSETV1
OK: PM
SA,EA,P,A,B,X,K=
000100 011100 001000 000000 000000 000000 000000
OK:
```

PMA Filename [1/A]

Loads the Prime Macro Assembler and starts assembly of a source file Filename from the current UFD. A is the A register setting that specifies listing detail and input/output devices. If A is not specified, the default value is:

A	000777	Normal listing detail, all input and output files on disk

For other values, refer to the Program Development Software User Guide and the PMA Reference Manual (MAN 1673).

Unless it is preceded by BINARY and LISTING commands, the assembler automatically opens Unit 2 to write a binary file named B←XXXX, and opens Unit 3 to write a listing file named L←XXXX, where XXXX is the first four letters of the input filename. The assembler closes any units that it opens. (Units opened by BINARY and LISTING commands are not closed.) PMA is an external command.

PRERR

Prints the message stored in ERRVEC and the first six locations of ERRVEC in octal. The PRERR command is useful in debugging a program. On encountering an error condition, DOS or DOS/VM sets up an internal vector called ERRVEC with several pieces of information. One of these pieces is an error message, unless the user has called a system subroutine with a non-zero alternate return. Refer to Appendix I for a description of ERRVEC.

Using the system subroutine ERRSET (Refer to Section 5), a user may set the content of the error message and have the message printed or not printed depending upon the alternate return being zero or nonzero in a user subroutine. If the user routine was the last routine to set ERRVEC, PRERR prints the user-stored message.

PSD

Loads and starts the interactive debugging program which assumes control and waits for a command string. For details, refer to the Program Development Software User Guide. To return to DOS, enter the command string R 30000, R 50000 or R 70000 (under DOS) or pressing CTL-P under DOS/VM. PSD occupies location 60000 to 65777.

PSD20

PSD20 is a version of PSD for 16K DOS. PSD20 occupies locations 20000 - 25777.

PTCPY

Loads PTCPY, a utility program that duplicates and verifies paper tapes using the high speed reader-punch. Operation is controlled by P-register and sense switch settings. PTCPY is an external command. For details, see The Operators Guide.

To return to DOS, restart the processor from the systems terminal at location '30000; or '50000; or '70000, depending on system configuration. Under DOS/VM, the command ASSIGN PTR and ASSIGN PTP must be given before PTCPY is invoked.

PTRED

Edits files read from paper tape. Refer to the description of ED and the Program Development Software User Guide.

RESTORE Filename

Restores a program Filename in the current UFD from disk to high-speed memory using the SA and EA values SAVEd with the file. The SAVEd RVEC parameters (refer to next side head) are also loaded into RVEC to be ready for a START command RESTORE is an internal command. Example:

```
RESTORE *GENFIL
OK:  PM
SA: EA,P,A,B,X,K=
000200 011710 001000 075072 000001 177771 006001
OK:
```

RVEC Parameters: The commands RESTORE, RESUME, SAVE, PM and START process a group of optional parameters associated with the DOS RVEC vector. These parameters are stored on disk along with a starting address (SA) and ending address (EA), for every program saved by the SAVE command.

Initial values for these parameters are usually specified in the SAVE command or by the loader's SAVE command that stored the program on disk.

Each parameter is a 16-bit processor word, represented by up to six octal digits.

Parameter	Processor Memory Location	Definition
SA	-	Starting Address (first memory word used by program)
EA	-	Ending Address (last memory word used by program)
PC	7	P Register (Program Counter)
A	1	A Register (Arithmetic)
B	2	B Register (Arithmetic)
X	0	Index Register
Keys	--	Status keys associated with INK, OTK instructions

The RVEC parameters are optional in the command string. Any item that is specified replaces the previous value in RVEC, which is saved with the program. Thus, for any parameters that are not specified, the value previously stored in RVEC is saved with the program.

RVEC parameters specified in RESUME or START commands replace the previous values in RVEC. Also, when a program returns to DOS through the EXIT subroutine, RVEC is loaded from the processor values in effect at the time of exit. Only the SAVE command alters the values of RVEC stored on disk with the program.

RESTORE returns a program from disk to memory and loads the SAVE parameters into RVEC in preparation for a START command.

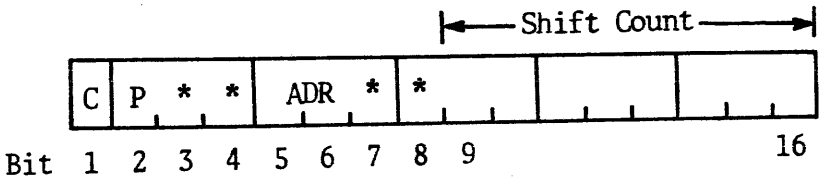
START sets the processor registers to the values currently stored in RVEC and starts execution at location PC. The START command may also specify new parameters to override (and replace) the previous values in RVEC.

RESUME combines the functions of RESTORE and START.

PM lists the current values of the RVEC parameters.

External commands have RVEC parameters that can be modified at the time the command is started. (e.g., PMA Filname 1/740).

Keys: The item [Keys] among the RVEC parameters refer to the processor status keys handled by the INK and OTK instructions. These are represented by a single 16-bit word in the following format:



where:

C = State of C (Carry) bit

P = Arithmetic mode; 0 - single precision, 1 double precision

* = Must be zero

ADR = Addressing Mode:

Bits 5-16	Mode
0XXX	16K Sectored
2XXX	32K Sectored
6XXX	32K Relative
4XXX	64K Relative

Shift Count = Bits 9-16 of Location 6, which may contain a normalize shift count

If [Keys] are not specified, they are unchanged.

RESUME Filename [PC] [A] [B] [X] [KEYS]

RESUME is equivalent to a RESTORE and START command, combined. The program Filename in the current UFD is loaded from disk to high-speed memory, using the SAVEd values of SA and EA. RVEC is loaded from the SAVEd RVEC parameters or from any new values specified in the command string. The processor registers and keys are then set from RVEC and the program is started at location PC. RESUME is an internal command.

RTOSRA

Establishes an RTOS mapped random access file using 128-word structure. Usage instructions are printed when the command is invoked. RTOSRA is an external command.

RT128F

RTOS off-line utility command to read and write non-DOS (128-word segment format) disk. See RTOS manual for details.

CAUTION: Do not use this command under DOS or DOS/VM except as directed in the RTOS User Guide.

SAVE Filename SA EA [PC] [A] [B] [X] [KEYS]

Saves the content of high-speed memory from SA (starting address) to EA (ending address) as a file named Filename in the current UFD. SAVE is an internal command.

As discussed at the beginning of this Section, the contents of the DOS vector RVEC are saved along with the program. RVEC may be altered by new parameters specified in the SAVE command string before the program and parameters are stored. For any parameters that are not specified, the previous values of RVEC remain in effect and are stored with the program. The RVEC parameters are used to initialize the processor registers and keys when the program is RESTOREd or RESUMEd. Example:

```
SAVE PROG1 200 2600 1000 0 0 0 0
```

This command saves the program PROG1 from locations '200 to '2600. Execution starts at '1000, the A, B and X registers are set to 0, and all bits of the keys are set to 0 (carry bit is 0, arithmetic mode is single precision, addressing mode is 16K Sectored, and shift count is zero). Start this save at '200 to preserve address links in Sector 0.

All FORTRAN programs begin with ELM, Enter Load Mode. If macro assembler (PMA) users have ELM as the first instruction in the program, there is no need to set the keys after loading. The preferred way to save a memory image is to use the loader SAVE command.

SHUTDN

The DOS command SHUTDN performs tasks necessary to shutting down DOS in an orderly manner. Refer to the examples in Section 3 for use of the DOS SHUTDN command. SHUTDN has extended capabilities in DOS/VM, refer to Section 6. SHUTDN is an internal command. SHUTDN must also be entered before closing down a DOS system or changing disk packs. The command does some incidental DOS housekeeping that makes sure all the information in memory buffers is written to disk properly.

SIZE Filename

Gives the size of Filename in records. Example:

```
OK:  SIZE  PRPLOT

GO
000002  RECORDS IN FILE
OK:
```

SLIST Filename

Prints the content of the file Filename at the users terminal. SLIST is often used to obtain source listings of short program or data files. SLIST is an external command.

SORT

Sorts an ASCII file and writes the sorted file in the current UFD. The SORT program requests input and output files, number of columns, and starting and ending columns for the sorting operation. SORT is an external command. Its format is:

```
SORT
SORT  BRIEF
SORT  SPACE
SORT  MERGE
```

The options BRIEF; SPACE; or MERGE, or a combination of these options, may be entered following the command SORT. Only two options can be implemented at a time. (Note, the names of the options may be abbreviated: BR, SP or ME.) The meaning of these options when specified is as follows:

Option	Meaning
BRIEF	SORT program messages are not printed at the users terminal.
SPACE	Any blank lines are deleted from the SORT output file.
MERGE	A maximum of ten unsorted files can be merged at a time. The SORT program asks for the names of the merged files. The user at the terminal types the filenames on one line, separated by spaces.

Reverse Sorting

Sorting can be specified to be in descending order by typing the letter R separated by a space after the ending column of the desired keys.

Command File

The SORT command can be run from a command file, since it does not close Unit 6.

```
OK, SORT
GO
SORT PROGRAM PARAMETERS ARE:
  INPUT FILE NAME -- OUTPUT FILE NAME FOLLOWED BY
  NUMBER OF PAIRS OF STARTING AND ENDING COLUMNS.
INFILE OUTPUT 3
  INPUT PAIRS OF STARTING AND ENDING COLUMNS
  ONE PAIR PER LINE--SEPARATED BY A SPACE.
  FOR REVERSE SORTING ENTER "R" AFTER DESIRED
  ENDING COLUMN--SEPARATED BY A SPACE.
1 5
15 25
30 35

BEGINNING SORT


 PASSES        3    ITEMS      266


OK,
```

Respond to the first inquiry with the input file name, output file name and number of pairs of columns.

Respond to each inquiry with the appropriate starting and ending column numbers (character positions).

During operation of the SORT program, the console DATA indicators display a count of the number of passes completed. When the sort is completed, SORT prints the number of passes and number of items (lines in the input file), and return to DOS.

SPOOL [Filename] ***DOS/VM***

Queues a copy of a file in the UFD SPOOL for off-line printing on the DOS/VM system configuration line printer. SPOOL allows a user to get output printed without specifically ASSIGNing the printer and then waiting until the printing operation is complete before being able to issue another command at the terminal. Example:

```
OK, FTN  MYPROG

OK,  SPOOL   L←MYPRO

GO

YOUR SPOOL FILE IS PRNT10

OK, FTN  NEWPRG
```

Using SPOOL, terminals are not tied up waiting for the printer, and terminals and files can be used while copies of the files are being printed.

One terminal in the DOS/VM configuration must be dedicated to running SPOOL (i.e., SPOOL must be logged-in as if it were a user).

The SPOOL program copies the specified file Filename into the UFD SPOOL and changes its filename to prevent naming conflicts. Each file in the UFD SPOOL is deleted after it is printed.

There are two programs in the UFD SPOOL that control printer output. The SPLCEN program prints files in UFD SPOOL on the Centronics line printer, and the SPLMPC program prints files in the UFD SPOOL on the high-speed line printer. Only entire files can be printed using SPOOL.

After a successful file copy of Filename to the UFD SPOOL, SPOOL responds:

YOUR SPOOL FILENAME IS PRNTnn

where nn is a two-digit decimal integer that is part of the new filename in the SPOOL directory.

SPOOL Output Format

The SPOOL filename is printed on the header page before the file is printed. A header page with the UFD of origin and Filename is generated as the first page of each SPOOL job.

Errors

An INPUT FILE ERROR or LINE SIZE ERROR results in an error message at the terminal dedicated to SPOOL. The file causing the error is deleted and SPOOL continues, printing the next file in UFD SPOOL.

A DISK FULL error results in an error message being printed at the user terminal, and the copy of the file, Filename, in the UFD SPOOL is deleted without being printed.

DEFAULT

SPOOL typed with no Filename argument opens File Unit 2 for writing in the SPOOL directory. SPOOL responds by typing the SPOOL filename. A user program may then write directly to File Unit 2. When the program finishes, the user may close File Unit 2 with the command:

```
CLOSE ALL
```

or may close the unit at the end of the program. The file produced in this manner is subsequently printed by SPLCEN or SPLMPC, whichever is appropriate.

Example 1:

```
OK, PMA FILE
GO
NO ERRORS
OK, SPOOL  L←FILE
GO
YOUR SPOOL FILENAME IS  PRNT1Ø
OK, DELETE L←FILE
OK,
```

In the above example, a user assembles the program named FILE and generates the listing, L←FILE, and a binary file B←File. The user then issues the command: SPOOL L←FILE. This command causes L←FILE to be copied to PRNT1Ø in the UFD SPOOL. If SPLMPC, or SPLCEN, is running in UFD SPOOL (logged in on another terminal); the file PRNT1Ø is printed on the line printer. The user may then DELETE the file L←FILE since what is desired is a printed copy of the listing.

Example 2:

```
OK, SPOOL
GO
YOUR SPOOL FILENAME IS PRNT1Ø
OK, PMA FILE
GO
NO ERRORS
OK, CLOSE ALL
OK,
```

In this example, the user issues the command SPOOL with no Filename argument before invoking the assembler. SPOOL opens PRNT1Ø in UFD SPOOL for writing on File Unit 2. The command: PMA FILE first checks if Unit 2 is open. Because Unit is open, PMA does not open and write L←FILE in the users UFD; instead, it outputs the assembly listing to the file already open on File Unit 2, which happens to be PRNT1Ø in the UFD SPOOL. When the assembly is done and PMA returns to command level, PMA leaves File Unit 2 open. The user gives the CLOSE ALL command which closes Unit 2. (Note, the user could have not given this command but proceeded to invoke a series of assemblies or compilations with the result that a listing file consisting of a series of listings would have been created). After File Unit 2 is closed, and if SPLCEN or SPLMPC is running, the file PRNT1Ø is printed on the line printer.

LOGGING IN AND STARTING UP THE SPOOL DAEMON

To start SPLMPC or SPLCEN at a terminal, proceed as follows (user input is underlined):

```
LOGIN  SPOOL  SPLOUT
OK, ASSIGN CENPR
OK, RESUME SPLCEN
GO
```

For the High-Speed Printer:

```
LOGIN  SPOOL  SPLOUT
OK, ASSIGN PR1
OK, RESUME SPLMPC
GO
```

The SPLMPC or SPLCEN program looks for files with names: PRNT1Ø, PRNT11, PRNT12, etc. in the current UFD and prints them if any exist. SPLMPC or SPLCEN always processes files first-in, first-out (FIFO). An INPUT FILE ERROR or a LINE SIZE ERROR results in an error message at the terminal from which SPOOL was logged in. The spool-file (PRNT1Ø, 11, ... etc.) is deleted and the next file is processed.

If a user decides not to print a file that is queued for printing by SPOOL, it is possible to ATTACH to the UFD SPOOL and DELETE the appropriate file (named PRNTnn where nn is a number 10, 11, 12, etc.).

If printing of the file to be deleted has started, the attempt to DELETE it fails. However, the user can request the operator at the terminal from which SPOOL was logged-in to stop the file from printing; or the operator may stop printing a file if he perceives that the file is incorrect. The operator or user proceeds to do this by:

CONTROL-P	(Operator presses QUIT)
QUIT, CLOSE ALL	
OK, ATTACH SPOOL SPLOUT	
OK, DELETE PRNTnn	(nn = number for whatever file is appropriate)
OK, RESUME SPLMPC	(or RESUME SPLCEN)
GO	

CAUTION

SPOOL reuses available names. Thus, after PRNT1Ø is printed and deleted, the name PRNT1Ø is available for use by SPOOL again and may be given to a subsequent Filename argument in a subsequent SPOOL request. If deleting files from the UFD SPOOL, be sure you are deleting the right one.

START [PC] [A] [B] [X] [Keys]

Initializes the processor's registers and keys from the command line (or from RVEC, for any values not specified in the command line) and starts execution at location PC. This command assumes a program has been loaded into memory by a previous RESTORE, RESUME, or LOAD command. START is an internal command.

START can also restart a program that has returned control to DOS (for example, because of an error, a FORTRAN PAUSE or CALL EXIT statement). If START is typed without a value for PC, the program resumes at the PC value where execution was interrupted. To restart the program at a different point, specify an octal starting location as the PC value.

STARTUP Pdisk0 [Pdisk1] [Pdisk2] [Pdisk3] ... [Pdisk8]

Initializes the configuration of disk drives by relating physical disk drive numbers to DOS logical disk unit numbers. STARTUP is an internal command. Physical device numbers for disks are those shown in Table 3-1.

The logical-to-physical assignment depends on the order in which the physical device numbers are listed as parameters in the STARTUP command. The physical device number specified in the Pdisk0 position is assigned as logical disk unit 0, the physical device number specified in the Pdisk1 position is assigned as logical disk unit 1, and so on.

The number of parameters indicate to DOS the number of logical drives assigned to the system. Example:

STARTUP 3 0 1

This command makes the following logical/physical disk assignments:

Logical Unit	Physical Unit
0	3
1	0
2	1
3	Not Assigned

STARTUP has some extended capability in DOS/VM; refer to Section 6.

STATUS

Lists the current UFD - the logical device upon which the UFD resides, the low boundary of DOS plus buffers, the open file units, and the physical-to-logical device correspondence. STATUS lists physical device numbers as described in Table 3-1 and Section 3. Example:

```
OK:  STATUS

UFD=GOUDY   0

DOSLO  67000  FUNITS

LDEV      PDEV
 0         01
 1         00
 2         04
```

In DOS/VM, the STATUS command prints the packnames of the disks also. Rather than typing the current UFD, the login UFD is typed. Example:

```
OK,  STATUS

UFD=GOUDY  0

FUNITS

DISK      LDEV    PDEV
TSDISK     0       01
COMMAND    1       00
DUD        2       04
```

The disk name (Packname) is the name of the DSKRAT on that disk pack. The DSKRAT name can be changed by the CNAME command.

The STATUS command may be used to monitor the usage of DOS/VM. When entered at the system terminal, the STATUS command prints status information that consists of the information given at the user terminal and; in addition, prints the paging device, the command device, and a list of current logged-in users, and the devices that each user has currently assigned. Disks assigned to a user are printed as: DISK <octal number>. Following each user name in the list, the user terminal number and the numbers of the physical disks currently used by the user are printed. A disk is considered to be in use by a user (1) if his home UFD or current UFD resides on the disk or (2) if the user has opened a file on that disk. Some typical instances where the STATUS command must be used are:

1. Prior to mounting a new disk pack to determine what physical disk assignments are available.

2. After a request that all users release a given disk or disks to determine that they have done so before shutting down the given disk or disks.

3. As a check that all users have logged out before shutting down DOS/VM. (No harm to the system results if the users of a particular disk are still logged-in when the disk or the system is shut down. However, users will be disconnected and the message: DISK d DETACHED; YOUR FILES CLOSED will be printed at their terminal.

Example: STATUS comamnd and response when issued at system terminal.

STAT

```
USER = SYSTEM  0

FUNITS

DISK            LDEV        PDEV
TS#1             0           250
DUD#2            1         40250
COMAND           2             4
ETCH3            3             0
TS#2             4         20250
MD6V2            5             6
TRANS            6         50250
DOSDVM           7         60250

PAGDEV = 10250 COMDEV = 250

USER    LINE     PDEVS
SPOOL     3      40250  PR1
PDAVIS    4      20250  MT0    DK1
PDAVIS    5      40250  DK20 DK21
GRUBIN    7      50250
GREATA    8      250
SPORER    9      20250
```

SVCSW ***DOS/VM***

The SVCSW command controls the handling of SVC instructions in the virtual memory environment. Syntax:

```
SVCSW  0
SVCSW  1
SVC    0
SVC    1
```

The normal mode (SVC 0) causes all SVC instructions to be trapped and processed by the system supervisor. If the SVC SWITCH is ON (SVC 1), almost all SVC instructions cause a virtual trap, and SVC instructions are handled through the users location 65. The class of SVC instructions always processed by the DOS/VM operating system regardless of the SVCSW command are those determined by FUNCTION code 5XX (currently the SVC's are RREC, WREC (for reading and writing to disk), TIMDAT (for obtaining the time and date from DOS/VM), and RECYCL.

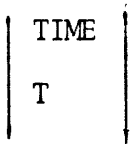
The SVC switch is initialized to 0 by the LOGIN command. The SVCSW command allows a special version of DOS called VDOS32 to be run under DOS/VM.

TIME ***DOS/VM***

The TIME command prints the current value stored in the time accounting registers. The three values printed are the same as the three values in the logout message, namely:

Connect time	(hours, minutes)	Time since LOGIN
Compute time	(minutes, seconds)	Time accumulated executing commands or using programs (does not include paging time).
Paging time	(minutes, seconds)	Time accumulated taking page faults and bringing pages into memory.

The syntax is:



Example:

```
T
```

Typical Response:

```
00'51  01'32  00'28
```

UNASSIGN ***DOS/VM***

The UNASSIGN command may be entered at the user terminal to which a device is currently ASSIGNed or may be entered at the supervisor terminal. The UNASSIGN command, entered at the system terminal, unconditionally deassigns the peripheral assigned to any user. Entered from a user terminal, UNASSIGN only deassigns the device that was previously assigned to the user. On selected devices, this command turns off the device and clears the associated I/O buffers. Syntax:

```
| UNASSIGN Device |
|                 |
| U  Device       |
```

where device is a previously assigned device named as shown in Table 4-4.

From the system terminal, this command is useful to release a device if the user who assigned it has forgotten to log out and has left his terminal.

Example:

```
UNASSIGN CENPR
```

unassigns the Centronics printer.

```
U  PTR
```

unassigns the paper tape reader.

Before a disk may be assigned to a terminal, it must not be assigned to either DOS/VM or another user. If the disk is assigned to DOS/VM, it must be released using the SHUTDN command at the supervisor terminal. A disk that has been ASSIGNed by a user cannot be entered as an argument in the STARTUP command. The supervisor terminal can UNASSIGN a device that may be assigned. Devices ASSIGNed by another user are released when the LOGOUT command is invoked by that user.

USERS ***DOS/VM***

The USERS command prints the number of users currently logged into DOS/VM. This command is useful to estimate how fast response will be. A user may decide on the basis of this command whether or not to run a long program. Example:

```
OK, USERS

 USERS = 7

OK,
```

VDOS32 ***DOS/VM***

Starts a version of DOS that may be run under DOS/VM.
Example:

```
SVC 1
VDOS32
```

VRTSSW ***DOS/VM***

The VRTSSW command allows setting the virtual sense switches. The 16-bit configuration specified by the numeric parameter of the VRTSSW command, is stored and made available. Syntax:

```
VRTSSW [XXXXXX]
```

where XXXXXX is an octal number that specifies a 16-bit configuration; when XXXXXX is not specified, it value is 0.

The 16-bit configuration specified by the numeric parameter of VRTSSW is stored and made available to the user when a program written in PMA executes an INA 1620 (read sense switches) instruction. For further details, see the Assembly Language Reference Manual.
Example:

```
V 10100
```

The virtual sense switches are initialized to 0 by the LOGIN command.

WARNING: The instructions, skip on sense switch, always refer to the actual sense switches, not to the virtual sense switches.

* [Comment]

The internal command named, *, indicates the beginning of a comment line. * must be followed by a space and have a correct command line form (1 to 3 names followed by 0 to 9 octal parameters).
Example:

```
* PROGRAM.1  JULY.14.1974
```

This command is useful to include comment lines in COMMAND files.

SECTION 5

FILE SYSTEM AND TERMINAL I/O LIBRARY

INTRODUCTION

DOS and DOS/VM provide the user with a powerful and general file system. The key definitions of SEARCH, PRWFIL, and ATTACH are complicated. To keep things straight, the definitions of these file system subroutines have been written with mnemonic keys (Refer to Appendix C).

CALLING AND LOADING LIBRARY SUBROUTINES

When a FORTRAN user calls a subroutine, a call to the required subroutine is automatically inserted in the FORTRAN object program by the compiler.

After a FORTRAN or Macro Assembler main program is loaded, library subroutines are loaded by using the loaders LIB (or LI) command.

CALLING SEQUENCE NOTATION

The following conventions apply to the FORTRAN calling sequence formats described in the rest of this section. For assembly language calling, refer to the PMA Manual.

Items in capital letters are to be reproduced literally. Items in initial caps are variables to be assigned actual names or values by the user. For example, the calling sequence:

```
CALL CMREAD (Array)
```

means that the user must enter CALL CMREAD, as specified, but may coin his own array name. Common abbreviations such as Funit, Ldisk, etc. are defined in the Foreword.

File names and UFD names used in routines such as ATTACH, RESTOR, etc., may be specified either by a Hollerith string or an array name. The Hollerith form allows the file or UFD name to be expressed literally in a 6-character Hollerith string such as 6HFILNAM. If an array name is used instead, it must designate a 3-word integer array that contains the file or UFD name. For example, the user could specify an array NAM that contains filename FILNAM in the following form:

```
NAM(1) FI
NAM(2) LN
NAM(3) AM
```

In either the Hollerith or Array form, the name must be specified as exactly six characters; if the actual name has fewer than six characters, it must be left justified and the Hollerith string (or array) filled with space characters (b). For example, the filename FIL1 should be treated as follows:

```
6HFIL1bb  or  NAM(1) or FiL1  = FI
              NAM(2)          = L1
              NAM(3)          = bb
```

Numerical values such as Funit, Ldisk, etc. must be specified by decimal integer expressions. The error return Altrtn must be set by an ASSIGN statement to the value of a statement number within the user's program. (The form $n, where n is the statement number, is also acceptable.)

Example: The ATTACH subroutine has the general form:

```
CALL ATTACH (Ufd, Ldisk, Password,  Key, Altrtn)
```

The user might code an actual call to this subroutine as follows:

```
CALL ATTACH (6HUSER1 , 0, PWD, 000001, $50)
```

where:

a. 6HUSER1, literally identifies the user's UFD, 'USER1 ';

b. The USER1 UFD is on logical disk unit 0 (Ldisk is an integer)

c. The user stores his current password in 3-word integer array PWD

d. The variable KEY (declared as integer mode in the user's program) controls the way that the file is referenced and the home-UFD setup;

e. In case of uncorrectable error, control passes to statement label 50 in the user's program.

FILE SYSTEM AND TERMINAL I/O SUBROUTINES

DOS and DOS/VM provide a collection of subroutines that simplify disk input-output, permit user programs to communicate with the DOS supervisor and file structure, and provide various input-output and control functions. The subroutines SAVE, RESTOR, RESUME and ATTACH have the same effect as the commands of the same name, but they are called from, and return control to, a user program. The calling sequence provides the parameters that are normally entered from the terminal. Most routines, like SEARCH, SAVE and RESTOR are implemented by code within DOS itself. A small interlude program executes a supervisor call to DOS or DOS/VM to do the work in each case.

Subroutines from this group are loaded from the main library file FTNLIB if they are called in a user's FORTRAN program. They are described in this section in alphabetical order:

ATTACH	D$INIT	PRWFIL	T1IN	TNOU
BREAK$	ERRSET	RECYCL	T1OU	TNOUA
CMREAD	EXIT	RESTOR	TIMDAT	TOOCT
C1IN	FORCEW	RESUME	T$CMPC	UPDATE
CNAME	GETERR	RREC	T$LMPC	WREC
COMINP	GINFO	SAVE	T$MT	
COMANL	PRERR	SEARCH	T$SLC	

ATTACH

The ATTACH subroutine has the same effect as the ATTACH internal command. The calling sequence is:

```
CALL ATTACH  (Ufd, Ldisk, Password, Key, Altrtn)
```

Definition of ATTACH

To access files, the file system must be attached to some User File Directory. This implies that the file system has been supplied with the proper file directory name and either the owner or nonowner password, and the file system has found and has saved the name and location of the file directory. After a successful attach, the name, location, and owner/nonowner status of the UFD is referred to as the current UFD. As an option, this information may be copied to another place in the system, referred to as the home UFD. The user gets owner status if he gives the owner password or gets nonowner status if he gives the nonowner password. The owner of a file directory can declare on a per-file-basis what rights a nonowner has over the owner's files. The nonowner password may be given only under DOS/VM. (Refer to the description of the commands PASSWD and PROTECT in Section 4 for more information.)

In attaching to a directory, the subroutine ATTACH specifies where to look for the directory. ATTACH either specifies a file directory in the master file directory (MFD) on a particular logical disk or a file directory in the current UFD, or the home UFD as the directory to be attached. ATTACH may specify a file unit number on which a segment directory is open. In the segment directory reference, the file directory to be attached is the one whose beginning disk address is given by the word at the file pointer of the file unit. Syntax:

ATTACH is used as in the following call:

```
CALL ATTACH (NAME, LDISK, PASSWORD, KEY, ALTRTN)
```

KEY is composed of two subkeys that are combined additively. They are REFERENCE and SETHOME. All calls require a REFERENCE subkey. The REFERENCE subkeys are shown in the following table:

REFERENCE	Octal Value	Meaning
MFDUFD	0	attach to NAME in MFD on LDISK
CURUFD	2	attach to NAME in current UFD.
SEGUFD	4	attach to directory whose location on the disk is given by the word at the file pointer of the file unit given by NAME (1). The file unit opened previously by a call to SEARCH must be an open segment directory.

The SETHOME subkeys are required on all calls; these subkeys are shown in the following table:

SETHOME	Octal Value	Meaning
---	0	do not set home UFD to current UFD after attaching.
SETHOM	1	set home UFD to current UFD after attaching.

The meaning of the remaining parameters on a call to ATTACH is as follows:

NAME — If the key is 0 and NAME is 0, the home UFD is attached.

If the reference subkey is MFDUFD or CURUFD, NAME is either a six-character Hollerith expression or the name of a three word array that specifies a Ufdname to be attached.

If the reference subkey is SEGUFD, NAME is a file unit on which a segment directory is open.

LDISK — If the reference subkey is MFDUFD, LDISK is the logical disk on which the MFD is to be searched for UFD NAME. LDISK must be a logical disk that has been started up by the STARTUP command. The special LDISK octal code 100000 signifies: search all started-up logical devices in order 0, 1, 2 ... n and attach to the UFD whose NAME appears in the MFD of the lowest numbered logical device. The special LDISK octal code 177777 signifies: search the MFD of the Ldisk currently attached for NAME.

If the reference subkey is CURUFD or SEGUFD, or NAME is 0, LDISK is ignored and is usually specified as 0.

PASSWORD — If the reference subkey is MFDUFD, CURUFD, or SEGUFD, PASSWORD is either a six-character Hollerith expression or the name of a three-word array that specifies one of the passwords of UFD NAME. If the password is blank, it is specified as three words of two blank characters.

ALTRTN — An integer variable assigned the value of a label is the user's FORTRAN program to be used as an alternate return in case of error. If this argument is 0 or omitted, an error message is printed and control returns to DOS or DOS/VM if any error should occur while using ATTACH.

A UFD attached through a segment directory reference does not have a name. On LISTF, such a UFD is listed with a name of six asterisks.

If an error is encountered and control goes to Altrtn; ERRVEC(1), a DOS vector, is set to the error type as follows:

Code	Message
AH	Name NOT FOUND
AL	No UFD ATTACHED
AR	Not a UFD (detected by DOS/VM only)

A user obtains ERRVEC through a call to GETERR. The error 'Name NOT FOUND' is printed if one of the following errors occur:

1. KEY bad.
2. NAME is not found in the specified directory.
3. LDISK is out of range or not started up.
4. In a segment directory reference, NAME(1) is a closed unit or the unit is at end of file.

If the error BAD PASSWORD is obtained, the alternate return is never taken, and both the home UFD and current UFD are set to 0 to indicate that no UFD is attached. This feature is a system security measure to prevent a user from writing a program to try all possible passwords on a UFD.

Examples of ATTACH:

```
CALL ATTACH  ('JHNDOE',  -1, 'JJJ', 0, ERR)
```

Searches for the UFD, JHNDOE, in the MFD (as specified in the Key) on the current logical device. If JHNDOE is found and the password, JJJ, matches the recorded password, then UFD JHNDOE is attached. The current UFD (now JHNDOE) is not set as the home UFD (This is specified in the Key). The DOS vector that points to the current UFD is set to this new directory.

BREAK$

The calling sequence is:

```
CALL BREAK$ |(.TRUE.) |
            |(.FALSE) |
```

Under DOS/VM, the BREAK$ routine, called with argument .TRUE:

```
CALL BREAK$  (.TRUE.)
```

inhibits the CTL-P or BREAK$ key from interrupting a running program.

```
CALL BREAK$  (.FALSE)
```

enables the CTL-P or BREAK$ characters to interrupt a running program. The LOGIN command initializes the user terminal so that the CTL-P or BREAK keys cause interrupt.

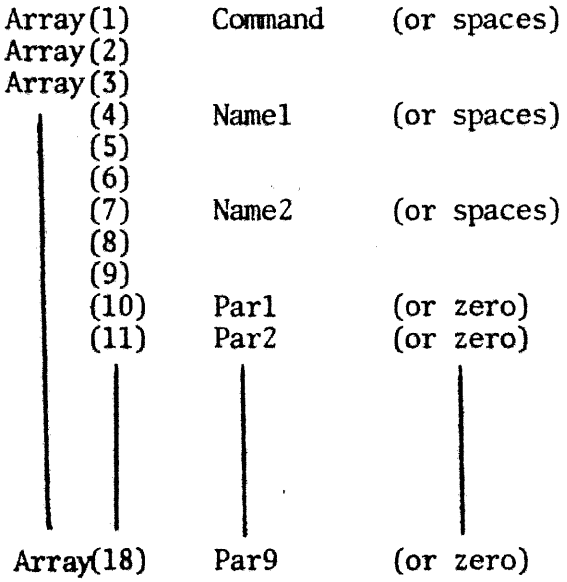
Under DOS, BREAK$ has no effect.

CMREAD

The calling sequence is:

```
CALL  CMREAD  (Array)
```

CMREAD reads 18 words which represent the last command line typed into Array as follows:



The command line may then be accessed directly from ARRAY. The 'Name's are normally UFD's or filenames and the 'Par's are parameters.

C1IN

This routine gets the next character from the terminal if the command stream comes from a terminal or from a Command (text) file if the command stream comes from there. The calling sequence is:

```
CALL  C1IN (Char)
```

The next character of a command file is read and loaded into Char . If the character is CR or a line feed, Char is set to NL (new line).

The command file must be specified by a preceding COMINPUT command. The user's program that contains the call to C1IN is usually started by one of the command lines in the command file.

CNAME

The CNAME routine allows the same action at user program level as the CNAME command allows at command level. The calling sequence is:

```
CALL  CNAME (Oldnam, Newnam, Altrtn)
```

CNAME changes the name of Oldnam in the current UFD to Newnam. The user must have owner status to the UFD. The arguments are:

Oldnam	A filename to be changed
Newnam	The new filename for Oldnam
Altrtn	If not 0, control goes to Altrtn if any error occurs. If 0, an error message is printed and control returns to DOS/VM if any error occurs.

If an error is encountered and control goes to Altrtn, ERRVEC(1) is set to the error type as follows:

Code	Message
CA	Newnam BAD NAME
CZ	Newnam DUPLICATE NAME
SH	Oldnam NOT FOUND
SI	Oldnam IN USE
SL	NO UFD ATTACHED
SX	Oldnam NO RIGHT

A user obtains ERRVEC through a call to GETERR. CNAME does not run on DOS, only DOS/VM.

COMINP

The COMINP routine allows the user to perform the same action at program level as the user command COMINPUT allows at command level. Refer to Section 4 for details of the COMINPUT command. Briefly, COMINP causes DOS or DOS/VM to read commands from a file rather than a terminal. The calling sequence is:

```
CALL  COMINP  (Name, Funit, Altrtn)
```

The arguments are:

Name	Either a three-word array containing a filename of a command file or the words TTY, CONTIN, or PAUSE.
Funit	A File Unit number (range 1 to 16; 1-15 under DOS) that is to be used for reading the command file.
Altrtn	If not 0, control goes to Altrtn in the event of an error while opening Name. If 0, an error message is printed and control returns to the operating system in the event of an error while opening Name.

If an error is encountered and control goes to Altrtn, ERRVEC(1) is set to the error type as follows:

Code	Message
SD	UNIT NOT OPEN
SH	Name NOT FOUND
SI	Name IN USE
SI	UNIT IN USE
SL	NO UFD ATTACHED
SX	Name NO RIGHT

A user obtains ERRVEC through a call to GETERR.

COMANL

COMANL causes a command line to be read from the terminal or from a command file, depending upon the source of the command stream. The calling sequence is:

```
CALL  COMANL
```

Example:

```
CALL COMANL
CALL CMREAD  (ARRAY1)
```

Assume a user wishes to get a file name typed at the terminal via a program. The user program calls COMANL followed by CMREAD (ARRAY1). The filename is contained in the first three words of the array, ARRAY1.

D$INIT

The D$INIT routine is called to initialize a disk device. The calling sequence is:

```
CALL  D$INIT  (Pdisk)
```

when Pdisk is the physical disk number to be initialized. D$INIT initializes the disk controller and performs a seek to cylinder 0 on Pdisk. D$INIT must be called prior to any RREC or WREC calls. Pdisk must be assigned by the DOS/VM ASSIGN command before calling this routine. D$INIT is not normally used by users but is used by system utilities such as FIXRAT, COPY, and MAKE

ERRSET

ERRSET sets ERRVEC, a system vector, then takes an alternate return or prints the message stored in ERRVEC and returns control to the system. ERRSET has these forms:

```
CALL ERRSET (Altval, Altrtn)                          (Form 1)
CALL ERRSET (Altval, Altrtn, Messag, Num)             (Form 2)
CALL Errset (Altval, Altrtn, Name, Messag, Num)       (Form 3)
```

In Form 1, Altval must have value 100000 octal and Altrtn specifies where control is to pass. If Altrtn is 0, the message stored in ERRVEC is printed and control returns to the system. Forms 2 and 3 are similar; therefore, the arguments are described collectively as follows:

Altval	A two-word array that contains an error code that replaces ERRVEC(1) and ERRVEC(2). Altval(1) must be not equal to 100000 octal.
Altrtn	If Altrtn is nonzero, control goes to Altrtn. If Altrtn is zero, the message stored in ERRVEC is printed and control returns to DOS.
Name	The name of a three-word array containing a six-letter word. This name replaces ERRVEC(3), ERRVEC(4), and ERRVEC(5). If Name is not an argument in the call, ERRVEC(3) is set to 0.
Message	An array of characters stored two per word. A pointer to this message is placed in ERRVEC(7).
Num	The number of characters in Message. Num replaces ERRVEC(8).

If a message is to be printed, six characters starting at ERRVEC(3) are printed at the terminal and ERRVEC(8) characters from a message pointed to by ERRVEC(7) are printed at the terminal. If ERRVEC(3) is 0, only the message pointed to by ERRVEC(7) is printed. The message stored in ERRVEC may also be printed by the PRERR command or the PRERR subroutine. The contents of ERRVEC may be obtained by calling subroutine GETERR.

EXIT

The EXIT subroutine provides a way to return from a user program to DOS or DOS/VM, that prints OK; (or OK,) at the terminal and resumes control. The calling sequence is:

```
CALL EXIT
```

The user may open or close files or switch directories, and restart a FORTRAN program at the next statement by typing S (i.e., START).

FORCEW

Calling sequence is:

```
CALL  FORCEW  (0, Funit)
```

The FORCEW subroutine, under DOS/VM, immediately updates to the disk the file that is currently open on Funit. Normally this action is not needed since the system automatically updates all changed file system information to the disk at least once per minute. Under DOS, the FORCEW routine acts as a no-operation (i.e., it does nothing).

GETERR

A user obtains ERRVEC contents through a call to GETERR.

Calling sequence is:

```
CALL  GETERR  (Xervec, n)
```

GETERR moves n words from ERRVEC into Xervec.

On an alternate return:

error code (returned in B register);
ERRVEC(1)

alternate value (returned in A register);
ERRVEC(2)

On a normal return:

PRWFIL:
ERRVEC(3) record number
ERRVEC(4) word number

Key of read/write convenient:
ERRVEC(2) no. of words transferred

SEARCH:
ERRVEC(2) File type

GINFO

Calling sequence is:

```
CALL GINFO  (Xervec, n)
```

GINFO moves n words in Xervec.

The information acquired is:

DOS	DOS/VM
1. low bound of DOS and buffers	1. 0
2. high bound of DOS	2. 0
3. count of started devices (from STARTUP command)	3. max. possible device count
4. data word count of current device	4. data word count of device 0
	5. data word count of device 1
	.
	.
	.
	12) data word count of device 8

PRERR

Calling sequence is:

```
CALL PRERR
```

PRERR prints an error message on the users terminal.

Example of Use

A user wants to retain control on a request to open a unit for reading if the name was not found by SEARCH. To accomplish this, the user calls SEARCH and gets an alternate return. He then calls to GETERR and determines if another type of error occurred other than NAME NOT FOUND. The user then wishes DOS or DOS/VM to print the error message, but to keep control, he calls PRERR.

PRWFIL

Definition of PRWFIL

PRWFIL is used to read, write, and position a file open on a file unit. A typical call to PRWFIL will read into a user buffer N words from a file open on Funit starting at the file pointer in the file. A user may instead move the file pointer forward or backward relative to its current position or move the file pointer to an absolute position in the file. The two operations of reading and positioning or writing and positioning may be combined in a single call with position occurring either before or after the read or write operation. Syntax:

PRWFIL is used as in the following call:

```
CALL PRWFIL (KEY, FUNIT, PBUFFER, NWORDS, POSITION, ALTRTN)
```

KEY is composed of three subkeys that are combined additively. They are RWKEY, POSKEY, and MODE. The POSKEY is only required on those calls in which positioning is requested. Subkeys whose values are 0 may be omitted from the call. The PRWFIL call may be represented as:

```
CALL PRWFIL (RWKEY+POSKEY+MODE,FUNIT,PBUFFER, NWORDS,POSITION,ALTRTN)
```

The RWKEY subkeys are shown in the following table.

RWKEY	Octal Value	Meaning
PREAD	1	read NWORDS from FUNIT into a buffer whose address is in PBUFFER.
PWRITE	2	write NWORDS from a buffer whose address is in PBUFFER to FUNIT.

The POSKEY subkeys are shown in the following table:

POSKEY	Octal Value	Meaning
PREREL	0	move the file pointer of FUNIT POSITION words relative to the current position before reading or writing.
POSREL	20	move the file pointer of FUNIT POSITION words relative to the current position after reading or writing.
PREABS	10	move the file pointer of FUNIT to an absolute position specified by POSITION(1) and POSITION(2) before reading or writing.
POSABS	30	move the file pointer of FUNIT to an absolute position specified by POSITION(1) and POSITION(2) after reading or writing.

The MODE subkeys are shown in the following table:

MODE	Octal Value	Meaning
--	0	read or write NWORDS.
PCONV	400	read or write a convenient number of words. The number transferred is NWORDS. See explanation below.

The meaning of the remaining parameters in a call to PRWFIL are as follows:

FUNIT a file unit number 1 to 16 (1 to 15 for DOS) on which a file has been opened by a call to SEARCH or a command. PRWFIL actions are performed on this file unit.

PBUFFER reading or writing is done beginning at the memory location whose address is contained in PBUFFER. PBUFFER is therefore a pointer to a user buffer or array used in reading or writing. (Use the LOC function of FORTRAN to generate a pointer.)

NWORDS If the mode subkey is 0, NWORDS is the number of words to be transferred to or from a file unit and a user buffer. If NWORDS is 0, no words are transferred.

If the MODE subkey is PCONV, NWORDS is the maximum number of words to be transferred. The number actually transferred is a number between 1 and NWORDS that is convenient and fast for PRWFIL to transfer. If NWORDS is 0, no words are transferred. The user can find how many words were transferred from ERRVEC(2).

For either mode, NWORDS may be between 0 and 65535.

POSITION If the POSKEY is PREREL or POSREL, POSITION is a single signed integer word for relative positioning. Positioning is forward and backward from the file pointer depending on the sign of POSITION. If POSITION is 0, no positioning is done.

If the key is PREABS or POSABS, POSITION is a two-word integer array (record-number, word-number) for absolute positioning. If POSITION is (0,0) (both values 0), the file pointer is moved to the beginning of the file.

ALTRTN An integer variable assigned the value of a label in the user's FORTRAN program to be used as an alternate return in case of uncorrectable errors. If the argument is 0 or omitted, an error message is printed and control returns to DOS or DOS/VM if any error occurs while using PRWFIL.

If an error is encountered and control goes to ALTRTN, ERRVEC(1) is set to the error type. This is a two-character code as follows:

Code	Message	Meaning
PD	PRWFIL UNIT NOT OPEN	bad key or file unit not open for read/write.
PE	PRWFIL EOF	end of file reached on read or position.
PG	PRWFIL BOF	beginning of file reached on read or position.
DJ	DISK FULL	no room left on disk

A user obtains ERRVEC through a call to GETERR, which is described in this section. A user may wish to handle one type of error and have the system type all other error messages and return to DOS or DOS/VM. The user can call PRERR to print the error message that would have been printed without ALTRTN.

On a PRWFIL EOF or PRWFIL BOF error, ERRVEC(2) is set to the number of words left to be transferred in the read or write requests. On all normal returns from PRWFIL, ERRVEC(3) and ERRVEC(4) are set to the file pointer of the file as a two-word array (record-number, word-number). On a call with the PCONV subkey, ERRVEC(2) is set to the number of words read.

On a DISK FULL error, the file pointer is set to the value it had at the beginning of the call. The user may, therefore, delete another file and restart the program by typing START. This feature only works with DOS/VM.

The positioning operation of PRWFIL is now discussed in more detail. For every open file, the system maintains a file pointer of a file. Because a file may contain more than 65,535 words, the largest unsigned integer that can be represented in a 16-bit word, the file pointer occupies two words. The method of representation chosen is two words, the first of which is the record number and the second of which is a word number. Each record contains 440 words of data, corresponding to one disk record so the word number has a range of 0 to 439. The record number has a range of 0 to 32767. When a file is opened by a call to SEARCH, the file pointer is set so the next word read is the first word of the file. The position pointer contains record 0, word 0, or briefly (0,0). If the user calls PRWFIL to read 490 words and does no positioning, at the end of the read operation the file pointer is (record 1, word 50) or briefly (1,50). The user is cautioned that the number of data words per record (440), although the same for all disk-like devices in DOS, is not promised to be 440 for all times and all devices. The user must call GINFO to determine the data record size. The data record size is needed to convert a (record number, word number) representation of the file pointer into one number (possibly in floating point notation).

A call to read or write N words causes N words to be transferred to or from the file starting at the file pointer in the file. Following a call to transfer information, the file pointer is automatically moved to the end of the data transferred in the file. Using a POSKEY of PREABS or POSABS, the user may explicitly move the file pointer to (record number, word number) before or after the data transfer operation. Using a POSKEY of PREREL or POSREL, the user may explicitly move the file pointer forward POSITION words from the current position, if POSITION is positive. Using a POSKEY of PREREL or POSREL, the user may move the file point backward POSITION words from the current position, if POSITION is negative. The maximum position that can be moved in the call is therefore plus or minus 32767 words. Positioning takes place before or after the data transfer, depending on the key. If NWORDS is 0 in any of the calls to PRWFIL, no data transfer takes place, so PRWFIL does only a pointer position operation. On normal returns from PRWFIL, ERRVEC (3) and ERRVEC (4) contain the file pointer as (record number, word number).

The MODE subkey of PRWFIL is now discussed. In most cases, the user wants to transfer a specific number of words on a call to PRWFIL. In these cases, the MODE is 0 and is normally omitted in PRWFIL calls. In some cases, such as in a program to copy a file from one file directory to another, a buffer of a certain size is set aside in memory to hold information, and the file is transferred a buffer-full at a time. In this case, the user doesn't care how many words are transferred at each call to PRWFIL, as long as the number of words is less than the size of the buffer set aside in memory.

In fact, the user would prefer to use a number of words convenient to the system, so that his program runs as fast as possible. The PCONV subkey is used for this purpose. In the call to PRWFIL; NWORDS, or less, are transferred. The number of words transferred is a number convenient to the system. The number of words actually transferred is put in ERRVEC (2).

For an example of PRWFIL use in a program, refer to Appendix H.

RECYCL

The RECYCL subroutine is called under DOS/VM to tell the system to cycle to the next user. It is a "I have nothing to do for now" call. Under DOS, RECYCL does nothing. The calling sequence is:

```
CALL  RECYCL
```

RESTOR

RESTOR has the same effect under program control as the RESTORE command. The calling sequence is:

```
CALL  RESTOR  (Vect, Filename, Altrtn)
```

RESTOR performs the inverse of the SAVE operation. The SAVEd parameters for a Filename previously written to disk by SAVE are loaded into the 9-word array VECT. The program itself is then loaded into high-speed memory using the starting and ending address provided by VECT (1) and VECT (2).

If an error is encountered and control goes to Altrtn, ERRVEC(1) is set to the error type as follows:

Code	Message
SH	Name NOT FOUND
SI	UNIT IN USE
SI	Name IN USE
SL	NO UFD ATTACHED
SX	NO RIGHT
PE	PRWFIL EOF

RESUME

RESUME has the save effect under program control as the RESUME command. The calling sequence is:

```
CALL  RESUME  (Filename)
```

RREC

Subroutine RREC reads one disk record from a disk into a buffer in memory. RREC may optionally scatter the information recorded in the disk record into one, two or three buffers. For instance, the first X words of the record may be sent to buffer A, the next Y words of the record may be sent to buffer B and the last Z words of the record may be sent to buffer C. Before RREC is called, the disk must be assigned by the DOS/VM ASSIGN command and D$INIT must be called to initialize the disk.

The RREC routine is not used normally by users but is used by system utilities such as FIXRAT, MAKE and COPY.

The calling sequence is:

```
CALL  RREC  (Bptrs, Blen, N, Ra, Pdisk, Altrtn)
```

where:

Bptrs — is an array of dimension N giving a list of buffer pointers.

Blen — is an array of dimension N giving a list of buffer lengths (number of words).

N — bits 9-16 contain the dimension of Bptrs and Blen (1, 2, or 3)
bit 1 set means do current record address check
bit 2 set means ignore checksum error
bit 3 set means read an entire track beginning Ra into a buffer 3520 words long beginning at the buffer pointed to by Bptrs (1). This feature may only be used if RREC is running under DOS and is reading a driver connected to the 4001/4002 controller.

Ra — is the disk record address. Legal addresses depend on the size of the disk.

Size	Ra Range
Floppy disk	0-303
1.5M disk pack	0-3247
3.0M disk pack	0-6495
30M disk pack	0-64959
128K fixed head disk	0-255
256K fixed head disk	0-511
512K fixed head disk	0-1023
1024K fixed head disk	0-2047

Pdisk — is the physical disk number of the disk to be read. Pdisk numbers are the same numbers available for use in the ASSIGN and STARTUP commands.

Altrtn — is an integer variable in the user's program to be used as an alternate return in case of uncorrectable disk errors. If this argument is 0 or omitted, an error message is printed if any error occurs.

If an error is encountered and control goes to Altrtn, ERRVEC is set as follows:

Code	Message	Meaning
ERRVEC(1) = WB ERRVEC(2) = 0	on supervisor terminal: 10 times DISK RD ERROR Pdisk Ra Status on user terminal: UNRECOVERED ERROR	disk hardware or WRITE PROTECT error
ERRVEC(1) = WB ERRVEC(2) = CR	on user terminal: 10 times DISK RD ERROR Pdisk Ra Status followed by UNRECOVERED ERROR	current record address error

See Appendix J for a description of status error codes.

Notes: The sum of the buffer lengths, Blen, must be between 0 and 448. If this number is not 448 and Pdisk is 20-27, (diskette) a checksum error always is generated. This can be bypassed by setting N bit 2 = 1 to ignore the checksum error. No check is made for legality of Ra.

On a DISK NOT READY, RREC simply waits for the disk to become ready under DOS/VM and prints no message. Under DOS, RREC prints a single error message and waits for the disk to become ready.

On any other read error, an error message is printed at the system terminal followed by a seek to cylinder zero and a reread of the record. If 10 errors occur, the message UNRECOVERED ERROR is typed to the user or Altrtn is taken.

The parameters Bptrs and Blen allow scatter-gather operation for up to three of the physical records.

SAVE

SAVE has the same effect under program control as the SAVE command. The calling sequence is:

```
CALL  SAVE  (Vect, Filename)
```

The user sets up a nine-word vector VECT before calling SAVE. VECT(1) must be set to an integer which is the first location in memory to be saved and VECT(2) must be set to the last location to be saved. The rest of the vector may be set up at the programmer's option.

		Location
VECT(3)	P Register	7
VECT(4)	A Register	1
VECT(5)	B Register	2
VECT(6)	X Register	0
VECT(7)	Keys	--
VECT(8)	Spare	--
VECT(9)	Spare	--

SAVE writes, to the named disk file, the nine-word vector VECT followed by the memory image starting at VECT(1) and ending at VECT(2).

SEARCH

For some program examples that show the use of SEARCH, refer to Appendix H.

Definition of SEARCH

SEARCH is used to connect a file to a file unit (open a file) or disconnect a file from a file unit (close a file). After a file is connected to a unit; PRWFIL and other routines may be called, either to position the current-position pointer of a file unit (file pointer) or to transfer information to or from the file (using the file unit to reference the file).

On opening a file, SEARCH specifies allowable operations that may be performed by PRWFIL, and other routines. These operations are read only, write only or both read and write.

On opening a file, SEARCH also specifies where to look for the file or where to add the file, if the file does not already exist, and also SEARCH specifies the file is to be opened for writing or both reading and writing. SEARCH either specifies a filename in the currently attached user file directory or a file unit number on which a segment directory is open. In the segment directory reference, the file to be opened or closed is the one whose beginning disk address is given by the word at the current position pointer of the file unit.

On creating a new file, the user specifies to SEARCH the file type of the new file.

The subroutine SEARCH may be used to perform actions other than opening and closing a file. SEARCH may delete a file, rewind a file unit, or truncate a file.

Upon opening a file, SEARCH sets the file pointer to the beginning of the file. Subroutines PRWFIL, and others cause information be transferred to or from the file unit starting at the file pointer of the file. After the transfer, the pointer is moved past the data transferred. A call to SEARCH to rewind a file causes the file pointer to be set to the beginning of the file. Subsequent calls to PRWFIL, and other routines cause information transfer to occur as if the file had just been opened. A call to SEARCH to truncate a file causes all information beyond the file pointer to be removed from the file. This call is useful if one is overwriting a file with less information than originally contained within the file.

Syntax:

SEARCH is used as in the following call:

```
CALL SEARCH (KEY, NAME, FUNIT, ALTRTN)
```

KEY is composed of three subkeys that are combined additively. They are ACTION, REFERENCE and NEWFILE. Not all subkeys are required on every call, and subkeys whose values are zero may be omitted from the call. The SEARCH call may therefore be represented as:

```
CALL SEARCH (ACTION+REFERENCE+NEWFILE, NAME, FUNIT, ALTRTN)
```

All calls require an ACTION subkey. The ACTION subkeys are shown in the following table:

ACTION	Octal Value	Meaning
OPNRED	1	open NAME for reading on FUNIT
OPNWRT	2	open NAME for writing on FUNIT
OPNBTH	3	open NAME for both reading and writing on FUNIT
CLOSE	4	close file by NAME or by FUNIT
DELETE	5	delete file NAME
REWIND	7	rewind file on FUNIT
TRNCAT	10	truncate file on FUNIT

The REFERENCE subkeys are shown in the following table:

REFERENCE	Octal Value	Meaning
UFDREF	0	search for file NAME in the current user file directory as defined by a previous ATTACH and perform the action in the ACTION subkey on the specified file.
SEGREF	100	perform the action specified in the ACTION subkey on the file whose disk location is given by the word indicated by the file pointer of the file unit specified by NAME(1). This file unit must be an open segment directory.

Only those calls to SEARCH that reference a file in a UFD or Segment Directory need the reference key. Calls that reference file units do not need this key, and it is ignored.

The following table lists the NEWFIL subkeys:

NEWFIL	Octal Value	Meaning
NTFILE	0	new threaded (SAM) file
NDFILE	2000	new directed (DAM) file
NTSEG	4000	new threaded (SAM) segment directory
NDSEG	6000	new directed (DAM) segment directory
NEWUFD	10000	new user file directory (SAM)

Only those calls to SEARCH that generate a new file require a NEWFIL subkey. On other calls, this subkey is ignored.

The name of the remaining parameters in a call to SEARCH are as follows:

NAME If the reference subkey is UFDREF, NAME is either a six character Hollerith expression or the name of a three-word array that specifies a filename (existing or not).

If the reference subkey is UFDREF and NAME(1) is -1, the current UFD is opened. NAME = -1 must only be used in configuration with ACTION subkeys 1, 2, or 3. Owner status of the current UFD is required.

If the reference subkey is SEGREF, NAME is a file unit (1-16; 1-15 under DOS) on which a segment directory is open.

On calls in which the ACTION key requires only a file unit to specify the file to be acted-on, NAME is ignored and is usually specified as 1. If -1 is specified, then name is the current UFD.

FUNIT On calls that require a file unit number, FUNIT is a number 1 to 16 (1-15 under DOS). On calls that require no unit number, FUNIT is ignored and usually specified as 0.

ALTRTN ALTRTN is an integer variable assigned the value of a label in the user's FORTRAN program to be used as an alternate return in case of uncorrectable errors (e.g., attempting to open a file that is already open). If this argument is 0 or omitted, an error message is printed; and control returns to DOS or DOS/VM, if any error should occur while using SEARCH.

If an error is encountered and control goes to ALTRTN, ERRVEC (1) is set to the error type, a two-character code as follows:

Code	Message	Meaning
SA	BAD CALL TO SEARCH	some parameter in call is invalid.
SD	UNIT NOT OPEN	attempt to truncate or rewind a file on a closed unit.
SD	Name OPEN ON DELETE	self-explanatory.
SH	Name NOT FOUND	file Name not in UFD.
SI	Name IN USE	file Name is already open.
SI	UNIT IN USE	file unit is already open.
SK	UFD FULL	self-explanatory.
SL	NO UFD ATTACHED	self-explanatory.
SQ	SEG-DIR ERROR	*SEG-DIR ERROR
SX	NO RIGHT	access rights violation.
DJ	DISK FULL	no room left on disk.

*SEG-DIR ERROR: Meaning

1. If attempting to open an existing file in the segment directory means:

   a. the segment directory unit specified in NAME is not open for reading.

   b. the file pointer of the segment directory unit is at end of file, therefore points to no disk address.

   c. the file pointer of the segment directory unit points to a 0 entry.

2. If attempting to open a new file in the current segment directory means:

   The segment directory unit specified in NAME is not open for both reading and writing.

A user obtains ERRVEC through a call to GETERR, which is described in this section. Any of the above errors cause a control to go to ALTRTN. A user may wish to handle one type of error and have the system type all other error messages and return to DOS or DOS/VM. The user can call PRERR to print the error message that would have been printed without ALTRTN.

ERRVEC(2) is set to a file type on a normal return of a call to SEARCH to open a file, using action keys of OPNRED, OPNWRT, or OPNBTH. The codes are:

ERRVEC(2)	File Type
0	threaded file (SAM)
1	directed file (DAM)
2	threaded segment directory (SAM)
3	directed segment directory (DAM)
4	user file directory (SAM)

Access Rights and Call to SEARCH

Under DOS/VM, the access rights of files are checked when a user attempts to open a file through a call to SEARCH. Under DOS, access rights are not checked.

A SEARCH call that creates a new file gives that file default access rights. Default access rights are: owner has all rights; non owner has no rights (Refer to Section 4 for a detailed description of access).

Adding and Deleting Files

The action of SEARCH in adding or deleting files from file directories is now explained in more detail. For references to user file directories, a call to SEARCH to open a file for writing or both reading and writing causes SEARCH to look in the current User File Directory for the file. If the file is not found in the UFD, the file name and beginning disk address of a new file is appended to the UFD, and the file is opened for the appropriate activity. Currently, UFDs are restricted to 72 files. An attempt to open a new file to a full UFD generates the message: UFD FULL. A call to delete a file from a UFD causes the name and beginning disk address to be removed from the UFD and causes the UFD to be shortened.

For references to segment directories, a call to SEARCH to open a file for writing or reading and writing causes SEARCH to examine the word at the file pointer of the referenced segment directory file unit. If the word is not zero, SEARCH considers the word to be a beginning record address of an already created file. SEARCH opens the file for writing or reading and writing. If the word is zero, SEARCH writes the beginning disk address of a new file in that word and opens the file. If the file pointer is positioned at the end of file, the file is lengthened one word and SEARCH writes the beginning disk address of a new file in that word, and opens the file. A call to delete a file from a segment directory causes the beginning disk address of a file at the file pointer of the segment directory to be replaced by zero. The segment directory is not shortened. An attempt to open a file for reading in a segment directory whose file pointer points to zero or whose file pointer is at end-of-file, generates a SEG-DIR error. In no case is the file pointer of a segment directory moved. Generating a segment directory and filling it with files is quite involved. Examples are presented in Appendix H under the title "Example" and in Appendix C.

Closing and Opening Files

On a call to close a file, SEARCH attempts to close file NAME and generates an error message or goes to the alternate return if NAME is found. FUNIT is ignored unless NAME is 0. If NAME is 0, SEARCH ensures that FUNIT is closed. That is, it closes FUNIT if FUNIT is open but does not generate an error message if the file unit is closed. Example:

```
CALL SEARCH (1, 'OBJECT', 1, ERR)
```

Searches for a file, OBJECT, in the current UFD and opens it for reading.

The user is allowed to open the current UFD for reading via a call to SEARCH. The calling sequence for this feature is:

```
CALL SEARCH (1, -1, Funit, Altrtn)
```

This call opens the current UFD for reading on Funit. The user must have owner access rights to the UFD; i.e., the owner password must have been given in the most recent call to ATTACH (or ATTACH command). Control goes to Altrtn if there is no UFD attached, if Funit is already in use, or the user does not have owner rights to the UFD.

T1IN

The calling sequence is:

```
CALL T1IN (Char)
```

T1IN reads a character from the terminal into Char, and echoes CARRIAGE RETURNS for LINE FEEDS. (Also, LINE FEED is returned if CARRIAGE RETURN is typed.)

T1OU

The calling sequence is:

```
CALL T1OU (Char)
```

T1OU types out Char. If Char is LF, both CR and LF are typed.

TNOU

The calling sequence is:

```
CALL TNOU (Array, Nchars)
```

TNOU prints Nchars characters from Array and adds CARRIAGE RETURN and LINE FEED characters at the end of Nchars.

TNOUA

The calling sequence is:

```
CALL  TNOUA  (Array, Nchars)
```

TNOUA prints Nchars characters from Array but does not add a CARRIAGE RETURN and a LINE FEED.

TOOCT

The calling sequence is:

```
CALL  TOOCT  (Number)
```

TOOCT types the ASCII representation of Number converted to octal as an unsigned six-digit number.

TIMDAT

The calling sequence is:

```
CALL  TIMDAT  (Array, Num)
```

TIMDAT may be called to pick up additional useful information, namely the user's unique number on the system and his login UFD name. The information is obtained by increasing the Array size and Num before calling TIMDAT.

TIMDAT returns the date, time, CPU time, and paging time used since LOGIN in an array as follows:

```
Array (1)  two ASCII characters representing month.
  |    (2)  two ASCII characters representing day.  Example:  30
  |    (3)  one ASCII characters representing year.  Example:  4
  |    (4)  integer time minutes.
  |    (5)  integer time seconds.
  |    (6)  integer time ticks.
  |    (7)  integer CPU time used, seconds.
  |    (8)  integer CPU time used, ticks.
  |    (9)  integer paging time used, seconds.
  |   (10)  integer paging time used, ticks.
  |   (11)  integer number of ticks per second.
  |   (12)  user number.
  |   (13)
  |   (14)  six-character login name, left justified.  Example:  MSMITH
  V   (15)
```

Num words of Array are set. This routine only runs under DOS/VM.

T$CMPC

The calling sequence is:

CALL T$CMPC (Unit, Buffer-Address, Word-Count, Instruction, Status-Vector)

The T$CMPC routine is the raw data mover that moves a card of information from the MPC card reader to the user's space.

T$CMPC is called by the IOCS card reader driver I$AC03. The user normally reads cards under program control using either a FORTRAN READ statement or a call to I$AC03 (Refer to the Subroutine Library Manual). However, it is possible to call T$CMPC directly. The arguments to T$CMPC are:

Unit — Card reader number. (Currently ignored. Generally only one card reader is connected to a DOS (DOS/VM) configuration).

Buffer-Address — A pointer to a buffer to hold a card of information read from the card reader.

Word-Count — The number of words to be read from the current card.

Instruction — The instruction required to be sent to the card reader. Valid instructions are:

Instruction		Meaning
100000 (octal)	=	read status
40000 (octal)	=	read card in ASCII format
60000 (octal)	=	read card in BINARY format

Status-Word — is a three-word vector that contains device code, status of reader, and number of words transferred. Possible status of the card reader is as follows:

Octal Value	Condition
200	ON-LINE
40	Illegal ASCII
20	DAX overrun
4	Hopper Empty
2	Motion Check
1	Read Check

Example:

```
40  DO   I  =  1, 23

50  CALL  T$MPC  (0, CARDS, 40, 40000, STATUS)

60  CALL  O$....

    GO to 40
```

Reads an 80 character card of ASCII data and places the contents in CARDS.

Card Reading Operation

Under DOS/VM, card reader input is buffered. The user must insert the card deck in the card reader, then give the command:

```
ASSIGN  CR1
```

About ten cards are read, enough to fill up the input buffer. The user then starts up the program that uses the card reader. If T$CMPC is called and the buffer is empty, the user is placed in INPUT-WAIT state. Later, when the buffer is no longer empty, the user is rescheduled by the operating system and the call to T$CMPC is retried.

The user may issue a status-request call to check if the input buffer is empty. If the buffer is empty, the ON-LINE status is not set. Using this feature, a user may check for input, then read a card if one is available, or do another computation if no card is available.

Under DOS, card reader input is not buffered; and the card reader is never OFF-LINE.

T$LMPC

The calling sequence is:

```
CALL T$LMPC (Unit, Buffer-Address, Word-Count, Instruction, Status-Vector)
```

The T$LMPC routine is the raw data mover that moves information from the user to one line on the MPC line printer.

T$LMPC is called by the IOCS line printer driver O$AL06. The user normally prints lines under program control using either a FORTRAN WRITE statement or a call to O$AL06. However, it is possible to call T$LMPC directly. The arguments are:

Unit	Line Printer unit (currently ignored).
Buffer-Address	A pointer to a buffer to hold information to be printed on the line printer. Information is expected to be packed two characters per word.
Word-Count	Number of words to print on the current line.
Instruction	The instruction required to be sent to the line printer. Valid instructions are:

Instruction (Octal)	Meaning
100000	Read Status
40000	Print a line
20012	Skip a line
20014	Skip to top of page
200XX	Skip on control tape channel

Status-Vector	is a three-word vector that contains device code, status of printer, and a space. Possible printer status is as follows:

Octal Value	Condition
200	ON-LINE
100	Not Busy

Under DOS/VM, line printer output is buffered. If T$LMPC is called and the buffer is full, the user is placed in OUTPUT-WAIT state. Later, when the buffer is no longer full; the user is rescheduled, and the T$LMPC call is retried. The user may issue a status request call to check if the buffer is full. If the buffer is full, then the Not-Busy status is reset. Using this feature, a user program may check that the buffer is not full, then output a line, or do another computation if the buffer is full.

Under DOS, output is not buffered, and control does not return to the user until printing is complete.

T$MT

The calling sequence is:

```
CALL T$MT (Unit, Buffer-Address, Word-Count, Instruction, Status-Vector)
```

The T$MT routine is the raw data mover that moves a record of information from one of four magnetic tape drives to the user address space, or vice-versa. T$MT is called by the IOCS routines concerned with controlling, reading and writing both seven- and nine-track magnetic tapes.

(For details, refer to the Subroutine Library Manual and/or the Magnetic Tape Controller User Guide). The user normally controls, reads, and writes magnetic tape under program control using either FORTRAN READ, WRITE, REWIND, and END FILE statements or calls to the appropriate IOCS driver. However, it is possible to call T$MT directly. The arguments are:

Unit	Magnetic Tape Drive (=0, 1, 2, or 3)
Buffer-Address	A pointer to a buffer from which to read or write a record of information.
Word-Count	Number of words to transfer. This number must be between 0 and 512 words.
Instruction	The instruction request to the magnetic tape driver.

Magnetic tape I/O is not buffered under DOS/VM. A call to T$MT returns immediately before the operation is complete. When the magnetic tape operation is completed, the Status Flag in the user space is set to 0. Therefore, a user program may loop waiting for completion and do another computation while waiting. If a user initiates another call to T$MT before the first call has completed its magnetic tape operation, the second call does not return to the user until the first magnetic tape operation has completed.

Under DOS, T$MT does not return to the user until the magnetic tape operation is completed.

T$SLC

The driver T$SLC is available on the master disk and provides user control of a synchronous multi-line communications device.

Control

The driver is loaded in supervisor space. A user program communicates with the driver via FORTRAN-format calls to T$SLC0. The driver communicates with the user address space via buffers in the user address space specified by the user program. There is a data structure provided by the user that is used by the driver. It is referred to as the control block. The control block is created by the user program in the user address space. It contains pointers to the user status buffer and pointers to buffers containing a message to be transmitted or buffers set to receive a message. The details of the data structure are sumamrized in the subsequent paragraphs. A special control block is required for each line.

The communications lines must be assigned to a user space before they can be used. The proper command is:

```
          ┌0┐
          │1│
ASSIGN SMLC │2│
          └3┘
```

The ASSIGN command is given at the user terminal. One or more lines may be assigned to a user space.

Timing

The user space program runs asynchronously with message transfers. A call to T$SLC0 returns immediately after executing the control function required. The progress of the communication must be monitored by the user program by examination of the user space status buffer contents. For interpretation of the status codes, see the Prime Computer User Guide for Synchronous Multiline Controller (UG-0001 Rev. 2).

Hardware Requirements

The SMLC driver assumes the presence of a 520X synchrous multi-line controller with a 5246 SMLC option. The address of the controller is 56_8.

Software Requirements

DOS/VM: File TSSLC in UFD DVBIN on Master Disk Vol. I is a DOS/VM executable memory image file with the synchronous-line controller option. It can be created by file C←LSLC (see UFD DVSRC on Master Disk Vol. II). In particular, file TMAIN (UFD DVSRC) must be assembled with the B-register set to 20004_8 and the modules that comprise T$SLC0 (refer to the SMLC User Guide) must be locked in memory.

There is a memory conflict among special drivers: the same memory and table entries are used by T$SLC0, the Gould printer/plotter code, and the digital input/output controller code.

User Level Software Responsibilities: A user address space program is given direct control of most of the functionality of the SMLC controller; therefore, the prospective user is assumed to know the User Guide. A specific limitation is that no more than four message blocks may be chained at a time in a given direction (transmit or receive) for a given line.

Controller status is collected as it is produced. This status is moved from interrupt response code buffers in the supervisor address space to user-space buffers at the next possible DOS/VM cycle (after any currently executing and interrupted supervisor code). However, the user bus/program does not get a chance to execute and act on the reported status until its turn in the round-robin cycle. If system usage is heavy enough, there will be excessive delay in line response by the user-level program.

All details of implementation of a communications protocol are left to the user program with one exception: the driver program automatically disables an active transmitter when the LAST CHARACTER OUT status is detected for that line.

Information that is provided the user program in the user program's status buffer consists of all status words received from the controller plus two special codes. One is a code indicating the time at which the LAST CHARACTER OUT (LCT) status was detected by the driver interrupt code. This time is always inserted following a LAST CHARACTER OUT status word in the status data stream. The time is taken from $VCLOK_0$. The value can be related to the (seconds, tics) time value obtained from a call to TIMDAT as follows:

$$\text{LCT time (sec)} = \text{floor } [(\text{status time} - \text{vqutm}_0)/\text{clock}]$$
$$\text{LCT time (tics)} = \text{remainder } [\text{LCT time (sec)}]$$

where: $\text{vqutm}_0 = -60 \quad \text{clock}$
clock = buf (11) of call to TIMDAT

The status time is given modulo ("one minute")

The other status code indicates that the stream of controller status data has overflowed either an internal supervisor buffer or the user program status buffer. If this is detected, status information has been lost. The status buffer overflow code is the integer -1 (supervisor buffer) or -2 (user buffer).

User Calls to the SMLC

The form of the user call to the supervisor is (in FORTRAN):

CALL T$SLC0 (Key, Line, Loc (Block), Nwds)

where: $1 < \text{Key} \leq 5$;

$0 < \text{Line} \leq 3$;

Loc (Block) is the memory address of a buffer used in the call;

Nwds is the word count of Block.

The values and meaning for Key are as follows:

Key	Meaning
1	User control block is undefined. Status information is no longer moved to user program space. The state of controller is not altered. Requires two arguments (key, line).
2	Control block is defined to be "block". The block is structured as in Table 5-1. It defines an area to store status information and, optionally, a message chain for reception or transmission.
3	Buffer Block contains four or five data words to be sent to the controller. These control words configure the line, set line control, define the programmable sync character and optionally set the internal programmable character-time clock. Refer to Table 5-1 for the block structure.
4	Buffer Block contains one word to be used as the next data set control word. See "OTA 01XX" in the SMLC User Guide.
5.	Buffer Block contains one word which is used as the next receive/transmit enable word. See "OTA 14XX" in the SMLC User Guide. Half-duplex looping for odd-even line pairs is not allowed.

Table 5-1. Structure of SMLC Hardware Configuring Block

Word	Meaning
0	Receiver line configuration word. See "OTA 00XX" in the SMLC User Guide.
1	Transmitter line configuration word. See "OTA 00XX" in the SMLC User Guide.
2	Line Control Word. See "OTA 02XX" in the SMLC User Guide.
3	Synchronizing characters. See "OTA 03XX" in the SMLC User Guide.
4	Clock control constant. This word is optional. Note that this word controls the clock rate for all lines on the controller. See "OTA 17XX" in the SMLC User Guide.

UPDATE

The calling sequence is:

```
CALL  UPDATE  (Key 1, 0)
```

The possible value for Key is:

Value	Meaning
1	Update CUFD (current UFD); DSKRAT buffers to disk, if necessary; and undefine RAT in memory.

WREC

Subroutine WREC writes the disk record to a disk from a buffer in memory. The arguments and rules of the WREC call are identical to those of RREC except for bits 1 and 2 of N which have no meaning on write. For a call to write a record on the diskette, the buffer length Blen must be 448 words.

The calling sequence is:

```
CALL WREC  (Bptrs, Blen, N, Ra, Altrtn)
```

The meaning of the parameters is the same as described under RREC in this section, except that the function of the command is to write the specified records instead of to read them. Like RREC, WREC is available only under DOS/VM. The user is cautioned that indiscriminate use of WREC could cause destruction of the operating system.

An attempt to write on a write protected disk generates the message:

```
DISK WT ERROR     Pdisk    Ra    Status
WRITE PROTECT
```

on the supervisor terminal and the message UNRECOVERED ERROR at the user terminal; ERRVEC(1) will contain error code WB, unless Altrtn is taken. Other write errors are retried ten times similar to read errors (Refer to RREC).

SECTION 6

DOS/VM - OVERVIEW AND STARTUP

DOS/VM SYSTEM OVERVIEW

DOS/VM achieves a sharing of the computer resources among a community of up to 31 simultaneous users and in addition, provides each user with a virtual memory environment.

The resources shared are the central processor, high-speed memory, the file system, and the peripheral devices. Each user is provided with a terminal to interact with DOS/VM. Each user is provided with a 64K word virtual memory space. Any user can access files on disks using the same commands and system subroutines that are available when running under DOS. Other peripheral devices, such as the paper tape reader, may be used in the same manner as they are used under DOS, provided they are first assigned to the user, using the ASSIGN command. Under DOS/VM, users are protected from interfering with each other, and user privacy is assured. No user can peek into another user's memory to find out what the other user is doing, and no user can alter another user's memory. Under both DOS and DOS/VM, disk files are protected by passwords on file directories.

Sharing Files

Sharing of files is possible under DOS/VM. Two or more users may be attached to the same UFD at the same time. Furthermore, two or more users may have the same file open for reading, and thus may be reading the same file at the same time. File interlocks are provided, as under DOS, to prevent one user from reading the file while another is writing. This interlock may be modified by the DOS/VM system configurator.

File Access Protection

Under DOS/VM, a user (hereafter called the owner) has the ability to open his file directories to other users giving restricted access rights. The owner of a file directory can declare the access rights that nonowner users have over each file. File access protection is not available under DOS.

The declaration of access can be made on a per-file basis, thus the owner has a degree of flexibility in the manner that file access is specified. Access rights are separated into three categories.

Read Access (includes execute access)

Write Access (includes overwrite access and append access)

Delete/Truncate access

The access rights to a file are declared and specified through the PASSWD and PROTECT commands. (Refer to Section 4).

The owner of a UFD can establish two passwords for access to any file in the UFD. An owner password is required by the owner to obtain owner privileges, and a nonowner password (if any) is required to obtain nonowner priveleges.

The PROTECT command replaces the existing protection keys on a file. It is used by an owner to specify the access rights to be given other users of a specific file.

Bypassing Bad Memory

DOS/VM includes features to help Field Service or users to detect and bypass bad memory chips. These features are categorized as follows:

- On a START, DOS/VM performs a simple data and parity check of all memory locations above 32K.
- If memory chips are known to be bad, DOS/VM can be modified (and SAVEd) so as not to use these bad pages (chips).
- If the system crashes while running, a self-contained routine can be started that tests all available memory to locate any memory call that contains incorrect parity.

When any user of the system LOGS IN or LOGS OUT, the program LOGIN in CMDNC0 is RESUMEd if it exists. This program may be custom-written by a given installation to perform special LOGIN/LOGOUT functions such as accounting or restricting system access.

The program is RESUMEd with BREAK inhibited so as to prevent the user from exiting LOGIN via Control-P or BREAK. The LOGIN program performs a CALL BREAK$ (.FALSE.) before exiting.

The command line that called LOGIN is available via CMREAD.

Accounting information is available via TIMDAT.

The LOGIN program exits via CALL EXIT and must not encounter any uncontrolled errors that result in EXIT being bypassed.

Inactivity Timeout

Users logged in at a terminal but inactive are automatically logged out after N inactive minutes, where N is a system configuration parameter. A user is considered inactive if the terminal is waiting for a DOS/VM command or a user program is waiting for terminal input or card reader input. The default is 1000 minutes, which effectively disables this feature.

DOS/VM SYSTEM CONFIGURATION

Upon obtaining a master disk from Prime, the system configurator (user) must install a DOS/VM for his installation into CMDNC0. DOS/VM supports 1 to 31 users and a variety of peripheral devices except the SMLC. DOS/VM is delivered as a run file in the UFD, DVBIN, called TSAMLC. This version supports a system containing a System Option Controller or Option A Controller, and one or two 8 or 16 line asynchronous multi-line controllers. The run file (TSAMLC) determines what controllers are connected to the machine and configures itself accordingly. The user may configure the following combinations:

AMLC Hardware	Paging Device Required	No. of Terminal Users
One 8 line AMLC	512K fixed head disk	5
One 8 line AMLC	1.5M platter	7
One 16 line AMLC	1.5M platter	11
One 16 line AMLC	3.0M platter or partition	15
One 16 line and one 8 line AMLC	3.0M platter or partition	23
Two 16 line AMLCs	3.0M platter and 1.5M platter or two 3.0M platters	31

All AMLC lines are set to run at 1200 baud. System configurators who wish to set lines for other speeds should see the discussion of "Changing Configuration Table. The file, DOS/VM, in the UFD CMDNC0 is a copy of TSAMLC.

Systems without an AMLC use the serial interface to connect to user terminals. These systems require the system configurator to copy file TS330 from UFD DVBIN on the master disk to CMDNC0 and rename it DOS/VM. TS330 supports 1 to 4 user terminals running at 110 baud. It requires a 330 cycle clock. TS300 in UFD DVBIN supports 1 to 4 user terminals running at 300 baud for output and 75 baud for input. It requires a 300 cycle clock. A 1.5M platter is required for paging. TSSMLC in UFD DVBIN is the same as DOS/VM but it also supports the SMLC controller.

The system configurator may wish to limit all users to a 32K address space to run more users with less disk space allocated for paging. This is especially true for those installations that use a fixed head disk for paging. To configure DOS/VM for 32K address space per user, do the following:

1. Run command file CMKM32 using the COMINPUT command un UFD DVBIN. This command file generates the page maps for DOS/VM for 32K per user as a file called PAGMAP.

2. Incorporate the custom file PAGMAP into DOS/VM by doing the following:

```
User:       RESTOR   TSAMLC

Response:   OK

User:       RESTOR   PAGMAP

Response:   OK

User:       SAVE  DOS/VM  60  64777  1000

Response:   OK

User:       FUTIL

Response:   >

User:       TO  CMDNC0

Response:   >

User:       COPY  DOS/VM

Response:   >

User:       QUIT

Response:   OK
```

The version of DOS/VM requires disk space for paging as follows:

AMLC Hardware	Paging Device	No. of Users
One 8 line AMLC	256K fixed head disk	5
One 8 line AMLC	512K fixed head disk	6
One 8 line AMLC	1.5M Platter	7
One 16 line AMLC	1.5M Platter	15
One 16 line AMLC	512K fixed head disk	13
One 16 line AMLC and one 8 line AMLC	1.5M Platter	21
Two 16 line AMLC	3.0M Platter	31

A system configurator may wish to custom-modify the DOS/VM page maps to:

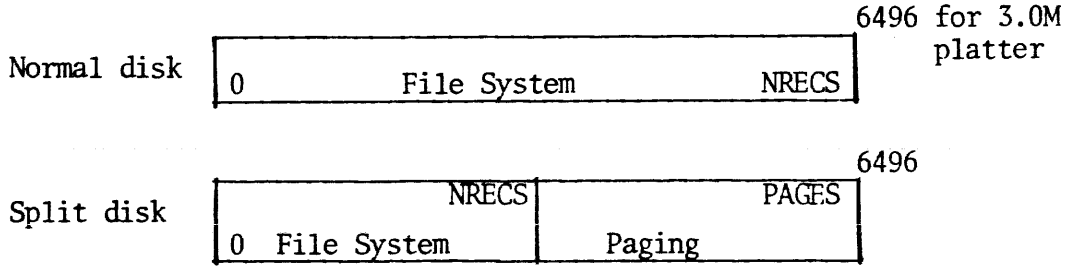
. run some users with 64K address space and other users with 32K address space,

. run some users on the fixed head disk and others on a moving head disk.

To accomplish this, the system configurator must modify the source programs MAKM64 or MAKM32 found in UFD DVBIN to generate the appropriate page maps. These programs generate page maps based on a table at the beginning of the program. The page maps are generated as file PAGMAP by running the command files CMKM64 or CMKM32 and incorporating these maps into DOS/VM as described above.

The system configurator may wish to use part of a disk surface for paging and the rest for the file system.

On a normal disk, the file system uses space from 0 to NRECS, where NRECS is the number of 448 word records that may be written on the disk. On a split disk, NRECS must be specified as something less than the maximum and the remainder of the disk space is used for paging. This is shown by the following diagram:



The amount of paging space required is calculated as follows:

Paging space on the moving head disk is 256 records per user (64K address space) plus 352 records for the supervisor.

Paging space on the fixed head disk is 128 records per user (64K address space) plus 176 records for the supervisor.

For an n-user DOS/VM configuration, where n is the number of users, the amount of paging space needed is:

PAGES = 256 X n + 352 records on MHD

PAGES = 128 X n + 176 records on FHD

Example:

For a 6 user system on a moving head disk: PAGES = 256X6+352=2488 and NRECS = 6496-2488=2008. This assumes the disk is one platter of a 6.0M disk drive.

To make a split disk, mount a scratch pack on the drive and do the following:

User: MAKE

Response: PHYSICAL DEVICE =

User: Type number of Physical Device

Response: RECORD SIZE

User: (Carriage Return)

Response: RECORDS

User: 3720 ($=2008_{10}$)

Response: DEVICE NUM RECORD COUNT

PARAMETERS OK?

User: YES

Response: VIRGIN DISK?

User YES

Response: VERIFY DISK

User YES

Response: DISK CREATED (after a while)

The system configurator must then copy the BOOT into the MFD, use the UFDCPY feature of FUTIL to copy the UFD's: CMDNCO and DOS from the master disk.

NOTE

The split disk must be used as the command device under DOS/VM.

Refer to the sideheads CONFIG and STARTUP in the following paragraphs:

Many systems are shipped with just one disk - a 30M word disk. It is suggested that the user does not run the 30M word disk as a split disk. When a disk is shipped, it is partitioned into two subdisks; a 3.0M disk partition number 250 with the rest of the disk blank. The system configurator should make a 3.0M word partition for paging if DOS/VM is for 23 or fewer users or two 3.0M word partitions if DOS/VM is for 24 or more users. The system configurator does this using the command MAKE, to make physical device 10250, for example. The rest of the disk must then be made as 1 or more partitions. If it is made as one partition, the disk number would be 24250.

Tables 6-1 and 6-2 provide a guide to disk addresses for system configurators. They show disk space required for the supervisor and up to 31 users on both 32K and 64K configurations.

CHANGING CONFIGURATION TABLE

The baud rate for the AMLC configuration can be changed easily by modifying the line configuration table for lines 0-6. The line configuration table is identified by a comment in the DOS/VM source program; its location is currently '55044. The values that may be specified are:

110 baud:	XX0033	2033
300 baud:	XX0213	2213
1200 baud:	XX0313	2313

A 100005 IS HALF DUP
000005 IS FULL DUP

Table 6-1. Disk Space Required for 32K Per User

User		MHD Disk Addresses		FHD Disk Addresses
Supervisor		0-351		0-175
1		352-479		176-239
2		480-607		240-303
3		608-735		304-367
4		736-863		368-431
5		814-991	limit 256K disk=512	432-495
6		992-1119		496-559
7		1120-1247		560-623
8		1248-1375		624-687
9		1376-1503		688-751
10		1504-1631		752-815
11		1632-1759		816-879
12		1760-1887		880-943
13		1888-2015	limit 512K disk=1024	944-1007
14		2016-2143		1008-1072
15		2144-2271		
16		2272-2399		
17		2400-2527		
18		2528-2655		
19		2656-2783		
20		2784-2911		
21		2912-3039		
22	limit 1.5M platter = 3248	3040-3167		
23		3168-3295		
24		3296-3423		
25		3424-3551		
26		3552-3779		
27		3780-3907		
28		3908-4035		
29		4036-4163		
30		4164-4291		
31		4292-4419		
32		4420-4547		

Table 6-2. Disk Space Required for 64K Per User

User		MHD Disk Addresses		FHD Disk Addresses
Supervisor		0-351		0-175
1		352-607		176-303
2		608-863	limit 256K	304-431
3		864-1119		432-559
4		1120-1375		560-687
5		1376-1631	limit 512K	688-815
6		1632-1887		816-1044
7		1888-2143		
8		2144-2399		
9		2400-2655		
10		2656-2911		
11	limit 1.5M platter = 3245	2912-3167		
12		3168-3423		
13		3474-3779		
14		3780-4035		
15		4036-4291		
16		4292-4547		
17		4548-4803		
18		4804-5059		
19		5060-5315		
20		5316-5571		
21		5572-5827		
22		5828-6083		
23	limit 3.0M platter = 4956	6084-6339		
24		~~0-255~~ 3684	pagedev 2	
25		256-511	"	
26		512-767	"	
27		768-1023	"	
28		1024-1279	"	
29		1280-1535	"	
30		1536-1791	"	
31		1792-2047	"	

DOS/VM SYSTEM INITIALIZATION

Once the system configurator has installed and appropriately modified DOS/VM in CMDNC0 as explained in the preceding paragraphs, DOS/VM is started from DOS by the system operator at the beginning of each day.

The steps to get DOS/VM running are:

1. Turn on machine and disks, and bootload DOS as explained in Section 3.

2. STARTUP Command-Disk-Number

3. ATTACH CMDNC0

4. DOSVM

Response is:

```
DOSVM   REV  X.X

XX.X  K  MEMORY IN USE

PLEASE ENTER CONFIG AND DATE
```

At this point, DOS/VM is running. The operator must give SETIME, CONFIG, and STARTUP commands before the system is ready for the users. The operator may optionally give the DISKS command which defines the disks that user's are allowed to access outside of the DOS/VM file system. The STARTUP command starts the disks that users are allowed to access through the DOS/VM file system.

If DOS/VM fails to type its introductory message and halts, it usually means that DOS/VM has detected bad memory. Refer to Appendix for a description of how to recover from the situation. The following example shows a typical operator procedure to bring up DOS/VM for use. After the example, the operator terminal commands issued at the start and during the running of DOS/VM are explained in detail. An example startup of DOS/VM follows:

```
OK:  STARTUP  1
OK:  A CMDNC0
OK:  DOSVM
GO

DOSVM   REV 7.0
---------------

64.0 K  MEMORY IN USE


PLEASE ENTER CONFIG AND DATE.

OK,  CONFIG  12  5  1
OK,  SE -0423  -0905
OK,  STARTUP  1  0  2  4
OK,  UDIN  (6)  LOGGED IN AT 09'07  0423
JHNDOE  (3)  LOGGED IN AT 09'10  0423

...
```

DOS/VM SYSTEM-TERMINAL COMMANDS

When started, DOS/VM prints QUIT on every active user terminal and waits for a command at the supervisor terminal. A typical sequence of command is:

SETIME

CONFIG

STARTUP

These commands and their arguments are described in the following paragraphs. All system commands are issued at the supervisor terminal.

After DOS/VM is started, the DOS system terminal becomes the DOS/VM system terminal. Unless the USRASR command is given, the system terminal can only be used for a restricted set of operator commands, given in this section. Since the system terminal operates in the supervisor address space, any external commands, and in addition, the RESTOR, RESUME, and START commands will crash DOS/VM if they are given at the system terminal. Normally, the system terminal is used only to STARTUP and SHUTDN disks, UNASSIGN devices, check STATUS, and to collect a record of LOGIN and LOGOUT messages.

CONFIG

The CONFIG command defines five system parameters that are specified once per system session. The CONFIG command is disabled after its first use during a session.

```
CONFIG   Nuser   Pagedev1  Comdev   [Availm]  [Pagedev2]
```

Nuser is an integer less than or equal to octal 40 that defines the number of users, including the supervisor. (e.g., for a four-user system, enter 5; for a seven-user system, enter 10).

Pagedev1 is a physical disk number that specifies the device to be used for paging. See Table 3-1 and Table 4-3 for possible values.

Comdev specifies the physical device number initially assigned as logical 0. When a user invokes an external command, the command directory, CMDNC0, is searched on this device. If Comdev and Pagedev are the same, the disk is considered to be split into a file system and a paging part. The boundary between the partitions is defined by the DSKRAT header, and it may be set by the MAKE program. (See the paragraph on configuration at the beginning of this section.

Availm is an optional argument that defines available physical storage. It corresponds to the last sector number (octal) to be used. If Availm is omitted, all of available memory is used. The values for Availm and associated storage used are as follows:

blank or 0	all of memory (must be at least 32K)
'117	40K
'137	48K
'157	56K
'177	64K
...	...
'777	256K

Pagedev2 CONFIG may specify either one or two disk devices on which paging is to take place.

The CONFIG command uses the range of acceptable Pagedev and Comdev codes (Physical Disk Numbers) as shown in Table 3-1.

Example: A system terminal operator may wish to specify two paging devices, for example, to run a 30-user system using 3.0-million word disk packs. If each user's virtual space is set to be 64K, only 23 users will fit on a 3.0-million word disk pack. Thus the use of two paging devices is required. The command line:

```
CONFIG  30  0  0  0  1
```

would allow paging device to be physical devices 0 and 1 for a 30-user system.

DISKS [NOT] Pdisk0 [Pdisk1] ... [Pdisk8]

The DISKS command may only be given from the supervisor terminal. The DISKS command adds the specified physical disk(s) to the assignable disks table or, removes the specified physical disks from the assignable disks table. Pdisk0 ... Pdisk8 are physical disk numbers. No more than ten disks may be entered into the assignable disks table. A physical disk number must be specified in this table before a user may invoke the ASSIGN command to assign that disk.

When the optional argument NOT is specified in the DISKS command line, the subsequently specified physical disks are removed from the assignable disks table. Removing a physical disk number from the table does not cause the disk to be unassigned; the operator must give the UNASSIGN command in order to unassign a disk from a user.

Example:

```
OK, DISK 1 20250 50250 60250 70250 10020 110250 20252
```

adds the specified physical disks (disk 1) and partitions (20250, etc.) to the assignable disk table. These disks and partitions may now be ASSIGNed by the users or operators. The command sequence:

```
OK, DISK NOT 20250
OK, UNASSIGN 20250
```

removes the physical disk partition 020250 from the assignable disks table and unassigns that partition.

MESSAGE

The DOS/VM command MESSAGE provides a message facility that allows a user, at a user terminal, to communicate with the operator, at the supervisor terminal; allows an operator, at the system terminal, to communicate with all users at all terminals connected to the system; or allows an operator to communicate with a specific user at a specific terminal. The format for user to operator messages is:

```
MESSAGE

text of message
```

where (text of message) is a one-time message. Two lines are printed at the supervisor terminal. Their format is:

```
*** uu  hh'mm

text of message
```

where: uu is the user number; hh'mm is the time of day in hours and minutes.

The format of operator to user messages is:

```
MESSAGE        ALL     [ NOW ]
```

The operator can send messages to all users or to a single user. When the parameter ALL is specified, the message is sent to all users. The parameter -uu is a minus followed by the user number. When the parameter -uu is specified, a message is sent to the user specified. If the optional argument NOW is not specified, the message is stored in a Broadcast Buffer (ALL) or a Single User Buffer (-uu). The message is printed at the users terminal when that user returns to DOS/VM command level. A message that is in the Broadcast Buffer is also printed after LOGIN.

If the argument NOW is specified, the message is printed immediately. This is an unfriendly thing to do if the user is in the midst of a sensitive operation. When NOW is specified, stored messages are not affected.

Also, when NOW is specified, the format of the message at the user terminal is:

```
*** BULLETIN ***

text of message
```

If the operator attempts to send a message to a single user before the previous message to a single user has been received, the error line:

MESSAGE NOT SENT

is printed at the systems terminal.

To cancel a stored message, a null line must be entered as the text of message.

SETIME

The SETIME command sets date and time. It can be entered at any time during system operation. Syntax:

SETIME -mmddy -hhmm

where mmddy are digits that represent the month, day and year (last digit only), and hhmm are digits that represent the time in hours and minutes. The two arguments to SETIME must be separated by spaces and start with a minus sign as the first character. Example:

SETIME -09294 -1630

sets the data and time: September 29, 1974, 4:30 PM.

STARTUP

The STARTUP command defines a list of physical disk devices to be used by DOS/VM. A disk is considered started if it has been mentioned in a previous STARTUP command. Additional disks may be started if the new list in a subsequent STARTUP command does not conflict with the list in a previous STARTUP, and if no user has assigned a disk specified in the list. Syntax:

STARTUP Comdev [Pdev1 Pdev2 ... Pdevn]

where Comdev and Pdev1 ... Pdevn are items in a list of physical disk (device) numbers. The argument, Comdev, must be specified in the initial STARTUP command; the remaining device numbers are specified optionally. The order of the list defines the logical number sequence of the devices (e.g., Comdev is logical 0, Pdev1 is logical 1, etc.). Physical Device codes are listed in Table 3-1.

Comdev must match the Comdev specified in the CONFIG command. Example:

```
STARTUP  2  7  3
```

defines that physical devices 2, 7 and 3 are to be used with DOS/VM and associates the following logical device numbers with the physical device numbers specified: 2 is logical 0; 7 is logical 1; and 3 is logical 2. In DOS/VM, logical device numbers may also be specified as arguments to the STARTUP command. In this case, they must be followed by a slash and the associated physical device number. Examples:

```
STARTUP  0/2  1/7  2/3

STARTUP 4/100250
```

SHUTDN

The SHUTDN command performs tasks necessary to shutting down the DOS/VM system in an orderly manner. Syntax:

```
SHUTDN  ALL

SHUTDN  Pdev0  [Pdev1  Pdev2 ... Pdevn]
```

The command form: SHUTDN ALL performs a complete system shutdown. All user files are closed, physical disks are closed, and the DOS/VM system shuts down by inhibiting interrupts, exiting page mode, stopping the system clock and halting.

If the SHUTDN command is issued with a list of physical devices (Pdev0 ... Pdevn), the listed devices are closed by closing all files opened on the listed devices and by detaching all users attached to the listed devices. Refer to Section 3 and Table 3-1 for a complete discussion of physical device numbers. Then, the specified disks are not available for DOS/VM file I/O operations until the devices are specified on a subsequent STARTUP command. SHUTDN must be given before shutting down or changing a disk pack on a drive if that drive is currently started up with the STARTUP command. The STATUS command can be used to list the devices currently started up. Unlike the STARTUP command, the Pdev's do not have to be given in logical drive order.

CAUTION: Do not shut down the physical device associated with logical 0. If this is done, DOS/VM loses the command directory (from its memory, not the disk). To recover, STARTUP the disk and ATTACH CMDNC0.

Example of Selective Shutdown

Assume the initial STARTUP command was:

```
STARTUP  1  0  6  4
```

The operator wishes to replace the pack on physical device 6, which is logical device 2. The operator gives the command SHUTDN 6, stops the drive, replaces the pack and restarts the drive. The operator then gives the command:

```
STARTUP 2/6
```

to startup physical drive 6 as logical drive 2.

STATUS

The STATUS command may be used to monitor the usage of DOS/VM. When entered at the system terminal, the STATUS command prints status information that consists of the information given at the user terminal and, in addition, prints a list of current logged-in users. Following each user name in the list, the user terminal number and the numbers of the physical disks currently used by the user are printed. Also, devices that a user has assigned are listed after the number of the physical disk that is currently used. A disk is considered to be in use by a user (1) if his home UFD or current UFD resides on the disk or (2) if the user has opened a file on that disk. Some typical instances where the STATUS command must be used are:

1. Prior to mounting a new disk pack to determine what physical disk assignments are available.

2. After a request that all users release a given disk or disks to determine that they have done so before shutting down that disk or disks.

3. As a check that all users have logged out before shutting down DOS/VM. (No harm to the system results if the users of a particular disk are still logged-in when the disk or the system are shut down. However, the users files are closed and a message is printed at the terminal to that effect.

Examples:

1. Example of a STATUS command issued at the supervisor terminal:

```
OK, STATUS

USR = SYSTEM

FUNITS


DISK          LDEV              PDEV
TS             0                 250
SPOOLD         1                 250
DUD            2               20250
DSKRAT         3                   0
ADMIN          4              110250
ETCH           5               40250
PMFII          6              100250
MD6V2          7                   4
TRANS         10               50250

PAGEDEV = 10252  COMDEV = 250

USER        LINE              PDEVS
JOEL           2             110250
SPOOL          3              252 PR1
COHEN          4              0 PTR     PUNCH
MERRIC         5              250
GOUDY          8              20250
PODUSK        13              250
JDOAKS        16              110250

OK,
```

2. Example of a STATUS command issued at a user terminal:

```
OK, STAT

USR = GOUDY

FUNITS

DISK          LDEV              PDEV
TS             0                 252
SPOOLD         1                 250
DUD            2               20252
ETCH           3               40252
LSTFIL         4               50250
DOSDVM         5               60252
WORKII         7              100252
ADMIN         10              110252
```

USRASR

The USRASR command allows the supervisor terminal to be associated with a different address space to allow it to be used as a user terminal. After invoking USRASR, it is still possible to invoke supervisor commands at the supervisor terminal. Syntax:

```
USRASR  Usrno
```

where Usrno is a user number. Example:

```
USRASR  4
```

Restrictions: The USRASR works only if the associated communications line is not enabled on the AMLC. If connected to a current loop bit-banger line, the input leads must be shorted (or a terminal must be connected to the line).

Return: To return to operations as a normal supervisor terminal, type:

```
USRASR  1
```

WARM RESTART FOR DOS/VM

If DOS/VM halts because of an error or because of a machine-check, it is usually possible to restart DOS/VM. The procedure for a Warm Restart is as follows:

1. At the control panel set the rotary switch to STOP/STEP. Press MASTER CLEAR.

2. START at 1001 for machine check or 1002 for no machine check. Store the starting address (e.g., 1001) in Location 17 and set rotary switch to RUN.

3. Set START.

4. At each terminal connected to DOS/VM at the time of the halt, type:

   ```
   S
   ```

   followed by a CARRIAGE RETURN.

SECTION 7

INPUT/OUTPUT WITH DOS/VM

I/O VIRTUALIZATION

Since all user programs running under DOS/VM are executed in restricted mode, all I/O instructions executed by a user program cause traps to the supervisor.

Since user I/O instructions in virtual memory operation cause a trap, a mechanism is provided for user programs to perform a supervised form of I/O. This is accomplished by defining a functional means of allowing certain devices to operate via user I/O commands (I/O virtualization). These devices are listed in Table 7-1 along with the implemented values of the Virtual Memory Systems Controller Board Control Word values for input and output, and the associated port to which the devices are connected.

Port No.	CONTROL WORD VALUES		Device
	Input (Bits 11, 12)	Output (Bits 13-16)	
1	00	000 (or 10 (octal))	User Terminal
2	01	100 (4 (octal))	CENPR (J2)
3	10	010 (2 (octal))	CE2PR (J3)
4	11	001 (1 (octal))	CARDR (J4)

Table 7-1. System Controller Board Control Word, Device, and Port Relationships

A subset of all possible I/O functions that can be performed with a given device are defined, and the DOS/VM system provides a mechanism for calling the supervisor to perform these I/O functions.

The Prime DOS/VM operating system provides a functional interpretation of most I/O instructions relating to the virtual memory systems controller. With the System Controller Option, these I/O instructions include the following:

OCP 4, OCP 104

INA 4, INA 1004, INA 1204, INA 1304

OTA 4, OTA 104

SKS XX04

SYSTEM CONTROLLER CONTROL WORD

For every user terminal connected (logged-in) to the DOS/VM operating system, a register is maintained that stores a virtual-memory-systems-command. (The instructions to initialize this register are OCP 4, OCP 104). The control word is set equal to the A-register by the instruction OTA 104, and the control word can be read by executing the instruction INA 1204.

The control word may also be set by the ASRCWD command. Only the port select fields of the control word (Bits 11-16) are used when the INA 4, OTA 4 instruction sequence is executed.

INPUT/OUTPUT BUFFERS

I/O with Port 1 selected is performed through the user terminal buffers maintained by the operating system supervisor. I/O is always full-duplex. Ports 2, 3, and 4 have three associated buffers. Access to these buffers is only allowed if the corresponding I/O device has been assigned to the user's process by means of the ASSIGN command.

DATA TRANSFERS

Input: Execution of the INA 4 (INA 1004) instruction causes a transfer of a character from the buffer associated with the assigned device to the A-register. If the buffer is empty, the user's process is placed in INPUT-WAIT state, and the supervisor cycles to service another user's process. The user process is rescheduled when the requested input arrives.

Output: Execution of the OTA 4 instruction causes a transfer of a character from the A-register to the buffer associated with the assigned device. If the buffer fills up, the user's process is placed in the OUTPUT-WAIT state. Users are removed from the OUTPUT-WAIT state once per second. At that time, the user process is rescheduled to the location following the OTA instruction; no skip occurs.

Emptying and Filling Buffers: The device interface modules (interrupt routines) empty and fill their associated buffers. The physical device may be different than the logical I/O device. For example; when an INA 4, OTA 4 instruction sequence is executed in the virtual memory system, the system performs output on the serial interface in the CPU using ISI, OSI instructions.

SKIPS

DOS/VM, on encountering an SKS instruction, always skips (with the exception of SKS 704, skip if receiver ready, and SKS 604). The SKS 704 skips only if there is input available either in the buffer associated with the user terminal or in the port that is specified by the virtual control word. The virtual control word is initially set to the user terminal, either by the ASRCWD command or the OTA 104 instruction. SKS skips only if there is room in either the output buffer associated with the user terminal, or in the port that is specified by the virtual control word.

A user program may SKS for terminal input and input a character if one is available or do other computation if no character is available. Previously, the SKS would always skip and a subsequent INA instruction issued to an empty user terminal input buffer would put the user into input-wait state until a character was typed. Similarly, a user program may SKS for room in the terminal output buffer and output a character if there is room or if not do some other computation instead. Previously, the SKS would always skip and an OTA issued to a full terminal output buffer would put the user into output-wait state until the buffer became less than full. No existing IOCS routines or other teletype routines such as T1IN, TIOU, TNOUA etc. in the FORTRAN library does an SKS 704 or SKS 604.

Paper Tape Reader

To interface a paper tape reader with virtual memory, interpretation of the following instructions is provided:

OCP XX01 (treated as NOP's)

SKS XX01 (always SKIP)

INA 1, INA 1001

Execution of the INA 1 (INA 1001) instruction causes a transfer of a character from the paper tape reader buffer to the A-register, and the INA instruction skips. If the buffer is empty, the INA is handled as NOP. The reader must be ASSIGNed by the user. An interrupt routine (PTRDIM) maintains the buffer full by reading the paper tape as long as there is room in the buffer.

Paper Tape Punch

To interface a paper tape punch with virtual memory, interpretation of the following instructions is provided:

OCP XX02 (treated as a NOP)

SKS XX02 (always a SKIP)

OTA XX02 (output character)

Execution of the OTA XX02 instruction causes a transfer of a character from the A-register to the paper tape punch buffer, and the OTA instruction skips. If the buffer is full, the user process goes into OUTPUT-WAIT state for up to one second. A restart is then made to the location following the OTA (no SKIP). An interrupt routine (BRPDIM) punches characters from the punch buffer until the buffer is empty. The punch must be ASSIGNed by the user.

CPU Control Panel

To interface the CPU control panel with virtual memory, interpretation of the following instructions is provided:

INA 1620 (read sense switches)

OTA 1720 (output lights)

A virtual sense-switch-register and a lights-register are maintained for each user that is logged-in. The sense-switch register is set by the VRTSSW command and read by the instruction INA 1620. The lights-register is set from the A-register by executing an OTA 1720 instruction. The lights-register is displayed on the control panel by entering the memory address on the panel sense switches and setting the ADDRESS/DATA switch to DATA. The memory address is computed by taking the sum of 12377 plus the terminal number (number typed on login).

Disk

The disk interfaces with virtual memory through a supervisor call (SVC) instruction to perform a READ or WRITE operation on a single physical record of a physical disk. The disk must be assigned to the terminal by the ASSIGN command. Refer to RREC and WREC in Section 5. For information about the SVC instruction, refer to the Systems Reference Manual and the PMA User Guide.

Magnetic Tape

Input/output operations for magnetic tape are effected by DOS/VM through SVC calls. Refer to T$MT in Section 5.

MPC Line Printer

Output to the parallel interface line printer is accomplished through SVC calls. Refer to T$LMPC in Section 5.

MPC Card Reader

Input from the parallel interface card reader is controlled through SVC calls. Refer to T$CMPC in Section 5.

SVC VIRTUALIZATION

To allow debugging or execution of other operating systems, DOS/VM allows virtualization of all SVC calls except a class of SVC's considered exclusive to DOS/VM. (Function codes XXX5XX). This capability is turned off on LOGIN and can be set by the following commands:

SVCSW 1 turn-on virtual SVC handling

SVCSW 0 turn-off virtual SVC handling

If the SVCSW is turned-on, the SVC instruction executed by a user program that has a word following the SVC that is not of the form XXX5XX, results in a virtual trap through location '65.

Example:

Assume that a version of DOS that performs disk I/O using the DOS/VM RREC/WREC SVC calls is stored in the UFD CMDNC0 under the name, VDOS32. Thus, a user may ASSIGN a disk to a terminal, turn on SVC calls and run DOS. The following sequence shows a typical operation.

User Input	Effect
ASSIGN DISK 2	assigns physical disk 2 to user.
AS DISK 3	assigns physical disk 3 to user.
SVC 1	turn on SVC virtual memory interface.
VDOS32 GO	bring DOS into virtual memory and start execution. DOS types its usual message and types OK:
STARTUP 2 3	informs DOS to use physical disks 2 and 3.
A LIB	attach to any desired UFD.
FIXRT	performs VIRTUAL FIXRAT
COPY GO FROM-TO: 2 3	copy physical 2 to physical 3.
SHUTDN	direct DOS to perform normal clean up functions prior to shutting down.
Press 'QUIT'	return to DOS/VM command level.

User Input	Effect
UNASSIGN DISK 2	release physical disk 2.
UNASSIGN DISK 3	release physical disk 3.
SVC	turn off SVC virtual memory interface.

Table 7-2 is a list of SVC codes used by DOS/VM.

OTHER VIRTUALIZATION

Unimplemented Instructions (UII), floating point exceptions (FLEX) and Procedure Stack Underflow (PSU) are also virtualized (i.e., these cause interrupts that vector the trap location in the users virtual address space. For optimal performance, the appropriate hardware configuration is recommended.

SVC Number		Associated Call
100	ATTACH	(ufdnam, ldev, passwd, key, altrtn)
1	SEARCH	(key, name, unit, altrtn)
2	SAVE	(rvec, name)
3	RESTOR	(rvec, name, altrtn)
4	RESUME	(name)
5	EXIT	
6	ERRTN	(altrtn, a1, a2, a3)
7	UPDATE	(1,0)
110	GETERR	(buff, nw)
1	PRERR	
2	GINFO	(buff, nw)
3	CNAME	(oldnam, newnam, altrtn)
4	ERRSET	(altval, altrtn, a1,a2,a3)
5	FORCEW	(key, unit)
200	READ	(unit, buff, nw, altrtn)
1	WRITE	(unit, buff, nw, altrtn)
2	RDLIN	(unit, line, nw, altrtn)
3	WTLIN	(unit, line, nw, altrtn)
300	PRWFIL	(key, unit, LOC(buff), nw, posv, altrtn)
500	RREC	(pbav, nwv, nchn, ra, pdev, altrtn)
1	WREC	(pbav, nwv, nchn, ra, pdev, altrtn)
2	TIMDAT	(buff, nw)
3	--	reserved
4	--	reserved
5	RECYCL	
6	D$INIT	(pdev)
7	BREAK$	(onoff)
510	T$MT	(unit, LOC(buff), nw, inst, statv)
1	T$LMPC	(unit, LOC(buff), nw, inst, statv)
2	T$CMPC	(unit, LOC(buff), nw, inst, statv)
600	COMANL	
1	CLIN	(char)
2	CMREAD	(buff)
3	COMINP	(name, unit, altrtn)
700	T1IN	(char)
1	T1OU	(char)
2	TNOU	(msg, cnt)
3	TNOUA	(msg, cnt)
4	TOOCT	(num)
1000	T$MT	See 510
1	T$SLC	(key, line LOC(buff), nw)
1100	T$LMPC	See 511
1200	T$CMPC	See 512

Table 7-2. SVC's Numbers Used by DOS/VM

Other SVC numbers are available for the user implemented SVC's.

APPENDIX A

Table A-1. File and Header Formats

File Record Header Format

Word	Content	Remarks
0	"This" record address	Consists of the DOS address of the record.
1	Parent Record Address or Beginning Record Address (BRA)	If record is a beginning record, this word contains a pointer to the parent (immediately superior) segment directory, or UFD.
2	Forward	Records forward pointer to the next record. (May be a null pointer if last record).
3	Backward	Records backward pointer to the immediately preceding record. (May be a null pointer if first record).
4	Data Count	Records number of words of data in this record (excludes header).
5	Type	Only in the beginning record. 0 = SAM File 1 = DAM File 2 = SAM Segment Directory 3 = DAM Segment Directory 4 = SAM User File Directory
6	Spare 1	Reserved
7	Spare 2	Reserved

All remaining words in the record may be used to store ASCII character pairs or 16-bit words. Data is assumed to continue from the last word in the record to the eighth word of the physical record specified in Word 1. The forward and backward pointers make it easy for DOS to traverse a file in either direction, and at the same time provide a large measure of protection against snowballing disk errors. The pointer to the beginning record address makes it possible to identify a "lost" record.

Table A-2. UFD FORMATS

Word	Content	Remarks
	UFD Header, where:	
0	Word Count = 8 (size of header)	
1-3	Owner Password	Six ASCII characters
4-6	Nonowner Password	
7	Spare	
8-13	UFD Entries	See table below
14-19		
...		

UFD ENTRY FORMAT

Word	Content	Remarks
		Word numbers are relative to beginning of entry
0	BRA	Beginning Record Address of file
1-3	Filename	Currently six ASCII characters
4	Spaces	Reserved for future use
5	Protection keys	Bits 1-8 owner protection Bits 9-16 non-owner protection

Figure A-1 shows UFD file format and use schematically.

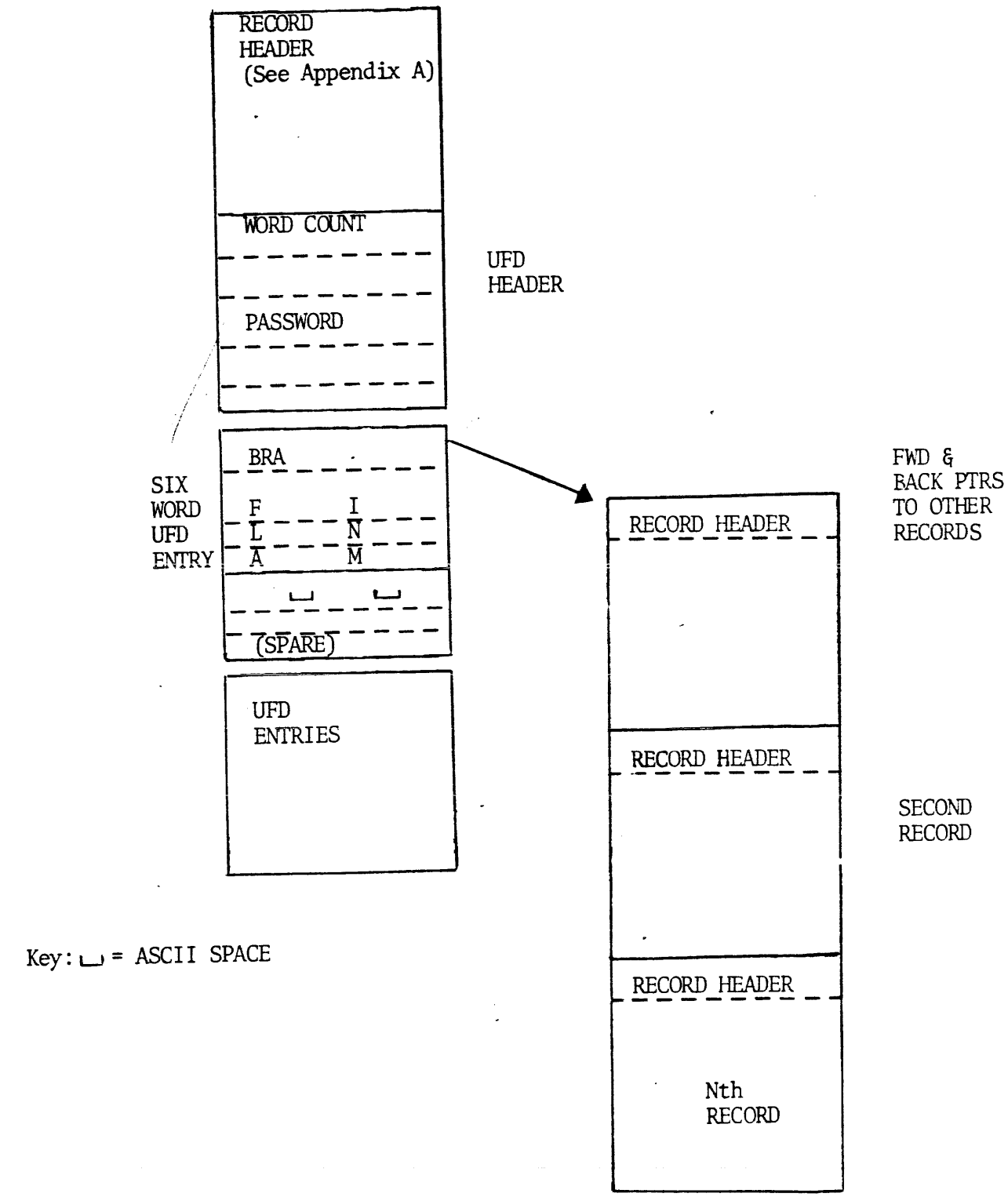


Figure A-1. UFD File Format and Use

Table A-3

FORMAT OF DSKRAT

The DSKRAT file has a special header block as follows:

Word	Contents	Meaning
0	WRDCNT	Words in header block (5).
1	RECSIZ	Disk record size.
2	NRECS	Number of records for file system.
3*	CYLS	Cylinder count.
4	HEADS	Head count for disk or partition

*not currently used by DOS file system.

WRDCNT allows an expansion of the block size while still maintaining a compatible disk. The header is followed by DSKRAT data, a 1 bit for each record in the file system (NRECS).

During all file transactions, DOS and DOS/VM update the DSKRAT file to reflect the state of records occupied or released, as files or portions of files are added or deleted. The DSKRAT file also contains data on the total disk record count.

In the Virtual Memory Operating System (DOS/VM), the name of DSKRAT file(s) may be obtained by using the STATUS command. Example:

```
OK, STATUS

UFD=BOUDY    0

FUNITS
4

DISK    LDEV    PDEV
TSDISK    0       01
DUD       1       00
PCBRD     2       02
COMAND    3       04


OK,
```

APPENDIX B

BOOTSTRAPS

BOOTSTRAPS

Prime bootstraps are either control panel boots (and key-in substitutes) or mass storage resident second-level bootstraps.

CONTROL PANEL BOOTS

Control Panel μ-code

A control panel can have either 256 or 512 16-bit words of PROM from which bootstrap programs can be loaded into memory. After pressing MASTER CLEAR and dialing the selection switch to the LOAD position, pressing the START switch causes the control panel μ-code to read PROM locations '0 to '50 into memory locations '6 to '56 and begin execution in 16S mode at the address loaded into Location 7. This initial program, the pre-boot, can then read succeeding PROM locations into memory with the following instruction sequence:

```
LDA (PROM address)
OTA '1720 (address setup - same as display data lights)
INA '1420 (input PROM location)
STA (memory location)
```

For some applications, the initial '51 words may be sufficient to code a complete bootstrap. Caution must be exercised when coding a program to execute in the register file (locations '0 to '37) because some instructions alter registers.

Prime Pre-Boot

The Prime pre-boot saves the A-register in location '57 and then selects among three classes of bootstraps and stores the appropriate code from the PROM into memory. The three classes of bootstraps are auto-start, paper tape, and mass storage boots. The user selects the desired boot by setting Sense Switches 14, 15 and 16 as follows:

SS =	14	15	16	Code	
	0	0	0	=0	Auto-start
	0	0	1	=1	ASR paper tape (MDL format)
	0	1	0	=2	High speed paper tape (MDL format)
	0	1	1	=3	Fixed head disk
	1	0	0	=4	Moving head disk
	1	0	1	=5	Magnetic tape
	1	1	0	=6	Floppy disk (Diskette)
	1	1	1	=7	Spare

Device Specific Boots

Auto-Start (0): Enters 64R mode and jumps to the location specified in Sense Switches 1 to 10 ('100 to '177700). If no address is specified, a default of '1000 is used.

Paper Tape (1 & 2): Modifies itself for either ASR or high speed paper tape (by sense switches) and reads a second-level MDL boot into memory. This boot requires that the first nonzero frame on the tape be '20 and the next two frames be '004/'010 = '2010 = JMP '10. If the initial A-register setting (saved by the pre-boot in location '57) is to be used, it must be saved before location '57 is loaded by the bootstrap. The first zero frame on the tape causes the JMP '10 instruction in location '20 to be skipped. When execution starts at location '21, the following locations have been set up:

Location	Contents	Instruction (X=1 for PTR, =4 for ASR)	
2	'3000X	OCP	X
3	'13100X	INA	'100X
'10	'13100X	INA	'100X
'11	'002010	JMP	*-1
'12	'141240	ICR	
'13	'13000X	INA	X
'14	'002013	JMP	*-1
'15	'050000	STA	0,1
'16	'140114	IRX	
'17	'100040	SZE	
'20	'002010	JMP	'10 (from tape)

Mass Storage (3-7): Performs further selection for fixed head disk (FHD), moving head disk (MHD), magnetic tape, diskette and spare, all of which are loaded by the pre-boot.

FHD (3): Sense Switch 13 is used to select between controller 4001 (SS 13 reset) and controller 4002 (SS 13 set). The boot reads record 0 (448 word DOS record format) of the disk starting at location '770 and begins execution a '1000 (via a JST '777). This boot waits for the drive to come ready and retries on status errors.

MHD (4): Moving head disks come in two varieties: two platter drives (3M or 6M words) On either 4000 or 4002 controllers and 20 surface drives on 4001/4002 controller. Sense Switches 11, 12 and 13 are used as follows (X - don't care):

SS = 11	12	13	
0	X	0	4000, upper surface
1	X	0	4000, lower surface
0	0	1	4002, lower surface
1	0	1	4002, upper surface
X	1	1	4001, (20 surface)

In all cases, record 0 (448 words DOS record format) of the selected surface of physical drive 0 is read into memory starting at location '770 and execution begun at '1000 (via a JST '777). This boot waits for the drive to come ready and retries on status errors.

MT (5): Sense Switch 12 is used to select between 9 track (SS 12 reset) and 7 track (SS 12 set) drives. The boot starts up the drive, insures that the tape is set a loadpoint (space forward, abort, and rewind), and reads one tape record into memory starting at location '200 and through '7777 (4K). Execution begins at '1000 (via a JST '777).

FLOPPY (6): Reads record 0 (track 1, sector 1) into memory starting at location '770. To maintain IBM compatibility, the boot alternately tries to read a 448 word DOS record and a 64 word IBM record. Execution then begins at '1000 (via a JST '777). This boot waits for the drive to come ready and retries on status errors.

SPARE (7): Intended as a user-supplied down line loader. Currently, halts at location '57.

PROM Generation

Generation of control panel PROM is a three-step operation: write, assemble (PMA) and load (LOAD) the control panel boot program (CPBOOT); generate a PROM simulator paper tape with the CPBGEN program; and physically make the PROM.

CPBOOT: CPBOOT is the standard Prime control panel bootstrap program. It resides on UFD=AIDS on the master disk. There are three general rules for generating a control panel boot:

1. It must be loaded at '1000 in 16S mode, but executable in sector zero (all sector bits reset).

2. All unused locations in sector '1000 must be set to 0.

3. A maximum of 256 locations can be used (512 for larger control panels).

Rules 1 and 2 are satisfied by the use of absolute offsets to the proper values and added to all addresses in memory reference instructions and an initial instruction sequence of:

```
D16S
ABS
ORG   '1000
BSZ    512
ORG   '1000
```

Offsets are computed and used as follows:

```
       PBD     EQU     6-*             (*='1000)
'1000  PB2     DATA    7               at '1000, to be loaded at 6
'1001          LDA     *-1+PBD         (*='1001, '1001-1+6-'1000=6)
'1002          LDA     PB2+PBD         (PB2='1000, '1000+6-'1000=6)
                .
                .
                .
       OVER    EQU     '1042           for subsequent device boot offsets
                .                      actual location ='50
                .
                .
       STD     EQU     OVER+PBD-*      (*='1051)
'1051  START   ANA     S1+STD          (S1='1056, '1056+'1042+6-'1000-'1051='55)
'1052          SNZ
'1053          JMP*    STE+STD         (STE='1060, '1060+'1042+6-'1000-'1051='57)
'1054          E64R
'1055          JMP*    1
'1056  S1      DATA    '177700
       STE     EQU     *+1             (next location contains '1000)
'1057          DATA    '1000
```

The number of locations in the auto-start boot is computed by STE-START= '1060-'1051=7. Each device boot, as well as the pre-boot, defines its own base offset to make the code easier to read. The use of these bases as memory reference modifiers requires that no literals be used. A FIN pseudo-op is placed at the end of each boot to allow easy identification of any literals so that they may be replaced with appropriately named locations. Since the boot program must be wholly contained within sector 1 (no LOAD generated cross sector links), instructions of the form LDA -1, 1 cannot be used. The command file C+CPB, also on UFD=AIDS, produces a SAVE file named *CPB. CPBOOT occupies 240 ('360) locations of PROM.

CPBGEN: CPBGEN punches a PROM simulator tape of locations '1000-'1777. Since the control panel μ-code expects the PROM to contain the one's complement of the desired locations, CPBGEN performs a preliminary backscan and inverts all locations except the unused trailing zeroes (a 0 is inherently more reliable than a 1 in PROM). Since PROM comes in 8 X 512 bit chips and the PROM simulator loads two parallel 8-bit banks, CPBGEN first punches all left bytes and then all right bytes. A given byte is punched as two ASCII hexadecimal digits followed by an ASCII apostrophe. For example, the bit pattern 10100110 is viewed as 1010/0110 = C6 and is punched as '303/'266/'247. A TAPE-ON ('222) turns the reader on and a TAPE-OFF ('224) turns the reader off. The final tape format is:

```
leader (48 inches): TAPEON: left bytes (0-'777): TAPEOFF:
blank tape (48 inches): TAPEON: right bytes (0-'777): TAPEOFF:
trailer (48 inches)
```

CPBGEN resides on UFD=AIDS of the master disk and the command file C←CPBG produces a run file named *CPBG. To punch a paper tape of CPBOOT, the following sequence of instructions must be used:

```
FILMEM                  insure sector '1000 filed with zeroes ;
RESTORE     *CBP        into '1000-'1777
ASSIGN      PUNCH       (DOS/VM only)
RESUME      *CPBG       at '2000
UNASSIGN    PUNCH       (DOS/VM only)
```

Physical PROM: To create the physical PROM, load the paper tape produced by CPBGEN into the PROM simulator and verify that the load is good. Use the PROM simulator to blow the actual PROM chips and insert them into the control panel.

Key-In Substitues for Control Panel Boots

Since the auto-load control panel PROM function is optional on some Prime computers, hand keyed-in programs are necessary. Because programs keyed in are likely to disappear after one use, these programs can be as long as desired, but should be as short as possible.

SECOND LEVEL DISK BOOTS (BOOT)

The control panel disk bootstraps (FHD/MHD/FLOPPY) read one 448 word DOS record from record 0 of the selected device into memory starting at location '770 and begin execution at '1000 in 16S mode. Regardless of the device booted from, the second level boot is the same and can, in turn, read DOS into memory from any DOS disk in the system. The source is named BOOT and resides in the UFD FILAID on the master disk. The run file on record 0 is also named BOOT and resides in the MFD.

APPENDIX C

CREATING SEGMENT DIRECTORIES AND FILES

Ufd's may be created at command level by the CREATE command. However, building a segment directory is a bit more involved. The following two source program listings show a means of creating a directory and files within the created directories. The program KEYCOM provides mnemonic keys for PRWFIL, SEARCH, and ATTACH to make programming easier. The second, GENFIL, creates segment directories that are both threaded (SAM) and directed (DAM) and shows how to create SAM and DAM files within the created directories by programming means.

C KEYCOM JPC 30 MAY 1974

```
C KEYCOM  JPC 30 MAY 1974
C  PROVIDES MNEMONIC KEYS FOR PRWFIL, SEARCH, AND ATTACH
      INTEGER PREAD, PWRITE, PREREL, PREABS, POSREL, POSABS, PCONV,
     X    OPNRED, OPNWRT, OPNBTH, CLOSE, DELETE, REWIND,
     X    TRNCAT, UFDREF, SEGREF, NTFILE, NDFILE, NTSEG, NDSEG, NEWUFD,
     X    MFDUFD, CURUFD, SEGUFD, HOMUFD, SETHOM
C
      DATA PREAD, PWRITE, PREREL, PREABS, POSREL, POSABS, PCONV
     X   / :1,    :2,     :0,     :10,    :20 ,   :30,    :400/
      DATA OPNRED, OPNWRT, OPNBTH, CLOSE, DELETE, REWIND, TRNCAT
     X   / 1,     2,      3,      4,     5,      7,      8   /
      DATA UFDREF, SEGREF, NTFILE, NDFILE, NTSEG, NDSEG, NEWUFD
     X    /:0,    :100,   :0,     :2000,  :4000, :6000, :10000/
      DATA MFDUFD, CURUFD, SEGUFD, HOMUFD, SETHOM
     X   / 0,     2,      4,      6,     1    /
```

```
C GENFIL, JPC 30 MAY 1974
C GENERATES FILE STRUCTURES FOR FIXRAT TO CHECK
C
C LOCAL VARIABLES
      INTEGER BUF(2000),BUF1(2000)
C
C KEYCOM CONTAINS MNENONIC KEYS FOR SEARCH, PRWFIL, AND ATTACH
C
$INSERT KEYCOM
C
      DATA BUF/2000*:123456/
      DATA BUF1/2000*:654321/
C
C GENERATE DAM FILE LONGER THAN 440 RECORDS
      CALL SEARCH(OPNWRT+NDFILE+UFDREF,'DAMFIL',1)
      DO 20 I=1,500
C
C LOC IS A FUNCTION THAT RETURNS A POINTER TO ITS ARGUMENT.
C
      CALL PRWFIL(PWRITE,1,LOC(BUF),440)
20    CONTINUE
      CALL SEARCH(CLOSE,0,1)
C
C GENERATE THREADED SEGMENT DIRECTORY
      CALL SEARCH(OPNBTH+NTSEG+UFDREF,'TSDIR ',1)
C
C OPEN NEW THREADED FILE ON UNIT 2 IN CURRENT POSITION OF
C SEG DIR OPEN ON UNIT 1
      CALL SEARCH(OPNWRT+NTFILE+SEGREF,1,2)
C
C WRITE STUFF IN THE FILE, THEN CLOSE IT
      CALL PRWFIL(PWRITE,2,LOC(BUF),2000)
      CALL SEARCH(CLOSE,0,2)
C
C RELOCATE POINTER IN SEG DIR TO EOF
      CALL PRWFIL(PREAD+PREREL,1,0,0,1)
C
C OPEN ANOTHER THREADED FILE IN SEG DIR, WRITE STUFF, THEN CLOSE
      CALL SEARCH(OPNWRT+NTFILE+SEGREF,1,2)
      CALL PRWFIL(PWRITE,2,LOC(BUF1),2000)
      CALL SEARCH(CLOSE,0,2)
C
C CLOSE SEGMENT DIRECTORY
      CALL SEARCH(CLOSE,0,1)
C
C GENERATE DIRECTED SEGMENT DIRECTORY
      CALL SEARCH(OPNBTH+NDSEG+UFDREF,'DSDIR ',1)
C
C WRITE A THREADED FILE IN IT
      CALL SEARCH(OPNWRT+NTFILE+SEGREF,1,2)
```

```
C  GENFIL.  JPC  30 MAY 1974


      CALL PRWFIL(PWRITE,2,LOC(BUF1),2000)
      CALL SEARCH(CLOSE,0,2)
C
C CLOSE SEG DIR
      CALL SEARCH(CLOSE,0,1)
C
C
C GENERATE DIRECTED SEG DIR LONGER THAN 1 RECORD
C OPEN DIRECTED SEG DIR NAMED LONSEG ON UNIT 1 IN CURRENT UFD
      CALL SEARCH(OPNSTH+NDSEG,'LONSEG',1)
C
C SENERATE 500 DIRECTED FILES IN SEGMENT DIR
      DO 10 I=1,500
        CALL SEARCH(OPNWRT+NDFILE+SEGREF,1,2)
        CALL PRWFIL(PWRITE,2,LOC(BUF),10)
        CALL SEARCH(CLOSE,0,2)
        CALL PRWFIL(PREAD+PREREL,1,0,0,1)
10    CONTINUE
C
C CLOSE SEG DIR
      CALL SEARCH(CLOSE,0,1)
C
      CALL EXIT
C
      END
$0
```

APPENDIX D

DATA BASE MANAGEMENT

GENERAL INFORMATION

This appendix describes the features of DOS and DOS/VM software that facilitate development of Data Base Management (DBM) System. The data base structures are defined and then used in several examples that show the speed and flexibility of Prime supporting software.

Features of DOS and DOS/VM that facilitate DBM are:

- Sequential file access (SAM)
- Directed file access (DAM)
- Segmented structures with multiple growth points
- Relative and absolute positioning
- Pre-and post-access positioning
- Expandable file dimensions
- Security at the UFD, sub-UFD, and segment directory levels
- Multiple user access to any file (DOS/VM)
- FORTRAN callable file manager
- Associative buffering (DOS/VM)
- 2400 RPM Moving Head Disks
- 30 Megaword storage per device

DEFINITION OF DATA BASE MANAGEMENT

The objective of DBM systems is to use the processing power of the computer to collect and organize data and to make data easily accessible to the user. DBM systems consist of a set of programs that create and maintain complex data structures known as data bases, and a set of library procedures that enable users to access, modify, and report on the content of the data bases.

Data Base Terminology

To make the discussions and examples that follow meaningful, it is necessary to establish a data base structure. The data base structure consists of three basic structures: data items, data entries, and data sets.

The data item is the smallest accessible data element. Each data item is a value and is referenced by a data item name. Usually, many data item values are referenced by the same data item name. For example:

DATA ITEM NAMES	DATA ITEM VALUES
CITY	DENVER, BOSTON, MIAMI
STATE	COLORADO, MASS, FLORIDA
ZIP	01767, 01752, 07353

The data item is defined as N words (or bytes) of a physical disk record.

The data entry is an ordered collection of related data items and is defined by an ordered listing of the data item names. Data entries are all the same length and are stored in physical disk records. For example:

DATA ENTRY		DATA ITEM NAMES		
		NAME	CITY	STATE
DISK RECORD	N	SMITH	DENVER	COLORADO
	N+1	JONES	BOSTON	MASS
	N+2	GREEN	MIAMI	FLORIDA

The data set is a collection of data entries sharing a common definition. A data set name references any or all of the data entries of a data set. The number of data entries in a data set is limited by available disk space.

There are two types of data sets: master data sets and detail data sets.

Detail data sets contain "line item" information, e.g., in the detail data set PERSONNEL, each person's location, education, etc., is stored.

Master data sets serve as indices to detail data sets. The data entries of a master data set contain pointers to corresponding detail data sets.

In general, access to data within a data base is carried out at the data entry level. Each CALL to a DBM procedure accesses some or all of the data items within a data entry. The functions provided by DBM procedures include adding a new data set, deleting a data entry from a data set, reading some or all of the data items of a data entry, and changing the values of items in a data entry.

Accessing the Data Base

Although access time to specific data in a data base is dependent on the structure of the master and detail data sets, the speed and flexibility of the underlying disk file manager is also significant.

SAM files are a linear array of records threaded with forward and backward pointers. Therefore, to access the last record of a lengthy SAM file (data set) of 47 records, all previous records must be read 47 access times to locate and read the last record. However, the same data set using a DAM file structure would require only three access times to read the same record. The DAM file consists of a record directory maintained by the file system. To access any record in a DAM file takes one disk accses to read the directory, and one additional access to read the desired record if it is not the first record in the target file. For 30M word disk, with an average total disk access time of 47.2 ms, the difference is roughly 2-1/2 seconds vs. 1/10 of a second.

However, for applications where files are only one or two records in length, the SAM file structure is as fast or faster. DAM files require one disk access just to retrieve the record directory.

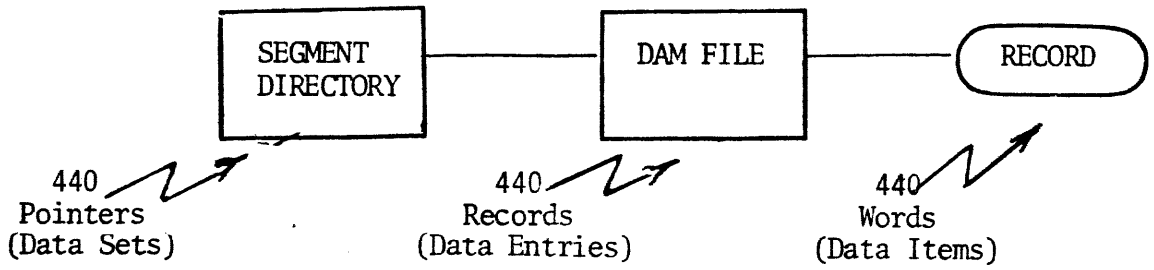
Another feature of the SAM file is the way in which the file system computes the best method for locating a record within the file. If the pointer to a SAM file with ten records is positioned at record #8, and the next access addresses record #3, the file system determines whether traversing the file backwards or positioning at the beginning record and traversing forward is faster. In this example, the latter method is selected; thus three accesses instead of five will be made.

Positioning in a SAM or DAM file can be done relatively with a + or - 32767 words (+ or - 74 records) parameter or absolutely with record number, word number parameters. The moving of the file pointer can be pre- or post- the disk access.

For example, let a data set be defined as a SAM or DAM file in a segment directory. A segment directory is a named SAM or DAM file that contains pointers (physical record addresses) to SAM or DAM files (data sets). Thus, each data set has its own growth point. Any number of data sets can be grouped together in a single segment directory providing disk space is available. The segment directory is then defined as the data base.

A tree structure for a file may be developed consisting of a single segment directory and 440 DAM files (see Figure D-1). To access a single word out of 85 million (three times the capacity of a 20 surface device) requires only four disk transfers. If repeated accesses are performed over the entire tree, each additional access requires at most three additional disk transfers. If repeated accesses are done within a local part of the file (193,600 words), each additional access requires, at most, one additional disk transfer.

Figure D-1



In Figure D-1, the segment directory is shown with only 440 pointers, this is not a limit. The segment directory can be expanded just like any ordinary file. However, a segment directory with 440 pointers can directly address 7,744,000 words of data. Segment directories can have "holes". For example, directory entries 1, 3, and 5 can contain valid data set pointers while entries 2 and 4 are not used. This is useful when data sets are arranged logically in a segment directory and additional data sets need to be incorporated later.

FILE SYSTEM PERFORMANCE

No file in the system has a fixed length providing disk space is available. However, as files become larger, an eventual decrease in performance occurs. For the SAM file, more and more disk accesses are necessary to traverse the file. The DAM file, however, has a boundary where performance falls off. The DAM file directory can handle only 440 record addresses; so as the file becomes larger, the 441st, 442nd, etc. records are not directly addressable and must be read sequentially. This only occurs, however, when DAM files exceed 193,600 words.

DISK ACCESS TIME

30 million word disk:

	SEEK	ROTATION	ACCESS TIME
Average =	35 ms	12.5 ms	47.5 ms
Maximum =	70 ms	25. ms	95 ms

FILE SECURITY

The DOS/VM file system has passwords and access attributes associated with the user file directories (UFD's) and sub-UFD's. Both the owner and nonowner passwords are defined with the command PASSWD. The PROTECT command allows the association of access attribute with files in a UFD to limit their use if desired. The UFD is always a SAM file and contains up to 72 named files, segment directories, and sub-UFD's. Once a user is attached to a UFD or sub-UFD, he has access privileges to files in that UFD within limits that may be defined by the PASSWD and PROTECT commands (Refer to Sections 2 and 4). UFD's can be created that contain executable DBM programs for different levels of security. The user, although attached to the UFD has no way to dump, modify, or delete the executable programs if the command directory (CMDNCØ) was empty or protected by an owner password unknown to him, or by a combination of passwords and protection attributes.

Under multi-user (DOS/VM), more than one user has access to a file simultaneously, provided it is opened for reading only. If on the other hand, the file is opened for writing by one user, other users are prevented through a locking algorithm from reading or writing it.

Under DOS/VM, associative buffering is implemented with 32 buffers. Each buffer contains one disk record (440 words). A least recently used (LRU) algorithm is used when a record not in the buffer is accessed. This greatly decreases access time for a DBM system because the master data set directories (the indexes to all data) tend to remain in buffers because of the high number of references to them.

To further illustrate the capability of the file system, several examples are given that show different data base structures, and the access times and resources required when traversing them.

Example #1:

This example uses a file structure consisting of a segment directory with 440 SAM files and then with 440 DAM files to show the difference in average access time. All the files are five records long, (2200 words).

440 x 2200 (968,000 words)

SAM FILES			DAM FILES		
Seg. Dir.	440	(SAM)	Seg. Dir.	440	(SAM)
	[1]		DAM Dir.	[1]	
	[2]	SAM FILE		[2]	DAM FILE
	[3]	(2200 Words)		[3]	(2200 Words)
	[4]			[4]	
	[5]			[5]	

DISK ACCESSES		DISK ACCESSES	
OPEN Seg. Dir.	1	OPEN Seg. Dir.	1
OPEN SAM file	1	OPEN DAM file	1
AVG file access	2	AVG file access	1
TOTAL	4	TOTAL	3

Example #2:

This example is similar to Example #1 but with much larger files. All files are 50 records long (22,000 words).

440 x 22,000 (9,680,000 words)

DISK ACCESSES		DISK ACCESSES	
OPEN Seg. Dir.	1	OPEN Seg. Dir.	1
OPEN SAM file	1	OPEN DAM file	1
AVG file access	24.5	AVG file access	1
TOTAL	26.5	TOTAL	3

Example #3

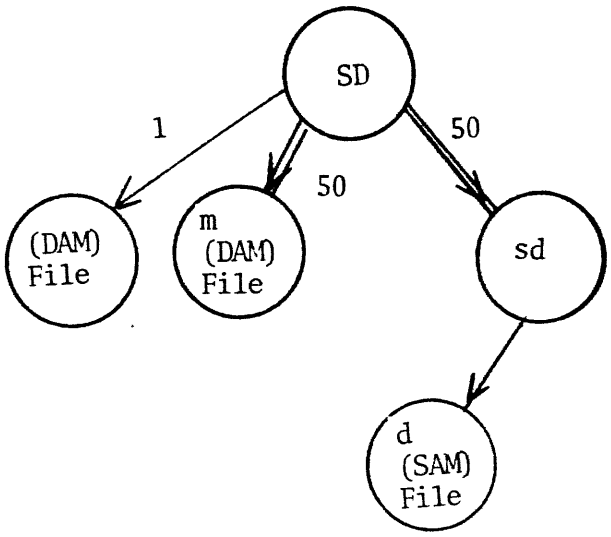
This example uses two levels of segment directories and the SAM/DAM comparison. The first directory contains pointers to ten other segment directories which each contain pointers to 440 files. The files are five records long (2200 words).

10 x 440 x 2200 (9,680,000 words)

SAM FILES			DAM FILES			
1st Seg. Dir.	10	(SAM)	1st Seg. Dir.	10	(SAM)	
2nd Seg. Dir.	440	(SAM)	2nd Seg. Dir.	440	(SAM)	
	[1]		DAM Dir.		[1]	
	[2]				[2]	DAM FILE
	[3]	SAM FILE			[3]	(2200 words)
	[4]	(2200 words)			[4]	
	[5]				[5]	

DISK ACCESSES		DISK ACCESSES	
OPEN Seg. Dir. #1	1	OPEN Seg. Dir. #1	1
OPEN Seg. Dir. #2	1	OPEN Seg. Dir. #2	1
OPEN SAM file	1	OPEN DAM file	1
AVG file access	2	AVG file access	1
TOTAL	5	TOTAL	4

Example #4 shows a more complicated file structure. A segment directory which contains one pointer to a master index file (DAM), 50 pointers to master data sets (DAM files), and 50 pointers to data segment directories. The data segment directories contain the pointers to the detail data sets (SAM files of 880 words).



The time and resources to read randomly into this file structure follow.

Initialize	DISK ACCESSES	BUFFERS	UNITS
OPEN SD	1	1	1
OPEN sd	1	1	1
OPEN M	2	2	1
	4		
Read master directory to select index			
ACCESS M	1	0	0
Read index			
OPEN m	2	2	1
ACCESS m	1	0	0
Read data			
OPEN SAM	1	1	1
ACCESS SAM	.5	0	0
	5.5	7	5

The maximum number of buffers available under DOS is 16, under DOS/VM there are 32. Example #4 shows seven buffers open which occupies 3.1K words of memory. There are 16 units available under both DOS-DOS/VM.

APPENDIX E

ALL ABOUT FIXRAT

INTRODUCTION

The external command FIXRAT loads and restarts a maintenenance program that checks the DOs file integrity on any disk pack. FIXRAT fully supports nested UFDs and nested Segment Directories. FIXRAT handles all current available disks. The command FIXRAT runs under either DOS or DOS/VM

FIXRAT DESCRIPTION

The external command FIXRAT runs either under DOS or DOS/VM; it loads and starts a maintenance program that checks the file integrity. Before reading this document, the user should read a description of the file structure found in Section 2. Existing DOS users should also read this section for a description of Segment Directories, nested directories and FIXRAT printout options.

FIXRAT reads every record in every file, UFD, and segment directory, and checks that information in each record header is consistent with record headers in the rest of the file and consistent with the file directory that contains the record.

Any inconsistencies generate an error message. FIXRAT also builds a record availability table (RAT) from the existing file structure and compares it to the DSKRAT file for agreement. If discrepancies are found, FIXRAT prints an error message.

If requested, FIXRAT will not only check the file structure but also repairs pointers (if possible) or truncates or deletes defective files and generates a corrected DSKRAT file. Up to two repetitions of FIXRAT may be necessary to repair a damaged file structure. The recommended procedure is to repeat FIXRAT until an error free printout is obtained.

FIXRAT must be run whenever there is reason to expect that the file structure is damaged - for example, if a program being debugged runs wild and writes over part of DOS. Until the user gains experience with the syste, he should run FIXRAT at the close of every operating session. Never attempt to run FIXRAT after a COPY has aborted.

The suggested procedure to maintain a disk pack is to run FIXRAT every morning and, if no errors occur, to copy the pack onto a daily backup pack. If any files are truncated or deleted from the pack, they are copied from the daily backup disk, if they exist there, to the

disk-pack before copying that pack onto an updated daily backup disk. The owners of the bad files must be notified that those files have been copied from the backup and any modifications to those files may have been lost.

Running FIXRAT

The command is:

```
FIXRAT  [OPTIONS]
```

If the word OPTIONS is included, FIXRAT requests printout options, otherwise, FIXRAT prints the name and number of records used (in octal) in the MFD and in each directory in the MFD. When entered, FIXRAT asks the question:

```
FIX  DISK ?
```

If the answer is YES, .CR., FIXRAT truncates or deletes defective files and generates a corrected DSKRAT file in addition to checking the file structure and repairing all file structure errors. FIXRAT then asks the question:

```
PHYSICAL  DISK  DRIVE  =
```

The user types the physical disk drive in octal on which FIXRAT is to be run followed by .CR., FIXRAT then prints the disk pack identification (which is the name of the DSKRAT) and begins processing the file structures. The DSKRAT is always the first file in the disk pack ID. The default name given to DSKRAT is DSKRAT.

The following is a sample DOS file structure:

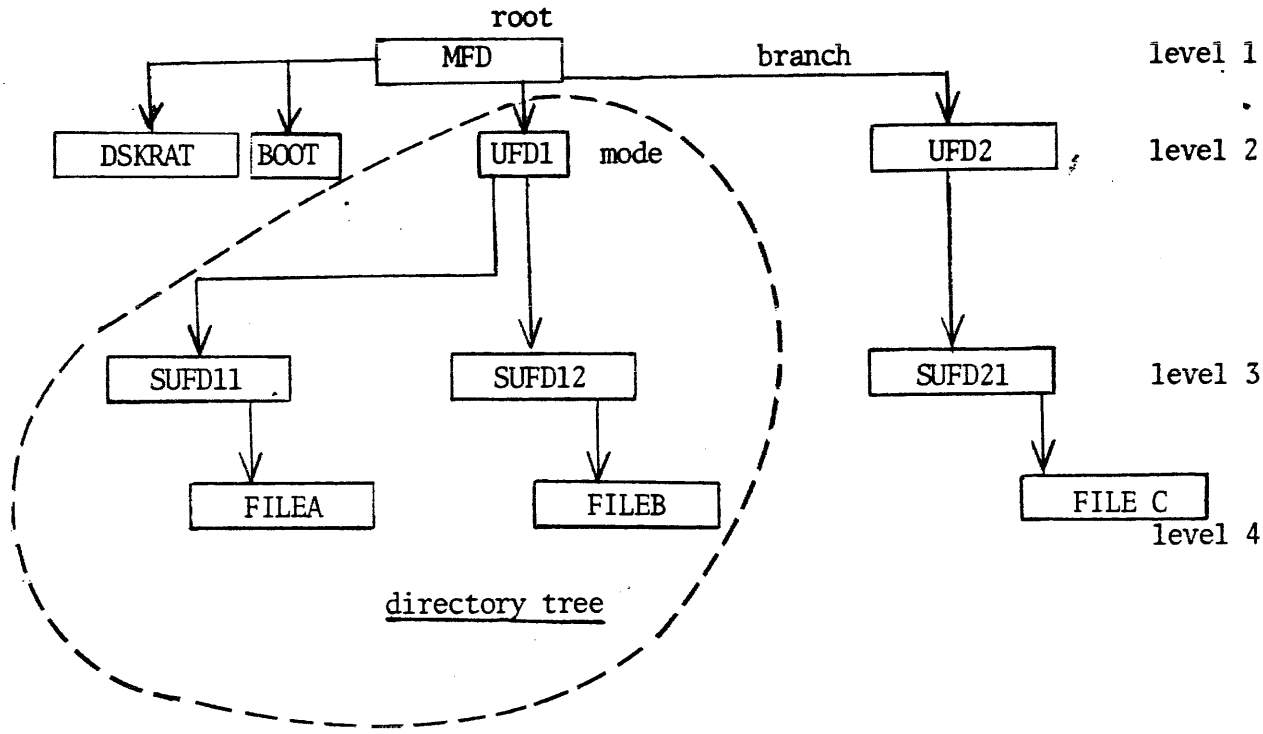


Figure E-1. Sample File Structure

The file structure on any disk pack is a tree structure where the MFD is the root or trunk of the tree, the links between directories and files or subdirectories are branches; and the directories and files are nodes.

A directory tree consists of all files and subdirectories that have their root in that directory. In Figure E-1, the directory tree for UFD1 is circled. The level of a file is the depth of that file in the tree. For example, as shown in Figure E-1, the MFD is at level 1 in the tree, UFD1 is at level 2 in the tree, and FILEC is at level 4.

FIXRAT traverses the file structure as shown by the snaked line generating typeout at the various points below.

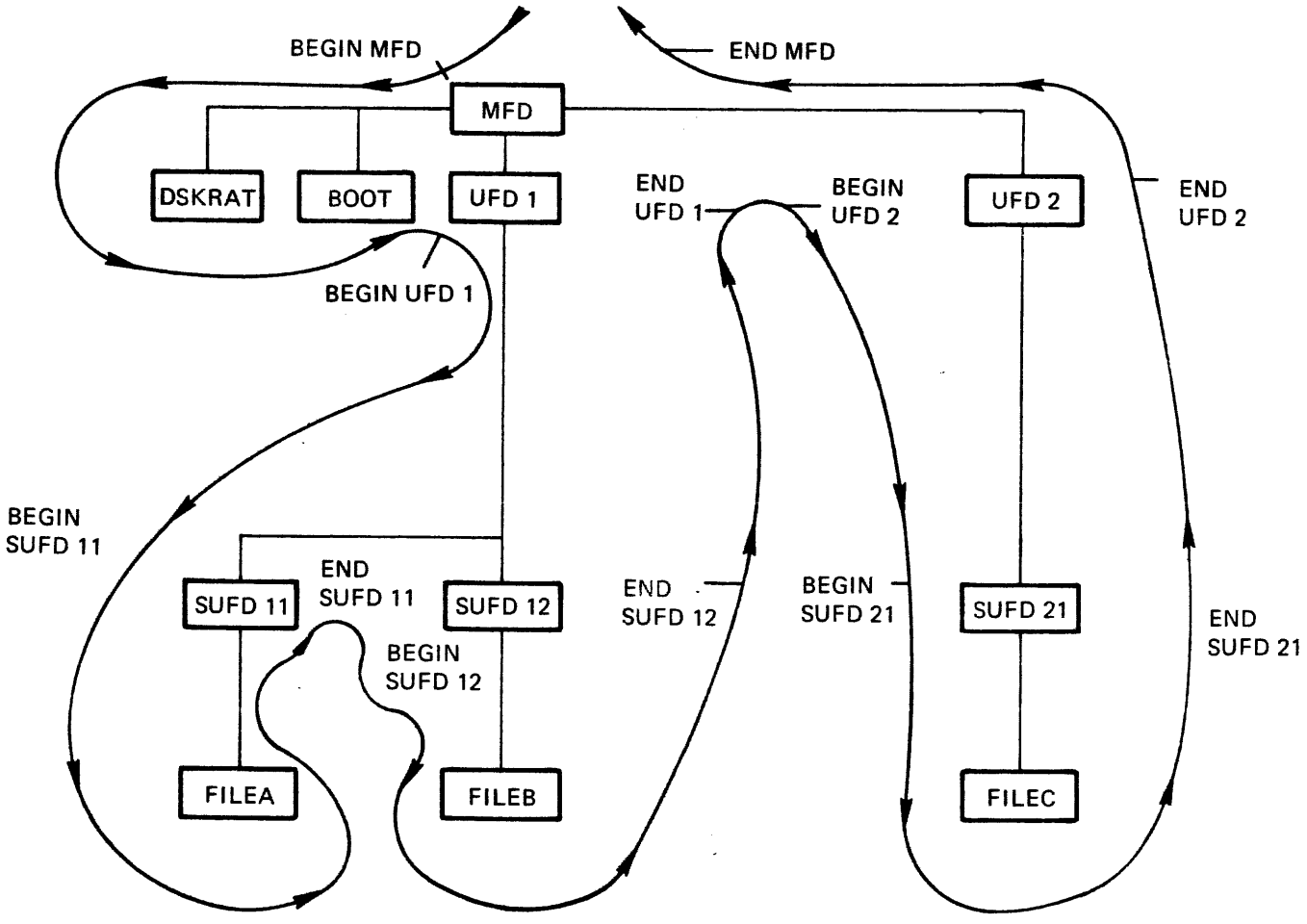


Figure E-2. Typical FIXRAT Traverse of File Structure

The terminal output appears as follows for the above file structure:

```
BEGIN   MFD
 BEGIN   UFD
  BEGIN   SUFD11
  END     SUFD11    10
  BEGIN   SUFD12
  END     SUFD12    10
 END     UFD1      21
 BEGIN   UFD2
  BEGIN   SUFD21
  END     SUFD21    10
 END     UFD2      11
END     MFD       35
RECORDS USED  =  35
RECORDS LEFT  =  6223
DSKRAT  OK
OK,
```

FIXRAT prints the word BEGIN followed by the directory name when beginning processing of a directory tree. On leaving a directory tree, FIXRAT prints END Directoryname followed by the number of records (in octal) used by all files and directories in the directory tree. In the above example, the number following MFD ,35, is the same as the total number of records used as the MFD directory tree and consists of all files and directories on the disk pack. FIXRAT indents the printed output one space for each level down in the tree in which the directory is located. This format makes it easy to understand the relationship of each directory to the other directories in the tree. To prevent excessive output, FIXRAT as a default, only prints out directory names at levels 1 and 2 in the tree corresponding to the MFD and all directory names in the MFD file. Unless OPTIONS are specified, FIXRAT processing of the tree shown in Figure E-1 generates the following default output rather than the complete output shown above:

```
BEGIN   MFD
 BEGIN   UFD1
 END     UFD1   21
 BEGIN   UFD2
 END     UFD2   11
END    MFD     35
RECORDS USED =  35
RECORDS LEFT = 6223
DSKRAT  OK
OK,
```

If the command FIXRAT OPTIONS is given, FIXRAT asks the question FIX DISK? and PHYSICAL DISK DRIVE =, as before, and also asks:

```
TYPE DIRECTORIES TO LEVEL =
```

The user must type an octal number corresponding to the lowest level in the tree structure that directory names are to be printed. The following table describes the output:

LEVEL	Output
blank	all directories
1	MFD only (level 1 directory)
2	MFD and all directories in MFD file (level 2 directories)
3	all output for level 2 and all directories at level 3 (level 3 directories)
etc.	etc.

FIXRAT will then ask:

TYPE FILE NAMES ?

If the answer is YES, followed by .CR., FIXRAT prints all filenames in all directories, indented appropriately. This option is useful to list the contents of a disk. Note that unless the user requests suppression of directory name output by answering the TYPE DIRECTORIES TO LEVEL = question with the parameter one, directories are printed three times - twice as directories and once as files.

FIXRAT will then ask:

TYPE FILE CHAINS ?

If the answer is YES, followed by .CR., FIXRAT prints the disk address of all records in all files on the disk. All files consist of one or more records chained together by pointers. This option is useful to see how files are scattered on a disk. FIXRAT begins processing the disk after this point as it would if the OPTIONS parameter was not specified.

Following the file structure analysis printout, FIXRAT prints the number of records used on the pack and the number of records left on the pack for file system use. Finally, FIXRAT compares a record availability table built from the existing file structure against the DSKRAT. If they match, FIXRAT types DSKRAT OK and exits to DOS or DOS/VM. If they do not match, FIXRAT types DSKRAT FILE DIRECTORIES MISMATCH.

FIXRAT Output Example

The following is sample FIXRAT output generated after all questions have been answered:

```
DISK  PACK  ID  IS  DSKRAT

BEGIN  MFD
 BEGIN  CMDNC0
 END    CMDNC0     000021
 BEGIN  DOS
 END    DOS        000011
 END    MFD        000035
 RECORDS  USED  = 000035
 RECORDS  LEFT  = 006223
 DSKRAT  OK
 OK:
```

The first line prints the disk pack identification as the name of the DSKRAT. The DSKRAT is always the first file in the MFD, so FIXRAT prints the name of the first file as the disk pack name. The default name given to the DSKRAT is DSKRAT.

The next section of output concerns FIXRAT examining the file structure on the disk for consistency. This ample output is generated from a disk that contains only two directories, CMDNC0 and DOS, in the MFD. If either of these directories contains subfile directories, FIXRAT traces the nested directory structure but does not print the name of the subfile directories. Each directory is printed twice: following the word BEGIN when FIXRAT enters the directory and following the word END when FIXRAT is finished processing the directory and any subfile directories nested within it. Directories that are files in the MFD are indented one space when typed to show the nested structure.

Following the directory name, FIXRAT prints a number that is the number of records used in the directory and all files nested within that directory. Since all files on a pack are nested within the MFD, the number of records used in the MFD always matches the number of records used on the disk pack.

Following the file structure analysis typeout, FIXRAT prints the number of records used on the pack and the number of records left on the pack for file system use.

Finally, FIXRAT will compare a record availability table (RAT) built from the existing file strucrure against the DSKRAT. In the preceding example, they match and FIXRAT prints:

```
DSKRAT   OK
```

and exits to DOS or DOS/VM.

If the RAT and DSKRAT totals do not match, FIXRAT prints:

DSKRAT, FILE DIRECTORIES MISMATCH

If the user typed YES to the question FIX DISK?, FIXRAT repairs the DSKRAT and types:

DSKRAT FIXED

then exits to DOS or DOS/VM. Otherwise, FIXRAT asks the user: FIX DISK? If the user answers YES, .CR. the DSKRAT is repaired. This option is useful if there are no file structure errors but there is a bad DSKRAT.

If the user typed YES to the question FIX DISK? asked at the beginning, FIXRAT repairs the DSKRAT and types DSKRAT FIXED, then exits to DOS or DOS/VM. Otherwise, FIXRAT asks the user FIX DISK?. If the user answers YES, followed by .CR., the DSKRAT is repaired, DSKRAT FIXED is typed, and control exits to DOS or DOS/VM. This option is useful if there are no file structure errors but there is a bad DSKRAT.

Broken File Structure Messages

When FIXRAT detects a problem in the file structure, it prints an error in the following format:

reason for error

FILE - filename BAD RECORD = octal record address.

DIRECTORY PATH = list of directories

FILE DELETED, FILE TRUNCATED or blank

The directory path is the list of nested file directories needed to get from the MFD to the bad file. For example, if FILEC in Figure E-2 was broken, the directory path would be MFD, UFD2, SUFD21. Because all files have the MFD as a root, "MFD" is not printed as part of the path.

After printing the directory path, FIXRAT prints how it disposed of the bad files. If the FIX DISK question was answered NO, FIXRAT does nothing to the file, therefore prints nothing. Otherwise, FIXRAT either truncates the file before the bad record and prints FILE TRUNCATED, or finds no part of the file can be saved, removes the file directory entry from a UFD, or zeroes the entry in a segment directory and prints FILE DELETED.

Segment Directories

A segment directory may contain references to files, other segment directories and User File Directories (UFD). The distinction between a UFD and a segment directory is that entries in a UFD are referenced by name and those in a segment directory by position. Recall that each entry in a UFD consists of a one-word disk address that is the beginning

record address of the file, followed by a six letter name and two spare words. (Refer to Appendix A.) In a segment directory, FIXRAT prints the absolute position of the file in the segment directory as an octal number-pair (record number, word number). For example, the first entry in a segment directory is printed as (0, 0), the second entry as (0, 1), the 440th as (0, 440), and the 441st entry as (1, 0). Notice that as with user file directories, indentically named files in different segment directories represent unique files.

If FIXRAT is requested to FIX DISK and detects a bad file, it either truncates or deletes the file depending on where in the file a problem is detected. If FIXRAT deletes a file, the action taken depends on the type of directory the file is entered in. If the directory is a UFD, FIXRAT removes the entry from the directory similar to the action of the DELETE command. If the directory is a segment directory, FIXRAT sets the entry to zero. On the next pass, FIXRAT skips the zero entry. The convention, then, is that a zero in a segment directory represents a null file.

PITFALLS AND RESTRICTIONS

Bad BOOT

If the BOOT file in the MFD is accidentally deleted or broken, DOS will allocate record number 0 to the next new file. FIXRAT will complain if any file except the BOOT in the MFD contains record 0. The message given is:

```
BAD DISK ADDRESS  BAD RECORD = 0
```

If this occurs, RESTOR from a good MFD and SAVE the BOOT into the MFD before doing anything else.

Directory Nesting Limit

FIXRAT will trace nesting of directories to a depth of 100 levels only.

Writing Into Directories

Since directories may be nested, the possibility of accidentally writing bad data into directories is increased. This generates a bad file structure detected by FIXRAT. To minimize this possibility, it is suggested that users preface all except those in the MFD by U← and all segment directories by S←.

Deleting Directories

Do not delete a UFD or segment directory before deleting all files contained in the directory. If this is not done, the records used by files in the directory are not returned to the DSKRAT. When the next file FIXRAT is run, the message DSKRAT, FILE DIRECTORIES MISMATCH is given, and the records of files not deleted explicitly are not recovered for use.

FIXRAT ERROR MESSAGES

This appendix lists all error messages generated by FIXRAT and gives an expanded explanation of them. The user should be familiar with the details of the file structure. Error messages are of the form:

reason for error

FILE = filename BAD RECORD = octal number

DIRECTORY PATH = list of nested directories

Description of Messages

DSKRAT BAD

This message is obtained if the DSKRAT file contains any bad record pointers, the DSKRAT date header word is not 5, or the number of data words in the DSKRAT file does not match (NRECS+15)/16-+5. If the DSKRAT is BAD, FIXRAT reconstructs it using parameters typed by the user in response to the following questions. If the user types CARRIAGE RETURN to any of the questions, default values are used. The questions are:

```
INPUT  OCTAL  RECORD  SIZE  =
INPUT  OCTAL  FILE  SPACE  RECORD  COUNT  =
INPUT  OCTAL  CYLINDERS  =
INPUT  OCTAL  HEADS  =
```

The default values are respectively, 700, 6260, 313, 2. FIXRAT types these values back to the user for verification then asks "OK?". If the answer is YES, FIXRAT repairs the DSKRAT and continues, or else it requests the parameters again.

BAD DISK ADDRESS

A pointer to a disk record is out of range. Acceptable range is between 1 and NRECS -1, where NRECS is the number of records available for file system use. NRECS is stored in the DSKRAT data header. A record address of 0 is acceptable only for the disk bootstrap loader file BOOT in the MFD.

BAD RECORD ID

The first word of a record contains a number unequal to the record address of the record. This message is preceded by 10 disk error messagse as this problem could indicate a disk drive problem.

FIXRAT has difficulty determining whether the error is a disk drive error or a broken file. This case occurs if a record has a bad record identification word. The disk driver retries 10 times producing 10 disk error messages, then returns to FIXRAT, which prints the message BAD RECORD ID. Be sure to allow FIXRAT 10 disk error messages before assuming there is disk drive trouble.

BRA POINTER MISMATCH

The second word of the second record (or greater) of a file does not point to the beginning record of the file.

FATHER POINTER MISMATCH

The second word of the first record of a file does not point to the beginning record address of the file directory of the file.

BACK POINTER MISMATCH

The back pointer of a record, word 4, does not point to the previous record of the file, or if the current record is the first record of a file, the back pointer is not 0.

BAD WORD COUNT

The data word count, word 5 of a record is not between 0 and 440. Note that it is OK for a record to contain a word count of 0 which indicates an empty record.

BAD FILE TYPE

Word 6 of the first record of a file is not between 0 and 4, the legal file types for Rev. 5 DOS.

TWO FILES POINT TO SAME RECORD

Two files point to the same first record. FIXRAT prints the name of the second file only. This error may occur if the DSKRAT is changed by a user overwriting DOS. Records already used have been erroneously made available to new files.

BAD DAM POINTER

A DAM data file or DAM segment directory has a bad index in the first record of the file. The nth index of the file must point to the nth record of the file for all records of the file, or this message is given. This error is repaired by FIXRAT.

UFD LONGER THAN RECORD

A UFD is longer than 1 record. DOS expects all UFDs to be only 1 record long.

BAD UFD HEADER

Data word 1 of a UFD file does not contain 8 (decimal), the first word of a UFD header.

DIRECTORIES NESTED TOO DEEP

Directories may be nested to a depth of 100 levels. FIXRAT cannot follow the directory tree because the user has nested directories to more than 100 levels.

BAD STRUCTURE MESSAGES

```
FILE = MFD        BAD RECORD = 7
DIRECTORY PATH = MFD
FIXRAT ABORTED
```

A MFD has been altered and damaged. The best action to take is to copy the backup disk onto the "daily user disk" and continue from there.

```
DSKRAT NOT IN MFD
FIXRAT ABORTED
```

The DSKRAT has been accidentally deleted from the MFD. Suggested action is same as above.

```
RECORD READ OK NOW CHECKS BAD
POSSIBLE DRIVE ERROR, FIXRAT ABORTED
```

Suggested action is to run the disk diagnostic on a scratch pack at this point.

```
DIRECTORY RECORD READ OK NOW CHECKS BAD
POSSIBLE DRIVE ERROR, FIXRAT ABORTED
```

Suggested action is same as above.

CHECK FOR MFD INTEGRITY

FIXRAT checks that the first three entries in the MFD are DSKRAT, MFD and BOOT. The DSKRAT may have any name and the name is used on the disk pack ID (identification). The error messages thay may arise as a result of one of these entries missing are:

```
DSKRAT NOT IN MFD, REPLACE IT?

MFD NOT IN MFD, REPLACE IT?

BOOT NOT IN MFD, REPLACE IT?

MFD HAS BAD NAME, REPLACE?
```

If YES (followed by CR) is responded to each of these questions, the specified action asked in the message is performed. The user must not delete or alter the DSKRAT, MFD, or BOOT since these are system files used by DOS and DOS/VM.

FIXRAT and 30-Million Word Disk

FIXRAT supports the 30-million word disk. If the 30-million word disk is treated as a single disk device (no partitioning), the disk numbers 0, 1, 2, 3, attached to the controller are 5252, 5253, 5254 and 5256, respectively. If the disk is partitioned, disk numbers include head-offset and number-of-heads information. If the user gives an incorrect disk number, one of the following messages is printed at the terminal:

DEVICE, DSKRAT DIFFER IN HEAD COUNT. ABORT?

DISK READ ERROR with status of 17777

The user must restore FIXRAT (via the RESTOR command) with the correct disk number, which is the one used normally with the large disk on DOS, DOS/VM and COPY (Refer to Table 3-1).

APPENDIX F

FILE UTILITY

(FUTIL)

INTRODUCTION

FUTIL is a file utility command that provides commands for the user to copy, delete, and list files and directories. FUTIL has an attach command that allows attaching to subdirectories by giving a directory pathname from either the MFD or the home-UFD to the subdirectory. FUTIL allows operations not only with files within User-File-Directories (UFD's) but also files within segment directories. For complex operations, FUTIL may be run from a command file.

FILE STRUCTURE

A user should be generally familiar with the Prime file structure. Refer to Section 2 entitled "File Structures". Some new terms that are used to describe FUTIL commands are now described. Figure F-1 is a sample file structure.

The DOS/VM file structure on any disk pack is a tree structure where the MFD is the root or trunk of the tree, the links between directories and files or subdirectories are branches; and the directories and files are nodes. A directory tree consists of all files and subdirectories that have their root in that directory. In Figure F-1, the directory tree for UFD1 is circled. An MFD directory pathname consists of a list of directories and passwords necessary to move down the tree from the MFD to any directory. For example, the MFD pathname for SUFD11 is:

```
MFD  MFDPASSWORD  >  UFD1 UFD1PASSWORD  >  SUFD11 UFD11PASSWORD
```

The character ">" separates directories in the pathname and suggests that one is proceeding down a tree structure.

An MFD directory pathname may optionally omit the MFD and may optionally include the logical disk number of the pack or the packname. Examples:

```
UFD1 UFD1PASSWORD  >  SUFD11 SUFD11PASSWORD
< 1 > UFD1 UFD1PASSWORD  >  SUFD11 SUFD11PASSWORD
< TDISK  >  UFD1 UFD1PASSWORD  >  SUFD11 SUFD11PASSWORD
```

MAN 1675

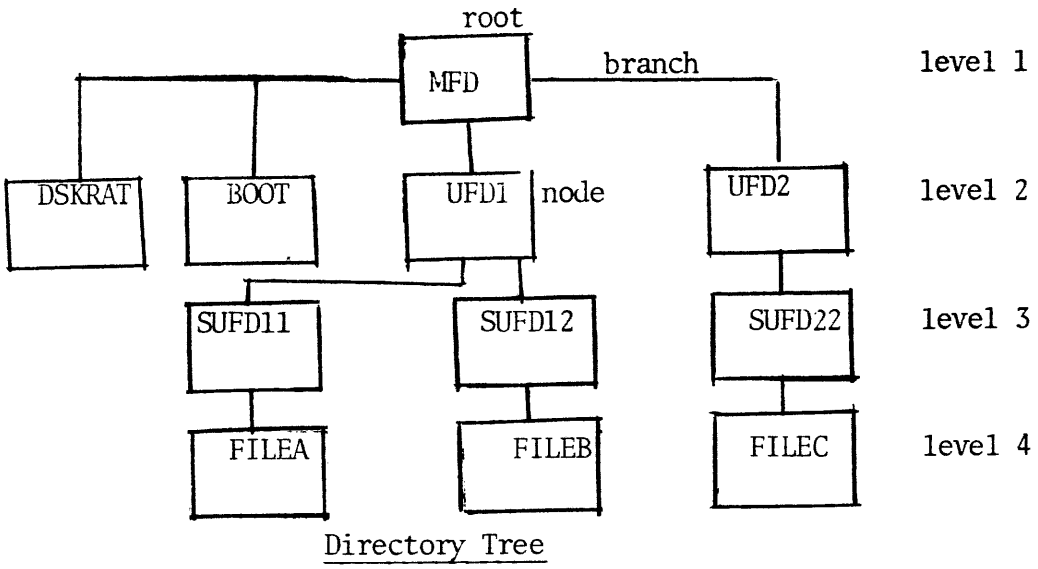


Figure F-1. Sample File Structure.

The logical disk number may optionally follow the first UFD as follows:

```
UFD1 UFD1PASSWORD 1  >  SUFD11 SUFD11PASSWORD
```

If no pack name or disk number is given, the logical disk referred to is the lowest numbered logical disk in the MFD in which UFD1 appears. A user, using the ATTACH or DOS/VM LOGIN command may specify a particular user-file-directory in the file structure as the home-UFD. Additional FUTIL ATTACH commands may refer to either the MFD or the home-UFD as the starting point. A home-UFD directory path name consists of a list of directories and passwords necessary to move down the tree from the home-UFD to any directory that has the home-UFD as the root. For example, if the home-UFD is UFD1, the home-UFD path name to SUFD11 is:

```
*  >  SUFD11 SUFD11PASSWORD
```

"*" represents the home-UFD. The home-UFD path-name to UFD1 is simply *.

A user-file-directory is a file that consists of a header and a number of entries (0-72). Each entry consists of 1 to 6 character filenames, protection attributes of the file, and a disk record address pointer to the file. A segment directory is a file consisting of an unlimited number of entries, each entry being a disk record address pointer to the file. A null pointer indicates no file at that entry. To refer to a particular file in a segment directory, a user must specify the file position of the entry in the segment directory. Refer to Section 5 (PRWFIL) for details of file positioning. A user may specify the position as an absolute position (record-number, word-number) where record-number is between 0 and 32767 and word-number is between 0 and 439. There are 440 data words in each disk record so there are 440 files in each record of segment directory. The first file can be referred to as (0,0), the second as (0,1), the 440th file as (0,439), and the 441st file as (1,0). The construction (record-number, word-number) is referred to as a segment directory filename. In FUTIL, arguments to the commands are either user-file-directory filenames or segment directory filenames depending on the directory type the file is under. Furthermore, names specified as parameters to the LISTF command of FUTIL are of either type depending on the directory type.

DESCRIPTION OF FUTIL COMMANDS

To invoke FUTIL, type FUTIL. When loaded, FUTIL prints the prompt character, >, and awaits a command string from the user terminal. To terminate long operations such as LISTF, type CTRL P and restart FUTIL at 1000. A user must type a command followed by a carriage return and wait for the prompt character before using the next command. The erase character " and the kill character ? may be used to modify the command string as in other commands such as the text editor ED. In the following description of commands, underlined letters represent the abbreviation of the command or argument. [] surround optional arguments. ... means the previous element may be respected.

* Indicates following information is a comment.

QUIT

return to DOS/VM

FROM Directory-pathname

where Directory-pathname is of format:

<Ldisk Directory [Password] [Ldisk]> Directory [Password]

<Packname>

FROM defines the FROM directory in which files are to be searched for the commands COPY, COPYSAM, COPYDAM, DELETE, LISTF, TRECPY, TREDEL, UFDCPY, and UFDDEL. The directory is defined from the directory-pathname whose format is given above. The pathname may contain up to 10 directories that may be segment directories as well as user-file-directories. If segment directories are specified, the user must have read access rights to them. If any error is encountered, the FROM directory is set to home-UFD. The first directory in the pathname may be *, which refers to the home-UFD. The default FROM directory is the home-UFD.

Examples:

FROM 0 CARLSO

Set FROM directory to CARLSO on logical disk 0. CARLSO must be in the MFD on logical disk 0 and have a blank password.

FROM CARLSO ABC

Search the MFD on all started disks for CARLSO in logical disk order 0 - 8. Set the FROM directory to the first directory named CARLSO that is found. One of the passwords of CARLSO must be ABC.

FROM <TSDISK> CARLSO > SUB1 > SUB2

Set the FROM directory to SUB2. SUB2 must be a directory in SUB1; SUB1 must be a directory in CARLSO; and CARLSO must be a directory in the MFD on a disk with pack name TSDISK. The directories CARLSO, SUB1, and SUB2 must have a blank password.

FROM *

Set the FROM directory to the home-UFD. The home UFD is normally the last UFD the user has logged into, or attached to with either the ATTACH or FUTIL ATTACH command. If one were logged into CARLSO, the above command sets the FROM directory effectively to CARLSO. This command does not have to be given again if the user changes the home UFD. Furthermore, this command does not have to be given at all unless the FROM directory has been made something other than the home UFD, since home UFD is the default. Example:

```
FROM *   > SUB1
```

Sets the FROM directory to SUB1. SUB1 must be a directory in the home UFD and have a blank password.

TO Directory-Pathname

TO defines the TO directory in which files are searched for the commands COPY, COPYSAM, COPYDAM, TRECPY, and UFDCPY. The directory is defined from the Directory-pathname. The pathname may contain at most 10 directories that may be segment directories as well as UFD's. If segment directories are specified, the user must have read, write, and delete/truncate access to them. The first directory in the pathname may be *. The default TO directory is the home UFD. If any error is encountered, the TO directory is set to home UFD (*).

ATTACH Directory-Pathname

ATTACH moves the home UFD to the directory defined by the directory-pathname. The pathname may contain at most 10 directories. The first directory in the pathname may be *. The last directory in the pathname must be a UFD. If segment directories are specified within the pathname, the user must have read access rights to them.

COPY FILEA [FILEB] [, FILEC [FILED]] . . .

Copy FILEA in the FROM directory to FILEB in the TO directory and optionally FILEC in the FROM directory to FILED in the TO directory. If FILEB is omitted, the new file is given the same name as the old file. FILEA and FILEC must be SAM or DAM files and cannot be directories. Read access rights are required for FILEA and FILEC. If FILEB exists prior to the copy, it must be a SAM or DAM file and the user must have read, write, and delete/truncate access rights to the target file (FILEB in this case). If FILEB exists, it is deleted, then FILEA is copied to FILEB. The file type of FILEB will be the same as FILEA.

Examples:

```
COPY  FILEA
```

Copies FILEA in the FROM directory to FILEA in the TO directory.

```
COPY  FILEA  ,  FILEB  ,  FILEC
```

Copies FILEA, FILEB, and FILEC in the FROM directory to FILEA, FILEB, and FILEC in the TO directory.

```
COPY  FILEA  FILEB
```

Copies FILEA in FROM directory to FILEB in TO directory.

```
COPY  FILEA1  FILEA2,FILEB1  FILEB2,FILEC1  FILEC2
```

Copies FILEA1, FILEB1, and FILEC1 in the FROM directory to FILEA2, FILEB2, and FILEC2 in the TO directory.

```
COPY  (0,0)
```

In this case, the FROM directory and TO directory must each be segment directories. Copy the file at position (0,0) of the FROM directory to position (0,0) of the TO directory. There are no access rights attached to these files, so DOS/VM checks instead the access rights of the directory. A user cannot set the FROM and TO directories if they are segment directories without access rights to them. No spaces are allowed in the name (0,0).

```
COPY  (0,0)  (0,1)
```

Copies the file at position (0,0) of the FROM directory to position (0,1) of the TO directory, both of which are segment directories.

COPYSAM FILEA [FILEB] [, FILEC [FILED]] . . .

same as COPY but also sets file type of FILEB and FILED to SAM instread of copying the type of FILEA and FILEC.

COPYDAM FILEA [FILEB] [, FILEC FILED] . . .

same as COPYSAM but sets file type of FILEB and FILED to DAM

TRECPY DIRA [DIRB] [, DIRC [DIRD]]

Copies the directory tree specified by directory DIRA to directory DIRB, and optionally DIRC to DIRD. DIRB and DIRD must be new directories. If DIRB is omitted, DIRA is taken as the name of the directory to be copied to. A directory tree consists of all files and subdirectories that have their root in that directory. DIRA and DIRC must be in the FROM directory. DIRB and DIRD are created in the TO directory. Read access rights are required for DIRA and DIRC but no access rights are required of files or subdirectories within them.

DIRB and DIRD are created with the same directory type and passwords as DIRA and DIRC and with default access rights. The names, access rights and passwords of all files and subdirectories copied are also copied.

Example:

```
FROM  MFD
TO    MFD
TRECPY     CARLSO     CARNEW
```

Copies the directory tree specified by CARLSO in the MFD to a new directory, CARNEW, in the MFD.

UFDCPY Copies all files and directory trees from the FROM directory to the TO directory. The user must have owner rights in the FROM directory. Furthermore, all files and directories in the FROM directory must have read access rights. Files already existing in the TO directory with names identical to those in the FROM directory are replaced. The user must have read, write, and delete access rights to files that are to be replaced.

Directories already existing in the TO directory with names identical to those in the FROM directory cause the copy operation to stop. Files and directories created in the TO directory will have the same file type as the old files with default access rights. The names, access rights and passwords of all files and subdirectories within directory trees being copied are also copied. Other existing files and directories in the TO directory are not affected. UFDCPY is effectively a merge of two directories. Both the FROM and the TO directory must be UFD's.

Example:

```
FROM        CARLSO
TO          CARNEW
UFDCPY
```

copies all files and directories from CARLSO in the MFD to CARNEW in the MFD. Note that unlike the example for TRECPY, the user has not specified the MFD as the FROM directory, therefore, does not need to know the MFD password. In the example, CARNEW exists prior to the UFDCPY. With the TRECPY example, CARNEW does not previously exist.

DELETE FILEA [FILEB] . . .

Deletes FILEA and optionally FILEB from the FROM directory. FILEA and FILEB cannot be directories. The user must have read, write and delete access rights to each file specified. If FILEA and FILEB are in a segment directory, read, write, and delete rights are required for the FROM directory.

Examples:

```
DELETE    FILEA

DELETE    FILEA  FILEB  FILEC  FILED
```

TREDEL DIRA [DIRB] . . .

Deletes the directory tree specified by directory DIRA and optionally delete DIRB from the FROM directory. DIRA and DIRB must be directories. The user must have read, write, and delete rights to the DIRA and DIRB; however read, write, and delete rights are not required for files and subdirectories nested within DIRA or DIRB. If FILEA and FILED are in a segment directory, read, write, and delete access rights are required for the FROM directory. Note that the operating system DELETE command must not be used to delete directories because it does not free the disk space used by files within the directory for system usage. TREDEL correctly frees disk space to the system.

UFDDEL

Deletes all files and directory trees (specified by directories) within the FROM directory. User must give the owner password in the FROM command and have read, write, and delete access to all files and directories within the FROM directory. These rights are not required for files and subdirectories nested within the directories in the FROM directory.

LISTF [Level] [LSTFIL] [PROTEC] [SIZE] [TYPE]

Lists the FROM directory pathname, the TO directory pathname and all files and directory trees in the FROM directory at the terminal. LISTF optionally follows each filename by its protection attributes, size in disk records, and file type. If the LSTFIL option is given, the list of files is sent to a file named LSTFIL in the home UFD instead of to the terminal. At a later time, a user may print that file on a line printer. Level is a number specifying the lowest level in the FROM directory tree structure to be listed. (See Figure F-1). The following list describes the output.

Level	Output
0	the FROM directory name
1	the FROM directory and all files and directories within it. (level 1 directories)
2	all output at level 1 and all files and directories within level 1 directories

If the level is omitted, the default is 1.

The protection attribute of each file is printed as

Owner-Key Nonowner-Key

These keys are numbers with a range 0-7 that have the following meanings:

0	no access allowed
1	read access only
2	write access only
3	read and write access
4	delete/truncate only
5	delete/truncate and read
6	delete/truncate and write
7	all access allowed

The possible file types are:

SAM	for SAM file
DAM	for DAM file
SEGSAM	for SAM segment directory
SEGDAM	for DAM segment directory
UFD	for User File Directory

LISTF traverses the file structure as shown by the snaked line in Figure F-2 generating printed messages in sequence as shown in the circles adjoining the snaked line.

Using LISTF to produce a list of the sample file configuration shown in Figure F-2, the output level is set to 3 and with the SIZE option, the printed list appears as follows:

```
FROM-DIR = *          MFD

FROM-DIR = *          MFD
TO-DIR   = *

BEGIN MFD             1

DSKRAT      1 BOOT              1

BEGIN UFD1            1
 BEGIN SUFD11          1

  FILEA       1

 END    SUFD11         2
 BEGIN  SUFD12         1

  FILEB       1

 END    SUFD12         2
END    UFD1           5
BEGIN  UFD2           1

  FILEC       1

 END    SUFD21         2
END    UFD2           3
END   MFD            11
```

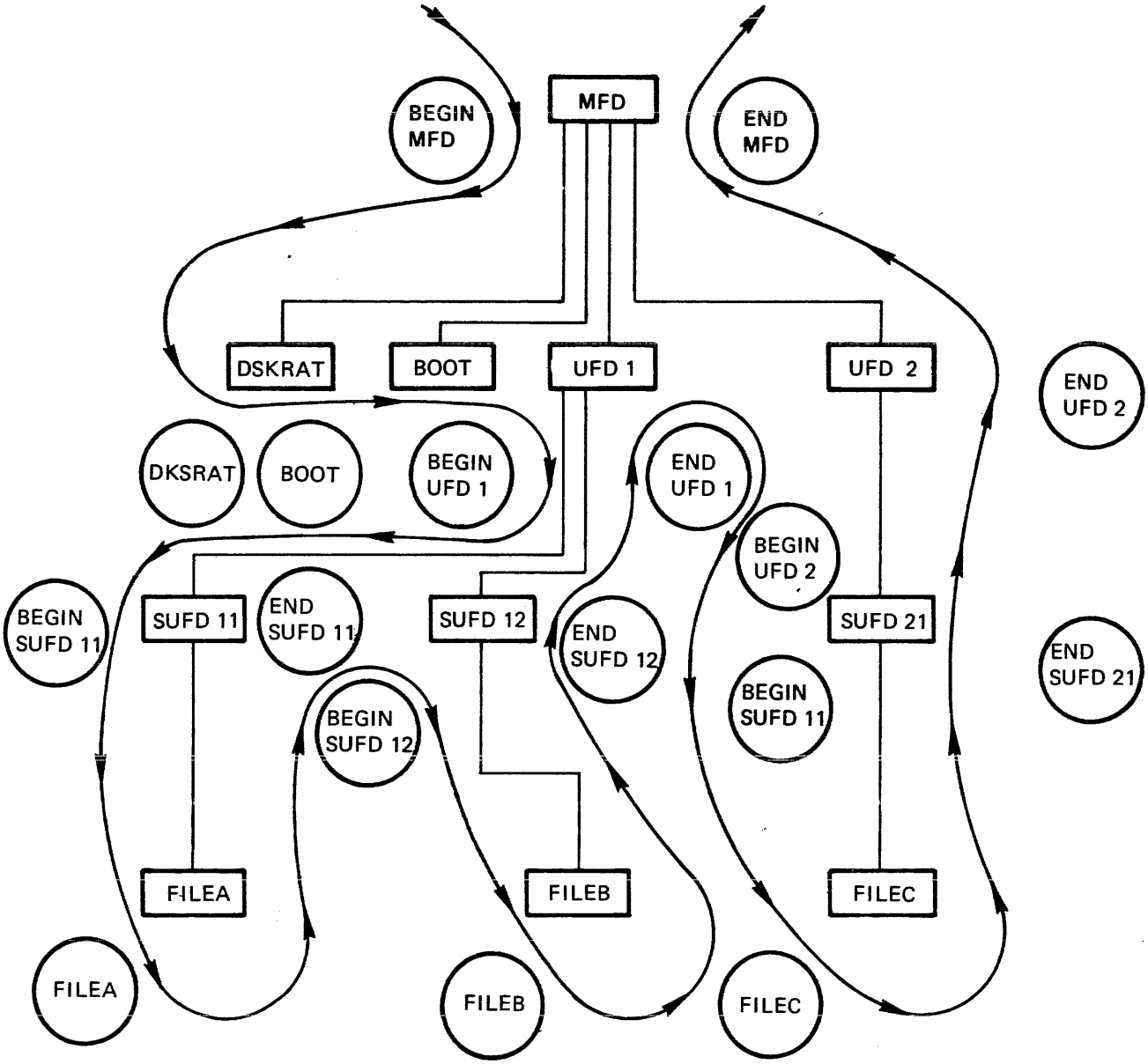


Figure F-2. Typical Traverse of Directory Tree by FUTIL during LISTF.

LISTF upon encountering a directory prints the word BEGIN followed by the name of the directory and its size in records. On leaving a directory, LISTF prints END followed by the number of records used by all files and directories within the directory tree headed by the directory file. On encountering a file, LISTF simply prints its name and size, squeezing as many file names as fit on each line. LISTF skips a line whenever a directory follows a file or a file follows a direcotry. LISTF does not count records in files lower than the specified Level in the FROM directory tree.

In the above example output, the number following MFD, 11, is the total number of records used by the MFD directory tree and consists of all files and directories on the disk pack. LISTF indents the printed output one space for each level down in the tree the directory is located. This format makes it easy to understand the relationship of each directory to other directories in the tree.

RESTRICTIONS

In using FUTIL under DOS/VM, certain operations may intérfere with the work of other users. For example, a UFDCPY command to copy all files from a UFD currently used by another logged-in user may fail. If any file in that directory is open for writing by that user, UFDCPY encounters the error FILE ALREADY OPEN, and aborts. If the user attempts to open one of his files for writing while UFDCPY is running, the user may encounter that error. The FUTIL: LISTF, and TRECPY commands cause the same interaction problems. Other FUTIL commands such as COPY and DELETE can also interfere with the other user, but the problem is not as serious as only one file is potentially involved in a conflict. To prevent the conflicts, users working together and involved in operations using each other's directories must coordiante their activities. If two users consistently use the same UFD at the same time, they must avoid the LISTF command of FUTIL, and use the system LISTF command instead.

FUTIL operations when using the MFD must be done carefully. Never give the command TREDEL MFD as the command deletes every file on the disk except the MFD, DSKRAT and BOOT. A LISTF or UFDCPY of the MFD must be done only if no other user is using any files or directories on that disk. A UFDCPY of the MFD to the MFD of another disk has the effect of merging the contents of two disks onto one disk. A user must be sure there is enough available space on the TO disk before attempting this operation if it aborts. Also, the names of files and directories on the two disks may conflict. To avoid the name conflict, it may be desirable to UFDCPY the MFD of one disk into a UFD on another disk. Each directory originally on the FROM disk becomes a subdirectory in that UFD on the TO disk. For example, the contents of 10 diskettes could be copied into 10 User File Directories on a 1.5M disk pack. Note, a UFDCPY of an MFD does not copy the DSKRAT, MFD or BOOT to the TO directory.

The effect of a UFDCPY from the MFD of a disk in use to the MFD of a disk that was newly produced by the MAKE command is to reorganize the disk files so that all files are compacted; that is, files have their records close to each other on the new disk. After such a compaction, the access time to existing files on the new disk is effectively reduced from the access time on the old disk. Furthermore, new files tend to be compact since all free disk records are also compacted. The use of a compacted disk may improve the performance of DOS/VM or DOS.

Users must not abort COPY or DELETE operations under DOS, but allow them to run to completion. Aborting a COPY or DELETE operation may cause a directory to contain incorrect entries. For example, this may result in a file with a pointer mismatch or bad file structure or a directory with a partial entry. DOS or DOS/VM will not run correctly with a directory with a partial entry. FIXRAT must be run immediately if these conditions are encountered. Under DOS/VM, critical areas of code are surrounded by calls to BREAK$, a subroutine that inhibits the CNTRL-P key. As a result, interruption of FUTIL does not result in a bad file structure.

Error Messages

The following error messages are generated by FUTIL. In many cases, FUTIL prints error messages generated by DOS or DOS/VM and retains control, so users must be familiar with operating system error messages. The list given here includes messages that may be encountered by FUTIL. Most messages are preceded by a filename identifying the file causing the error. Some of the error messages have the format:

reason for error

FILE = filename

DIRECTORY PATH = directory pathname

?

Unrecognizable command

ALREADY EXISTS

An attempt has been made to TRECPY, to a file that already exists, or UFDCPY has attempted to copy to a directory that already exists. If you intend to do the operation, the file in the TO directory must first be deleted.

BAD NAME

A segment directory filename was given to a command that expected a UFD filename or vice versa. The type of filename must match the type of directory the file is contained in.

BAD PASSWORD

An incorrect password has been given in a FROM, TO or ATTACH command. DOS/VM does not allow FUTIL to maintain control in case of a bad password so the FUTIL command must be given to restart FUTIL. The FROM directory and TO directory are reset to home UFD in this case.

BAD SYNTAX

The command line processed by FUTIL is incorrect.

CANNOT ATTACH TO SEGDIR

The last directory in the directory pathname to an ATTACH command is a segment directory. It must be a UFD, because ATTACH sets the home UFD to the last directory in the path.

CANNOT DELETE MFD

User has given the UFDDEL command while attached to the MFD. This is not allowed.

DIRECTORIES NESTED TOO DEEP

Directories may be nested to a depth of 100 levels. User has attempted to exceed this limit.

DISK ERROR

May indicate a disk error or may indicate a FUTIL attempt to process a badly constructed segment directory. Running FIXRAT (Appendix E) is recommended.

DISK FULL

The disk has become full before FUTIL has finished a copy operation. For operations involving many files, some files are not copied, creating only partially copied directories that may be of limited use. It is suggested that the user delete such a structure immediately to prevent confusion as to what has been copied.

IN USE

Indicates a FUTIL attempt to process a file in use by some other user. It may also indicate an attempt to copy a directory to a subdirectory within itself.

IS A DIRECTORY, CANNOT COPY TO IT

Same as ALREADY EXISTS.

NO RIGHT

User has attempted an operation on a file that violates the file access rights assigned to that file. These rights may be changed by the DOS/VM PROTEC command, if the user has given the owner password on ATTACH.

NO ROOM USE DOS32

User is using FUTIL under DOS16 and has attempted an operation that has caused FUTIL to run out of room. This message is not likely to occur as long as all segment directories processed are SAM segment directories.

NO UFD ATTACHED

Self-explanatory.

NOT A DIRECTORY

User has given a directory-pathname which includes a file that is not a directory.

NOT FOUND

Self-explanatory.

POINTER MISMATCH

Indicates a bad file structure. Running FIXRAT is recommended.

PRWFIL EOF

User has attempted to reference a nonexistent file beyond the end of a segment directory.

SEG-DIR ER

User has attempted to reference a file in a segment directory with an entry of 0, which indicates file does not exist or the user has attempted to reference a file beyond the end of the segment directory.

UFD FULL

Self-explanatory.

UNRECOVERED ERROR

Indicates either the user has attempted to write to a write-protected disk, or disk error, or an attempt to process a bad file structure. Running FIXRAT is recommended if the disk was not write-protected.

APPENDIX G

LIBRARIES

DOS MASTER DISK

For systems to operate under DOS, all standard PRIME software is supplied in the form of files stored on a master DOS disk. The user can generate a listing of the contents of the master disk with the help of the FIXRAT command. (For example, see Appendix T.) Of the many files shown in the listing, those in the UFD's MFD, CMDNC0 (or other command UFD), and LIB (Library) are most important to DOS and the user. These files contain saved command execution programs and both the source and object forms of all standard library subroutines.

Contents of MFD

The MFD of a master disk typically contains the following entries:

Each entry in the MFD is itself a UFD. On the Mster Disk only, each entry in the MFDis an index of all UFD's in the system. Some typical UFD's that are actively accessed by DOS itself or the user are:

DSKRAT	Contains the Disk Record Availability Table.
MFD	MFD itself.
AIDS	Source modules for various external utility commands.
BINED	Source modules and command files for EDB.
BOOT	Contains the bootstrap program that is ready by a DOS BOOT paper tape program or by panel LOAD microcode.
DOS	Run file of DOS.
DVBIN	DOS/VM support. Includes three different sets of DOS/VM supervisor for three different configurations (Refer to Section 6).
ED	Source modules and command files for ed and FILED.
INDEX	Command files, including INDEX1 and INDEX2 that list all files in Volumes I and II of the master disk.
IOCS	Source modules for IOCS.
LIB	Object versions of all standard library programs. For details of contents refer to the Subroutine LIBRARY manual.
MATHLB	Source modules for the library of matrix manipulation subroutines.
RTOS 1, 2, 3, 4	Source modules and command files for RTOS.
U-CODE	Microcode source.
SPARE	An empty UFD.
SRCLIB	Contains general-purpose source programs to be assembled as required by user.
T & M	Test and maintenance programs. (For information on the use of these programs, refer to the Prime Installation and Maintenance Manual.)

SPARE2, etc. are empty UFD's available to store the user's program and files. The names can be changed by the CNAME command. New UFD's may be created by the CREATE command.

UFD's such as PMA, FLIB1, etc. contain source and object versions of Prime software for the sophisticated user who wants to generate custom versions of PMA, the library, the loader, etc. (Prime provides source files as a convenience only, and does not guarantee operation of versions that are modified and assembled by the user.)

Contents of Command File CMDNC0

The command file UFD (typically CMDNC0) contains the names of the SAVEd program files that execute DOS external commands.

Contents of LIB

The UFD LIB contains source and object versions of all standard Prime FORTRAN/Math/I-O library routines. Library software is described in the Subroutine Library manual. LIB also includes the IOCS subroutines.

FORTRAN library subroutines satisfy all the ANSI standard functions. Also included are a collection of arithmetic and formatted I/O operations. These are run-time operations and are invisible to the FORTRAN programmer. These subroutines exist where it would be inefficient for the compiler to generate in-line code because of its size, especially when the time to execute the CALL is a small fraction of the subroutines execution time.

Contents of SRCLIB

The UFD SRCLIB is reserved for source programs or subroutines to be assembled or compiled by the user. Among other programs, SRCLIB contains the microcode assembler. This macro-package must be inserted before any microcode instructions that are contained in the source file. For further information, refer to the Prime Microcoders Handbook.

The FORTRAN/Math Library is an extensive collection of subroutines that perform mathematical operations and functions, mode conversions and input/output operations. Math and conversion routines are provided for all modes of variables: single and double precision, fixed and floating point, complex and integer.

Scaled Fixed Point Math and Trig Functions

These PMA-compatible subroutines are most useful where high speed is critical and the programmer is willing to forego the convenience of floating point. The user should read the listing carefully for calling sequence and handling of results.

Function	Single Precision	Double Precision
ADD	--	DADD
SUBTRACT	--	DSUB
MULTIPLY	MPY	DMPY
DIVIDE	DIV	DDIV
ROUND	ROND	RODD
TWO'S COMPLEMENT	--	TWOS
ARCTANGENT	ATNX1	DATNX1
COSINE	COSX1	DCOSX1
SINE	SINX1	DSINX1
EXPONENTIAL BASE 2	EX2X1	DEX2X1
EXPONENTIAL BASE E	EXEX1	DEXEX1
LOG BASE 2	LG2X1	DLG2X1
LOG BASE E	LGEX1	DLGEX1
SQUARE ROOT	SQRX1	DSQRX1

To use these functions in a CPU that does not have the high-speed arithmetic options, the user must force-load the unimplemented instruction package.

FORTRAN Math/Trig Functions

The following routines support the standard ANSI FORTRAN functions and are also assembly language compatible. (Obtain listings from source files.)

Function Name	Source Filename*	Argument Mode	Result Mode	Function Definition
SIN	SINCOS	REAL	REAL	Sine(R) (Radians)
DSIN		DOUBLE	DOUBLE	
CSIN		COMPLEX	COMPLEX	
COS	SINCOS	REAL	REAL	Cosine(R) (Radians)
DCOS		DOUBLE	DOUBLE	
CCOS		COMPLEX	COMPLEX	
ATAN	ATAN-2	REAL	REAL	Arctangent(R)
DATAN		DOUBLE	DOUBLE	
ATAN2	ATAN-2	REAL(2)	REAL	Arctan(R1/R2)
DATAN2		DOUBLE(2)	DOUBLE	
TANH		REAL	REAL	Hyperbolic Tan(R)
SQRT		REAL	REAL	Square Root
DSQRT		DOUBLE	DOUBLE	
CSQRT		COMPLEX	COMPLEX	
EXP		REAL	REAL	E**R
DEXP		DOUBLE	DOUBLE	
CEXP		COMPLEX	COMPLEX	
ALOG		REAL	REAL	Natural Log(R)
DLOG-2		DOUBLE	DOUBLE	
CLOG		COMPLEX	COMPLEX	
ALOG10	ALOG	REAL	REAL	Common Log(R)
DLOG10		DOUBLE	DOUBLE	
ABS		REAL	REAL	Absolute Value
IABS		INTEGER	INTEGER	
DABS		DOUBLE	DOUBLE	
CABS		COMPLEX	REAL	SQRT(R**2+I**2)
AMOD		REAL(2)	REAL	R1(MOD R2)
MOD		INTEGER(2)	INTEGER	
DMOD		DOUBLE(2)	DOUBLE	
AINT		REAL	REAL	Truncate to Integer
INT	IFIXINT	REAL	INTEGER	
IDINT		DOUBLE	INTEGER	

* Same as function name unless otherwise specified.

FORTRAN Math/Trig Functions (Cont'd)

Function Name	Source Filename*	Argument Mode	Result Mode	Function Definition
AMAX0	MAX0	INTEGER (>1)	REAL	Choose Largest Argument
AMAX1	MAX1	REAL (>1)	REAL	
MAX0		INTEGER (>1)	INTEGER	
MAX1		REAL (>1)	INTEGER	
DMAX1		DOUBLE (>1)	DOUBLE	
AMIN0	MIN0	INTEGER (>1)	REAL	Choose Smallest Argument
AMIN1	MIN1	REAL (>1)	REAL	
MIN0		INTEGER (>1)	INTEGER	
MIN1		REAL (>1)	INTEGER	
DMIN1		DOUBLE (>1)	DOUBLE	
FLOAT		INTEGER	REAL	Change Argument Mode
IFIX	IFIXINT	REAL	INTEGER	
SNGL		DOUBLE	REAL	
C$52		COMPLEX	REAL	
AIMAG		COMPLEX	REAL	
DBLE		REAL	DOUBLE	
CMPLX		REAL(2)	COMPLEX	
SIGN		REAL(2)	REAL	Value of R1 with Sign of R2
ISIGN		INTEGER(2)	INTEGER	
DSIGN		DOUBLE(2)	DOUBLE	
DIM		REAL(2)	REAL	Positive Difference
IDIM		INTEGER(2)	INTEGER	
CONJG		COMPLEX	COMPLEX	Complex Conjugate
OR	F$SHFT	INTEGER(2)	INTEGER	16-BIT Logical OR
SHFT		INTEGER(2)	INTEGER	Shift A1 by A2 Bits
SHFT		INTEGER(3)	INTEGER	Shift A1 by A2, Then A3 Bits
LT		INTEGER(2)	INTEGER	Save Left A2 Bits of A1
RT		INTEGER(2)	INTEGER	Save Right A2 Bits of A1
LS		INTEGER(2)	INTEGER	Shift A1 Left by A2 Bits
RS		INTEGER(2)	INTEGER	Shift A1 Right by A2 Bits

* Same as function name unless otherwise specified.

Special FORTRAN Subroutines

Subroutine Name	Source Filename*	Definition
OVERFL		Check for Arithmetic Overflow
SLITE		Set Panel Lamp
SLITET	SLITE	Test and Clear Panel Lamp
SSWTCH	SLITE	Test Sense Switch
RND		Random Number Generation
IRND		Random Number Generation
DISPLAY		Random Number Generation

Miscellaneous Conversion and Compiler Support Routines

These routines are invoked by the Compiler, but may also be called by the user in assembly language programs. (See source program listings for details.)

The names are formed according to the following conventions:

Functions	Modes
A = Add	1 = Integer
C = Convert	2 = Real
D = Divide	5 = Complex
E = Exponentiation	6 = Double precision
F = FORTRAN Utility	8 = Exponent
H = Store (hold)	
I = Input	
L = Load	
M = Multiply	
N = Negate	
O = Output	
S = Subtract	
Z = Clear (zero)	

* Same as subroutine name unless otherwise specified

Subroutine Name	Source Filename*	Definition
AC1-AC5	ACCN	Pseudo-accumulators 1-5
ARG$		Convert Address from Indirect to Direct
A$52		C=R+C
A$55		C=C+C
A$62		D=R+D
A$81		D=D*(2**I)
C$12		Convert I to R
C$16		Convert I to D
C$21		Convert R to I
C$25		Convert R to C
C$26		Convert R to D
C$52		Convert C to R
C$61		Convert D to I
C$62		Convert D to R
C$81		Convert D to I (the exponent)
D$52	MD$22	C=C/R
D$55		C=C/C
D$62		D=D/R
E$11		I=I**I
E$21		R=R**I
E$22		R=R**R
E$26		D=R**D
E$51		C=C**I
E$61		D=D**I
E$62		D=D**R
E$66		D=D**D
F$AT		Transfer a Variable Number of Arguments
F$BN		Rewind
F$DN		Backspace
F$ER		Print Error Message
F$FN		End File Statements
F$HT	F$ER	Process Pause and Halt
F$IO		Format Conversion
F$RN		FORTRAN READ (Calls IOCS F$R1-8)
F$TR		FORTRAN TRACE Statements
F$WN		FORTRAN WRITE (Calls IOCS F$W1-8)

Note:
R=REAL
C=COMPLEX
D=DOUBLE PRECISION
I=INTEGER
L=LOGICAL

* Same as subroutine name unless otherwise specified.

Subroutine Name	Source Filename*	Definition
H$55		Store Complex Number
L$55		Load Complex Into Pseudo Accumulators
M$52		C=C*R
M$55		C=C*C
M$62		D=D*R
N$55		C=-C
S$52		C=C-R
S$55		C=C-C
S$62		D=D-R
Z$80		Replace Binary Exponent with Zero

*Same as subroutine name unless otherwise specified.

IOCS

The Input Output Control Subsystem was developed in order to achieve uniformity of I/O and a degree of device independence. The user (of FORTRAN IV, Macro Assembler, DOS, etc.), by specifying a logical function number and a logical I/O transfer function, can reach a physical unit through a series of tables that are initialized by the operator.

In addition to this logical/physical mapping, the Input Output Control Subsystem also contains all the device dependent input/output subroutines. Also, there are some very basic teletype and paper tape I/O routines which other subroutines may use to communicate with these devices. There are tables to allow for tab stops whether physically in the hardware as in a typewriter, or simulated by using the "form" key in a teletype.

Input/Output Control System (IOCS) Summary:

Logical Unit (LUNIT) Conventions	
1	source/binary input
2	listing output
3	binary output
4-8	others as required

Standard Device Number (DEVNO) Assignments	
0	pseudo output device
1	ASR
2	Paper tape reader/punch
3	Card reader/punch
4	Line printer
5	Mag tape
7	Disk/diskette

Initialization

CALL SETIOS (flag) Assign logical units to physical units.

bit 1	= zero
bits 2-4	= DEVNO for LUNIT = 5
bits 5-7	= DEVNO for LUNIT = 4
bits 8-10	= DEVNO for LUNIT = 1
bits 11-13	= DEVNO for LUNIT = 2
bits 14-16	= DEVNO for LUNIT = 3

CALL ATTDEV (LUNIT, DEVNO, PUNIT) Equate a logical unit (LUNIT) to a device number (DEVNO) and a physical unit (PUNIT) i.e. which mag tape transport or, for disk, which DOS file unit number.

Logical Unit I/O

CALL CONTRL (KEY, NAME, LUNIT, ALTRTN) Perform control function (KEY) for file name (NAME) on logical unit (LUNIT); ALTRTN is entry point if command cannot be satisfied.

KEY =			
	1 = open for read	5 = delete file
	2 = open for write	6 = read/write by record
	3 = open for read/write	7 = rewind
	4 = close	

CALL RDASC (LUNIT, BUFR, N, ALTRTN) Read ASCII record of "N" words into buffer (BUFR) from logical unit (LUNIT).

CALL WRASC (LUNIT, BUFR, N, ALTRTN) Write an "N" word ASCII record.

CALL RDBIN (LUNIT, BUFR, N, ALTRTN) Read an "N" word binary record.

CALL WRBIN (LUNIT, BUFR, N, ALTRTN) Write an "N" word binary record.

Physical Device Formatted I/O

CALL C$X (KEY, NAME, PUNIT, ALTRTN) Perform control function

X = A = ASR L = Printer
P = Paper tape M = Mag tape
C = Cards D = Disk/diskette

CALL I$XASC ([PUNIT,] BUFR, N, ALTRTN) Read ASCII record.

If X = M or D, specify PUNIT; otherwize omit.

CALL O$XASC ([PUNIT,] BUFR, N, ALTRTN) Write ASCII record.

CALL I$XBIN ([PUNIT,] BUFR, N, ALTRTN) Read binary record.

CALL O$XBIN ([PUNIT,] BUFR, N, ALTRTN) Write binary record.

Physical Device Unformatted I/O

CALL T1IN read one teletype character
CALL T1OU type one teletype character
CALL T1IB basic teletype input
CALL T1OB basic teletype output
CALL P1IN read one paper tape character
CALL P1OU punch one character on paper tape
CALL P1IB basic paper tape input
CALL P1OB basic paper tape output

REAL TIME LIBRARY

Executive Functions

RQST	Request program	RTNBLK	Return block
SCHED	Schedule label	RTNSTG	Return a string of blocks
CONCLK	Connect clock		
DISCLK	Disconnect Clock	PBLEND	Put block at end of queue
CONINT	Connect interrupt		
DISINT	Disconnect interrupt	PBLTOP	Put block at top of queue
TERM	Terminate		
XWAIT	Wait	PSTEND	Put string at end of queue
FBLCK	Fetch Block		
CFBLCK	Conditional fetch block	PSTTOP	Put string at top of queue
		TAKBLK	Take block from top of queue
		CHECKB	Check block count
		ERPRNT	Error print
		GTBLPQ	Get block, put on queue

ISA FORTRAN Extensions

START	
TRNON	
DELAY	
WAIT	

FORTRAN Interrupt Extensions

INTSET	Set entry point
INTACK	Execute parameter
ISKED	Exit interrupt code
FETPAR	Fetch parameter(s)

Real Time Equivalents of FORTRAN/IOCS Subroutines

RTF$HT	HALT and PAUSE statements
RTF$ER	ERROR print out
RTFLEX	Floating point exceptions
RTIOS	Basic teletype driver (IASC, OASC)

File Management Extensions

ABRTST Return to real time program after ABORT from file management system.

RTEXIT Equivalent to CALL EXIT. Returns control to RTOS executive.

RTGETA Equivalent to GETA. Fetches alternate return value passed by file management system.

DOSINT Interface to file management systems teletype routines

Real Time Virtual Instruction Package (VIP)

All the VIP available are written so they can be interrupted and a single copy of VIP can serve multiple programs.

RTOS System Components

RTOS Executive Modules

RTEXEC	RTOS executive
FIFO	Queueing routine for parameter passing
SYSLDR	Loader for 128 word block via MHDLNG driver
DOSLDR	Loader for DOS SAVE files
FM	File management system
SIM	Sample system information module

Off-Line DOS Commands

FILBLK	Transfer data to/from random access file, 128 word blocks, under FM.
RTOSRA	Generate 128 word block random access file under FM.
RT128F	Transfer data to/from non-DOS disk in 128 word format.

On-Line Utilities

UDROOT	Base segment of overlayed on-line utilities.
UDDOS	Overlay for background supervisor.
UDEXEC	Overlay for executive function capability.
UDDBUG	Overlay for TAP.
UMEXEC	Non-Overlay version of executive functions.
UMDBUG	Non-Overlay version of TAP.
UMXD	Non-Overlay version of both TAP and executive functions.

Real Time Device Drivers

ASR	Teletype
TASR	Terminating version of ASR
MHDSHT	Moving head disk, DOS compatible
MHDLNG	MHDSHT plus 128 word format
FHD	Fixed head disk
MTUD	Mag tape
PTPDRV	Paper tape punch driver
PTRDRV	Paper tape reader driver

Real Time Device Drivers (cont)

AISD	Analog input system
AOSD	Analog output system
DISD	Digital input system
DOSD	Digital output system
AMLC	Asynchronous Multiline Controller

Real Time Device Exercisers

M9TST	9 track mag tape
M7TST	7 track mag tape
ADTST	Analog input system
DATST	Analog output system
DITST	Digital input system
DOTST	Digital output system
TAMLC	Asynchronous multiline controller
DSKTST	Disk
ASRTST	Teletype
PTPTST	Paper tape punch
PTRTST	Paper tape reader

MATRIX LIBRARY

Command format: XMYYY, where X = 'blank' = real
D = double precision
I = integer
C = complex

XMIDN(A, N) A (N,N) = I(N,N) Identity matrix

XMCON(A,N,M,K) A(N,M) = K(N,M); aij = k

XMSCL(A,B,N,M,K) A(N,M) = K*B(N,M); aij = kbij

XMTRN(A,B,N) $A(N,N) = B(N,N)^T$; aij = bji

XMADD(A,B,C,N,M) A(N,M) = B(N,M)+C(N,M); aij = bij + cij

XMSUB(A,B,C,N,M) A(N,M) = B(N,M)-C(N,M); aij = bij - cij

XMMLT(A,B,C,N1, N2, N3) A(N1,N3) = B(N1,N2)*C(N2,N3)

XMINV(A,B,N,WORK,NP1, NPN,IERR) $A(N,N) = B(N,N)^{-1}$
(X≠I) Scratch area WORK of size NP1 X NPN
where NP1=N+1; NPN=N+N

IERR returned as 0 = inversion successfully completed
1 = matrix was non-invertible
2 = NP1≠N+1 or NPN ≠ N+N

XMADJ(A,B,N,IW1,IW2,IW3,IW4,IERR) A(N,N) = adjoint of Δ B(N,N)
IW1,IW2,IW3,IW4=work area X(N)

IERR = 0 adjoint found successfully
= 1 (N<2, no adjoint possible
aij = signed cofactor of Bij

XMDET (DET,B,N,IW1,IW2,IW3,IW4,IERR)

DET = determinant of B(N,N)
IERR = 0 determinant found successfully
= 1 N = 0, no determinant possible

XMCOF (COF,B,N,IW1,IW2,IW3,IW4,I,J,IERR) COF=(I,J) signed cofactor of B(N,N)

IERR = 0 cofactor successfully found
= 1 no cofactor found or subscript error

XLINEQ(X,Y,A,WORK,N,NP1,IERR) Solve for X in Y(N)=A(N,N) X(N)
(X=I)

IERR = 0 solution successfully found
= 1 matrix was singular
= 2 NP1≠N+1

PERM(IPERM,N,IW1,IW2,IW3,LAST [,RESTRT]) Compute the next permutation of N elements.

COMB(ICOMB,N,NR,IW1,IW2,IW3,LAST [,RESTRT]) Compute the next combination of NR out of N elements.

VIP LIBRARY (Virtual Instruction Package)

The following subroutines are provided to execute unimplemented instructions (UII's)

High Speed Arithmetic

DLD	Double precision load	PIM	Position for integer multiply
DST	Double precision store	PID	Position for integer divide
DAD	Double precision add	MPY	Multiply
DSB	Double precision subtract	DVD	Divide
		NRM	Normalize

PRIME 300

EAA	Effective address to A-register	JLE	Jump if less than or equal
		JGT	Jump if greater than
ENTR	Enter subroutine on stack	JLT	Jump if less than
		JGE	Jump if greater than or equal
CREP	Call recursive entry procedure	JDX	Jump and decrement X
		JIX	Jump and increment X
RTN	Return from subroutine	JSX	Jump and store X
JEQ	Jump if equal		
JNE	Jump if not equal		

Floating Point Arithmetic

Precision		
Single	Double	
FLD	DFLD	Load
FST	DFST	Store
FAD	DFAD	Add
FSB	DBSB	Subtract
FMP	DFMP	Multiply
FDV	DFDV	Divide
FCS	DFCS	Compare and Skip
FCM	DFCM	Complement
FLX	-	Load floating index
FLOT	-	Float
INT	-	Fix as integer
FRAC	-	Fix as fraction
FRN	-	Round up
FSZE	-	Skip if zero
FSNZ	-	Skip if not zero
FSMI	-	Skip if minus
FSPL	-	Skip if plus
FSLE	-	Skip if less than or equal to zero
FSGT	-	Skip if greater than zero

APPENDIX H

USE OF DOS FILE SYSTEM

INTRODUCTION

This appendix gives guidance in and examples of how to use the file system. The expanded key definitions of SEARCH, PWRFIL, and ATTACH have been rewritten in this appendix with mnemonic keys. The following examples use variables defined and initialized by the insert file KEYCOM.

A user wishing to use these keys must have the statement INSERT KEYCOM in his FORTRAN program after the storage specifiecation statements and before any data statements. The user will have to copy KEYCOM to the appropriate UFD before compiling the program(s). This appendix provides examples of use of the file system. (Refer to "Examples".)

The following example programs are:

Program Name	Function
KEYCOM	Provides mnemonic keys for PRWFIL, SEARCH, and ATTACH.
SAMWRT	Writes a SAM data file.
DAMWRT	Writes a DAM data file.
REDFIL	Reads a SAM or DAM file or unlimited length and prints the largest integer in the file. This program also shows how to use alternate return.
RDLREC	Reads logical record number n from a file of fixed-length recorded.
CRTSEG	Creates a segment directory.
REDSEG	Reads file on a segment directory and prints a specified word (record) in that file.
RDVREC	Reads logical record number n from a file on variable-length records.
GPTRFL	Generates a pointer file that consists of two-word pointers to each logical record in another file.

```
C KEYCOM  JPC 30 MAY 1974
C  PROVIDES MNEMONIC KEYS FOR PRWFIL, SEARCH, AND ATTACH
      INTEGER PREAD, PWRITE, PREREL, PREABS, POSREL, POSABS, PCONV,
     X    OPNRED, OPNWRT, OPNBTH, CLOSE, DELETE, REWIND,
     X    TRNCAT, UFDREF, SEGREF, NTFILE, NDFILE, NTSEG, NDSEG, NEWUFD,
     X    MFDUFD, CURUFD, SEGUFD, HOMUFD, SETHOM
C
      DATA PREAD, PWRITE, PREREL, PREABS, POSREL, POSABS, PCONV
     X   / :1,    :2,     :0,     :10,    :20 ,   :30,    :400/
      DATA OPNRED, OPNWRT, OPNBTH, CLOSE, DELETE, REWIND, TRNCAT
     X   / 1,     2,      3,      4,     5,      7,      8   /
      DATA UFDREF, SEGREF, NTFILE, NDFILE, NTSEG, NDSEG, NEWUFD
     X    /:0,    :100,   :0,     :2000,  :4000, :6000, :10000/
      DATA MFDUFD, CURUFD, SEGUFD, HOMUFD, SETHOM
     X   / 0,     2,      4,      6,     1    /
```

```
C SAMWRT, CARLSON JULY 10, 1974
C
C PROGRAM SAM-WRITE TO WRITE A SAM DATA FILE
C
C THE FILE IS 1000 WORDS WRITTEN FROM ARRAY BUFF.
C
C RESTRICTIONS: SAMFIL SHOULD NOT EXIST BEFORE RUNNING THE PROGRAM.
C
C
      INTEGER BUFF(1000),PBUFF,FUNIT1
C
C VARIABLE DEFINITIONS:
C BUFF- ARRAY TO BE WRITTEN TO A FILE
C PBUFF- POINTER TO BUFF
C FUNIT1- CONTAINS 1, REFERS TO FILE UNIT 1
C
C ROUTINES CALLED
C LOC,SEARCH,PRWFIL,EXIT
C
C
C KEYCOM CONTAINS FILE KEY DEFINITIONS
C
$INSERT KEYCOM
C
C
      DATA FUNIT1/1/
C
C
C INITIALIZE BUFFER CONTENTS
      DO 10 I=1,1000
        BUFF(I)=I
10    CONTINUE
C
C LOC RETURNS A POINTER TO ITS ARGUMENT
C
      PBUFF=LOC(BUFF)
C
C OPEN A NEW SAM DATA FILE CALLED SAMFIL IN THE CURRENT UFD
C FOR WRITING ON FILE UNIT 1.
C ON MOST CALLS THE UFDREF KEY IS OMITTED SINCE ITS VALUE IS 0.
C
C THE FOLLOWING STATEMENT WILL BE COMPILED AS IF IT WERE WRITTEN
C TEMP=OPNWRT+NTFILE+UFDREF
C CALL SEARCH(TEMP,'SAMFIL',FUNIT1,0)
C THE USE OF MULTIPLE MNEMONIC KEYS WILL GENERATE MORE CODE THAN
C THE USE OF NUMERIC KEYS.
C
      CALL SEARCH(OPNWRT+NTFILE+UFDREF,'SAMFIL',FUNIT1,0)
C
C WRITE 1000 WORDS FROM BUFF INTO FILE UNIT 1.
C
      CALL PRWFIL(PWRITE,FUNIT1,PBUFF,1000,0,0)
C
C CLOSE FILE.  THIS RELEASES FILE UNIT 1 FOR REUSE AND INSURES
C ALL FILE BUFFERS HAVE BEEN WRITTEN TO THE DISK.
C
      CALL SEARCH(CLOSE,0,FUNIT1,0)
C
C RETURN TO DOS
C
      CALL EXIT
C
      END
$0
```


MAN 1675

```
C DAMWRT, CARLSON, JULY 10,1974
C
C PROGRAM DAM-WRITE TO WRITE A DAM DATA FILE
C
C NOTE THAT THE ONLY DIFFERENCE FROM PROGRAM SAMFIL IS THE
C NEW FILE KEY SUPPLIED TO SEARCH IN CREATING THE FILE.
C
C RESTRICTIONS: DAMFIL SHOULD NOT EXIST BEFORE RUNNING THIS PROGRAM.
C
C
      INTEGER BUFF(1000),PBUFF,FUNIT1
C
C VARIABLE DEFINTIONS
C BUFF- ARRAY TO BE WRITTEN TO A FILE
C PBUFF- POINTER TO BUFF
C FUNIT1- CONTAINS 1, REFERS TO FILE UNIT 1
C
C ROUTINES CALLED
C LOC,SEARCH,PRWFIL,EXIT
C
$INSERT KEYCOM
C
      DATA FUNIT1/1/
C
C
C INITIALIZATION
C
      DO 10 I=1,1000
        BUFF(I)=I
10    CONTINUE
C
      PBUFF=LOC(BUFF)
C
C OPEN A NEW DAM DATA FILE CALLED DAMFIL IN THE CURRENT
C UFD FOR WRITING ON FILE UNIT 1.
C
      CALL SEARCH(OPNWRT+NDFILE+UFDREF,'DAMFIL',FUNIT1,0)
C
C WRITE 1000 WORDS FROM BUFF INTO FILE UNIT 1
C
      CALL PRWFIL(PWRITE,FUNIT1,PBUFF,1000,0,0)
C
C CLOSE FILE
C
      CALL SEARCH(CLOSE,0,FUNIT1,0)
C
C RETURN TO DOS
C
      CALL EXIT
C
      END
$0
```

```
C REDFIL, CARLSON, JULY 10,1974
C
C PROGRAM READ-FILE TO READ A SAM OR DAM FILE OF UNLIMITED LENGTH
C AND PRINT THE LARGEST INTEGER IN THE FILE.
C
C THIS PROGRAM SHOWS HOW TO USE THE ALTERNATE RETURN FEATURE
C OF SEARCH AND PRWFIL AND HOW TO USE GETERR AND PRERR IN
C CONJUNCTION WITH THE ALTERNATE RETURN.  NOTE THAT THE PROGRAM
C DOESN'T CHECK IF THE FILE IS SAM OR DAM.  TO THE USER, SAM AND
C DAM FILES ARE FUNCTIONALLY EQUIVALENT.
C
C RESTRICTIONS: NONE
C
C
      INTEGER BUFF(100),PBUFF,UERVEC(4),FUNIT1,LARGST,FNAME(3),N
C
C VARIABLE DEFINITIONS
C BUFF- BUFFER TO HOLD INFORMATION READ FROM FILE
C PBUFF- POINTER TO BUFF
C UERVEC- USER ERROR VECTOR.  HOLDS ERROR VECTOR OBTAINED FROM DOS
C FUNIT1- CONTAINS 1,  USED TO REFER TO FILE UNIT 1
C LARGST- VARIABLE TO HOLD LARGEST INTEGER IN FILE
C FNAME- HOLDS A FILE NAME
C
C ROUTINES CALLED
C LOC, SEARCH, PRWFIL, GETERR, PRERR, EXIT
C
$INSERT KEYCOM
C
      DATA FUNIT1/1/
C
C
C INITIALIZATION
C
      PBUFF=LOC(BUFF)
      LARGST=-32767
C
C ASK USER FOR FILE NAME.  FORTRAN UNIT 1 IS THE USER TERMINAL.
C
10    WRITE(1,1000)
1000  FORMAT('TYPE FILE NAME')
C
C READ FILE NAME
      READ(1,1010)(FNAME(I),I=1,3)
1010  FORMAT(3A2)
C
C OPEN FNAME IN THE CURRENT UFD FOR READING ON FILE UNIT 1.
C IF ANY ERROR, GO TO LABEL 100.
C
      CALL SEARCH(OPNRED,FNAME,FUNIT1,$100)
```


MAN 1675

```
C
C READ FILE 100 WORDS AT A TIME.  SET LARGST TO THE LARGEST INTEGER
C READ.  WHEN END OF FILE IS REACHED, THE ALTERNATE RETURN OF
C PRWFIL SENDS CONTROL TO LABEL 50.
C
30     CALL PRWFIL(PREAD,FUNIT1,PBUFF,100,0,$50)
C
C 100 WORDS READ INTO BUFF, SET LARGST
C
       DO 40 I=1,100
         IF (LARGST.LE.0.AND.BUFF(I).GE.0) LARGST=BUFF(I)
C
C THE ABOVE TEST IS DONE BECAUSE IF BUFF(I)-LARGST IS GREATER
C THAN 32767, THE FOLLOWING COMPARISON FAILS DUE TO ARITHMETIC OVERFLOW
C
         IF (LARGST.LT.BUFF(I)) LARGST=BUFF(I)
40     CONTINUE
C
C LOOP BACK TO READ MORE DATA FROM FILE
C
       GO TO 30
C
C ALTERNATE RETURN TAKEN ON PRWFIL.  SET ERROR TYPE FROM ERRVEC
C THROUGH A CALL TO GETERR.
C
50     CALL GETERR(UERVEC,4)
C
C IF ERROR TYPE NOT END OF FILE (CODE 'PE'), PRINT THE
C ERROR MESSAGE WITH PRERR AND RETURN TO DOS.
C
       IF (UERVEC(1).EQ.'PE') GO TO 60
         CALL PRERR
         CALL EXIT
C
C END OF FILE.  NUMBER OF WORDS IN PRWFIL CALL LEFT TO BE
C TRANSFERRED IS IN UERVEC(2)
C N IS SET TO NUMBER OF WORDS TRANSFERRED ON LAST CALL.
C
60     N=100-UERVEC(2)
       IF (N.EQ.0) GO TO 80
C
C SET LARGST
C
       DO 70 I=1,N
         IF (LARGST.LE.0.AND.BUFF(I).GE.0) LARGST=BUFF(I)
         IF (LARGST.LT.BUFF(I)) LARGST=BUFF(I)
70     CONTINUE
C
C THE FOLLOWING PRWFIL CALL ACTS AS A NO-OPERATION ON THE FILE
C BUT PUTS THE FILE POINTER IN ERRVEC.
```

```
C
80      CALL PRWFIL(PREAD,FUNIT1,0,0,0,0)
        CALL GETERR(UERVEC,4)
C
C FILE POINTER IS (RECORD-NO.,WORD-NO.) IN UERVEC(3) AND UERVEC(4).
C IF FILE POINTER IS (0,0) AT THIS POINT, IT INDICATES THE FILE
C CONTAINS NO DATA.
C
        IF (UERVEC(3).EQ.0.AND.UERVEC(4).EQ.0) GO TO 110
C
C FILE NOT EMPTY, PRINT LARGST.
C
        WRITE(1,1020)LARGST
1020    FORMAT('LARGEST INTEGER IN FILE IS 'I6)
C
C CLOSE FILE AND RETURN TO DOS
C
90      CALL SEARCH(CLOSE,0,FUNIT1,0)
        CALL EXIT
C
C ERROR IN ATTEMPT TO OPEN FILE
C PRINT MESSAGE AND GET ERROR CODE.
C
100     CALL PRERR
        CALL GETERR(UERVEC,1)
C
C IF ERROR IS NAME NOT FOUND (CODE 'SH'), GO TO LABEL 10 TO ASK
C FOR A NEW NAME OTHERWISE GIVE UP AND RETURN TO DOS.
C
        IF (UERVEC(1).EQ.'SH') GO TO 10
        CALL EXIT
C
C FILE EMPTY
C
110     WRITE(1,1030)
1030    FORMAT('FILE EMPTY')
         GO TO 90
C
        END
$0
```


MAN 1675

```
C RDLREC, CARLSON, JULY 10,1974
C
C RDLREC- READ LOGICAL RECORD
C PROGRAM TO READ LOGICAL RECORD NUMBER N FROM A FILE CONSISTING
C OF FIXED LENGTH RECORDS
C
C IN THIS PROGRAM THE FILE ACCESSED IS CONSIDERED TO CONTAIN
C AN UNLIMITED NUMBER OF LOGICAL RECORDS, EACH RECORD CONSISTING
C OF M WORDS.  THE PROGRAM READS AND TYPES THE CONTENTS OF RECORD
C NUMBER N AS M INTEGERS.  THE FIRST RECORD OF A FILE IS RECORD NUMBER 0.
C NOTE THAT A LOGICAL RECORD IS MEARLY A GROUPING OR WORDS IN
C A FILE.  IT HAS NO RELATION TO THE PHYSICAL DISK RECORD.
C
C RESTRICTIONS: RECORD SIZE MUST BE BETWEEN 1 AND 1000
C               RECORD NUMBER MUST BE BETWEEN 0 AND 32767
C               (RECORD-SIZE)*(RECORD-NUMBER) MUST BE LESS THAN
C               8,388,608 (2**23) BECAUSE FLOATING POINT NUMBERS ONLY REPRESENT
C               6.8 DIGITS.
C               THE RECORD MUST BE IN THE FILE
C
C
      INTEGER PBUFF, BUFF(1000), FUNIT1, FNAME(3), RECSIZ, RECNUM, POSITN,
     X   ABSPOS(2)
C
      REAL FRECSZ, FRCNUM, FPOSTN, PRECSZ
C
C VARIABLE DEFINTIONS
C BUFF- BUFFER USED TO HOLD A LOGICAL RECORD
C PBUFF- POINTER TO BUFFER
C FUNIT1- CONTAINS 1, USED TO REFER TO FILE UNIT 1
C FNAME- HOLDS A FILE NAME
C RECSIZ- LOGICAL RECORD SIZE
C RECNUM- LOGICAL RECORD NUMBER
C POSITN- RELATIVE POSITION TO POSITION TO REQUESTED RECORD
C ABSPOS- ABSOLUTE POSITION TO POSITON TO REQUESTED RECORD
C FRCNUM- FLOATING POINT LOGICAL RECORD NUMBER
C FRECSZ- FLOATING POINT LOGICAL RECORD SIZE
C FPOSTN- FLOATING POINT POSITION NEEDED TO POSITION TO REQUESTED RECORD
C PRECSZ- PHYSICAL DISK RECORD DATA SIZE.  USED TO FORM
C          TWO WORD ABSOLUTE POSITION.
C
C ROUTINES CALLED
C LOC, SEARCH, FLOAT, INT, AMOD, GINFO, PRWFIL, EXIT, GETERR, PRERR
C
$INSERT KEYCOM
C
      DATA FUNIT1/1/
C
C
C INITIALIZATION
```

```
C
      PBUFF=LOC(BUFF)
C
C ASK FOR FILE NAME
C
      WRITE(1,1000)
1000  FORMAT('TYPE FILE NAME')
C
C READ FILE NAME
C
      READ(1,1010)(FNAME(I),I=1,3)
1010  FORMAT(3A2)
C
C OPEN FNAME IN THE CURRENT UFD FOR READING ON FILE UNIT 1
C
      CALL SEARCH(OPNRED,FNAME,FUNIT1,0)
C
C ASK FOR RECORD SIZE
C
20    WRITE(1,1020)
1020  FORMAT('TYPE RECORD SIZE')
      READ(1,1030)RECSIZ
1030  FORMAT(I6)
      IF (RECSIZ.GE.1.AND.RECSIZ.LE.1000) GO TO 30
         WRITE(1,1040)
1040  FORMAT('BAD RECORD SIZE')
         GO TO 20
C
C ASK FOR RECORD NUMBER.  FIRST RECORD IS NUMBERED 0.
C
30    WRITE(1,1050)
1050  FORMAT('TYPE RECORD NUMBER')
      READ(1,1030) RECNUM
      IF (RECNUM.GE.0) GO TO 35
      WRITE(1,1051)
1051    FORMAT('BAD RECORD NUMBER')
      GO TO 30
C
C CHECK IF RECORD IS MORE THAN 32767 WORDS FROM BEGINNING OF
C FILE  IF SO, USE ABSOLUTE POSITIONING ELSE USE RELATIVE
C POSITIONING.
C
35    FRECSZ=FLOAT(RECSIZ)
      FRCNUM=FLOAT(RECNUM)
      FPOSTN=FRECSZ*FRCNUM
      IF (FPOSTN.LT.8388608.) GO TO 40
        WRITE(1,1055)
1055    FORMAT('RECORD-NUMBER*RECORD-SIZE IS TOO LARGE')
        GO TO 20
40    IF (FPOSTN.GT.32767.) GO TO 100
```

```
C
C RECORD IS LESS THAN 32767 WORDS FROM BEGINNING, USE RELATIVE
C POSITIONING.
C NOTE THAT ABSOLUTE POSITIONING COULD HAVE BEEN USED FOR A RECORD
C ANYWHERE IN THE FILE, NOT JUST FOR THOSE RECORDS BEYOND WORD
C 32767.  RELATIVE IS SHOWN IN HERE ONLY FOR AN EXAMPLE.
C
      POSITN=RECSIZ*RECNUM
C
C POSITION TO THE RECORD AND READ RECSIZ WORDS INTO THE BUFFER
C
      CALL PRWFIL(PREAD+PREREL,FUNIT1,PBUFF,RECSIZ,POSITN,$300)
C
C GO TO 200 TO TYPE RECORD CONTENTS
C
      GO TO 200
C
C RECORD MORE THAN 32767 WORDS FROM BEGINNING OF FILE, USE
C ABSOLUTE POSITIONING
C GET PHYSICAL DISK DATA RECORD SIZE FROM DOS
C
100   CALL GINFO(BUFF,4)
      PRECSZ=FLOAT(BUFF(4))
C
C CALCULATE ABSOLUTE POSTION (RECORD-NUMBER,WORD-NUMBER)
C THAT RECORD STARTS AT AND PUT IN ABSPOS(1) AND ABSPOS(2)
C
      ABSPOS(1)=INT(FPOSTN/PRECSZ)
      ABSPOS(2)=INT(AMOD(FPOSTN,PRECSZ))
C
C POSITION TO THE RECORD AND READ RECSIZ WORDS INTO THE BUFFER
C
      CALL PRWFIL(PREAD+PREABS,FUNIT1,PBUFF,RECSIZ,ABSPOS,$300)
C
C RECORD READ, NOW TYPE IT.
C
200   WRITE(1,1060) RECNUM,RECSIZ
1060  FORMAT('RECORD ',I6,' CONTAINS ',I6,' ENTRIES AS FOLLOWS')
      WRITE(1,1070)(BUFF(I),I=1,RECSIZ)
1070  FORMAT(10I7)
C
C RETURN TO DOS AFTER CLOSING THE FILE
C
      CALL SEARCH(CLOSE,0,FUNIT1)
      CALL EXIT
C
C ERROR WHILE ATTEMPTING TO READ THE RECORD
C
300   CALL GETERR(BUFF,1)
      CALL PRERR
      IF (BUFF(1).EQ.'PE') GO TO 305
      CALL EXIT
C
C END OF FILE REACHED, REWIND FILE AND TRY AGAIN.
C
305   CALL SEARCH(REWIND,0,FUNIT1,0)
      GO TO 20
C
      END
$0
```

```
C CRTSEG, CARLSON, JULY 12, 1974
C
C CRTSEG- CREATE-SEGMENT-DIRECTORY
C THIS PROGRAM SHOWS HOW TO CREATE A SEGMENT DIRECTORY AND WRITE
C FILES INTO IT.
C
C RESTRICTIONS: SEGDIR SHOULD NOT EXIST BEFORE RUNNING THE PROGRAM.
C
      INTEGER PBUFF,BUFF(10),SGUNIT,FUNIT
C
C
C VARIABLE DEFINITIONS
C BUFF- BUFFER TO WRITE TO SEGMENT DIRECTORY FILES
C PBUFF- POINTER TO BUFF
C SGUNIT- CONTAINS 1, FILE UNIT USED FOR SEGMENT DIRECTORY
C FUNIT- CONTAINS 2, FILE UNIT USED FOR DATA FILES
C
$INSERT KEYCOM
C
      DATA SGUNIT,FUNIT/1,2/
C
C
C INITIALIZATION
C
      PBUFF=LOC(BUFF)
      DO 10 I=1,10
        BUFF(I)=I
10    CONTINUE
C
C OPEN A NEW SAM SEGMENT DIRECTORY CALLED SAMDIR IN THE
C CURRENT UFD FOR READING AND WRITING ON FILE UNIT SGUNIT.
C
      CALL SEARCH(OPNBTH+NTSEG+UFDREF,'SEGDIR',SGUNIT,0)
C
C OPEN A NEW SAM DATA FILE FOR WRITING ON FILE UNIT FUNIT.  WRITE
C THE DISK LOCATION OF THIS NEW FILE AT THE FILE POINTER OF
C THE SEGMENT DIRECTORY OPEN ON FILE UNIT SGUNIT.
C THE FILE POINTER POINTS TO WORD NUMBER 0 OF THE SEGMENT DIRECTORY.
C
      CALL SEARCH(OPNWRT+NTFILE+SEGREF,SGUNIT,FUNIT,0)
C
C WRITE 10 WORDS FROM BUFF INTO THE DATA FILE
C
      CALL PRWFIL(PWRITE,FUNIT,PBUFF,10,0,0)
C
C CLOSE THE DATA FILE
C
      CALL SEARCH(CLOSE,0,FUNIT,0)
C
C REPLACE BUFF WITH NEW DATA
```


MAN 1675

```
C
      DO 20 I=1,10
        BUFF(I)=I*10
20    CONTINUE
C
C OPEN A DIFFERENT NEW SAM DATA FILE ON FUNIT.  PUT THE
C DISK LOCATION IN WORD NUMBER 1 OF THE SEGMENT DIRECTORY.
C THIS IS DONE IN TWO STEPS.  FIRST BY POSITIONING THE FILE
C POINTER OF THE SEGMENT DIRECTORY FORWORD ONE WORD AND THEN
C BY CALLING SEARCH AS SHOWN ABOVE.
C
      CALL PRWFIL(PREAD+PREREL,SGUNIT,0,1,0)
      CALL SEARCH(OPNWRT+NTFILE+SEGREF,SGUNIT,FUNIT,0)
C
C WRITE 10 WORDS IN THE FILE
C
      CALL PRWFIL(PWRITE,FUNIT,PBUFF,10,0,0)
C
C CLOSE THE DATA FILE
C
      CALL SEARCH(CLOSE,0,FUNIT,0)
C
C CLOSE THE SEGMENT DIRECTORY
C
      CALL SEARCH(CLOSE,0,SGUNIT,0)
C
C RETURN TO DOS
C
      CALL EXIT
C
      END
```

```
C REDSEG, CARLSON, JULY 12, 1974
C
C REDSEG- READ-FILE-IN-SEGMENT-DIRECTORY
C THIS PROGRAM READS FILE NUMBER N IN A SEGMENT DIRECTORY AND
C TYPES WORD NUMBER M IN THAT FILE.  THE FIRST FILE IN THE DIRECTORY
C IS FILE NUMBER 0.  THE FIRST WORD IN THE FILE IS WORD NUMBER 0.
C
C RESTRICTIONS:  THE FILE NUMBER MUST BE BETWEEN 0 AND 32767.
C                THE FILE MUST BE IN THE SEGMENT DIRECTORY.
C                THE WORD NUMBER MUST BE BETWEEN 0 AND 32767.
C                THE WORD MUST BE IN THE FILE.
C
      INTEGER PBUFF,BUFF,SGUNIT,FUNIT,SEGDIR(3),UERVEC(2),FILNUM,
     X   WRDNUM
C
C VARIABLE DEFINITIONS
C BUFF- HOLDS WRDNUM WORD OF FILNUM FILE OF SEGDIR
C PBUFF- POINTER TO BUFF
C SGUNIT- CONTAINS 1, FILE UNIT USED FOR SEGMENT DIRECTORY
C FUNIT- CONTAINS 2, FILE UNIT USED FOR DATA FILE
C UERVEC- HOLDS ERROR VECTOR OBTAINED FROM DOS
C FILNUM- HOLDS FILE NUMBER OF SEGDIR TO READ
C WRDNUM- HOLDS WORD NUMBER OF NTH FILE TO READ
C SEGDIR- HOLDS SEGMENT DIRECTORY NAME
C
$INSERT KEYCOM
C
      DATA SGUNIT,FUNIT/1,2/
C
C
C INITIALIZATION
C
      PBUFF=LOC(BUFF)
C
C INSURE UNITS ARE CLOSED AND
C ASK FOR AND READ SEGMENT DIRECTORY NAME
C
10    CALL SEARCH(CLOSE,0,SGUNIT,0)
      CALL SEARCH(CLOSE,0,FUNIT,0)
      WRITE(1,1000)
1000  FORMAT('TYPE SEGMENT DIRECTORY NAME')
      READ(1,1010)(SEGDIR(I),I=1,3)
1010  FORMAT(3A2)
C
C OPEN THE SEGMENT DIRECTORY FOR READING ON SGUNIT
C
      CALL SEARCH(OPNRED+UFDREF,SEGDIR,SGUNIT,0)
C
C GET FILE TYPE FROM ERRVEC AND MAKE SURE FILE IS A SEGMENT DIRECTORY.
C ALLOWABLE TYPE CODES ARE SAMSEG AND DAMSEG.  VALUES 2 AND 3.
```

```
C
      CALL GETERR(UERVEC,2)
      IF (UERVEC(2).EQ.2.OR.UERVEC(2).EQ.3) GO TO 20
C
C NOT A SEGMENT DIRECTORY, TRY AGAIN.
C
      WRITE(1,1020)
1020  FORMAT('FILE IS NOT A SEGMENT DIRECTORY')
      GO TO 10
C
C ASK FOR FILE IN SEGMENT DIRECTORY
C
20    WRITE(1,1030)
1030  FORMAT('TYPE FILE NUMBER')
      READ(1,1040) FILNUM
1040  FORMAT(I6)
C
C ASK FOR  WORD IN FILE
C
      WRITE(1,1035)
1035  FORMAT('TYPE WORD NUMBER')
      READ(1,1040) WRDNUM
C
C TRY TO POSITION TO FILNUM FILE IN THE SEGMENT DIRECTORY.
C IF ERROR GO TO 100
C
      CALL PRWFIL(PREAD+PREREL,SGUNIT,0,0,FILNUM,$100)
C
C OPEN FILE IN SEGMENT DIRECTORY FOR READING ON FUNIT
C GO TO 120 IF ANY ERROR.
C
      CALL SEARCH(OPNRED+SEGREF,SGUNIT,FUNIT,$120)
C
C POSITION TO FILWRD WORD IN DATA FILE AND READ IT INTO BUFF
C GO TO 200 IF ANY ERROR.
C
      CALL PRWFIL(PREAD+PREREL,FUNIT,PBUFF,1,WRDNUM,$200)
C
C PRINT THE WORD, CLOSE FILES AND RETURN TO DOS
C
      WRITE(1,1050)WRDNUM,FILNUM,(SEGDIR(I),I=1,3),BUFF
1050        FORMAT('WORD'I6,' OF FILE'I6,' IN '3A2,' CONTAINS'I6)
50    CALL SEARCH(CLOSE,0,FUNIT,0)
      CALL SEARCH(CLOSE,0,SGUNIT,0)
      CALL EXIT
C
C FILE NOT IN SEGMENT DIRECTORY
C 'PE' IS THE CODE FOR PRWFIL EOF
C 'PG' SI THE CODE FOR PRWFIL BOF
C
```

```
100   CALL GETERR(UERVEC,1)
      IF (UERVEC(1).EQ.'PE'.OR.UERVEC(1).EQ.'PG') GO TO 110
        CALL PRERR
        GO TO 50
C
110   WRITE(1,1060)(SEGDIR(I),I=1,3)
1060  FORMAT('FILE NOT IN '3A2)
      GO TO 10
C
C ERROR IN ATTEMPTING TO OPEN FILE IN SEGMENT DIRECTORY
C
120   CALL GETERR(UERVEC,1)
C
C SEE IF SEGMENT DIRECTORY ERROR TYPE
C
      IF (UERVEC(1).EQ.'SQ') GO TO 130
        CALL PRERR
        CALL EXIT
C
C YES, FILE POINTER IF SGUNIT IS AT END OF FILE OR DISK ADDRESS
C OF FILE IS 0 INDICATING NO FILE AT THIS FILE POINTER.
C IN EITHER CASE, THE ERROR INDICATES THE REQUESTED FILE IS NOT
C IN THE SEGMENT DIRECTORY.
C THIS ERROR CODE IS ALSO GIVEN IF NO FILE IS OPEN FOR READING
C ON SGUNIT IF SEARCH IS OPENING AN EXISTING FILE IN A SEGMENT
C DIRECTORY OR IF NO FILE IS OPEN FOR BOTH READING AND WRITING
C ON SGUNIT IF SEARCH IS OPENING A NEW FILE IN A SEGMENT DIRECTORY.
C THESE ERROR CONDITIONS CAN NEVER OCCUR IN THIS PROGRAM.
C
130   GO TO 110
C
C WORD NOT IN FILE
C
200   CALL GETERR(UERVEC,1)
      IF (UERVEC(1).EQ.'PE'.OR.UERVEC(1).EQ.'PG') GO TO 210
        CALL PRERR
      CALL EXIT
C
210   WRITE(1,1070)WRDNUM,FILNUM,(SEGDIR(I),I=1,3)
1070  FORMAT('WORD'I6,' NOT IN FILE'I6,' IN '3A2)
      GO TO 10
C
      END
$0
```

```
C RDVREC, CARLSON, JULY 16,1974
C
C RDVREC- READ-VARIABLE-LENGTH-RECORD
C PROGRAM TO READ LOGICAL RECORD NUMBER N FROM A FILE CONSISTING
C OF A GROUP OF VARIABLE LENGTH RECORDS AND TYPE THE RECORD
C ON THE TERMINAL.
C
C THE FILE VARREC CONSISTS OF LOGICAL RECORDS. EACH LOGICAL
C RECORD CONSISTS OF A HEADER WORD WHICH CONTAINS THE SIZE
C OF THE RECORD FOLLOWED BY THE DATA IN THE RECORD.
C THE FIRST RECORD OF THE FILE IS RECORD NUMBER 0.
C
C THE METHOD USED IS TO FIRST GENERATE PTRFIL, AN
C ANCILLARY FILE OF 2 WORD POSITION POINTERS TO EACH RECORD
C IN THE FILE VARREC. THIS IS DONE BY THE PROGRAM GPTRFL
C (GENERATE POINTER FILE) FOLLOWING THIS PROGRAM. RDVREC
C USES THE NTH FILE POINTER IN PTRFIL TO ACCESS THE NTH LOGICAL
C RECORD IN VARREC. NOTE THAT PTRFIL NEEDS TO BE GENERATED
C ONLY ONCE. AFTER THAT THE USER CAN MAKE ANY NUMBER OF
C REFERENCES TO VARREC. FOR FAST ACCESS, BOTH PTRFIL
C AND VARREC SHOULD BE GENERATED AS DAM FILES. HANDLING OF PRWFIL
C ERRORS IS OMITTED TO SIMPLIFY THIS EXAMPLE.
C
C RESTRICTIONS: FILE VARREC MUST EXIST IN THE CURRENT UFD.
C               FILE PTRFIL MUST EXIST IN THE CURRENT UFD.
C               RECORD SIZE MUST BE BETWEEN 1 AND 1000.
C               THE RECORD REQUESTED MUST BE BETWEEN 0 AND 16383.
C               THE RECORD MUST BE IN THE FILE VARREC.
C
      INTEGER FUNIT, SIZE, RECNUM, ABSPOS(2), PABSPS, BUFF(1000),
     X   PBUFF, PBUFF2
C
C VARIABLE DEFINITIONS
C FUNIT- CONTAINS 1, USED TO REFER TO FILE UNIT 1
C SIZE- HOLDS SIZE OF LOGICAL RECORD
C RECNUM- HOLDS LOGICAL RECORD NUMBER REQUESTED
C ABSPOS- HOLDS FILE POINTER
C PABSPS- POINTER TO ABSPOS
C BUFF- HOLDS RECNUM LOGICAL RECORD
C PBUFF- POINTER TO BUFF
C PBUFF2- POINTER TO BUFF(2)
C
C ROUTINES CALLED
C SEARCH, PRWFIL, EXIT, LOC
C
$INSERT KEYCOM
C
      DATA FUNIT/1/
```

```
C INITIALIZATION
C
      PABSPS=LOC(ABSPOS)
      PBUFF=LOC(BUFF)
      PBUFF2=LOC(BUFF(2))
C
C ASK FOR RECORD NUMBER.  0ST RECORD IS NUMBERED 1.
C
      WRITE(1,1000)
1000  FORMAT('TYPE RECORD NUMBER')
      READ(1,1010) RECNUM
1010  FORMAT(I6)
C
C OPEN FILE OF 2-WORD FILE POINTERS CALLED PTRFIL ON FUNIT
C
      CALL SEARCH(OPNRED,'PTRFIL',FUNIT,0)
C
C POSITION TO REQUESTED FILE POINTER AND READ IT INTO ABSPOS
C
      CALL PRWFIL(PREAD+PREREL,FUNIT,PABSPS,2,RECNUM*2,0)
C
C CLOSE FUNIT
C
      CALL SEARCH(CLOSE,0,FUNIT,0)
C
C OPEN VARREC FILE
C
      CALL SEARCH(OPNRED,'VARREC',FUNIT,0)
C
C POSITION TO THE RECORD USING THE FILE POINTER IN ABSPOS AND
C READ THE RECORD SIZE INTO BUFF(1)
C
      CALL PRWFIL(PREAD+PREABS,FUNIT,PBUFF,1,ABSPOS,0)
C
      SIZE=BUFF(1)
      IF (SIZE.LT.1.OR.SIZE.GT.1000) GO TO 100
C
C READ THE REST OF THE BLOCK INTO BUFF(2)..BUFF(N)
C
      CALL PRWFIL(PREAD,FUNIT,PBUFF2,SIZE-1,0,0)
C
C WRITE THE RECORD TO THE TERMINAL
C
      WRITE(1,1020) RECNUM,SIZE
1020  FORMAT('RECORD'I6,' IS'I6,' WORDS AS FOLLOWS:')
      WRITE(1,1030) (BUFF(I),I=1,SIZE)
1030  FORMAT(10I7)
C
C CLOSE FILE AND RETURN TO DOS
90    CALL SEARCH(CLOSE,0,FUNIT,0)
      CALL EXIT
C
C RECORD SIZE ERROR
C
100   WRITE(1,1040)
1040  FORMAT('BAD RECORD SIZE')
      GO TO 90
C
      END
```

```
C GPTRFL, CARLSON, JULY 16, 1974
C
C GPTRFL- GENERATE-POINTER-FILE
C PROGRAM TO GENERATE A FILE PTRFIL OF 2-WORD FILE POINTERS
C TO EACH LOGICAL RECORD IN FILE VARREC. VARREC CONSISTS
C OF LOGICAL RECORDS EACH OF WHICH CONSISTS OF A HEADER WORD
C THAT CONTAINS THE SIZE OF THE RECORD FOLLOWED BY THE DATA
C IN THE RECORD.
C
C RESTRICTIONS: RECORD SIZE MUST BE BETWEEN 1 AND 1000.
C               PTRFIL SHOULD NOT EXIST BEFORE RUNNING THE PROGRAM.
C               VARREC MUST EXIST IN THE CURRENT UFD.
C
      INTEGER FUNIT,PTRUNT,UERVEC(4),PUERVC,PUERV3,PSIZE,SIZE
C
C VARIABLE DEFINITONS
C FUNIT- CONTAINS 1, REFERS TO FILE UNIT 1 ON WHICH VARREC IS OPEN
C PTRUNT- CONTAINS 2, REFERS TO FILE UNIT 2 ON WHICH PTRFIL IS OPEN
C UERVEC- USER ERROR VECTOR, HOLDS ERRVEC OBTAINED FROM DOS
C SIZE- HOLDS SIZE OF LOGICAL RECORD
C PSIZE- POINTER TO SIZE
C PUERVC- POINTER TO UERVEC
C PUERV3- POINTER TO UERVEC(3)
C
C ROUTINES CALLED
C LOC, SEARCH, PRWFIL, GETERR, PRERR, EXIT
C
$INSERT KEYCOM
C
      DATA FUNIT,PTRUNT/1,2/
C
C
C INITIALIZE
C
      PUERVC=LOC(UERVEC(1))
      PUERV3=LOC(UERVEC(3))
      PSIZE=LOC(SIZE)
C
C OPEN VARREC FOR READING ON FUNIT
C
      CALL SEARCH(OPNRED,'VARREC',FUNIT,0)
C
C OPEN A NEW DAM FILE PTRFIL FOR WRITING ON PTRUNT
C
      CALL SEARCH(OPNWRT+NDFILE,'PTRFIL',PTRUNT,0)
C
C SET SIZE FOR FIRST TIME THROUGH LOOP. SIZE IS SET SO
C NO POSITIONING TAKES PLACE ON 1ST CALL TO PRWFIL. ERRVEC(3)
C AND ERRVEC(4) ARE SET TO FILE POINTER OF 1ST RECORD.
C
```

```
      SIZE=1
C
C POSITION TO NEXT LOGICAL RECORD OF VARREC.  WE HAVE
C ALREADY READ ONE WORD OF RECORD SO TO GET TO BEGINNING OF
C NEXT RECORD WE MUST POSITON FORWORD SIZE-1 WORDS.
C
10    CALL PRWFIL(PREAD+PREREL,FUNIT,0,0,SIZE-1,$90)
C
C GET FILE POINTER FROM ERRVEC
C
      CALL GETERR(UERVEC,4)
C
C FILE POINTER IS IN UERVEC(3) AND UERVEC(4).  WRITE 2 WORD
C FILE POINTER INTO PTRFIL.
C
      CALL PRWFIL(PWRITE,PTRUNT,PUERV3,2,0,0)
C
C READ 1ST WORD OF NEXT LOGICAL RECORD INTO SIZE.  1ST WORD
C IS SIZE OF NEXT LOGICAL RECORD.
C
      CALL PRWFIL(PREAD,FUNIT,PSIZE,1,0,$100)
C
C IF SIZE OK, LOOP TO READ NEXT RECORD.
C
      IF (SIZE.GE.1.OR.SIZE.LE.1000) GO TO 10
C
C ERROR
C
      WRITE(1,1000)
1000  FORMAT('A RECORD HAS A BAD HEADER WORD')
      GO TO 110
C
C FILE ENDS IN MIDDLE OF A RECORD
C
90    CALL GETERR(UERVEC,1)
      IF (UERVEC(1).NE.'PE') GO TO 120
      WRITE(1,1010)
1010  FORMAT('FILE ENDS IN A PARTIAL RECORD')
      GO TO 110
C
C PRWFIL ERROR RETURN, CHECK TYPE.
C
100   CALL GETERR(UERVEC,1)
      IF (UERVEC(1).NE.'PE') GO TO 120
C
C
C FILE ENDS NORMALLY, CLOSE FILE AND RETURN TO DOS
C
110   CALL SEARCH(CLOSE,0,FUNIT,0)
      CALL SEARCH(CLOSE,0,PTRUNT,0)
      CALL EXIT
C
120   CALL PRERR
      CALL EXIT
C
      END
```

APPENDIX I

ERRVEC CONTENTS

ERRVEC consists of eight words whose contents are as follows:

Word	Content	Remarks
ERRVEC (1)	Code	Indicates origin of error and nature of error.
(2)	Value	On alternate return, this is the value of the A-register. On normal return, this may have special meaning, (e.g., refer to PRWFIL and SEARCH error codes).
(3) (4) (5) (6)	X X X X X X X X	ERRVEC (3), ERRVEC (4), ERRVEC (5), and ERRVEC (6) contain a six-character Filename of the routine that caused the error [ERRVEC (6) is available for expansion of names]
(7)	pointer to message	For DOS (DOS/VM) supervisor usage.
(8)	message	For DOS (DOS/VM) supervisor usage.

PRWFIL Error Codes

PD	UNIT NOT OPEN	
PE	PRWFIL EOF (End of File)	Number of words left. (Information is in ERRVEC(2))
PG	PRWFIL EOF (Beginning of File)	Number of words left. (Information is in ERRVEC(2))

PRWFIL Normal Return

ERRVEC (3) = Record Number

ERRVEC (4) = Word Number

PRWFIL Read-Convenient

ERRVEC (2) = Number of words read.

SEARCH Error Codes

ERRVEC (1) =	Meaning
SA	SEARCH, BAD PARAMETER
SD	UNIT NOT OPEN (truncate)
SD	UNIT OPEN ON DELETE
SH	<Filename> NOT FOUND
SI	UNIT IN USE
SK	UFD FULL
SL	NO UFD ATTACHED
SQ	SEG-DIR-ER
DJ	DISK FULL

SEARCH Normal Return

ERRVEC (2) = Type where Type has the following values:

Type =	Meaning
0	File is SAM
1	File is DAM
2	Segment Directory is SAM
3	Segment Directory is DAM
4	UFD is SAM

APPENDIX J

DOS ERROR MESSAGES AND

DISK ERRORS AND DISK STATUS WORD

DOS ERROR MESSAGES

MESSAGE	REMARKS
BAD <COMMAND-NAME>	EXAMPLE: BAD STARTUP
BAD CALL TO SEARCH	
BAD DAM FILE	
BAD PARAMETER	
BAD PASSWORD	
BAD RTNREC	
BAD SVC	BAD SUPERVISOR CALL
DEVICE IN USE	
DISK <X> NON DOS	
DISK FULL	
DK ERR	SEE DISK ERROR EXPLANATION BELOW.
DUPLICATE NAME	
FATAL ERROR IN DOSEXT	
<FILENAME> NOT FOUND	
<FILENAME> IN USE	
<FILNAME> ALREADY EXISTS	
ILLEGAL INSTRUCTION AT <OCTAL LOC. >	
<NAME> NOT ASSIGNED	
NO UFD ATTACHED	
NO VECTOR	USER HAS GOTTEN A UII, PSU, OR FLEX, OR TRAP TO A LOCAION THAT IS 0, OR SVC SWITCH IS ON AND USER GOT AN SVC TRAP AND LOCATION '65 WAS 0.
NOT A UFD	
POINTER MISMATCH	RUN FIXRAT
PROGRAM HALT AT <OCTAL LOC. >	
PRWFIL BOF	
PRWFIL EOF	
PRWFIL POINTER MISMATCH	
PRWFIL UNIT NOT OPEN	
SEG-DIR ER	
UFD FULL	
UFD OVERFLOW	
UNIT <X> CLOSED	THIS LINE AND THE NEXT TWO LINES
DISK <X> CLOSED	ARE PART OF THE SAME
YOUR FILES DETACHED	MESSAGE.
UNIT IN USE	
UNIT NOT OPEN	
UNIT OPEN ON DELETE	

DISK ERRORS

There is no alternate return caused by a detected disk error. A message is printed and the operation is retried forever, in DOS; in DOS/VM the operation is tried ten times.

DK ER P# <Physical device #> <DOS record address > <disk status word>

Status Word

The status word typed as the third octal number of a disk error depends on the type of controller as follows:

4000 Controller

Status Word	Meaning
177777	bad record identifier
177776	device not ready
100000	data transfer complete (good if present)
040000	read/write past end of record
004000	seek complete (good if present)
002000	write protect violation
000400	command error
000200	checksum error
000100	DMX overrun
000040	stack overflow

4001 Prime Controller

Status Word	Meaning
177777	bad record identifier
177776	device not ready
100000	bit 1 always set
040000	DMX overrun
020000	disk is write protected
010000	checksum error
000100	disk drive seeking
000040	disk drive seeking
000020	disk drive seeking
000010	disk drive seeking
000004	illegal seek
000002	malfunction detected

Diskette Controller

Status Word	Meaning
177777	bad record identifier
177776	device not ready
100000	normal end of instruction (good if present)
040000	sector not found
020000	checksum error on sector ID
010000	track error; head is mispositioned
002000	deleted data mark read
001000	DMX overrun
000400	checksum error, write protect violation of file inoperable on write or format

APPENDIX K

DISK DRIVE OPERATION

This appendix describes cartridge loading and operating procedures for the various PRIME disk drive options.

PERTEC MOVING HEAD DRIVES

Operating Controls

Operating controls for the PERTEC D3000 are shown in Figure A-1. Control functions are as follows:

CONTROL	FUNCTION
OFF/ON Switch-Indicator	Turns main power on and off. Indicator lights when power is on.
SAFE Indicator	Lights when drive is not rotating and is safe to install or remove disk packs.
RUN/STOP Switch Indicator	Push switch to start or stop drive motion. Indicator lights when drive is running.
READY Indicator	Lights when drive is up to speed and ready to communicate with CPU.
PROT/PTOR Indicators	Indicates write protection status of upper (removable) and lower (fixed) disk platters.
Unit Number Selector Thumbwheel	Determines physical unit number of disk drive.
WRITE PROTECT Switches (LOWER, UPPER) behind door	Assigns or removes write protection for upper (removable) or lower (fixed) platters.

Cartridge Handling And Storage

The magnetic coatings on the disk surface have the ability to retain recorded intelligence for an indefinite period. However, the physical recording medium is susceptible to damage. The disk cargridge must be properly handled and stored to maintain the integrity of the recorded data. A damaged or contaminated cartridge can impair or prevent recovery of data and can result in damage to the disk drive.

CAUTION

Do not attempt to install or use a cartridge which is suspected of contamination or damage.

A disk drive which has been damaged or contaminated due to use of a defective cartridge should not be operated with other cartridges until the disk drive has been inspected and/or reconditioned by qualified service personnel.

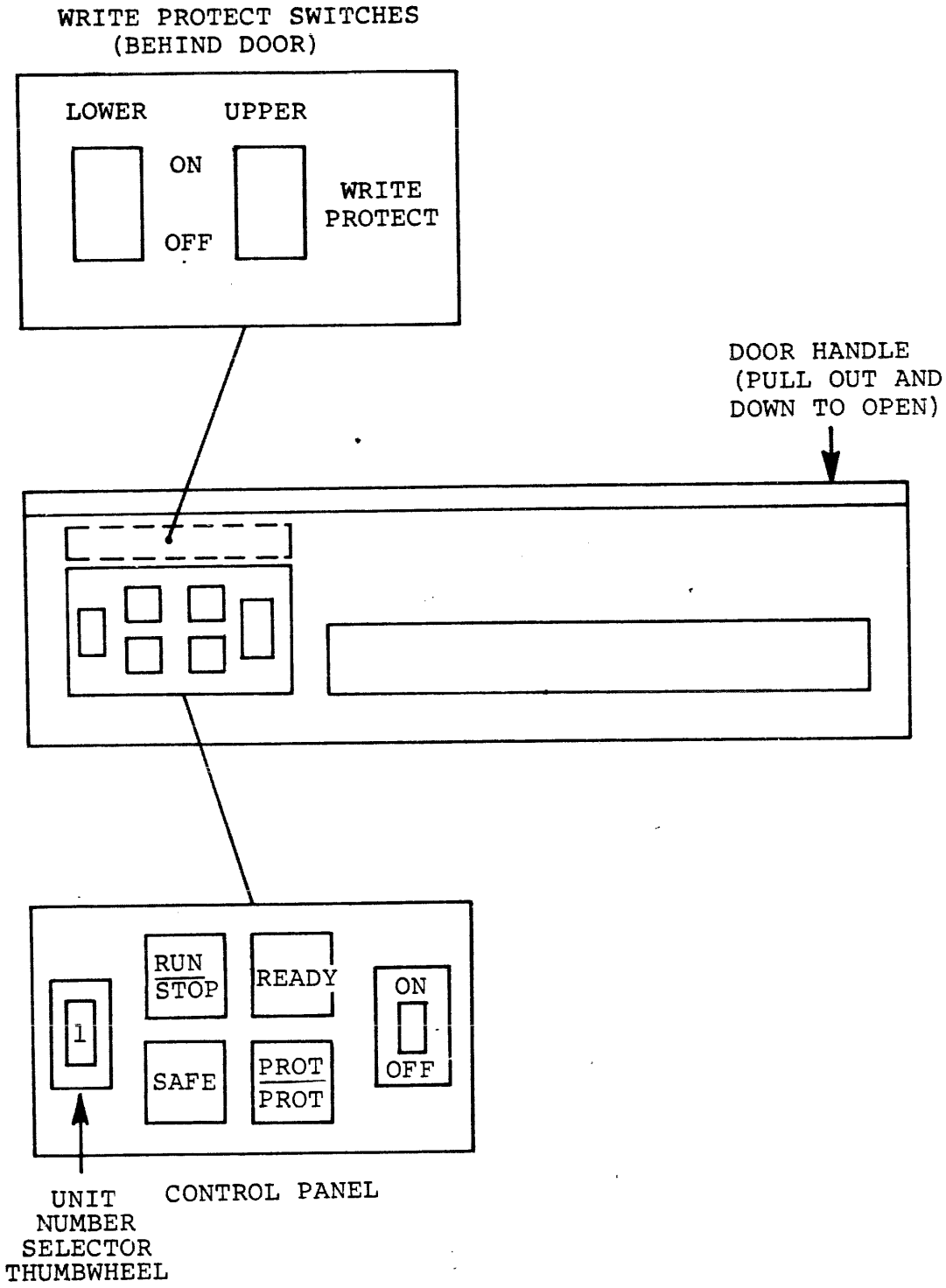


Figure K-1.

PERTEC D3000 Operating Controls

The following methods will ensure maximum protection of disk cartridges.

1. The head port door on front-loading cartridges should be kept closed when the cartridge is not inserted in a disk drive. This keeps dirt out and secures the disk internally.

2. Cartridges can be stored either horizontally or vertically. Front loading cartridges must always be positioned to avoid objects which could damage the hub or cause the air inlet door to be pushed open.

CAUTION

Do not stack cartridges more than five high.

3. Avoid exposure of the cartridge to magnetic flux in excess of 50 gauss or loss of stored data may result. The 50 gauss flux level is reached at a distance of approximately three inches from a motor, generator, transformer, or similar source.

4. Do not store the cartridge in direct sunlight. Temperatures outside the range of 33°F (0.6°C) to 140°F (60°C) should be avoided for non-operational storage.

5. If a cartridge is dropped, it should be inspected by a qualified service representative before it is used. Internal, as well as external, damage to the cartridge can result.

6. Top loading cartridges should be labeled only in the handle recess area. Placement of labels in areas other than these may cause improper operation or contamination.

Disk Drive Preparation

An initial check-out procedure must be performed by a qualified field service representative brfore the drive is operated by regular users.

Regular users should do the following initial preparation before attempting to insert a cartridge into the drive.

1. Position the power ON/OFF switch to the ON position and observe that the associated indicator becomes illuminated.

2. Observe that the SAFE indicator located on the operator control panel becomes illuminated within two seconds of the ON indicator illumination. The disk drive is now conditioned to safely insert or remove the cartridge.

Unloading a Cartridge

1. Verify that the SAFE indicator is illuminated.

CAUTION

Do not attempt to force removal of a disk cartridge when the safe indicator is extinguished. Failure to observe unsafe condition can result in damage to the equipment.

2. Grip the door handle formed by the top of the front bezel and move the handle out and downward, opening the door.

3. Grip the cartridge by the molded-in handle and pull the cartridge slowly out of the receiver.

4. Unless another cartridge is to be inserted immediately into the drive, close the disk drive door; this will exclude dirt and contamination from the interior of the drive.

Loading a Cartridge

1. Verify that the SAFE indicator is illuminated.

2. Grip the handle formed by the top of the front bezel and move the handle out and downward; this will open the disk drive door and move the cartridge receiver into position to accept a cartridge.

3. Grip the cartridge by the molded handle, and position the cartridge in the receiver opening. Make sure that the flap portion of the cartridge top is aligned between the guide rails at the top of the receiver. Slant the cartridge to match the slope of the bottom of the receiver.

4. Press the cartridge slowly but firmly most of the way into the receiver. Relax the grip on the cartridge handle and press the cartridge fully into the receiver, seating it completely within the receiver.

5. Close the door on the front of the disk drive by moving the door handle (top of the front bezel) up and toward the driver. As the door is closed, the cartridge will be positioned into the spindle.

CAUTION

If the cartridge is not properly inserted in the receiver, the door will not close. Do not attempt to force the door closed or damage to the cartridge and the disk drive will result; repeat steps 1 through 5.

Selecting Write Protection

When the disk drive is equipped with WRITE PROTECT switches, the operator should select the appropriate switch setting at the time the cartridge is inserted. The switches are mounted inside the door, behind the operator switch panel; the drive must be in a safe condition as indicated by the SAFE indicator in order to open the door and gain access to these switches.

To select protection for a particular platter, set the applicable switch to the ON position. To enable writing for a particular platter, set the applicable switch to the OFF position.

When a WRITE PROTECT switch is set to the ON position, the particular disk is protected from write operations regardless of any write commands which may be received. When a WRITE PROTECT switch is set to the OFF position, write operations may then be executed.

Starting The Disk Drive

After a cartridge is loaded, the disk drive may be started as follows:

1. Depress and release the RUN/STOP switch/indicator located on the operator control panel. Illumination of RUN/STOP indicates that there are no inhibiting conditions and that the drive mechanism is rotating the disk(s).

2. Observe that the READY indicator becomes illuminated within 60 seconds after actuating the RUN/STOP switch/indicator. The READY indicator indicates that the disk drive is ready to accept interface commands.

Stopping The Disk Drive

When it is desired to stop the disk drive while the RUN/STOP indicator is illuminated, perform the following procedure.

1. Depress and release the RUN/STOP switch/indicator.

2. Observe that the SAFE indicator becomes illuminated within 25 seconds. This indicates that the disk(s) has come to a stop and the cartridge may be removed or changed.

Designating Unit Number

When the disk drive is equipped with a Unit Number Selector Switch, the setting of this switch establishes the device (unit) number used during BOOT loading of DOS, etc. The operator should set the switch to the position required by the system and software operating procedures for the particular installation.

The switch is set by moving the thumbwheel up or down until the desired number appears in the window adjacent to the thumbwheel.

APPENDIX L

PRIME ASCII CHARACTER SET

The standard character set used by PRIME is the ANSI, ASCII 7-bit set shown in Figure C-1 . Control characters are described in Table C-1 .

The defined code set is basically a communications set complete with header and acknowledge procedures. The code is designed to allow a collating sequence, a 64 character subset, and subsetting of graphic and display motion primitives. Extensions to the communication aspects of ASCII regarding parity, control, are code extensions all contained in references 2, 3, and 4. An excellent survey of Data Communication Control Procedures is in reference 5.

PRIME USAGE

PRIME hardware and software uses standard ASCII for communications with devices. The following points are particularly important to PRIME usage.

1. Output Parity is normally transmitted as a zero (space) unless the device requires otherwise, in which case software will compute transmitted parity. Some controllers (e.g., MLC) may have hardware to assist in parity generations.

2. Input Parity is ignored by hardware and by standard software. Input drivers are responsible for making the parity bit suit the host software requirements. Some controllers (e.g., MLC) may assist in parity error detection.

3. The PRIME internal standard for the parity bit is zero. However, much existing software expects a one for the eighth bit. As a consequence, new software should be written so as to ignore the parity bit on internal characters.

INTERNAL STANDARDS

The following standards apply to internal usage of character codes, excluding communications and control functions. Internal Standards are composed of Storage Definitions and Table C-2 explains the internally redefined codes for the characters shown in Figure C-2.

	0	1	2	3	4	5	6	7
00	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL
01	BS	HT	LF	VT	FF	CR	SO	SI
02	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB
03	CAN	EM	SUB	ESC	FS	GS	RS	US
04	SP	!	"	#	$	%	&	'
05	(	)	*	+	,	-	.	/
06	0	1	2	3	4	5	6	7
07	8	9	:	;	<	=	>	?
10	@	A	B	C	D	E	F	G
11	H	I	J	K	L	M	N	O
12	P	Q	R	S	T	U	V	W
13	X	Y	Z	[	\	]	↑	
14	`	a	b	c	d	e	f	g
15	h	i	j	k	l	m	n	o
16	p	q	r	s	t	u	v	w
17	x	y	z	{	|	}	~	DEL

Figure L-1. ASCII Communications Codes

270 276

	0	1	2	3	4	5	6	7
00	NUL							BEL
01	BS	HT	NL	VT	FF	CR	RRS	BRS
02	RCP	RHT	HLF	RVT	HLR			
03								
04	SP	!	"	#	$	%	&	'
05	(	)	*	+	,	-	.	/
06	0	1	2	3	4	5	6	7
07	8	9	:	;	<	=	>	?
10	@	A	B	C	D	E	F	G
11	H	I	J	K	L	M	N	O
12	P	Q	R	S	T	U	V	W
13	X	Y	Z	[	\	]	↑	
14	`	a	b	c	d	e	f	g
15	h	i	j	k	l	m	n	o
16	p	q	r	s	t	u	v	w
17	x	y	z	{	|	}	~	

Figure L-2. Internal ASCII Codes

Basically, an internal message ("file") is composed of a number of ASCII lines terminated by a New Line character .NL. (012). The .NL. character is printed as a .CR. (Carriage Return) followed by a .LF. (Line Feed) followed (possibly) by a number of .NUL. (Null) characters for timing.

Within each ASCII line, carriage motion is defined by the following characters:

NAME	CODE	MEANING
.SP.	(240)	Space Forward One Position
.BS.	(210)	Space Backward One Position
.HT.	(211)	Physical Horizontal Tab
.VT.	(213)	Physical Vertical Tab
.FF.	(214)	Form Feed (Top of Form)
.CR.	(215)	Carriage Return
.RHT.	(221)	Relative Horizontal Tab, following byte Defines a number of .SP. to insert
.HLF.	(222)	Half Line Feed Forward
.RVT.	(223)	Relative Vertical Tab, following byte Defines a number of .LF. to insert
.HLR.	(224)	Half Line Feed Reverse

In addition, the following characters are used internally for specific device action.

.BEL.	(207)	Audible Alarm
.RRS.	(216)	Red Ribbon Shift
.BRS.	(217)	Black Ribbon Shift

The following characters are used for packing and compression:

NAME	CODE	MEANING
.NUL.	(200)	Allowed and ignored in any position
.RCP.	(220)	Relative Copy - following byte specifies number of characters to copy from corresponding positions of preceding line

VISIBLE STANDARDS

Several standards have been adopted for keyboard interfaces with standard software. Specifically:

"	(242)	Erase, i.e., ignore last character typed on the current line
?	(277)	Kill, i.e. restart current line
\ (shift L)	(234)	Logical Tab, i.e. space to logical tab of IOCS
↑	(236)	Logical Escape, visual escape for limited graphic devices
.CR.	(215)	Interpreted as .NL. on Keyboard
.LF.	(212)	Input

The logical escape conventions at present include:

↑ ddd	Three octal digit representation of unprintable character such as ↑007 (BEL)
↑ _	Backspace
↑ U	All subsequent letters are upper case
↑ L	All subsequent letters are lower case until end of line

Standard convention for PRIME Systems software is to formulate names for ASCII constants by the standardized names in Figure C-1 preceded by an "A", e.g.,

ANL	for New Line
ASP	for Space

REFERENCES

1. CACM, Vol 8, No. 4, April 1965, pg. 207

2. CACM, Vol 9, No. 9, Sept. 1966, pg. 695

3. CACM, Vol 9, No. 2, Feb. 1966, pg. 101

4. CACM, Vol 9, No. 10, Oct. 1966, pg. 759

5. SACM, Vol 4, No. 4, Dec. 1972, pg. 197

Table R-1. Control Characters

CONTROL CHARACTERS FOR COMMUNICATIONS

Code	Name	Use
201	SOH	Used at the beginning of a sequence of characters (a heading) containing address, routing, and possibly other information.
202	STX	Precedes a sequence of characters which is to be treated as an entity (a message or a message block) and passed to the destination station. STX terminates the heading, if any is present.
203	ETX	Terminates a sequence of characters (a message) begun with STX.
204	EOT	Terminates transmission.
205	ENQ	A request for a response from a remote station. It may be used to request station identification or status.
206	ACK	An affirmative acknowledgement returned to the sender from the receiver.
220	DLE	An "escape" character which changes the meaning of an immediately following string of characters. DLE was provided so that new control functions could be added using this extension character. Several two-character extension sequences have already been added.
225	NAK	A negative response returned to the sender by the receiver.
226	SYN	Used in synchronous transmission systems to provide a signal pattern from which synchronism may be attained or maintained. It is placed at the beginning of all transmitted character sequences and inserted in a sequence of characters in the absence of a data character to be transmitted.
227	ETB	Terminates a transmission block (heading or text) which is not the last block of message.

LEXICAL EQUIVALENTS

Code	Base Graphic	Alternate	
244	$	£	National Currency
237	_	←	Underbar
273	{	←	Conflict with 137
274	|	↑ or ¦	
275	}	→	
276	~	¬	Logical Not

Table R-2. Notes On Internal ASCII

Code	Name	Use
200	NUL	Filler
201	SOH	Print Header Line
207	BEL	Audible Response
212	NL	New Line, .CR. + .LF. on ASR
216	RRS	Red Ribbon Shift
217	BRS	Black Ribbon Shift
220	CRP	Relative Copy
221	RHT	Relative Horizontal Tab
222	HLF	Half Line Forward
223	RVT	Relative Vertical Tab
224	HLR	Half Line Reverse

APPENDIX M

SUMMARY OF DOS & DOS/VM COMMANDS

Command Syntax	Function	Remarks
ASRCWD Number	Change the virtual control word to select one of four devices for effective I/O.	DOS/VM only
ASSIGN Device ASSIGN Device WAIT ASSIGN DISK Number	Obtain complete control over a disk or peripheral device from the user terminal (Refer to Table 4-2 for device names and Table 3-1 for disk numbers). WAIT queues the assignment until the device is ready.	DOS/VM only. The disk assigned must be an assigned disk.
ATTACH Ufd ATTACH Ufd [Password] ATTACH Ufd [Password Ldisk] ATTACH Ufd [Password Ldisk Key]	Attach DOS (or DOS/VM user space) to the specified UFD.	For DOS/VM. Password may be owner or nonowner password.
AVAIL AVAIL ZERO AVAIL ONE AVAIL TWO ... AVAIL NINE AVAIL Packname	Print the number of disk records available for use on (1) the current logical disk; (2) the specified logical disk; (3) the logical disk specified by Packname.	Under DOS/VM, only AVAIL Packname is correct.
BASIC	Invoke the BASIC language interpreter, in order to write and execute programs in BASIC.	
BASINP	Load a paper tape containing programs written in BASIC language on a computer other than a Prime computer.	
BINARY Filename	Opens file specified by Filename for writing on File Unit 3, usually as a binary output file.	
CLOSE Filename CLOSE Filename [Funit...] CLOSE ALL	Closes the named files and specified file units; or if ALL is specified, closes all files and units.	
CMPRES Filename [Filename2]	Translate an input file into an output ASCII file using the relative copy character ('220)	EXPAND is the opposite of CMPRES.

Command Syntax	Function	Remarks
CNAME Oldname Newname	Change name of a file named Oldname to Newname.	
COMINPUT Filename	Read commands from the file specified by Filename in the current UFD, rather than from the terminal.	
COMINPUT Filename Funit	Read commands from the file specified by Filename or the logical unit specified by Funit.	
COMINPUT CONTINUE	Continue reading commands from a command file after a pause or interruption.	
COMINPUT PAUSE	Leave the current command input unit open and return to operating system command level.	
COMINPUT TTY	Read subsequent commands from the terminal.	Should be last line of command file or the last command file in a chain.
COPY	Copies and verifies a disk.	Use this command with acumen (i.e. know what you are doing).
CREATE Newufd	Create a new UFD, Newufd, in the current UFD.	
CRMPC Filename	Read cards from the parallel interface card reader and place their image in the file specified by Filename.	First card in deck should be #E.
CRSER Filename	Read cards from the serial interface card reader and place their image in the file specified by Filename.	First card in deck should be #E.
DBASIC	Invoke a version of BASIC that provides double precision arithmetic capabilities.	
DELAY DELAY [Minimum] DELAY [Minimum Maximum] DELAY [Minimum Maximum Rmargin]	Define a time function to be used to delay the printing of a character after a LINE FEED has been output to the terminal.	DOS/VM only.

Command Syntax	Function	Remarks
ED ED Filename	Load and start the system text editor (in INPUT mode if no Filename is specified; in EDIT mode if Filename is specified). Editor commands are: LINE Mode Editor Commands: APPEND String BOTTOM BRIEF CHANGE/String1/String2/[nG] DELETE [n] DELETE To String DUNLOAD Filename [n] DUNLOAD Filename To String ERASE Char FILE FILE Filename FIND String INPUT Device INSERT String LOAD Filename LOCATE String MODE PRUPPER MODE PRALL MODE PROMPT MODE NPROMPT MODE LINE MODE BOX MODIFY/String1/String2/[nG] MOVE Buffer1 Buffer2 NEXT [n] OUTPUT TTY OUTPUT OVERLAY String PAUSE PRINT [n] PTABSET Tab... PUNCH (ASR) n PUNCH (PTP) n QUIT RETYPE String SYMBOL Name Char TABSET Tab... TOP UNLOAD Filename [n] UNLOAD Filename To String VERIFY WHERE XEQ Buffer * [n]	Editor Commands are described in the Program Development Software User Guide in detail.

Command Syntax	Function	Remarks
	BOX Mode Editor Commands:	
	BOX v h ↑D# ↑D#	
	BOXIN Filename (MODIFY)	
	BOXIN Filename (OVERLAY)	
	BOXOUT	
	BRIEF	
	DISPLAY	
	ERASE Char	
	FILE Filename	
	FIND String	
	KILL Char	
	MODE PRUPPER	
	MODE PRALL	
	MODE PROMPT	
	MODE NPROMPT	
	MODE LINE	
	MODE BOX v h	
	MODIFY/String1/String2/[G]	
	MOVE Buffer1 Buffer2	
	OUTPUT	
	OVERLAY String	
	POINT v h ↑D# ↑D#	
	PRINT	
	PTABSET	
	QUIT	
	RFIND String	
	RLOCATE String	
	PPOINT v h ↑D# ↑D#	
	SYMBOL Name Char	
	VERIFY	
	WHERE	
	XEQ	
	*[n]	

Command Syntax	Function	Remarks
EDB Inputfile [Outputfile] EDB (PTR) [(PTP)]	Loads and starts the binary editor; EDB Commands are: BRIEF COPY Name COPY ALL ET FIND Name FIND ALL GENET [G] INSERT Name NEWINF [Name] OMITET [G] OPEN [Name] QUIT RFL SFL TERSE TOP VERIFY	For details of EDB commands, refer to the Program Development Software User Guide.
EXPAND Filename1 [Filename2]	Inverts the operation of CMPRES	
FILBLK	Reads or writes from high speed memory to any 128-word record in a previously created RTOS random access file.	Refer to RTOS User Guide.
FILMEM	Fills memory locations with zeroes from '100 to the top of 32K, except for those locations occupied by DOS.	Under DOS/VM, all locations from '100 to top of 32K are filled with zeroes.
FILVER Filename1 Filename2	Compares contents of file specified by Filename1 with contents of file specified by Filename2 for equivalence and prints message that verification is either confirmed or is not confirmed.	
FIXRAT FIXRAT OPTIONS	Loads and starts a maintenance program that checks file integrity of any disk pack.	For DOS/VM, the disk being checked must be ASSIGNED. Refer to Appendix E for complete details about FIXRAT.

Command Syntax	Function	Remarks
FTN Filename [1/A]	Loads Prime FORTRAN IV and starts compilation of a program	
FUTIL	Invokes a file utility that provides subsystem commands to copy, delete and list both files and directories. FUTIL commands are: ATTACH Directory Pathname COPY File1 [,File2...] COPYDAM File1 [,File2...] COPYSAM File1 [,File2...] DELETE File1 [,File2...] FROM Directory Pathname LISTF [level] [LISTFIL] [PROTECT] [SIZE] [TYPE] QUIT TO Directory Pathname TRECPY Dir1 [,Dir2...] TREDEL Dir1 [,Dir2...] UFDCPY UFDDEL	Refer to Appendix F for a detailed description of FUTIL.
HILOAD	See LOAD	
INPUT Filename	Opens an ASCII source file on Unit 1 for reading by a compiler or assembler.	
LBASIC	Invoke a version of BASIC with MAT and PRINT USING.	
LISTF	Print the current UFD name, the logical device, and all Filenames in the UFD at the terminal.	For DOS/VM, LISTF also prints O or N for owner or nonowner status.
LISTING Filename	Opens the file specified by Filename for writing on File Unit 2, usually as a listing output file.	
LFTN	Invokes a version of FORTRAN that can perform Sector 0 optimization.	

Command Syntax	Function	Remarks
LOAD	Loads and starts Prime's Linking Loader. LOAD provides the following commands: ATtach [Ufd][Password][Ldisk] [Key] COmmon Address EXecute [AReg][BReg][XReg] FOrce Filename [Loadpoint] [Linkstart][Linkrange] HArdware Definition INitialize [Filename][Loadpoint] [Linkstart][Linkrange] LOad Filename [Loadpoint] [Linkstart][Linkrange] LIbrary [Filename] MAp [Option] MOde Mode QUit REcover SAve Filename [AReg][BReg][XReg] SErbase Linkstart Linkrange VIrtualbase Startlinks To sector	Loader 60000-63777. P-register=61000. LOAD will now send maps to Disk Unit 2. Unit 2 must be open for writing.
LOAD20	See LOAD	Loader 20000-23777. P-register=21000
LOGIN	Connect to the DOS/VM system for a terminal session.	DOS/VM only.
LOGOUT	Give up user-access to the DOS/VM system. (Exit from a terminal-session).	DOS/VM only.
MACHK	For DOS, causes computer to operate in machine check mode.	DOS/VM default is machine check mode.
MAGSAV	Write all or part of the contents of a disk to magnetic tape.	
MAGRST	Read the contents of a magnetic tape, to a disk or portion of a disk.	

Command Syntax	Function	Remarks
MAKE	Creates a disk supported by DOS or DOS/VM that contains the following: DSKRAT MFD BOOT DOS CMDNC0	
MCG Filename	Translates results of microcode assembly into proper code for the ROM simulator.	
MDL	Punches paper tapes of specified sections of memory in a self-loading format.	
NUMBER	Utility to number or renumber a BASIC program.	
OPEN Filename Funit Key	Opens the file specified by Filename on the File Unit; Funit; Key specifies type of file and action to be taken.	
PASSWD Owner Password	Replace any existing Password in the current UFD with a new password.	DOS version of Password.
PASSWD Owner-Password Non-owner Password	Same as above, except assigns both owner and nonowner passwords.	For DOS/M. See Section 2 for a discussion of file access protection.
PASSWD	Replace existing passwords with null (no) password.	Both DOS and DOS/VM.
PROTECT Filename Key1 Key2	Open file directory giving restricted access rights to Filename as specified by Key1 and Key2.	DOS/VM only. Key1 or Key2=: 0 = No access 1 = Read only 2 = Read and write 3 = Delete only 4 = Delete, truncate, and read 5 = Delete, truncate, and write 6 = All access

Command Syntax	Function	Remarks
PM	Prints contents of the RVEC vector.	
PMA Filename [1/A]	Load the macro assembler and start assembly of Filename in the current UFD.	Default value of A is 000777 which signifies: normal listing detail, all input and output files on disk.
PRERR	Prints message stored in ERRVEC.	
PSD	Load and Start the interactive debugging program.	
PSD20	Invoke version of PSD for 16K DOS.	
PTCPY	Loads a utility program that duplicates and verifies paper tapes.	
PTRED	Edit files read from paper tape.	
RESTORE Filename	Restore Filemane in the current UFD to high speed memory, using the SA and EA values SAVEd with Filename.	Refer to the description of RVEC in Section 4.
RTOSRA	Establish RTOS mapped random access file.	
RT128F	RTOS off-line utility to read and write 128-word segment formatted files.	Use only as directed in the RTOS User Guide.
SAVE Filename	Save the content of high-speed memory using SA (starting address) to EA (ending address) on a file named Filename in the current UFD.	
SHUTDN	For DOS, shutdown the system (no parameters).	
SHUTDN [Pdisk...] SHUTDN ALL	For DOS/VM, shutdown the specified physical disk (Pdisk) or shutdown the entire system (ALL)	

MAN 1675

Command Syntax	Function	Remarks
SIZE Filename	Prints the size of Filename in records, at the terminal.	
SLIST Filename	Prints the content of Filename at the terminal.	
SORT SORT BRIEF SORT SPACE SORT MERGE	Sort an ASCII file and write the sorted file in the current UFD. BRIEF =: no messages SPACE =: delete blank lines from output MERGE =: merge (up to 10) unsorted files.	SORT command gives instructions (messages) as it is executing. SORT asks for names of files to be merged.
SPOOL [Filename]	Queues a copy of Filename in the UFD SPOOL for off-line printing. SPOOL typed with no Filename opens File Unit 2 for writing in the UFD SPOOL and prints them after they are closed(either by the user or the end of the program). Using SPOOL with no Filename argument is a convenient way to get listings and LOAD maps printed.	DOS/VM only.
START [PC][A][B][X][Keys]	Initializes the registers and keys from the command line (or from RVEC) and starts execution at the location PC.	START can also restart a program (refer to Section 4).
STARTUP Pdisk [Pdisk1...]	Initialize the configuration of disk drives by relating physical disks to logical disk unit number.	STARTUP has extended capabilities for DOS/VM (refer to Section 6).
STATUS	Print status information at the terminal.	STATIS information varies for DOS and DOS/VM; for details, refer to Section 4.
SVCSW	Controls the handling of SVC instructions in the virtual memory environment.	DOS/VM only.
TIME	Prints the current value of the time accounting registers.	DOS/VM only.

Command Syntax	Function	Remarks
UNASSIGN	Deassigns peripheral devices or disks.	DOS/VM only. UNASSIGN may be entered from either a user terminal or the system terminal (refer to Section 4).
VDOS32	Starts a version of DOS that may be run under DOS/VM.	DOS/VM only.
VRTSSW	Allows setting of the virtual sense switches.	DOS/VM only.
*	Indicates comment line.	* must be followed by a space and have the correct command line form (1 to 3 names followed by 0 to 9 parameters).

OBSOLETE COMMANDS

Names of Obsolete Commands	Replaced By
CARDIN	CRSER
LOAD74, LOAD40, LOAD70	HILOAD
PRINT	PRSER
FILCPY	FUTIL
BOOT	deleted
COPYVM	COPY
VFIXRT	FIXRAT
CNVT45	deleted
VMAKE	MAKE

APPENDIX N

FIXRAT OF MASTER DISK (REV 7)

```
BEGIN UPDCPY
BEGIN SLIST
BEGIN C_CPB
BEGIN CPBGEN
BEGIN C_CPBG
BEGIN CMPRES
BEGIN EXPAND
BEGIN PTCPY
BEGIN LCHR
BEGIN C_PTCP
END   AIDS2     185
BEGIN SPARE
BEGIN SPARE
END   SPARE       1
BEGIN BOSSRC
BEGIN BOSSRC
 BEGIN C1IN
 BEGIN GETBUF
 BEGIN RTNBUF
 BEGIN COMANL
 BEGIN CMREAD
 BEGIN ERRRTN
 BEGIN ERR
 BEGIN RESUME
 BEGIN RESTOR
 BEGIN SAVE
 BEGIN READ
 BEGIN BOSEXT
 BEGIN ERRS
 BEGIN COMINP
 BEGIN RTNREC
 BEGIN BOSCOM
 BEGIN GETREC
 BEGIN GINFO
 BEGIN PRWFIL
 BEGIN BSPISK
 BEGIN K3MAC
 BEGIN BOSVC
 BEGIN FSAT
 BEGIN CNEQV
 BEGIN SCHAR
 BEGIN GCHAR
 BEGIN MOVE
 BEGIN FILL
 BEGIN TEXTOK
 BEGIN TYPERS
 BEGIN BBOSEX
 BEGIN FSCG
 BEGIN BOSLOW
 BEGIN BCDEF
 BEGIN INTRO
 BEGIN C_FTN
 BEGIN C_PMA
 BEGIN GETERR
 BEGIN PRERR
 BEGIN WRITE
 BEGIN GRTNSB
 BEGIN C_BOSE
 BEGIN C_LDSS
 BEGIN *LBUT
 BEGIN LOABUT
 BEGIN ATTACH
 BEGIN PMAIN
 BEGIN SEARCH
 BEGIN BOSSUB
 BEGIN UPDATE
 BEGIN VDIO
END   BOSSRC    119
BEGIN FLIB1
BEGIN FLIB1
 BEGIN DATAN2
 BEGIN Z$80
 BEGIN C$81
 BEGIN C$16
 BEGIN DMOD
 BEGIN DSIGN
 BEGIN DABS
 BEGIN DBLE
 BEGIN DMAX1
 BEGIN DMIN1
 BEGIN DINT
 BEGIN E$62
 BEGIN E$61
 BEGIN ES26
 BEGIN ES66
 BEGIN AS62
 BEGIN SS62
 BEGIN MS62
 BEGIN DS62
 BEGIN AS61
 BEGIN SS61
 BEGIN MS61
 BEGIN DS61
 BEGIN CS26
 BEGIN CS62
 BEGIN CS61
 BEGIN AS81
 BEGIN DEXP
 BEGIN DSQRT
 BEGIN DLOG-2
 BEGIN DLOG10
 BEGIN C_FLB1
 BEGIN L_FLB1
 BEGIN BFLIB1
 BEGIN DATAN
 BEGIN DSNCS
END   FLIB1      54
BEGIN FLIB2
BEGIN FLIB2
 BEGIN CSQRT
 BEGIN CCOS
 BEGIN CSIN
 BEGIN CLOG
 BEGIN CEXP
 BEGIN CABS
 BEGIN ES51
 BEGIN AS52
 BEGIN SS52
 BEGIN MS52
 BEGIN DS52
 BEGIN AS55
 BEGIN SS55
 BEGIN MS55
 BEGIN DS55
 BEGIN CONJG
 BEGIN CS25
 BEGIN CMPLX
 BEGIN NS55
 BEGIN LS55
 BEGIN HS55
 BEGIN AIMAG
 BEGIN AS51
 BEGIN SS51
 BEGIN DS51
 BEGIN C_FLB2
 BEGIN L_FLB2
 BEGIN MS51
 BEGIN BFLIB2
 BEGIN REAL
 BEGIN CS52
END   FLIB2      37
BEGIN FLIB3
BEGIN FLIB3
 BEGIN ABS
 BEGIN AMOD
 BEGIN DIM
 BEGIN SIGN
 BEGIN RND
 BEGIN ATAN
 BEGIN ATAN2
 BEGIN ES22
 BEGIN SQRT
 BEGIN ALOG
 BEGIN CS216
 BEGIN C_FLB3
 BEGIN L_FLB3
 BEGIN TANH
 BEGIN ES21
 BEGIN EXP
 BEGIN AINT
 BEGIN BFLIB3
 BEGIN SINCOS
END   FLIB3      35
BEGIN FLIB4
BEGIN FLIB4
 BEGIN MAX0
 BEGIN MAX1
 BEGIN MIN0
 BEGIN FLOAT
 BEGIN IFXINT
 BEGIN CS18
 BEGIN CS21
 BEGIN MIN1
 BEGIN IRND
 BEGIN ES11
 BEGIN ISIGN
 BEGIN IDIM
 BEGIN AS21
 BEGIN SS21
 BEGIN MS21
 BEGIN DS21
 BEGIN C_FLB4
 BEGIN L_FLB4
 BEGIN BFLIB4
END   FLIB4      31
BEGIN FLIB5
BEGIN FLIB5
 BEGIN C_FLB5
 BEGIN L_FLB5
 BEGIN LGEX1
 BEGIN EXEX1
 BEGIN EX2X1
 BEGIN DSQRX1
 BEGIN DEXEX1
 BEGIN DEX2X1
 BEGIN DATNX1
 BEGIN COSX1
 BEGIN DCOSX1
 BEGIN DLGEX1
 BEGIN ATNX1
 BEGIN DLG2X1
 BEGIN DSINX1
 BEGIN LG2X1
 BEGIN DSUB
 BEGIN DMPY
 BEGIN DADD
 BEGIN RONB
 BEGIN RODB
 BEGIN TWOS
 BEGIN MPY
 BEGIN DIV
 BEGIN SINX1
 BEGIN BFLIB5
 BEGIN SQRX1
END   FLIB5      92
BEGIN FLIB6
BEGIN FLIB6
 BEGIN FSUII
 BEGIN ACCN
 BEGIN OVERFL
 BEGIN FSBN
 BEGIN FSBN
 BEGIN FSFN
 BEGIN SLITE
 BEGIN FSCG
 BEGIN FSSH
 BEGIN FSFLEX
 BEGIN FATI
 BEGIN C_FLB6
 BEGIN L_FLB6
 BEGIN BFLIB6
 BEGIN FSRN
 BEGIN FSDE
 BEGIN FSEN
 BEGIN FSTR
 BEGIN FSER
 BEGIN FSIO
 BEGIN FSWN
END   FLIB6      188
BEGIN AIDS
BEGIN AIDS
 BEGIN TAP
 BEGIN MBL
 BEGIN PSB
END   AIDS       102
END   MFD       3147
RECORDS USED(DECIMAL)=    3147
RECORDS LEFT=      181
BSKRAT OK
```

```
OK, AS DISK 0
OK, FIXRAT OPTION
@D
REV. 7.0
FIX DISK? NO
PHYSICAL DISK = 0
TYPE DIRECTORIES TO LEVEL (CR)
TYPE FILE NAMES? YES
TYPE FILE CHAINS? NO

DISK PACK ID IS MD7V2
BEGIN MFD
BEGIN MFD
 BEGIN MD7V2
 BEGIN BOOT
 BEGIN CMBNC0
 BEGIN CMBNC0
  BEGIN COPYB
  BEGIN BOSEXT
  BEGIN COPY
  BEGIN FIXRAT
 END   CMBNC0      26
 BEGIN BOS
 BEGIN BOS
  BEGIN *BOS32
  BEGIN *BOS24
  BEGIN *BOS16
 END   BOS         34
 BEGIN PMA
 BEGIN PMA
  BEGIN PMAIO1
  BEGIN PMAIO2
  BEGIN C_PMA1
  BEGIN C_PMA2
  BEGIN PMA
 END   PMA        163
 BEGIN FORTRN
 BEGIN FORTRN
  BEGIN FTNIO1
  BEGIN FTNIO2
  BEGIN C_FTN1
  BEGIN C_FTN2
  BEGIN FTN
 END   FORTRN     228
 BEGIN LDR
 BEGIN LDR
  BEGIN LOADAP
  BEGIN LOAD
  BEGIN C_LOAD
 END   LDR        118
 BEGIN BASIC1
 BEGIN BASIC1
 END   BASIC1       1
 BEGIN T&MSRC
 BEGIN T&MSRC
  BEGIN CPUT2
  BEGIN CPUT3
  BEGIN TTYT1
  BEGIN RTCT1
  BEGIN WDTT1
  BEGIN AMLCT1
  BEGIN MTUT1
  BEGIN FLT1
  BEGIN FLTPT1
  BEGIN C_FLTP
  BEGIN DFLT
  BEGIN DFT2
  BEGIN C_DFLT
  BEGIN GPIB
  BEGIN CPUT1
  BEGIN RAMP
  BEGIN DIGINP
  BEGIN LPTST1
  BEGIN PAGT1
  BEGIN FLTP0
  BEGIN MACIT1
  BEGIN IPCT1
  BEGIN DPCARD
  BEGIN DSCTST
  BEGIN TTYT2
  BEGIN RTCT2
  BEGIN HSMT1
  BEGIN K3MAC
  BEGIN MSLCD1
  BEGIN MSLCS1
  BEGIN MSLCS2
  BEGIN MSLCS3
  BEGIN MSLCS4
  BEGIN MSLCT1
  BEGIN A/DST1
  BEGIN DISCT1
  BEGIN HSRPT2
  BEGIN HSRTT2
  BEGIN DPIOT1
  BEGIN HSRPT1
  BEGIN DST1
  BEGIN DSKTT1
  BEGIN WCSP
  BEGIN DRATIT
  BEGIN P300T1
  BEGIN HSMT2
  BEGIN SDOT
  BEGIN C_DSKT
 END   T&MSRC    1251
 BEGIN DVSRC
 BEGIN DVSRC
  BEGIN DIGDIM
  BEGIN TFVDF
  BEGIN DIGIN
  BEGIN DILIB
  BEGIN TUTILS
  BEGIN DBDIM
  BEGIN ASCCOM
  BEGIN FSCG
  BEGIN FSAT
  BEGIN CNIOV
  BEGIN AMLDUF
  BEGIN C_FTN
  BEGIN CRDDIM
  BEGIN DVCDEF
  BEGIN LOCATE
  BEGIN TEXTOK
  BEGIN GCHAR
  BEGIN TFLIOB
  BEGIN DELAY
  BEGIN SCHAR
  BEGIN C_DELE
  BEGIN DFGETR
  BEGIN K3MAC
  BEGIN CPMAML
  BEGIN MPCINT
  BEGIN MPCDIM
  BEGIN PBDIOS
  BEGIN FORCEW
  BEGIN DVDISK
  BEGIN USRCOM
  BEGIN INIT
  BEGIN FATI
  BEGIN TTYPER
  BEGIN ASRDIM
  BEGIN DVFC
  BEGIN FSOR
  BEGIN MTINT
  BEGIN MTDIM
  BEGIN DVMCOM
  BEGIN NLKCOM
  BEGIN AMLDIM
  BEGIN DVFA
  BEGIN TMAIN
  BEGIN CPM330
  BEGIN CPM300
  BEGIN TSLCI0
  BEGIN SLCCOM
  BEGIN SLCDIM
  BEGIN TSSLC0
  BEGIN C_SLC
  BEGIN CPMSLC
  BEGIN DVFB
 END   DVSRC      282
 BEGIN FILAID
 BEGIN FILAID
  BEGIN RBOD
  BEGIN HCONVT
  BEGIN C_CNVT
  BEGIN CONVRT
  BEGIN AYENAY
  BEGIN BADDSK
  BEGIN RBOOT
  BEGIN C_BCOP
  BEGIN RBODEC
  BEGIN COPYB
  BEGIN C_COPY
  BEGIN BWREC
  BEGIN C_FIXR
  BEGIN FXOPMA
  BEGIN MOVE
  BEGIN FILL
  BEGIN FIXRAT
  BEGIN K3MAC
  BEGIN FIXCOM
  BEGIN MAKA
  BEGIN C_MAKE
  BEGIN BOOT
  BEGIN C_BOOT
  BEGIN COPY
  BEGIN MTDSK
  BEGIN MTDIOC
  BEGIN C_MTDS
  BEGIN MAKE
  BEGIN MAGRST
  BEGIN MAGSAV
  BEGIN MAGCOM
  BEGIN C_MSAV
  BEGIN C_MRST
  BEGIN SVRSTR
  BEGIN C_TRSM
  BEGIN C_FUTI
  BEGIN FUTCOM
  BEGIN FUTIL
  BEGIN FODEC
  BEGIN *BOOT
 END   FILAID     207
 BEGIN ED
 BEGIN ED
  BEGIN EDFLAG
  BEGIN EDCOM
  BEGIN EDMAIN
  BEGIN EDBOX
  BEGIN EDFSUB
  BEGIN PTREFS
  BEGIN EDIO
  BEGIN EDPSUB
  BEGIN C_ED
  BEGIN C_EDLI
  BEGIN C_PTED
  BEGIN C_DPTE
  BEGIN EFS
  BEGIN ED
  BEGIN *ED
  BEGIN *EDLI
 END   ED         150
 BEGIN BINED
 BEGIN BINED
  BEGIN EDBCOM
  BEGIN C_EDB
  BEGIN EDB
 END   BINED       14
 BEGIN AIDS2
 BEGIN AIDS2
  BEGIN PRSER
  BEGIN PRMPC
  BEGIN CRSER
  BEGIN CRMPC
  BEGIN SPOOL
  BEGIN SPLMPC
  BEGIN SPLCEN
  BEGIN AWAIL
  BEGIN CPBOOT
  BEGIN BOOTGN
  BEGIN C_BTGN
  BEGIN ASRBOT
  BEGIN PTRBOT
  BEGIN SIZE
  BEGIN NUMCOM
  BEGIN NUMBER
  BEGIN C_NUMB
  BEGIN FILVER
  BEGIN MCG
  BEGIN FILMEM
  BEGIN AIBMAK
  BEGIN PSG
  BEGIN FILCPY
```

MAN 1675

MAN1675

T

MAN1675