MOTOROLA

M68MDOS3(D2)

EXORdisk II/III OPERATING SYSTEM

User's Guide

MICROSYSTEMS

M68MDOS3(D2)
JUNE 1979

EXORdisk II/III Operating System User's Guide

Second Edition
Copyright 1979 by Motorola, Inc.
First Edition, December 1978

MANUAL ORGANIZATION

The purpose of this guide is to provide the user with the necessary information required to generate an MDOS system, to use the MDOS command programs, and to produce user-written programs that are compatible with MDOS. In addition, a brief summary is presented of the MDOS-supported software products which are currently available.

The User's Guide has been divided into two parts. Chapters 1 and 2 are intended for the new user of MDOS who is just receiving his system. It is essentially a manual within a manual that can be read as an entity by itself. It provides the basic concepts that are necessary for the installation of the EXORdisk and for the simplified operation of MDOS. Chapters 2 through 28 contain descriptions and examples of the basic forms of the most frequently used MDOS commands in the order in which they would most likely be used in a software development environment. The infrequently used commands are also summarized in order to direct the user to those chapters (command descriptions) as the need for their use arises.

Chapters 2 through 28 are intended as a detailed reference manual for those who need to know specific or extended information about the MDOS commands, the system structure, and the resident system functions.

TABLE OF CONTENTS Page

PART II -- ADVANCED MDOS USER'S GUIDE

TABLE OF CONTENTS Page

TABLE OF CONTENTS Page

TABLE OF CONTENTS Page

APPENDICES

CHAPTER 1

1. INTRODUCTION

The EXORdisk II is a single-sided, single-density, dual diskette drive storage system designed for use with the EXORciser or EXORterm. The EXORdisk III is a double-sided, single-density, dual diskette drive storage system designed for use with the EXORciser or EXORterm. The EXORdisk III can be expanded into a four-drive system.

With either the EXORdisk II or EXORdisk III system, the following items are also included: a floppy disk controller module, a floppy disk interconnection cable assembly, and a software disk operating system. An illustration of a typical EXORdisk system is shown in Figure 1-1.

The M6800 Diskette Operating System (MDOS) or M6809 Diskette Operating System (MDOS09), in conjunction with the EXORciser and EXORdisk II or EXORdisk III, provides a powerful and easy-to-use tool for software development. For the remainder of this manual, all references to MDOS will encompass both the M6800 version as well as the M6809 version, unless otherwise specified.

MDOS is an interactive operating system that obtains commands from the system console. These commands are used to move data on the diskette, to process data, or to activate user-written processes from diskette. All this can be accomplished with a minimum of effort; and since MDOS is a facilities oriented system, rather than a supervisory oriented one, a minimum of overhead is imposed.

In addition, an extensive set of resident system functions are provided for general development use. Such functions as dynamic space allocation, random access to data files, record I/O for supported and non-supported devices, as well as many register, string, and other diskette-oriented routines make MDOS a good basis for a user's application system.

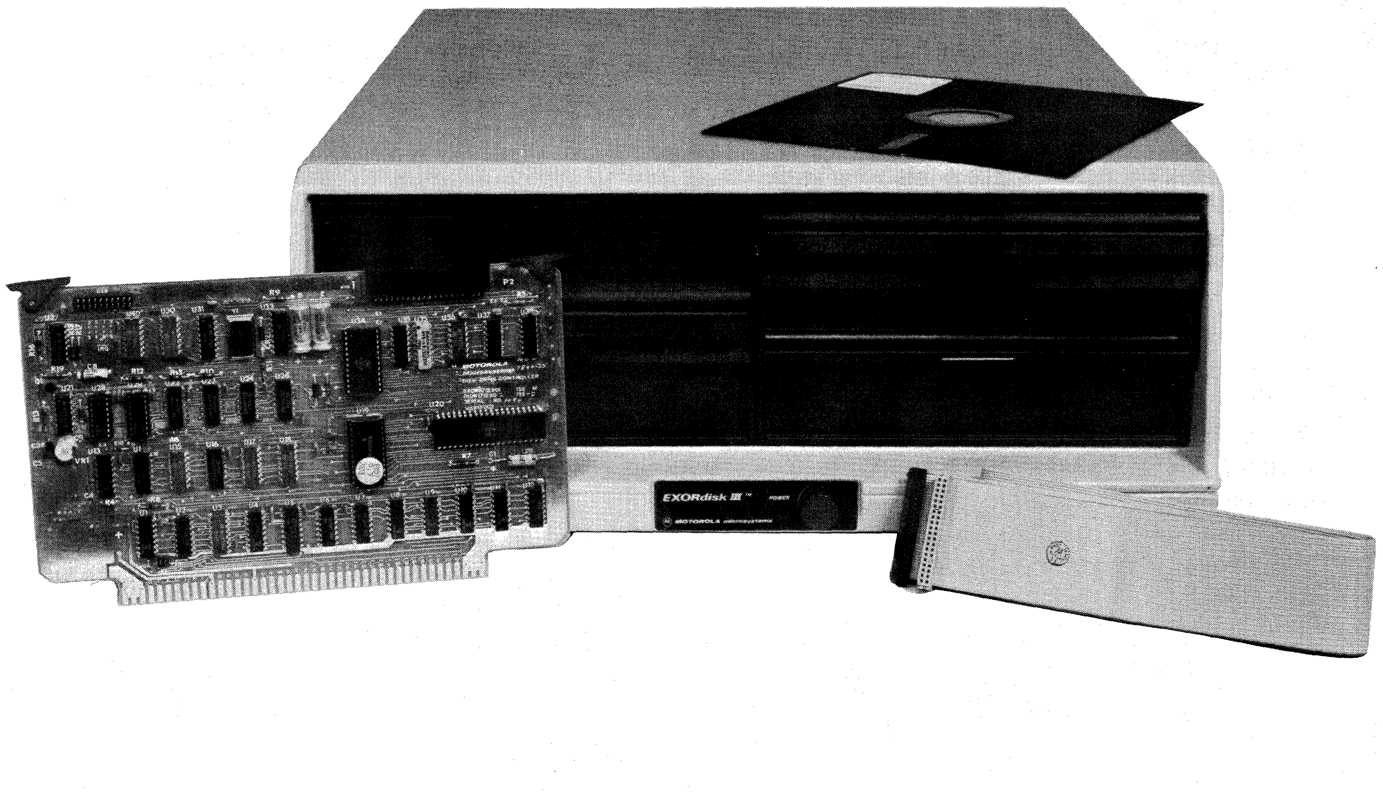


Figure 1-1. Typcial EXORdisk system.

1.1 Hardware Support Required

The minimum hardware configuration required to support MDOS consists of:

- -- an EXORciser or EXORterm with EXbug firmware
- -- 16K RAM
- -- EXORdisk II/III dual diskette drive unit
- -- EXORdisk II/III floppy disk controller module
- -- Interconnect cable
- -- ASR33 (TTY) or RS-232C compatible terminal

The EXORdisk II can read and write diskettes recorded in an IBM-3740-similar format (single-sided, single-density). The EXORdisk III can read and write all diskettes that the EXORdisk II can handle. In addition, diskettes formatted in the Motorola single-density, double-sided format can also be read and written. The double-sided diskettes cannot be used in the EXORdisk II.

The above minimum configuration will allow the user to run any of the MDOS commands that reside on the MDOS system diskette at the time of purchase. Other additional hardware may be required to run the MDOS-Supported software products. Such information is described in Appendix H.

1.2 Additional Supported Hardware

MDOS also supports a line printer and the reader/punch (record) devices of the system console. The line printer interfaces to the EXORciser through the printer interface module (MEX68PI) which consists of two PIA's plus the necessary buffering devices and address decoding. If the printer interface from an EDOS system is used instead, it must be modified for use with MDOS. The modifications consist of adding the following lines to the printer interface PIA:

1. Print select (high=selected) to PB0 (pin 10 of PIA)
2. Paper out (low=paper available) to PB1 (pin 11 of PIA)

The system console's automatic reader/punch (record) devices must be similar to a Teletypewriter's paper tape reader and punch. For a complete description of the system console requirements consult the "M6800 EXORciser User's Guide".

1.3 Software Support Required

No additional software is required to run the operating system as it comes shipped on the system diskette.

1.4 Program Compatibility

All of the MDOS commands and system files that are shipped on the system diskette must be used with that particular version of MDOS. MDOS commands and system files from other versions should never be intermixed. MDOS-Supported software products (see Appendix H) with version numbers 3.00 or greater must be used with MDOS 3.00. They will not operate correctly with prior versions of MDOS. In addition, prior versions of the M6800 Linking Loader (RLOAD, through version 2.03) will not operate with MDOS 3.00. Prior versions of other MDOS-Supported software products will work with MDOS 3.00.

Most user-written assembly language programs that were developed independently of MDOS can be executed on an MDOS system without reassembly; however, such programs will have to be converted into the memory-image file format before they can be loaded from diskette into memory (see section 2.8.5). Programs need only be changed when transferred to MDOS if:

1. They make assumptions about the initialization of the stack pointer after they are loaded into memory,

2. They are origined to load (initialize memory while loading) below hexadecimal location $20,

3. They make assumptions about the physical structure of diskette tables or files,

4. They utilize the diskette for input/output,

5. They make assumptions about the contents of the SWI and IRQ interrupt vectors.

If a user has prior EXORciser support software products which he has purchased from Motorola (e.g., editors, assemblers, or compilers), that software must be upgraded to be compatible with MDOS.

If a user has software that he has developed using previous versions of MDOS, then Appendix J should be consulted for a list of differences between MDOS 3.00 and prior versions that may affect programs running with MDOS 3.00.

1.5 Hardware Installation

The floppy disk controller module and drive unit should be inspected upon receipt for broken, damaged, or missing parts as well as for damage to the printed circuit board. The packing materials should be saved in case reshipping is necessary.

1.5.1 Four-drive system installation

The following procedure must be performed to install the four diskette drive version of the EXORdisk III. This section is not applicable to EXORdisk II systems or to dual-drive EXORdisk III systems. This procedure must be performed before the floppy disk controller module is installed (next section). It should be noted that in the four-drive configuration, all diskette controller originated lines must be terminated in the last drive of the daisy chain. When facing the front of the disk drive units, drive zero is on the left and drive one is on the right of one unit, while drive two is on the left and drive three is on the right of the other unit. Before the following modifications are made, both dual-drive units are identical.

1. The housings from both dual-drive units must be removed.

2. In the dual-drive unit that is to contain drives zero and one, the Terminator Network (Motorola P/N 51NW9626A01) should be removed from the socket XA22 on printed circuit board (pcb) for drive zero. The drive one pcb socket XA22 should not have the Terminator Network installed.

3. JPR 11 should be installed in the jumper area of the pcb for drive zero.

4. JPR 9 should be installed in the jumper area of the pcb for drive one.

5. The housing should be replaced on this dual-drive unit and the drives marked as zero and one.

6. On the other dual-drive unit the Terminator Network should be installed in socket XA22 of the pcb for drive three. There should be no Terminator Network installed on socket XA22 of the pcb for drive two.

7. JPR 11 in the jumper area of the pcb for drive two should be removed (if installed). JPR 8 should be installed.

8. JPR 9 in the jumper are of the pcb for drive three should be removed (if installed). JPR 10 should be installed.

9. The 50-pin ribbon cable that connects to P1 of the Controller Interconnect Board must be disconnected and insulated against contact with conductive material.

10. The housing on this dual-drive unit should be replaced and the drives marked as two and three.

11. The 50-pin ribbon cable (Motorola P/N 30BW1824X01) should be installed between drives zero/one and two/three.

1.5.2 Floppy disk controller installation

To install the floppy disk controller module into the EXORciser, the following steps should be followed:

1. The PWR keyswitch on the EXORciser should be turned OFF. CAUTION: Inserting the floppy disk controller module while power is applied to the EXORciser system may result in damage to components of the module.

2. Any other card in the EXORciser that responds to addresses between hexadecimal $E800 through $EC07, inclusive, must be removed from the system or configured for a different address range.

3. The floppy disk controller module can then be inserted into any available card slot. It is desirable to keep all of the cards in the EXORciser close together; it is specifically recommended that dynamic memory boards be kept as close to the MPU board as possible. When properly installed, the component sides of all cards should be facing the left-hand side of the EXORciser chassis (as viewed from the front). The EXORciser motherboard connectors are offset and keyed to prevent backward installation of cards.

4. The interconnect cable should then be attached to both the drive unit and the diskette controller module. CAUTION: The pin index mark on the connector must match up with the index mark on the cable. Damage to the module will result if the cable is installed the wrong way.

5. Power can now be applied to both the drive unit and to the EXORciser -- the hardware is installed. The operator should get into the habit of turning on the power in the following sequence: system console, EXORciser, EXORdisk, and line printer. The power off sequence should be the reverse: line printer, EXORdisk, EXORciser, and system console. No diskettes should be in a drive while the drive's or the EXORciser's power is being turned on or off.

1.6 Software Installation

There is no software installation that need be performed. All MDOS software is included on the diskette that is shipped with each EXORdisk. This diskette contains the operating system and a set of commands that comprise MDOS. It may or may not contain any of the MDOS-supported software products such as editors or assemblers. These products are dependent on the mode of system purchase.

CHAPTER 2

2. GENERAL SYSTEM OPERATION

This chapter provides the user with the basic concepts that are necessary for the simplified and typical operation of MDOS. It contains descriptions and examples of the initialization procedures and of the basic forms of the most frequently used commands. These examples clearly illustrate how MDOS is used to edit a program, to assemble it, to convert it into a loadable module, to load it and execute it, as well as some other useful operations. The commands are presented in a sequence that is commonly followed in a software development environment.

2.1 System Initialization

To initialize the operating system, power must first be applied to the EXORciser and to the diskette drive unit. No diskette should be in the drive while power is being turned on or off on either the drive or the EXORciser. Once the power is on, the following steps must be followed:

1. EXbug must be initialized and configured for the proper speed of the system console. If power has just been turned on for the first time, EXbug initialization is automatically performed by the power-up interrupt service routine in EXbug. If power is already on and MDOS is to be re-initialized, then either the ABORT or RESTART pushbuttons on the EXORciser's front panel must be depressed to initialize EXbug. The prompt "EXBUG V.R" will be displayed by EXbug indicating it is waiting for operator input. "V" indicates the version and "R" the revision number of the EXbug monitor in the system.

2. An MDOS diskette (one shipped from Motorola or one that has been properly prepared by the user (see section 2.8.10)) must be placed in drive zero. The door on the drive unit must then be closed in order for the diskette to begin rotating. For the side-by-side drives, drive zero is on the left side, as seen from the front. For the EDOS-converted systems using the vertically stacked drives, drive zero is the top one.

   The diskette must be oriented properly before being inserted into the drive. When the diskette is inserted properly, the label is facing up, and the edge of the diskette with the long narrow slot in the protective covering is inserted first. The labelled edge will be the last edge to be covered up as the diskette is inserted into the drive.

3. Operators with EXbug 2 in their systems will skip this step. The EXbug 1 command "MAID" must be entered. An asterisk (*) prompt will be displayed once MAID has been activated.

4. The MAID command "E800;G" must be entered if the debug monitor is EXbug 1. For EXbug 2 monitors, the EXbug command "MDOS" must be entered. Either command will give control to the diskette controller at the specified address. The controller will initialize the drive electronics and then proceed to read the Bootblock into memory. Once the Bootblock has been loaded, control is transferred to it. The Bootblock will then attempt to load into memory the remainder of the resident operating system.

2.2 Sign-on Message

If no errors occur during the initialization process, MDOS will display the message:

```
MDOS VV.RR (M6800)
=

MDOS09 VV.RR (M6809)
=
```

meaning that MDOS has been successfully loaded from disk and initialized. The "VV" and "RR" indicate the version and revision numbers of the operating system, respectively. In addition, an equal sign (=) is displayed as a prompt indicating that MDOS is ready to accept commands from the operator. The equal sign prompt is subsequently displayed each time the MDOS command interpreter gets control. The sign-on message showing the version and revision numbers is only displayed when MDOS is reloaded from the diskette.

2.3 Initialization Error Messages

If for some reason the drive electronics are not properly initialized, or if the diskette in drive zero cannot be read properly to load the Bootblock or the resident operating system, then a two-character error message will be displayed and control returned to the EXbug monitor.

The following errors can be produced during initialization. All two-character messages begin with the letter "E".

Message	Probable Cause
E1	A cyclical redundancy check (CRC) error was detected while reading the resident operating system into memory.
E2	The diskette has the write protection tab punched out. During the initialization process, certain information is written onto the diskette. The diskette is not damaged and can still be used for a system diskette; however, the write protection tab must first be covered with a piece of opaque tape to allow writing on the diskette.
E3	The drive is not ready. The door is open or the diskette is not yet turning at the proper speed. If the diskette has been inserted into the drive with the wrong orientation, the "not ready" error will be also generated. If a double-sided diskette is used in the EXORdisk II drives, this error will also occur. Closing the door, waiting a little bit longer before entering the "E800;G" or "MDOS" command, or turning the diskette around so it is properly oriented should eliminate this error.
E4	A deleted data mark was detected while reading the resident operating system into memory.
E5	A timeout interrupt occurred. This indicates that a diskette controller function was not completed within the allotted time. This error can also occur if the ABORT pushbutton is

	depressed while a diskette transfer is in progress.
E6	The diskette controller has been presented with a cylinder-sector address that is invalid.
	This error indicates some type of a hardware problem. For example, the error can be caused by missing or overlapping memory, bad memory, or pending IRQs that cannot be serviced.
E7	A seek error occurred while trying to read the resident operating system into memory.
	Like E6 errors, this one indicates some type of a hardware problem.
E8	A data mark error was detected while trying to read the resident operating system into memory.
E9	A CRC error was found while reading the address mark that identifies sector locations on the diskette.

The diskette controller errors E1, E4, E8, and E9 indicate that the diskette cannot be used to load the operating system; however, a new operating system can be generated on that diskette, making it useful again. Chapter 10, DOSGEN command, and chapter 15, FORMAT command, describe ways in which damaged diskettes can be regenerated. Depending on the extent of the errors, the diskette may be used in drive one to recover any files that may be on it (section 2.8.9).

The diskette controller error E5 can occur for a variety of reasons. The most common reason, and the most fatal, is the destruction of the addressing information on the diskette. If the addressing information has been destroyed (verified by using DUMP command to examine areas of diskette), the FORMAT command may be used to rewrite the addressing; however, information on the damaged diskette cannot be recovered. Occasionally, after a system has just been unpacked, the read/write head may have been positioned past its normal restore point on cylinder zero. In this case, trying the event which caused the error three or more times may position the head to the proper place. If this fails, the head will have to be manually repositioned past cylinder zero; however, this problem rarely occurs. The E5 errors can also occur if a user-written program accesses drives 1-3 without using one of the system functions and without first restoring the read/write head on that drive.

Even after the resident operating system has been successfully read into memory, certain errors can occur in the subsequent initialization procedure. During initialization the resident operating system cannot access the error message processor since it has not been initialized. Messages similar in format to those generated by the diskette controller are displayed to indicate such errors. They differ from the diskette controller errors in that the second character of the two-character message is a non-numeric character. The following errors can occur during initialization, but only after the resident operating system has been read into memory.

Message	Probable cause
E?	This error indicates that the Retrieval Information Block (RIB) of the resident operating system file MDOS.SY is in error. The operating system cannot be loaded. The diskette probably is not an MDOS system diskette, or the system files have been moved from their original places. The REPAIR command (Chapter 22) can be used to identify which files are missing or if their places have been changed.
EM	This error indicates that there was insufficient memory to accommodate the resident portion of the operating system. The memory requirements described in section 1.1 should be reviewed. If the minimum requirements are satisfied, then the existing memory should be carefully examined for bad locations.
EI	The version and revision of MDOS already loaded into memory are not the same as those on diskette. This error usually occurs as the result of switching diskettes in drive zero without following the initialization procedure outlined in section 2.1. The error can also occur if the ID sector has been damaged.

The error can be avoided if the initialization procedure is followed correctly every time a new system diskette is inserted into drive zero.

ER	The addresses of the Retrieval Information Blocks of the MDOS overlays are not the same as those at the time of the last initialization. This error may occur for the same reasons as the "EI" error.
EU	An input/output system function returned an error during the initialization. Errors of this sort indicate a possible memory problem or the opening of the door to drive zero while the initialization is taking place.
EV	One of the system files is missing or cannot be loaded into memory. If a system file is missing, the diskette has been improperly generated or the file was intentionally deleted. If a file cannot be loaded, then the diskette should be regenerated. The diskette may be used in drive one to save any files that may be on it (section 2.8.9). This error may also occur if the door to drive zero is opened while initialization is in progress.

2.4 Operator Command Format

After the sign-on message is displayed, MDOS is ready to accept commands from the operator. The equal sign prompt (=) indicates that the command interpreter is awaiting input via the console. Generally, the equal sign prompt will be redisplayed after each command has finished its function. The operator-entered command line must always indicate which command is to be executed. In addition, the file names that may be required by the command must be specified. Some commands also allow various options that can alter the way in which their functions are performed. These options are also entered on the command line. Each command line must be terminated with a carriage return. The command line has the following format:

```
<name 1> <name 2>,<name 3>,....,<name n>;<options>
```

where each <name i> (i=1 to n) has the form of a complete

MDOS file name (see section 2.7.1). The name of the command to be executed is always <name 1>. The remaining names and the options may not be required, depending on the individual command. The following lines:

```
DIR EDIT.CM:1;E
FREE
MERGE FILE1:1,FILE2:0,FILE3:1,FILE1:1
```

are valid examples of MDOS command lines. Section 2.8 describes in a simplified form the basic format (i.e., the command's name, what file names must be specified, and what options are available) of the most frequently used commands. PART II gives a complete and detailed description of all MDOS commands. In addition, Appendix H contains a summary of the command line formats of all MDOS-Supported software products.

Most frequently a "space" is used to separate <name 1>, the command name, from the other names which are typically separated by "commas". The "semicolon" always separates the options from the rest of the command line. The "space" and "comma" are the recommended separators since they make the command line the most readable; however, any character that will not be mistaken for an MDOS file name character, a suffix delimiter, a logical unit number delimiter, or a device name delimiter (see section 2.7.1) can be used as a separator. The use of special characters, although permitted, is not recommended because the command line becomes very unreadable.

2.5 System Console

The system console is used as the communications device between the operator and the operating system. MDOS messages are displayed on the console printer or display mechanism. MDOS commands, as well as operator inputs prompted by the commands, are entered via the keyboard. All command line input and most input to the various commands requires upper case, alphabetic characters. Numeric and special characters, of course, are case independent. To allow corrections to be made to any typed line before the terminating carriage return is entered, several special keys on the keyboard can be used. In addition, two other special keys serve to prematurely abort a command in progress or to "freeze" the display of messages on the console.

2.5.1 Carriage return key

The CARRIAGE RETURN key is used to terminate any operator response to an MDOS input prompt. This is true for the command line as well as all other input that may be required from the operator by the various commands. The

CARRIAGE RETURN will automatically perform both carriage return and line feed functions.

2.5.2 Break key

The BREAK key is used as a controlled-abort function key. Most MDOS commands that take a long time to complete their function periodically check to see if the BREAK key has been depressed. If it has, the command will come to a premature, but controlled, termination point.

The BREAK key should be used, whenever possible, as an alternative to using the EXORciser's ABORT or RESTART pushbuttons. The controlled abort that is achieved with the BREAK key ensures that all system tables are intact. Since termination is delayed until all critical diskette accesses have been completed, no file space is lost nor is any system table destroyed. Such precautions cannot be guaranteed if the ABORT or RESTART pushbuttons are used, since the operator has no way of knowing whether or not diskette data transfers are in progress.

2.5.3 Control-W

Control-W is actually a combination of two keys being depressed simultaneously: the CONTROL or CTL key and the W key. This combination is used to halt the display of information on the system console or printer. All commands that respond to the BREAK key abort function will also be "haltable" with the CTL-W key. Most MDOS commands that display more than a few lines of information on the console will occasionally check to see if the CTL-W key has been depressed. If a CTL-W is detected, the command will suspend processing until any other key on the console keyboard is depressed (except, of course, another CTL-W). This feature is particularly useful to hold the display for viewing on systems that have a CRT. In addition, if output is being directed to the printer, the CTL-W can be used to suspend printing until the paper is realigned.

2.5.4 Control-X

Control-X is actually a combination of two keys being depressed simultaneously: the CONTROL or CTL key and the X key. This combination is used to cancel the input line that was just entered by the operator (before a carriage return is depressed). All system input from the console supports CTL-X. Any characters entered on the current input line thus far will be deleted and input can be resumed from the beginning of the line. A carriage return and line feed will be sent to the console, so that the operator has a positive feedback that the line was cancelled.

2.5.5 DEL or RUBOUT

The DEL or RUBOUT key serves as a backspace key during console input. If the operator detects an error in the current input line (before a carriage return is depressed), the DEL key will cause the preceding character to be removed from the input line. The character that is removed will be echoed back to the console so that the operator has a positive feedback that a character was backed out of the line.

2.5.6 Control-D

Control-D is actually a combination of two keys being depressed simultaneously: the CONTROL or CTL key and the D key. This combination allows the operator to re-display the current input line (before a terminating carriage return is depressed). If the input line has had several characters backed out (see DEL key above), the line is very unreadable. The CTL-D key can, therefore, be used to show a "clean" copy of the line for operator inspection. The newly displayed line will be shown on the line following the current input line. Operator input is not terminated with the CTL-D key. Any remaining input must still be supplied, as well as the terminating carriage return.

2.6 Common Error Messages

Many error messages are common to the MDOS commands. In order to be aware of the most common errors, their descriptions are included here. These common error messages will be recognizable to the operator since they are prefaced with a pair of asterisks (**) and a two-digit reference number. Each command may, in addition, have a set of specific error messages that will not be displayed by other commands. These specific error messages will not have the asterisks or two-digit reference number. Such messages are explained along with each command's detailed description in PART II. A summary of the standard error messages can be found in Chapter 28. The messages are listed there in order of their two-digit reference numbers.

WHAT?

> The first name entered on the command line was not the name of a file in the diskette's directory. Most often this error occurs as the result of a mistyped command name.

** 01 COMMAND SYNTAX ERROR

The syntax of the command line parameters could not be interpreted. Most often this error refers to undefined characters appearing in the options field.

** 02 NAME REQUIRED

The file name required by the command as a parameter was omitted from the command line.

** 03 <name> DOES NOT EXIST

The displayed file name was not found in the diskette's directory. The file name must exist prior to using the command. The <name> is displayed to show which name of the multiple names specified as parameters caused the error.

** 04 FILE NAME NOT FOUND

The file name entered on the command line as a parameter does not exist in the diskette's directory. The file name must exist prior to using the command. No file name is displayed, since only one parameter is required by the command.

** 05 <name> DUPLICATE FILE NAME

The displayed file name already exists in the diskette's directory. The file name must not exist prior to using the command. The <name> is displayed to show which name of the multiple names specified as parameters caused the error.

** 06 DUPLICATE FILE NAME

The file name entered on the command line as a parameter already exists in the diskette's directory. The file name must not exist prior to using the command. No file name is displayed, since only one parameter is required by the command.

** 07 OPTION CONFLICT

The specified options were not valid for the type of function that was to be performed by the command. Several of the options are mutually exclusive and cannot be specified at the same time.

** 11 DEVICE NOT READY

Most frequently this indicates that a command is trying to output to the printer while the printer is not ready.

** 12 INVALID TYPE OF OBJECT FILE

Most frequently this indicates that an attempt was made to load a program into memory whose file does not have the "loadable" memory-image format, e.g., a source file.

** 13 INVALID LOAD ADDRESS

An attempt was made to load a program into memory that: 1) loads outside of the range of contiguous memory established at initialization; 2) loads over the resident operating system; 3) loads below hexadecimal location $20; or 4) loads beyond hexadecimal location $FFFF.

** 25 INVALID FILE NAME

A file name was specified that contained a family indicator (*), that began with a device name indicator (#), or that did not begin with an alphabetic character.

** 41 INSUFFICIENT DISK SPACE

A command is trying to create a file or to write into a file. Upon trying to allocate more file space, insufficient room remains on the diskette to accommodate the space requirements.

**PROM I/O ERROR--STATUS=nn AT h DRIVE i-PSN j

An unrecoverable error occurred while trying to access the diskette. The error status "nn" is a value returned by the diskette controller. The errors are of the same type that cause the initialization process to give control to EXbug; however, instead of beginning with the letter "E", the status (nn) begins with the digit "3". The second digit of the status corresponds directly to the diskette controller error number discussed in section 2.3. The "E" has been replaced by the "3". Thus, status

31 is the same as E1
32 is the same as E2
.
.
.
39 is the same as E9.

A memory address (only meaningful for system diagnostics) is substituted for the letter "h"; the drive number is substituted for the letter "i"; and the physical sector number (PSN) at which the error occurred is substituted for the letter "j".

2.7 Diskette File Concepts

In MDOS, a diskette file is a set of related information that is recorded more or less contiguously on the diskette. The information can be actual machine instructions that comprise a command or user program. The information can also be textual data, object program data, or any of the forms described in Chapter 24. The following section describes how files are named, created, deleted, and protected.

2.7.1 File name specifications

An MDOS file name specification consists of three parts: a "file name", a "suffix", and a "logical unit number". File names can be from one to eight alphanumeric characters in length, the first of which must be alphabetic. The alphabetic characters must be upper case letters. Valid file names could look like the following:

```
DIR
ASM3870
BACKUP
SO
BLOKEDIT
Z
```

In most cases, all that need be specified when a file name specification is called for is the file name. The suffix and logical unit number are usually given appropriate default values by the various commands.

The suffix can be either one or two characters in length. Like file names, suffixes must begin with an upper case alphabetic character. The rest of the suffix must be alphanumeric. A suffix is used to explicitly refer to a particular entry in the directory. That is, there may be several entries with the same file name but with different suffixes. In such cases, a file name reference alone would be ambiguous. Thus, the suffix is used to differentiate between entries with the same file name. Usually, suffixes designate a particular format of the file. Thus, a source file could have the suffix "SA". Its assembled object version could have the same file name but with the suffix "LX", and its executable version could have the same file name but with the suffix "LO". MDOS commands usually supply an appropriate default suffix when dealing with specific files.

If both file name and suffix are specified, they must be separated by a period (.). The following are examples of valid file name specifications using both file name and suffix:

```
BLOKEDIT.CM
Z.SA
PROC1.CF
DOCUMENT.Y
```

Since each diskette is a complete file system in itself, with complete directory and system files, it is possible to have directory entries with the same file names and suffixes on separate diskettes. Thus, the logical unit number is required to uniquely specify a directory entry on a given drive. Logical unit numbers consist of a single decimal digit (0, 1, 2, or 3). In most cases, MDOS commands supply a default value for the logical unit number. If a particular drive must be identified, it must be entered by the operator as a part of the file name specification. Logical unit numbers follow either the file name or the suffix depending on whether one or both are specified. The logical unit number must be separated from the file name or from the suffix by a colon (:). The following are examples of valid file name specifications using logical unit numbers.

```
BLOKEDIT.CM:0
TEST.X:1
DIR:1
Z456.D3:3
ASM:2
```

2.7.1.1 Family names

Some commands allow the operator to specify a family of file names. Family indicators can occur in either the file name or the suffix. An asterisk (*) is used as a family indicator. The family indicator represents all or part of a file name or suffix. For example,

```
FILE.*
```

would be a file name specification that includes all directory entries with the file name "FILE   " but with any suffix on the default drive. Similarly,

```
PROG*.SA
```

is a file name specification that includes all directory entries with "PROG" as the first four characters of their file names, regardless of what the remaining characters are, and with suffix "SA" on the default drive. The asterisk cannot have characters following it. Thus, the following file name specifications are invalid:

```
*PROG.SA
PROGRAM.*B
```

Not all commands allow file name specifications to contain the family indicator. The individual command descriptions should be consulted to see where family indicators are acceptable.

2.7.1.2 Device specifications

Some commands allow the operator to enter a device specification in the command line instead of a file name specification. Device specifications consist of two parts: a "device name" and an optional "logical unit number". Device names are two characters long, both of which must be alphabetic. A pound sign (#) is used as a leading character to indicate that the subsequent two-character sequence is a device name. For example,

```
#LP
#CN
```

are valid device names used for the line printer and the console, respectively. A device specification may be entered with a logical unit number. Logical unit numbers must follow the device name and must be separated from it by a colon (:). The individual command descriptions should be consulted to see where device specifications are allowed.

2.7.2 File creation

MDOS files are never explicitly created by the operator. All commands that write to output files will create them automatically if they do not exist. Files will be created according to the file name specification given on the command line. That is, if explicit suffixes and logical unit numbers are specified, the file will be created on the indicated drive. Otherwise, the appropriate default values supplied by the command will be used to create the file. Existing files are unaffected by the creation of a new file.

2.7.3 File deletion

Unlike file creation, file deletion is controlled explicitly by the operator via the DEL command which is described later. No other command program will delete existing files on the diskette. Exceptions to this are commands that automatically create an intermediate work file to perform the command's function. These intermediate files are deleted by the command as an automatic clean-up process.

2.7.4 File protection

All MDOS files can be configured with delete protection, with write protection, or with no protection. Delete protection will prevent the operator from inadvertently deleting the file (the protection can be changed by the operator so that the file can be deleted). Write protection will prevent any command from writing to that file as well as preventing deletion of the file. Normally, files are unprotected, allowing both writing to or deletion of the file. The NAME command, described later, can be used to set or to change a file's protection.

2.8 Typical Command Usage Examples

The following sections give simple, but meaningful, descriptions and examples of the most frequently used MDOS commands in a typical software development environment. No attempt is made in these sections to cover all capabilities and options of the described commands. The detailed command descriptions in PART II serve that purpose. After reading this section, the operator should be able to go "on-line" with MDOS and be able to display the directory of a diskette, create a source program file, assemble it, and load it into memory for testing. The commands to delete a file, to change its name or protection, to copy it between diskettes or to tape are also described. New MDOS diskette generation is discussed in the last part of this section.

It is assumed in the subsequent discussion that the system has been properly installed and initialized. Thus, a system diskette with the MDOS commands resides in drive zero. Command program files have a suffix of "CM" which is supplied as a default to the first file name that is entered on the command line. The default logical unit number that is supplied is ":0". In the command examples that follow, it will be seen that both suffix and logical unit number are not specified for the command name.

The following notation will be used in the description of the command line formats as well as throughout the remainder of the manual:

Notation	Meaning
$nnnn	Hexadecimal number "nnnn".
<>	Syntactic elements are printed in lower case and are contained in angle brackets, e.g., <options>, <name>.
[]	Optional elements are contained in square brackets. If one of a series of elements may be selected, the available list of elements will be separated by the word "or", e.g., [<tag1> or <tag2>].
{}	A required element that must be selected from the set of elements will be contained in curly brackets. The elements will be separated by the word "or".

All elements that appear outside of angle brackets (<>) must be entered as is. Such elements are printed in capital letters (if words) or printed as the actual characters (if special characters). For example, the syntactical element [;<options>] requires the semicolon (;) to be typed whenever the <options> field is used.

2.8.1 DIR -- Directory display

The DIR command is used to display the contents of a diskette's directory. Either the entire directory or selective parts of it can be displayed. The format of the command line for the DIR command is:

```
DIR [<name>] [;<options>]
```

The file name specification <name> indicates what to display. The <options> specification indicates how to display it. If DIR is entered by itself on the command line, it will display on the system console the file names of all user-generated files on drive zero. If no user-generated files exist on drive zero, a message will be displayed indicating that no directory entries were found. This is normally the case when DIR is used without any options on the system diskettes that are shipped with the new system. To display the system and the user-generated files, the "S" option can be placed into the options field:

```
DIR ;S
```

If drive one's directory is to be displayed, then a ":1" must be typed in place of the file name specification:

```
DIR :1;S
```

To direct the output of the DIR command to the printer, only one other option letter need be specified -- "L". Thus,

```
DIR :1;LS
```

will produce a listing of drive one's complete directory on the printer. The "S" and "L" can be in any order, as long as they follow the semicolon.

The DIR command can also be used to see if a specific file name exists on a given drive. This is accomplished by entering a complete file name specification (i.e., name, suffix, and logical unit number). Thus,

```
DIR EDIT.CM:1
```

will perform a directory search for the indicated file name specification on drive one. If the directory entry exists, its file name and suffix will be displayed. Otherwise, a message indicating that no entries were found will be displayed. Directory searches for specific file names do not require the "S" option to distinguish between system files and user files. Chapter 9 contains a complete description of the DIR command's use.

2.8.2 EDIT -- Program editing

The EDIT command is used to create and/or to change user-written source program and data files on diskette. The EDIT command, although an MDOS-Supported product which may be purchased separately, is mentioned here since it is such an integral part of the software development environment. The EDIT command, if not included on the MDOS system diskette, must be copied from the diskette on which it was shipped (see section 2.8.9). Once the EDIT command resides on the system diskette, it is invoked with the following MDOS command line:

```
EDIT <name>
```

If the EDIT command is not copied to the system diskette, it can be invoked from the diskette in drive one with the following command line:

```
EDIT:1 <name>
```

The only parameter supplied on the command line is the name of the file that is to be edited. If the file does not exist, the EDIT command will create the file; if the file already exists, then it will be used. The suffix "SA", which is typically used for ASCII source files, is automatically supplied as a default if no suffix is entered on the command line. Thus, the user need only specify the name of the file to be edited. Upon completion of an edit, the file name will be unchanged. That is, a user need not be concerned about renaming his files between edits. A complete description of the EDIT command's format and usage is found in the manual accompanying the EDIT command diskette, "M6800 Co-Resident Editor Reference Manual".

2.8.3 ASM or RASM -- Program assembling

The ASM and RASM commands for MDOS and RASM09 command for MDOS09 (hereafter called the assemblers) are used to assemble the source program files created with the EDIT command. The assemblers translate these source programs into object programs. The assemblers, although both MDOS-Supported software products which may be purchased separately, are mentioned here since they are such an integral part of the software development environment. If not included on the MDOS system diskette, the assemblers must be copied from the diskette on which they were shipped (see section 2.8.9). Once the assemblers reside on the system diskette, they are invoked with the following MDOS command line:

```
{ASM or RASM or RASM09} <name> [;<options>]
```

If the assemblers are not copied to the system diskette in drive zero, they can be invoked from the diskette in drive one by using the following command line:

```
{ASM:1 or RASM:1 or RASM09:1} <name> [;<options>]
```

The only required parameter is the name of the file that is to be assembled. Normally, this would be the name of the file specified in the previous description of the EDIT command. The assemblers will automatically supply the default suffix for both the source file that is read (SA) and for the object file that is created (LX, assuming that the OPT REL or OPT ABS assembler directive was not used). Such an object file will be in the standard, EXbug-loadable format. Such files cannot, however, be loaded by MDOS (see section 2.8.5). The object file will have the same file name as <name>, but a different suffix will be assigned to it to differentiate it from the source file.

Normally, a listing of the assembled program is desired. The assemblers will not produce a source listing unless the option to do so is specified in the <options> field. Thus, the command line to assemble a source program file named TESTPROG with source listing output would appear as:

```
{ASM or RASM or RASM09} TESTPROG;L
```

As with the DIR command, the "L" option directs the printed output to the printer. If a printer is not available, or if the program is short, the source listing can be produced on the system console by using the following option:

```
{ASM or RASM or RASM09} TESTPROG;L=#CN
```

If errors are detected during the assembly process, they will be included on the source listing. If no source listing is being produced, errors will automatically be displayed on the console. Typically, the software development process involves several iterations of the editing and assembly processes before an error-free object file is produced. The assemblers, however, require that the object file does not exist prior to the assembly process. Therefore, if a duplicate file name error message is displayed, the object file already exists. It must first be deleted before the assembly process can continue. The next section describes the process of file deletion.

During the iterative process of editing/assembling to obtain an error-free program, the object file created by the assembler can be suppressed by specifying the option "-O" in the options field. The command line

```
{ASM or RASM RASM09} TESTPROG;L-O
```

for example, will assemble the source program as in the above examples creating the listing on the line printer; however, the object file will not be created. Thus, the deletion of the object file between repetitive assemblies is not required since it is never created.

The "M6800 Resident Assembler Reference Manual" or the "M6800/M6801/M6805/M6809 Macro Assembler Reference Manual" should be consulted for a complete description of the assemblers' function, usage, and command format.

2.8.4 DEL -- File deletion

The DEL command is used to delete file names from the directory. The removal of a file's name from the directory makes the file unaccessible to any other process. The file itself is effectively deleted. Thus, in the subsequent descriptions, the phrases "delete a file name" and "delete a file" are equivalent. The format of the command line for the DEL command is:

```
DEL <name>
```

which will cause the specified file to be deleted. If the object file from the assembly process example above is to be deleted, for instance, the following command line would be entered:

```
DEL TESTPROG.LX
```

It should be noted that the suffix is specified. Since the DEL command is a general purpose command, like the DIR command, no default value for the suffix is supplied. Only those commands that can validly make an assumption about the type of file they will be dealing with (e.g., EDIT, ASM, RASM) will supply default suffixes.

The DEL command will display a message indicating that the file name was deleted or that the file name was not found. Chapter 8 contains a complete description of the DEL command's other capabilities.

2.8.5 EXBIN -- Creating program load module

The EXBIN command is used to convert the object file from the assembly process (assumes no OPT REL or OPT ABS in source file) into a file whose contents can be loaded into memory for execution. MDOS can only load programs into memory that are in memory-image files. Thus, the EXBIN command must be invoked after an assembly process to create the loadable file. The format of the command line for the EXBIN command is:

```
EXBIN <name>
```

The <name> specified on the command line is the name of the EXbug-loadable object file created by the assembler. Only the file name need be specified. The default suffix "LX" is automatically supplied by the EXBIN command. A file in the memory-image format will be created by the EXBIN process that has the same file name as <name>, but has the suffix "LO" to differentiate its file type. The following command line

```
EXBIN TESTPROG
```

will convert the file TESTPROG.LX:0 to its memory-image equivalent TESTPROG.LO:0. Thus, the processes of editing, assembling, and object file conversion can all be performed on a file by only referring to its file name. The suffix will be automatically supplied. Normally, EXBIN will not display any messages. The next section will describe how to load a program from a file into memory after it has been converted into the proper format. Chapter 14 contains the complete description of the EXBIN command.

2.8.6 LOAD -- Program loading/execution

The LOAD command is used to load programs from a memory-image file on the diskette into memory. After the program has been loaded, the debug monitor can be given control (for testing the program), or the program can be given control directly (for execution). The format of the command line for program loading is:

```
LOAD <name> [;<options>]
```

The name of the file whose contents are to be loaded is given as <name>. The default suffix "LO" is automatically supplied by the LOAD command. Thus, in normal software development, only a file's original source program name is required to take a user through the four processes of editing, assembling, object file conversion, and program loading.

The <options> field of the LOAD command line is used to specify whether the debug monitor or the loaded program is to be given control, and whether or not the program overlays the resident operating system. If the file TESTPROG from the previous examples was origined to the hexadecimal memory address $100, the following command line:

```
LOAD TESTPROG;V
```

would be used to load the program. The "V" option is used to specify that the program to be loaded will overlay the resident operating system. If the "V" option were left off the command line, an error message would be displayed. The absence of the "G" option letter means that the debug monitor will be given control after the program is loaded. So, the above example would be used to load TESTPROG into memory for testing.

If, on the other hand, the program TESTPROG has already been tested and works, the command line:

```
LOAD TESTPROG;VG
```

would be used to load and execute the program. No operator intervention is required to specify the starting execution address. This is only true if the starting execution address has been specified on the END statement of the source program during the assembly process.

Typically, most user-written programs that have been developed prior to receiving the MDOS system would be loaded and tested in this fashion. Programs that are developed with

MDOS as a basis (i.e., programs that use the resident system functions) are loaded without the "V" option. Chapter 18 describes the details of the LOAD command and should be consulted if more information is required.

CAUTION: AFTER THE DEBUG MONITOR HAS BEEN ENTERED VIA THE LOAD COMMAND, MDOS MUST NOT BE INITIALIZED VIA "E800;G" OR "MDOS" UNTIL EITHER THE ABORT OR RESTART PUSHBUTTON HAS BEEN DEPRESSED.

2.8.7 NAME -- File name changing

The NAME command allows file names and/or suffixes to be changed from their originally assigned values. Often, as a program is developed, its author decides that a file name other than the original one would be more appropriate and descriptive. The format of the command line for changing a file's name is:

```
NAME <name 1>,<name 2>
```

This command line requires the operator to enter two names. The first name, <name 1>, specifies the current or original name of the file. The default suffix "SA" is supplied automatically if none is given by the operator. The second name, <name 2>, indicates the new name that is to be assigned to the file now known by <name 1>. Thus, if the file from the above examples, TESTPROG, were to be given a more descriptive name, such as BLAKJACK, the following command would be used:

```
NAME TESTPROG,BLAKJACK
```

In this case, only the file name of the source file would be changed. Other files with the name TESTPROG but with suffixes other than "SA" would remain unaffected. The contents of the file that has its name changed are also unaffected -- only the name in the directory is changed.

2.8.8 NAME -- File protection changing

The NAME command is also used to change the protection attributes of a file. The command line format for changing a file's protection is:

```
NAME <name>;<options>
```

The <name> entry is required to identify the file whose attributes are to be changed. The <options> field contains the letters D, W, or X to indicate how the protection attributes are to be changed. The letters take on the following meanings:

D -- Set delete protection
W -- Set write protection
X -- Set no protection (remove existing protection)

Thus, if the file TESTPROG (source file) is to be protected against deletion, the following command line would be used:

```
NAME TESTPROG;D
```

If the memory-image file that was produced from the source of TESTPROG were to be write protected and delete protected, the following command line would be used:

```
NAME TESTPROG.LO;DW
```

The protection on this file could later be removed with the command line:

```
NAME TESTPROG.LO;X
```

Chapter 20 describes in more detail the other features of the NAME command.

2.8.9 COPY -- File copying

The COPY command is used to make a duplicate copy of a file on a single diskette, to move a file between two different diskettes, or to move a file between the console reader/punch (record) device and a diskette.

To make a duplicate copy of a file on the same diskette, the following command line is used:

```
COPY <name 1>,<name 2>
```

where <name 1> specifies the current name of an existing file, and <name 2> specifies the name of the duplicate copy. The default suffix "SA" and the default logical unit number zero are supplied for <name 1> if those parts of the file name specification are omitted. Normally, the destination file, <name 2>, does not exist. The COPY command, however, will alert the operator if <name 2> does exist, and ask him if that file should be overwritten. If <name 2> has a different logical unit number than the original file, the file will be duplicated on the specified drive. If the TESTPROG source file from the above examples is to be saved in a file called TEMP, the following command line would be used:

```
COPY TESTPROG,TEMP
```

The file TEMP will be created on the same drive as

TESTPROG, namely, drive zero. To copy TESTPROG to drive one, one need only specify the logical unit number (:1) after the second name.

The COPY command should be used to move the EDIT, ASM, and RASM commands from their separate diskettes onto the system diskette in drive zero. Since the names of the EDIT, ASM, and RASM commands are to be kept the same, the second name can be omitted completely. All that needs to be specified is the logical unit number. Thus,

```
COPY EDIT.CM:1,:0
COPY ASM.CM:1,:0
COPY RASM.CM:1,:0
```

would be the commands that are entered if the diskette in drive one contained these files. The suffixes "CM" are explicitly specified since neither the EDIT, ASM, or RASM commands are source programs.

A similar procedure would be followed to copy any files from a diskette in any drive to the system diskette in drive zero. If a diskette has been damaged or cannot be used to initialize MDOS, it may be placed in another drive in attempt to save any files that may be on it. The COPY command should be used to save files in this manner. If diskette controller errors occur during such a save process, the files cannot be recovered.

If a user has existing files on paper tape or cassette that are written in one of the standard record formats (i.e., records that end with a carriage return, line feed, null sequence -- see section 24.3) and which can be read via the console reader, the following command line can be used to transfer those files to diskette:

```
COPY #CR,<name 2>;N
```

where <name 2> is the name of the diskette file into which the tape file is to be written. The first parameter, #CR, specifies the console reader device, and the "N" option indicates that there is no MDOS header record on the tape file.

The above process can be changed slightly so that a file on diskette can be written to the console punch (record) device. For example,

```
COPY <name 1>,#CP;N
```

will transfer the file named by <name 1> to the console punch device, #CP, without the MDOS header information ("N" option). Chapter 7 describes in more detail the other features of the COPY command.

2.8.10 BACKUP -- MDOS diskette creation

New diskettes, or diskettes never before used on an MDOS system, must first be prepared for use with MDOS. The quickest way to generate a new MDOS diskette is to use the BACKUP command. Usually, a copy is retained of the original system diskette that was shipped with the EXORdisk II or III. This diskette should be used to generate subsequent MDOS diskettes. It is recommended that the original diskette not be used for development purposes. It should serve only as the master copy from which all other diskettes are generated.

A blank or scratch diskette should be placed into drive one. The master system diskette should be resident in drive zero. The following command line will then cause a complete copy of the master diskette to be created:

```
BACKUP ;U
```

The "U" option specifies that the entire surface of the diskette in drive zero is to be read and copied to the diskette in drive one. This process ensures that all sectors on the new diskette can be written to. Once the BACKUP command has been invoked in this way, it will display the following message:

```
BACKUP FROM DRIVE 0 TO 1?
```

to which the operator should respond with a "Y". Any other response will terminate the BACKUP process, leaving the diskette in drive one intact. The "Y" response will cause the diskette copy to take place.

As an added precaution, the two diskettes should be compared against each other after the BACKUP command has completed. This diskette verification is invoked with the following command line:

```
BACKUP ;UV
```

If any messages are displayed during the verification process, the diskette in drive one should not be used as a system diskette.

Chapter 3 describes the BACKUP command in detail. Chapter 10 describes an alternative method of generating new system diskettes.

2.9 Other Available Commands

Several other powerful commands are included with each MDOS diskette. These commands are not needed initially in becoming familiar with the system; however, they do provide helpful and necessary tools for the advanced software developer. A brief description of these commands is given here to shed some light on their utility.

2.9.1 BACKUP -- Diskette copying

The BACKUP command allows making copies of entire MDOS diskettes. Options exist for making complete copies, for file reorganization to consolidate fragmented files and available diskette space, for appending families of files from one diskette to another, and for diskette comparisons. Chapter 3 contains the complete description of the BACKUP command.

2.9.2 EMCOPY -- EDOS file conversion

The EMCOPY command allows files from a user's EDOS 2 system diskette to be copied to and catalogued on an MDOS diskette. Options exist for copying the entire diskette, selected files, or single files. Chapter 13 contains the complete description of the EMCOPY command.

2.9.3 BLOKEDIT -- File rearrangement

The BLOKEDIT command allows lines of text from one or more ASCII files to be selectively copied into a new file. This command can be useful in generating new program source files by copying routines from existing source files, or in rearranging existing files by copying their lines into a new sequence. Chapter 5 contains the complete description of the BLOKEDIT command.

2.9.4 LIST -- File display

The LIST command is used to print any ASCII file on either the system console or the printer. Options exist for numbering lines, specifying page formats, printing headings, and indicating starting and ending points. In addition, files can be accessed by their logical sector numbers for rapid access to any portion of a file. Chapter 17 contains the complete description of the LIST command.

2.9.5 MERGE -- File concatenation

The MERGE command allows one or more files to be concatenated into a new file. This command is useful in combining several smaller program modules or in building relocatable libraries to be used in conjunction with the

M6800 Linking Loader. Chapter 19 contains the complete description of the MERGE command.

2.9.6 BINEX -- EXbug-loadable file creation

The BINEX command allows memory-image files to be converted into an EXbug-loadable format for copying to tape. This command performs the inverse operation of the EXBIN command. BINEX is useful in the development of non-diskette-resident software with MDOS, since the object code can be written to tape after it has been tested. Chapter 4 contains the complete description of the BINEX command.

2.9.7 FREE -- Available file space display

The FREE command displays how many unallocated sectors and how many empty directory entries are on a diskette. Chapter 16 contains the complete description of the FREE command.

2.9.8 ECHO -- Echo console I/O on printer

The ECHO command can be used on an EXORciser II system to cause all input/output directed to the system console to also be printed on the line printer. Chapter 12 contains the complete description of the ECHO command.

2.9.9 PATCH -- Executable program file patching

The PATCH command allows changes to be made to memory-image files. An object file can be "fixed" due to minor bugs or assembly errors without having to re-edit and re-assemble its corresponding source file. The "fixes" can be entered using M6800 assembly language mnemonics or the equivalent hexadecimal operation codes. Chapter 21 contains the complete description of the PATCH command.

2.9.10 CHAIN -- MDOS command chaining

The CHAIN command allows predefined procedures to be automatically executed. A procedure consists of any sequence of MDOS command lines that have been put into a diskette file. Instead of obtaining successive command lines from the console, CHAIN will fetch commands from a file. This feature allows complicated and lengthy operations to be defined once, and then invoked any number of times, requiring no operator intervention. The additional capabilities of conditional directives to the CHAIN command at both compilation and execution time, and the capability of string substitution, permits an almost unlimited number of applications to be handled by a CHAIN file. Chapter 6 contains the complete description of the CHAIN command.

2.9.11 REPAIR -- System table checking

The REPAIR command allows the user to check and repair a malfunctioning or a non-functioning MDOS diskette. Errors in the system tables can be found, identified, and corrected with this command. Since MDOS performance is directly related to the correctness of these system tables, the REPAIR command is a useful diagnostic utility. Chapter 22 contains the complete description of the REPAIR command.

2.9.12 DUMP -- Diskette sector display

The DUMP command allows the user to examine the entire contents of any physical sector on the diskette. The sector can be displayed on either the system console or the printer. The display contains both the hexadecimal and the ASCII equivalent of every byte in the sector. The DUMP command allows opening of files so that they can be examined using logical sector numbers. Sectors can also be moved into a temporary buffer where changes can be applied before they are written back to diskette. Chapter 11 contains the complete description of the DUMP command.

2.9.13 FORMAT -- Diskette reformatting

The FORMAT command attempts to rewrite the sector addressing information on damaged diskettes. The command can be used to reformat either single-sided or double-sided diskettes; however, double-sided diskettes must be formatted with this command before they can be used with MDOS. Single-sided diskettes usually come pre-formatted in a compatible format. The FORMAT command will only work on systems that are operating at one of the standard clock frequencies of 1 MHz, 1.5 MHz, or 2 MHz. Chapter 15 contains the complete description of the FORMAT command.

2.9.14 DOSGEN -- MDOS diskette generation

The DOSGEN command allows specialized MDOS diskettes to be prepared. Diskettes that have bad sectors can have those sectors locked out so that the diskette can be used in an MDOS environment. DOSGEN will also create all system tables and files on the generated diskette. The DOSGEN command can be used to generate system diskettes on either single-sided or on appropriately formatted double-sided diskettes.

Chapter 10 contains the complete description of the DOSGEN command.

2.9.15 ROLLOUT -- Memory rollout to diskette

The ROLLOUT command is used for writing the contents of memory to diskette. The ROLLOUT command supports the dual-memory maps of EXORciser II as well as the single memory map of EXORciser I. Options exist for writing memory directly into a diskette file or for writing to a scratch diskette. Chapter 23 contains the complete description of the ROLLOUT command.

2.10 MDOS-Supported Software Products

Although the preceding list of commands provides the user with many powerful tools for software development, there are many other Motorola products which are capable of running in an MDOS environment, even though they were developed independently. These products are called MDOS-Supported software products. No attempt will be made in this User's Guide to comprehensively describe any MDOS-Supported software product. Appendix H contains a list (complete at time of publication) of all products that can be invoked from an MDOS diskette as a command. Each description will contain the additional hardware requirements, if any, the command line formats, and a brief discussion of the product's capabilities. MDOS-Supported software products will be received on separate diskettes. Section 2.8.9 describes how such products can be copied onto the system diskette.

2.11 Paper Alignment

All MDOS commands that output to the line printer will return the paper to its original position upon termination. Thus, if the paper is correctly aligned at the time MDOS is initialized, then the paper will never have to be aligned again. The paper should be placed so that the print line is positioned three lines before a perforation (assuming fan-fold forms). MDOS commands use the standard format of 66 lines/page.

CHAPTER 3

3. BACKUP COMMAND

The BACKUP command allows making copies of entire MDOS diskettes. Options exist for making complete copies, for file reorganization to consolidate fragmented files and available space, for appending families of files from one diskette to another, and for diskette comparisons. The BACKUP command will only copy MDOS-generated diskettes. The BACKUP command may also be used for copying single-sided diskettes onto double-sided diskettes.

3.1 Use

The BACKUP command is invoked with the following command line:

```
BACKUP [[:<s-unit>,]:<d-unit>] [;<options>]
```

where <s-unit> is the source logical unit number, <d-unit> is the destination logical unit number, and <options> can be one or more of the option letters described below.

If neither <s-unit> nor <d-unit> is specified on the command line, then zero will be used as the source unit and one will be used as the destination unit. Specifying only a single logical unit number on the command line will cause zero to be the source unit and the specified logical unit to be the destination unit. Both <s-unit> and <d-unit> must be valid logical unit numbers (0-3), <d-unit> cannot be zero, and the two numbers cannot be the same.

BACKUP will always copy from the source unit to the destination unit (unless diskette comparisons are specified).

If the command line is valid, the message:

```
BACKUP FROM DRIVE <s-unit> TO <d-unit>?
```

or

```
APPEND FROM DRIVE <s-unit> TO <d-unit>?
```

will be displayed where <s-unit> is the source unit number and <d-unit> is the destination unit number. In either case, a response of "Y" is required if BACKUP is to continue. Any other response will return control to MDOS. Further BACKUP action depends on the specified options. The options are divided into "Main Options" and "Other Options". Main

Options are mutually exclusive. That is, only one Main Option can be specified on the command line at a time. The Other Options can be included with the Main Options as described in section 3.6.

Main Options	Function
none	Copy all allocated space to destination diskette.
R	Reorganize diskette so that files are defragmented and free space is consolidated on destination diskette.
A	Append (copy) selective files to destination diskette.
V	Verify (compare) source and destination diskettes.

Other Options	Function
C	Continue if read/write errors occur.
D	Continue if deleted data mark errors occur.
I	Change ID sector during copy.
L	Use line printer for bulk of message printing.
N	Suppress printing of file names being copied.
S	Suppress printing of byte offsets during comparisons.
U	Include unallocated space in copy/verify process.
Y	If duplicate file name exists, delete old, copy new.
Z	If duplicate file name exists, suppress copy.

3.2 Diskette Copying

If no Main Options are specified, then the default BACKUP process will produce a physical sector copy of the source diskette on the destination diskette. Only the allocated space from the source diskette will be copied. The allocated space includes all file space and all areas locked out in the Lockout Cluster Allocation Table (see Chapter 24). Thus, only MDOS-generated diskettes can be copied using the BACKUP command, since other diskettes will not have an allocation table.

Since only the allocated space is copied, the minimum amount of disk space is copied, and the BACKUP process is completed in the minimum amount of time. Sometimes, however, it is desirable to obtain a complete copy, and not just a copy of the allocated space. In such cases, the "U" option can be used to force the copying of unallocated space as well as the allocated space.

A typical BACKUP process dialogue would look like the following:

```
=BACKUP
BACKUP FROM DRIVE 0 TO 1?
Y
=
```

and would produce a copy on the destination diskette of the source diskette's allocated space.

If an EXORdisk III system is being used, then the destination diskette cannot be a single-sided diskette if the source diskette is a double-sided diskette. The error message:

```
INVALID TO COPY/VERIFY FROM DOUBLE TO SINGLE SIDED
```

will be displayed and control returned to MDOS to indicate this condition. The opposite, however, is allowed. That is, a single-sided diskette can be in the source drive with a double-sided diskette in the destination drive.

3.3 File Reorganization

After an MDOS diskette has been used for a while, the file structure may become fragmented and new files can become scattered. The longer a diskette is used in a development environment, the more the total system performance may be degraded due to increased access time. File reorganization is supplied by the BACKUP command and constitutes one way to restructure MDOS diskettes, thereby improving the system's efficiency.

File reorganization improves system efficiency by:

1. Consolidating file segments,
2. Packing files more closely together,
3. Clustering related files together,
4. Operator selection to only copy desired files,
5. Reducing marginal diskette errors by rewriting files,
6. Consolidating directory space.

File reorganization is specified with the Main Option "R" on the BACKUP command line. Thus,

```
BACKUP :<s-unit>,:<d-unit>;R
```

would invoke the BACKUP command to reorganize the files on the source diskette in drive <s-unit> during the copy to the destination diskette in drive <d-unit>. The source diskette must be an MDOS diskette. It is unaffected by the reorganization. The message

```
BACKUP FROM DRIVE <s-unit> TO <d-unit>?
```

is displayed before any copying takes place. Unlike the complete copy process which will proceed immediately after the "Y" response is given by the operator, the reorganization process will perform the following initialization procedure: First the ID sector is copied (and optionally modified if the "I" option was specified). Second, the Lockout Cluster Allocation Table (LCAT) and the Cluster Allocation Table (CAT) are initialized (user locked out sectors are not copied during the reorganization process). Third, the directory sectors on the destination disk are zeroed. Fourth, the Bootblock is copied. Fifth, all of the file names from the source diskette's directory are read. They are then sorted into alphabetical order, first by suffix, then by file name. After the sorting has been completed the following message will be displayed:

```
ENTER FILE COPY SELECTION COMMANDS:
SAVE (S), DELETE (D), PRINT (P), QUIT (Q), NO MORE (CR)
S, D, P, Q, (CR):
```

indicating that the operator must enter file selection commands to specify which files from the source diskette are to be copied to the destination diskette. The first line of the message indicates that BACKUP has reached the file selection stage. The second line contains the function of each file selection command as well as the letter that must be used to issue that command. The third line is used as a prompt for the current and subsequent file selection commands.

Command	Letter	Function
SAVE	S	Include a certain file name or family of file names from the sorted directory in the set of files to be copied to the destination diskette.
DELETE	D	Exclude a certain file name or family of file names from the sorted directory from the set of files to be copied to the destination diskette.
PRINT	P	Display the set of file names from the sorted directory that are eligible to be copied to the destination diskette.
QUIT	Q	Terminate the BACKUP command and return to MDOS. No copying will take place; however, the destination diskette has been affected due to the reorganization option as explained above.
NO MORE	(CR)	Entered as a carriage return only. No more commands will be entered. The files to be copied have been selected. If no file selection commands were issued, all files in the sorted directory will be copied. Begin the copy process.

Both the SAVE and DELETE commands require file names to be specified as parameters. The format of the SAVE and DELETE commands are the same, except, of course, for the command letter:

```
{D or S} <name 1>[,<name 2>,...,<name n>]
```

The file names specified can contain the family indicator. The default suffix "SA" will be supplied if none is explicitly entered. For example, the SAVE command:

```
S *.CM,EQU,IOCB.*
```

will cause the family of files having the suffix "CM", the file EQU.SA, and the family of files having the name IOCB to be flagged as saved. The DELETE command:

```
D A*.CM,NOL,TEST.L*
```

will cause the family of files beginning with the letter "A" and having a suffix of "CM", the file NOL.SA, and the family of files named TEST with suffixes beginning with the letter "L" to be flagged as deleted.

After a SAVE or DELETE command has been entered, each file name of the sorted directory which has not already been marked as "saved" or "deleted" and which matches one of the <name i> (i=1 to n) will be marked as "saved" or "deleted". After all the file names from the SAVE or DELETE command line have been processed, a new prompt:

```
S, D, P, Q, (CR):
```

will be displayed. The operator can then enter further SAVE or DELETE commands as well as any of the other valid commands of the BACKUP file selection process.

Once a command other than SAVE or DELETE is entered one of two things happens to the sorted directory. If at least one SAVE command has been processed without error, then all file names in the sorted directory not marked as "saved" will be marked as "deleted". On the other hand, if no prior SAVE commands were used, then all file names not marked as "deleted" will be eligible for copying (marked as "saved").

The QUIT command can be entered at any time in response to the file selection command prompt. QUIT will cause the BACKUP process to be terminated and control returned to MDOS. The file selection commands entered thus far will have had no effect on the destination diskette; however, due to the reorganization option, the destination diskette will have had its basic system tables initialized as described above.

The NO MORE command, entered as a carriage return only, indicates that no more file selection commands will be given by the operator. If no file selection commands have been entered prior to the NO MORE command, then all file names in the sorted directory will be eligible for copying to the destination diskette. The copy process will begin.

The PRINT command will cause all names from the sorted directory which have not yet been flagged as "deleted" to be printed. The PRINT command also makes it impossible to enter further SAVE, DELETE, or QUIT commands. The PRINT command has its own sub-command structure that allows deletion of file names from the sorted directory. Along with each file name and suffix a two-digit, hexadecimal number that indicates the position of the file name within the sorted directory is displayed. Thus, the output from the PRINT command could look like:

```
00 BACKUP  .CM
01 BINEX   .CM
02 BLOKEDIT.CM
03 CHAIN   .CM
04 COPY    .CM
05 DEL     .CM
06 DIR     .CM
1D RLOAD   .CM
1E FORLB   .RO
1F EQU     .SA
20 IOCB    .SA
```

The range of numbers $07-1C, inclusive, is missing, indicating that they have been excluded from the sorted directory via prior SAVE and/or DELETE commands. If PRINT were the first command to be entered, then all file names in the sorted directory would be seen, and the range of numbers would be without gaps.

After the PRINT command has displayed all of the file names, a new prompt will be issued:

```
DELETE FILE NOS.:
```

to which the operator can respond with a number, a series of numbers or ranges of numbers separated by commas, a range of numbers, or a single carriage return. The numbers must be from the set of those displayed in front of the file names. These numbers are used to indicate which files are to be excluded from the sorted directory before files are copied to the destination diskette. For example, the following entry:

```
01-03,1E,06
```

would cause the file names with numbers

01, 02, 03, 06, and 1E

to be removed from the sorted directory before the file copy process begins. Another "DELETE FILE NOS." prompt will be displayed if a number was entered in response to a previous prompt. Thus, as many file names as desired can be excluded from the sorted directory. A carriage return response to the prompt has the same effect as the NO MORE command described above; i.e., it will end further command processing and cause the file copy process to begin.

After the files to be copied have been selected, the message

```
COPYING MDOS    .SY
```

will be displayed. This message will in turn be followed by similar messages for each of the eight remaining system files that must be copied to every diskette. The MDOS family of system files are not shown in the sorted directory since they must be copied. These system files are copied first so that they will be assured of residing in specific physical locations required by the MDOS initialization process. After the MDOS system files have been copied, the message:

```
STARTING TO COPY FILES
```

is displayed, followed by messages of the form:

```
COPYING <name i>
```

as each file from the selected files list is copied to the destination diskette.

Using the above example of the sorted directory and the file names deleted from it, the file copy messages would look like:

```
COPYING MDOS    .SY
COPYING MDOSOV0 .SY
COPYING MDOSOV1 .SY
COPYING MDOSOV2 .SY
COPYING MDOSOV3 .SY
COPYING MDOSOV4 .SY
COPYING MDOSOV5 .SY
COPYING MDOSOV6 .SY
COPYING MDOSER  .SY
STARTING TO COPY FILES
COPYING BACKUP  .CM
COPYING COPY    .CM
COPYING DEL     .CM
COPYING RLOAD   .CM
COPYING EQU     .SA
COPYING IOCB    .SA
=
```

After all eligible files from the sorted directory have been copied, BACKUP will return control to MDOS. The destination diskette will contain all of the selected files packed together as closely as possible, leaving as much free space as possible.

3.4 File Appending

The file append process allows selected single files or families of files to be copied from the source diskette to the destination diskette. The file append feature of the BACKUP command is similar to the reorganization feature except that the destination diskette is not initialized with new system tables or system files. Only the file selection and the file copying from the source diskette are performed.

The diskette in the destination drive is assumed to be a valid MDOS diskette. The file append process is invoked by using the Main Option "A" on the BACKUP command line:

```
BACKUP :<s-unit>,:<d-unit>;A
```

Instead of the "BACKUP FROM DRIVE <s-unit> TO <d-unit>?" message normally displayed by BACKUP, the message:

```
APPEND FROM DRIVE <s-unit> TO <d-unit>?
```

is shown. The operator must respond with a "Y" if the file append process is to continue. Like the file reorganization process, the file append process allows the operator to select which files are to be copied. The messages for file selection and the commands to the file selection process are explained in section 3.3, File Reorganization, and will not be discussed again here. After all files have been selected, they will be copied similar to the process described in section 3.3; however, the MDOS family of system files is not copied.

Since the destination diskette already contains entries in its directory, a possibility of file name duplication exists. In the event that one of the selected file names from the sorted directory duplicates a file name in the destination directory, the following message will be displayed:

```
<name> - DUPLICATION: IS IT TO BE COPIED?
```

The operator must respond with either an "N" or "Y". The "N" response will prevent the file from being copied to the destination diskette. The "Y" response will cause the prompt:

```
NEW NAME:
```

to be shown, to which the operator can respond with the new name that is to be assigned. If a valid file name and suffix are entered, they will be used as the name of the destination file. The default suffix "SA" will be supplied if none is explicitly entered. If only a carriage return is given as a response to the prompt, then the file on the destination diskette will be deleted (if it is unprotected) before the file from the source diskette is copied (which will retain its original name, in this case). If the destination diskette's duplicate file cannot be deleted, the message

```
CANNOT DELETE DUPLICATE NAME
```

will be displayed and the BACKUP command will be terminated.

The "Y" and "Z" options can be used in conjunction with the "A" option to indicate an automatic procedure in the event of file name duplication. The "Y" option will automatically cause an attempt to be made to delete the file on the destination diskette before the copy takes place. If the "Y" option is in effect, the file name duplication message from above takes on the following form:

```
<name> - DUPLICATION:  IS COPYING
```

to indicate that a "Y" was given as an automatic response to the "IS IT TO BE COPIED?" portion of the message. The "Z" option will cause the file name duplication message to take on the form:

```
<name> - DUPLICATION:  IS NOT COPIED
```

to indicate that an "N" was given as an automatic response to the "IS IT TO BE COPIED?" portion of the message.

The file append process causes space to be allocated on the destination diskette in contiguous blocks. If insufficient contiguous space should remain on the destination diskette for a given file, the file will not be copied. The error message

```
OBJECT FILE CREATION COPY ERROR
```

will be displayed and the BACKUP command will be terminated. The destination diskette may have sufficient space to accommodate the file; however, if the space is not contiguous, the above error occurs. To copy the file, the destination diskette should be run through the file reorganization process described in section 3.3, or the file must be copied via the COPY command (Chapter 7). After the last file has been copied to the destination diskette, control will be returned to MDOS.

3.5 Diskette Verification

The Main Option "V" invokes the verify process of the BACKUP command. The verify process allows a physical sector comparison to be made between the diskettes in the source and destination drives. The following command line, without the presence of other options, will cause the verify process to compare the diskettes' physical sectors based on the source diskette's allocation table:

```
BACKUP :<s-unit>,:<d-unit>;V
```

If any bytes in any sectors fail to compare, a sector message and a list of all offsets within the sector that did not compare is printed:

```
SECTOR nnnn
  OFFSET ii DR<s-unit>-jj DR<d-unit>-kk
```

where "ii" is the hexadecimal offset into physical sector "nnnn", "jj" is the hexadecimal contents of the sector's byte on the source diskette, and "kk" is the hexadecimal contents of the respective sector's byte on the destination diskette. If all sectors compare, no messages are displayed. After the verification has completed, control is returned to MDOS.

If an EXORdisk III system is being used, the destination diskette cannot be a single-sided diskette if the source diskette is a double-sided diskette. In such cases the message

```
INVALID TO COPY/VERIFY FROM DOUBLE TO SINGLE SIDED
```

will be displayed and control returned to MDOS. The opposite, however, is allowed; that is, a single-sided diskette can be verified against a double-sided diskette.

3.6 Other Options

The Other Options described briefly in section 3.1 cannot be used indiscriminately with any of the Main Options. This section serves to fully explain the use of each Other Option.

Other Option	Valid with Main Option	Function
C	any	The "C" option will cause the copy or verify process to continue even if a retryable read/write error occurred which could not be corrected. The retryable errors include CRC, seek, data mark, and address mark CRC errors. The "C" option will not cause read/write errors on Retrieval Information Blocks to be ignored.
D	any	The "D" option will cause the copy or verify process to continue even if a deleted data mark error is detected. This option allows the verification of diskettes that have had bad sectors locked out during the DOSGEN or REPAIR process (such sectors are flagged with a deleted data mark). The "D" option permits a user to copy the maximum amount of data from a bad source diskette to a good destination diskette.

Other Option	Valid with Main Option	Function
I	none, R	The "I" option indicates that the diskette's ID sector is to be modified by prompting the operator. The "I" option will cause the following prompt messages to be displayed. The operator can enter new information if that field of the ID sector is to be changed. If the field is to remain the same as on the source diskette, then only a carriage return need be entered.
L	any	The "L" option causes the output from the copy process or from the verification process to be directed to the line printer instead of the system console.
N	R, A	The "N" option will suppress the printing of the file names as they are being copied to the destination diskette. This option will not suppress the printing of error messages.
S	V	The "S" option will suppress the printing of the sector offset messages if sectors do not compare.

Prompt	Operator Response
DISK NAME:	Maximum of eight characters for diskette ID. Format is similar to that of a file name.
DATE(MMDDYY):	Six-digit numeric date. No check is made for valid months or days of the month.
USER NAME:	Maximum of twenty characters.

Other Option	Valid with Main Option	Function
U	none, V	The "U" option indicates that all physical sectors, both allocated and unallocated, are to be copied or verified. If "U" is not specified, only the allocated sectors, as mapped in the source diskette's allocation table, will be used.
Y	A	The "Y" option will cause a "Y" to be automatically given as a response to the file name duplication error message. This will automatically force the attempted deletion of the duplicate file on the destination diskette before the file is copied. The "Y" and "Z" options are mutually exclusive.
Z	A	The "Z" option will cause an "N" to be automatically given as a response to the file name duplication error message. This will automatically prevent the file on the source diskette from being copied to the destination diskette. The "Z" and "Y" options are mutually exclusive.

3.7 Messages

The following messages can be displayed by the BACKUP command. Not all messages are error messages, although error messages are included in this list. The standard error messages that can be displayed by all commands are not listed here.

BACKUP FROM DRIVE <s-unit> TO <d-unit>?

This indicates BACKUP will copy to the destination diskette in drive <d-unit> from the source diskette in drive <s-unit> if a "Y" response is given. Any other response will cause control to be returned to MDOS.

APPEND FROM DRIVE <s-unit> TO <d-unit>?

This indicates that BACKUP will perform the file append process if a "Y" response is given. Any other response will cause control to be returned to MDOS.

DISK NAME:

The "I" option has been specified. The operator is expected to respond with a new disk ID or a carriage return.

DATE(MMDDYY):

The "I" option has been specified. The operator is expected to respond with a new date or a carriage return.

USER NAME:

The "I" option has been specified. The operator is expected to respond with a new user name or a carriage return.

```
ENTER FILE COPY SELECTION COMMANDS:
SAVE (S), DELETE (D), PRINT (P), QUIT (Q), NO MORE (CR)
S, D, P, Q, (CR):
```

The "R" or "A" option has been specified. The file selection process is activated. The third line shows what the valid responses are.

S, D, P, Q, (CR):

This is a subsequent prompt from the file selection process. SAVE and DELETE commands can be entered until a P (print), Q (quit), or carriage return (NO MORE) is entered.

SYNTAX ERROR

This indicates a mistake in a response to a question or prompt from the BACKUP command. The entire line entered by the operator is ignored and a new response must be made.

STARTING TO COPY FILES

This indicates that files from the sorted directory are starting to be copied (R or A option).

NO FILES TO COPY

This indicates that there are no file names in the source directory (other than the MDOS system files) or that all of the file names from the sorted directory have been deleted. No files are copied if the "A" option is used. Only the MDOS family of system files will be copied if the "R" option is used.

<name> NOT FOUND

This indicates that a file name or a family of file names specified by a SAVE or DELETE command could not be found in the sorted directory.

COPYING <name>

This indicates that the file name specified by <name> is being copied to the destination diskette.

<name> - DUPLICATION: IS IT TO BE COPIED?

This indicates that the file name specified by <name> already exists on the destination diskette during the append process. Only a "Y" or "N" is accepted as a valid response.

NEW NAME:

This message is displayed if a "Y" is given in response to the preceding message. It allows the operator to assign a new file name to the file being copied from the source diskette. A carriage return response (no file name) will cause an automatic attempt to delete the duplicate destination file to be made, rather than assigning a new name to the source file.

<name> - DUPLICATION: IS COPYING

This indicates that the file name specified by <name> already exists on the destination diskette during the append process. The "Y" option caused an automatic attempt to delete the duplicate destination file to be made before the copy continues.

<name> - DUPLICATION: IS NOT COPIED

This indicates that the file name specified by <name> already exists on the destination diskette during the append process. The "Z" option caused the file to be skipped. The destination file is unaffected.

OBJECT FILE CREATION COPY ERROR

This usually indicates that insufficient contiguous space exists on the destination drive for the file being copied (A option). Occasionally, however, it may mean that an error was detected in the reading or writing of the file's Retrieval Information Block on the destination diskette.

CANNOT DELETE DUPLICATE NAME

This indicates that the duplicate file name on the destination diskette could not be deleted due to its protection attributes.

DELETE FILE NOS.:

The PRINT command displays this prompt to allow deletion of file names by entering their displayed numbers. The prompt will be redisplayed until a null response (carriage return) is given.

nn <name>

After the PRINT command is chosen during the file selection process, a list of all file names eligible for copying is displayed. The "nn" is a hexadecimal number that indicates the position of the name with respect to the total sorted directory. The <name>, of course, is the file's name and suffix.

SYSTEM SECTOR COPY ERROR

This indicates that a system sector could not be read from or written to. BACKUP cannot continue and control is returned to MDOS.

SECTOR nnnn

This indicates that the physical sectors "nnnn" did not compare during the verify process.

OFFSET ii DR<s-unit>-jj DR<d-unit>-kk

This indicates which bytes did not compare during the verify process. The "ii" is the hexadecimal offset into the sector, "jj" is the hexadecimal contents of the byte on the source unit <s-unit>, "kk" is the hexadecimal contents of the byte on the destination unit <d-unit>.

DIRECTORY READ/WRITE ERROR

This indicates that an internal system error was encountered while trying to access the directory of the source diskette. Errors of this type indicate a possible hardware problem.

SOURCE FILE COPY ERROR

This indicates that an internal system error was encountered while reading a Retrieval Information Block from a file on the source diskette. Errors of this type indicate a possible hardware problem.

INVALID TO COPY/VERIFY FROM DOUBLE TO SINGLE SIDED

This indicates that on an EXORdisk III system, the source diskette was double-sided while the destination diskette was single-sided. This is invalid.

3.8 Precautions with BACKUP

The following sections describe some of the precautions that should be taken when using the BACKUP command in the various environments that are supported by MDOS.

3.8.1 BACKUP and the CHAIN process

Since the BACKUP command has so many different paths that can be taken, it is generally recommended that BACKUP not be invoked from within a CHAIN process (see Chapter 6). The BACKUP process is so important to the protection of diskette files that the entire process should be supervised by the operator.

Diskette verification from within a CHAIN process using the BACKUP command is also infeasible. The CHAIN command writes intermediate information to the diskette in drive zero during its operation. Thus, if BACKUP with the "V" option is invoked from within a CHAIN process, and if drive zero is involved in the BACKUP process, then the two diskettes are guaranteed to be different.

3.8.2 Single/double-sided diskettes

On EXORdisk III systems the BACKUP command can be used to copy or verify from a single-sided diskette (source diskette) to a double-sided diskette (destination diskette); however, the reverse is not allowed.

When a single-sided diskette is copied to a double-sided diskette, the system tables (CAT and LCAT) are automatically adjusted so that they reflect the true amount of space available on the double-sided diskette. When a verify takes place, the CAT and LCAT will be different between the two diskettes; however, no verification error is displayed if the allocated parts of the tables are the same.

3.8.3 Four-drive systems

The BACKUP command has the capability of copying to or verifying with any of the three drives (1-3) in a four-drive system. It is not possible, however, for BACKUP to sense the difference between a two-drive and a four-drive system. Thus, due to the nature of the two-drive disk controllers with EXORdisk II, it is possible to destroy a diskette in drive one if BACKUP is invoked with the "R" option and if non-zero numbers are specified on the command line for <s-unit> and <d-unit>.

If the user has a two-drive system, it does not make any sense for him to enter logical unit numbers on the command line when invoking the BACKUP command, since the proper default is to copy from drive zero to drive one. If he were to specify to copy from drive two to drive three with the "R" option, then the diskette in drive one would be accessed and subsequently destroyed.

3.9 Examples

Many times it is desirable to differentiate the two identical copies of diskettes from each other by use of the ID sector information. The ID sector's contents can be changed during a diskette copy by using the "I" option.

```
=BACKUP ;I
BACKUP FROM DRIVE 0 TO 1?
Y
DISK NAME:NEWNAME
DATE(MMDDYY):010978
USER NAME:
=
```

All information to the right of the colons is supplied by the operator. The destination diskette will be given the disk name NEWNAME which will be printed on the heading lines of subsequent FREE and DIR command invocations (see Chapters 16 and 9, respectively). The date of the disk copy that is generated is January 9, 1978, and the same user name that was assigned to the source diskette during a previous BACKUP or during the initial DOSGEN process will be given to the destination diskette (indicated by carriage return response without any data).

The verification process using the two diskettes generated above will cause an error when comparing the ID sectors; however, the remainder of the diskettes are still compared. The offset messages of the discrepancies can be suppressed by also using the "S" option. Thus, the verification of the above example's generated diskettes would show the following operator-system interactions:

```
=BACKUP ;VS
SECTOR 0000
=
```

The following example assumes that no scratch or garbage files exist on the source diskette. Then, the reorganization process requires a minimum amount of operator interaction:

```
=BACKUP :1,:2;R
BACKUP FROM DRIVE 1 TO 2?
Y
ENTER FILE COPY SELECTION COMMANDS:
SAVE (S), DELETE (D), PRINT (P), QUIT (Q), NO MORE (CR)
S, D, P, Q, (CR):
COPYING MDOS    .SY
etc.
STARTING TO COPY FILES
COPYING BACKUP  .CM
etc.
=
```

It should be noted that no file selection commands were used. The resulting destination diskette will contain all files from the source diskette, but they may be in different places on the surface of the diskette. Thus, a reorganization process cannot be followed with a verification process for the same diskette pair. The "N" option could have been used in the above example to suppress the printing of the file names as they were being copied.

The last example shows the file append process. The example assumes that there is an MDOS diskette in drive 1. Also, it assumes that the diskette in drive zero has a family of files which are to be copied to the destination diskette. The family has file names which start with the letters "FOR".

The following shows the operator-system interactions:

```
=BACKUP ;A
APPEND FROM DRIVE 0 TO 1?
Y
ENTER FILE SELECTION COMMANDS:
SAVE (S), DELETE (D), PRINT (P), QUIT (Q), NO MORE (CR)
S, D, P, Q, (CR):S FOR*.*
S, D, P, Q, (CR):P
09 FORT    .CM
0A FORTLIB .RO
0B FORTNEWS.SA
0C FORTEST1.SA
0D FORTEST2.SA
0E FORTEST3.SA
0F FORTEST4.SA
10 FORTEST5.SA
DELETE FILE NOS.:
B-E,10
DELETE FILE NOS.:

STARTING TO COPY FILES
COPYING FORT    .CM
COPYING FORTLIB .RO
COPYING FORTEST4.SA
FORTEST4.SA - DUPLICATION:  IS IT TO BE COPIED?
Y
NEW NAME:FTEST
=
```

The file selection command SAVE was used to flag all file names beginning with FOR as eligible for copying. Then the PRINT command was used to see the eligible list of file names. The PRINT command terminates the use of the DELETE and SAVE commands. Thus, the PRINT command's delete file feature is used to remove any remaining file names from the eligible list. File names 0B, 0C, 0D, 0E, and 10 were deleted in this manner. A null response is required to terminate the PRINT command's input prompting. The last file to be copied turned out to have a duplicate file name existing on the destination drive. The operator responded with a "Y" indicating that he wanted to copy the file anyway. Since duplicate file names cannot exist, the append process lets the operator rename the source file before it gets copied. The new name assigned to the file on the destination diskette will be FTEST.SA (default suffix assigned).

CHAPTER 4

4. BINEX COMMAND

The BINEX command allows memory-image files to be converted into an EXbug-loadable format for copying to tape. This command performs the inverse operation of the EXBIN command (see Chapter 14). BINEX is useful in the development of non-diskette-resident software with MDOS, since the object code can be written to tape after it has been tested.

4.1 Use

The BINEX command is invoked with the following command line:

```
BINEX <name 1>[,<name 2>]
```

where <name 1> is the file specification of a memory-image file that is to be converted, and <name 2> is the file specification of a file that is to receive the results of the conversion. Only <name 1> is required to be entered on the command line. The default suffix "LO" and the default logical unit number zero will be supplied for <name 1> if those quantities are not explicitly given. The output file specification, <name 2>, is optional. If <name 2> is entered, it may be a partial file specification consisting of only a file name, a suffix, or a logical unit number (or any combination thereof). The unspecified parts of <name 2> will be supplied from the respective parts of <name 1>, with the exception of the suffix. The default suffix for <name 2> is "LX" to indicate its EXbug-loadable format. If no file specification is given for <name 2>, the output file will be created with the same file name as <name 1> but with the suffix "LX". If only a suffix is given for <name 2>, that suffix will be used instead of the default "LX". If no logical unit number is given for <name 2>, the output file will be created on the same drive as given for <name 1>. In any case, <name 2> must be a file specification for which no entry already exists in the directory.

Standard error messages will be displayed if <name 2> already exists, if <name 1> does not exist, or if <name 1> is of the wrong file format. If no errors are found on the command line, BINEX will write into the output file a name record, or S0 record, that contains the file name and suffix of <name 2>. Then, BINEX will convert the content of <name 1> into displayable ASCII characters and output them to <name 2> in the form of the EXbug S1 records (the "M6800 EXORciser User's Guide" contains a description of this record format).

The terminating S9 record will contain the starting execution address that was extracted from <name 1>'s load information.

The memory-image file, <name 1>, is unaffected by the entire BINEX process. The output file, <name 2>, can then be copied to tape (see Chapter 7, COPY Command) for use in a non-diskette environment.

4.2 Error Messages

No special error messages are displayed by the BINEX command. Only the standard error messages available to all commands are used.

4.3 Examples

Most frequently, the default suffixes and logical unit numbers suffice for BINEX operation. The following command line

```
BINEX TESTPROG
```

will produce the file TESTPROG.LX on logical unit zero from the memory-image file TESTPROG.LO, also on logical unit zero.

If the output file is to be created on a different drive than the input file, but the other default parameters are still to be applied, then only a logical unit number need be specified for <name 2> as in the following example:

```
BINEX TESTPROG,:1
```

which will create the file TESTPROG.LX on logical unit one.

If the file to be converted happens to reside on a drive other than zero, then that unit number will also be the default value of the logical unit number for the output file. Thus,

```
BINEX TESTPROG:2
```

will create TESTPROG.LX on drive two.

The last example illustrates the explicit naming of an output file and input file. In any case involving default values of which the operator is uncertain, it is always safe to explicitly use the full file specifications. For example,

```
BINEX TESTPROG.LO:0,FILEX.LT:0
```

will create FILEX.LT on drive zero.

CHAPTER 5

5. BLOKEDIT COMMAND

The BLOKEDIT command allows lines of text from one or more ASCII files to be selectively copied into a new file. This command can be useful in generating new program source files by copying routines from existing source files, or in rearranging existing files by copying their lines into a new sequence.

5.1 Use

The BLOKEDIT command is invoked with the following command line:

```
BLOKEDIT <name 1>,<name 2>
```

Both of the parameters are required by the BLOKEDIT command. <name 1> is the file specification of a command file, and <name 2> is the file specification of a new file which will be created. The new file will be written into as directed by commands in the command file.

Both file specifications are given the default suffix "SA" and the default logical unit number zero. <name 1> must be the name of a file that exists in the directory. <name 2> must not already exist. A standard error message will be displayed if either of these criteria is not met, or if <name1> is of the wrong file format.

5.2 BLOKEDIT Command File

The command file specified by <name 1> is the controlling factor in the execution of the BLOKEDIT command. The command file contains the names of the source files that are to be used for the extraction of data, the numbers of the lines within a particular source file that are to be copied into <name 2>, comments, and original text supplied by the user that is also to be copied into <name 2>. The command file must be created with the EDIT command, or a similar command, prior to using the BLOKEDIT command.

There are three kinds of lines that can appear in the command file:

1. Comment lines
2. Command lines
3. Quoted lines

The three types of lines that comprise the command file are discussed in the following sections.

5.2.1 Comment lines

A comment line is a line whose first character is an asterisk (*). For example:

```
*
* THESE THREE LINES ARE BLOKEDIT COMMENT LINES
*
```

The occurrence of comment lines in the command file is ignored by the BLOKEDIT command. Comment lines serve only to document the command file.

5.2.2 Command lines

A command line is recognized by the fact that its first character is an upper-case alphabetic character, a decimal digit, or a double quote character. For example,

```
FILENAME:1
5,75-80
"
```

are three valid command lines.

Command lines which begin with an upper-case alphabetic character indicate that a source file is being named. Such command lines are used to specify from which file the subsequent lines are to be copied. A source file can only be named by putting its file specification at the beginning of a command line. Optionally, the suffix and/or logical unit number can be specified in the standard format after the file's name. The default values of "SA" and zero are supplied automatically if no explicit references to suffix or logical unit number are made.

Command lines which begin with a decimal digit indicate that the command line will contain one or more numbers. These numbers represent the physical line numbers to be copied from a source file which has been named using the prior form of the command line. Physical line numbers can be up to five digits in length and must be in the range 1-65535, inclusive. More than one physical line number can appear on a command line if it is followed by a comma. A range of physical line numbers can be specified by separating the start and end of the range with a hyphen (-). For example,

```
5
12345
100-364
12,15,1-5,17-200,5-15,2,2
```

are valid forms of physical line number command lines. A source file's physical line numbers can be printed using the LIST command described in Chapter 17.

5.2.3 Quoted lines

A command line that begins with a double quote character (") indicates the beginning or the end of quoted lines. Any information that appears on the same line as the double quote is ignored. A quoted line is any line bounded by a pair of command lines which begin with a double quote character. All quoted lines will be copied directly from the command file into the new file, as is. Thus, it is possible to include original lines of text that will be copied into the new file in addition to the physical lines copied from the named source files. The following example illustrates the use of quoted lines:

```
" START OF QUOTED LINE SEQUENCE
LABEL LDAA #$FD . SET MASK
 LSRB .
 STAB TAB+4
 TAB .
*
* COMMENTS IN QUOTED LINES GET WRITTEN OUT
*
 JMP EXIT .
" END OF QUOTED LINE SEQUENCE
```

The first and the last lines of the example will be discarded by the BLOKEDIT command. The eight lines in between will be written as is into the new file.

5.3 Messages

The following messages can be displayed by the BLOKEDIT command. Not all messages are error messages, although error messages are included in this list. The standard error messages that can be displayed by all commands are not listed here.

CURRENT SOURCE FILE IS <name>

A command line containing the name of a source file has been processed. The name of source file is shown as <name>. This message is used to monitor the path of BLOKEDIT through the command file.

DONE. NEW FILE LINE COUNT IS nnnnn

The command file has been exhausted (end of file encountered) when this message is displayed. It indicates that no more command lines will be processed. The number of physical lines that were copied into the new file is given by the decimal number "nnnnn". After this message is displayed, control is returned to MDOS.

** 36 FILE EXHAUSTED BEFORE LINE FOUND

This message is displayed when the source file being read was exhausted (end of file encountered) before a specified physical line number was found. This is not a fatal error. The next command line from the command file will be processed.

** 38 INVALID LINE NUMBER OR RANGE

This error message can be displayed for several reasons. A line in the command file did not begin with an asterisk, a double quote, a decimal digit (0-9), or an alphabetic character (A-Z), and the line was not a quoted line. If the command line started with a digit, then the physical line number had a value outside of the range 1- 65535, or the starting number of a line number range was greater than the ending line number of the range. In any case, this is a fatal error. BLOKEDIT is terminated and control returned to MDOS. The command line in error is displayed prior to this message.

** 39 LINE NUMBER ENTERED BEFORE SOURCE FILE

This message indicates that the command file contained a line with a decimal digit in the first position before a source file was named. Processing cannot continue, so the BLOKEDIT command is terminated. The command line in error is displayed prior to this message.

5.4 Examples

In the following example it is assumed that the three source files EDIT.SA:1, ASM.SA:0, and LOAD.SA:0 contain some special utility subroutines that are to be extracted and placed into a new file UTILITY.SA:0. The physical line numbers of the routines can be determined by listing the source files on the console or printer (Chapter 17, LIST Command). With that information, the command file BLKCMD.SA:0 is created using the EDIT command:

```
*
* Define the first source file
*
EDIT:1
176-205
224-230
*
* Define the second source file
*
ASM.SA:0
" Insert a PAGE directive to separate routines
 PAGE
"
56-80,90-101,150-163
*
* Define the last source file
*
LOAD
" Insert another PAGE directive
 PAGE
"
27,28,29,30,31,32,33,34,35,36
37
38
39
40
*
* End of Command File
*
```

Then, the MDOS command line

```
BLOKEDIT BLKCMD,UTILITY
```

is used to invoke the BLOKEDIT command. During the processing, BLOKEDIT will display the following messages:

```
CURRENT SOURCE FILE IS EDIT    .SA:1
CURRENT SOURCE FILE IS ASM     .SA:0
CURRENT SOURCE FILE IS LOAD    .SA:0
DONE.  NEW FILE LINE COUNT IS 104
=
```

The new file will contain the indicated lines from the respective source files. Each set of lines copied from the source files has been separated from the next file's set of lines by a PAGE directive (causing paging when the UTILITY file is assembled). The PAGE directive was inserted using quoted lines.

BLOKEDIT can also be used to rearrange the lines of an existing file by copying them in a given sequence into the new file. The following command file:

```
PROG1
207-300,10-206,1-9
```

for example, could be used to shuffle the lines in the source file PROG1.SA:0. First, lines 207-300 would be copied into the new file. These would be followed by lines 10-206, which would be followed by lines 1-9.

The last example illustrates an error message displayed by BLOKEDIT. The command line in error is displayed prior to the error message. The initial five-digit number in front of the displayed command line gives the line's physical line number within the file (as displayed with the LIST command, Chapter 17).

```
=BLOKEDIT BLKCMD,TEMPEQU
CURRENT SOURCE FILE IS EQU     .SA:0
00002   56-34
** 38 INVALID LINE NUMBER OR RANGE
=
```

The error was caused by an invalid line number range. The starting number of a range must be less than or equal to the ending number of the range.

CHAPTER 6

6. CHAIN COMMAND

The CHAIN command allows predefined procedures to be automatically executed. A procedure consists of any sequence of MDOS command lines that has been put into a diskette file, known as a CHAIN file. Instead of obtaining successive command lines from the console, CHAIN will fetch commands from the CHAIN file. This feature allows complicated and lengthy operations to be defined once, and then invoked any number of times, requiring no operator intervention. The additional capabilities of conditional directives to the CHAIN command at both compilation and execution time, and the capability of string substitution, permit an almost unlimited number of applications to be handled by a CHAIN file.

6.1 Use

The CHAIN command is initially invoked by the following command line:

```
CHAIN <name 1> [;<arg 1>,.....,<arg n>]
```

The only required parameter is <name 1>, the file name specification of the diskette file that contains the procedure definition. The CHAIN file, <name 1>, is given the default suffix "CF", permitting the file name to be identified in the directory listing at a glance as being a CHAIN file. The default logical unit number is zero. The optional arguments, <arg i> (i = 1 to n), are CHAIN tag definitions which can be used to modify the compilation, content, or execution of a CHAIN file.

Two special forms of the CHAIN command line can be used to restart an aborted CHAIN process. These command lines are shown here, but are described in detail in section 6.6.

```
CHAIN N*
CHAIN  *
```

CHAIN executes a compilation phase and an execution phase. In the compilation phase, <name 1> is read from beginning to end. An intermediate file, CHAIN.SY:0, is created during the compilation. The intermediate file consists of lines to be used in the execution phase of the CHAIN process. This file will be automatically deleted upon the subsequent successful completion of the CHAIN process.

During the execution phase, CHAIN basically intercepts the system console input requests so that input can be supplied from the intermediate file. Each time an input request is made by a command that is invoked by the CHAIN process, the next line from the intermediate file will be read and passed to the command. As far as the command is concerned, it is receiving its input information from the operator at the console.

The CHAIN command only intercepts console input via the MDOS system function ".KEYIN" (see section 25.2). Therefore, only programs (commands or user-written programs) that use this system function will receive their input from the intermediate file. Programs which contain their own input routines, or which use the device independent I/O functions (see section 25.3) can be invoked by the CHAIN process, but the subsequent input to those programs must be supplied manually via the console.

The CHAIN command cannot be invoked from within a CHAIN process unless it is invoked from the last line of the intermediate file. An error message will be displayed if other types of CHAIN command recursion are attempted.

The CHAIN command will continue to supply information from the intermediate file until the end of the file is encountered. If, at that point, the next input request from the console is by the MDOS command interpreter, the CHAIN process will be properly terminated, MDOS will be re-entered, and commands will again be accepted from the operator at the console. If, however, the end of the intermediate file is encountered while a program is requesting console input, then the CHAIN process is aborted, an error message is displayed, and the currently active program will be stopped. Control will then be given to the MDOS command interpreter.

The diskette in drive zero must remain in drive zero throughout the execution of the CHAIN process, even if the "CF" file is compiled from drives other than zero.

6.2 Tag Definition, Assignment, and Substitution

The CHAIN command line can be parameterized with arguments that follow the CHAIN file specification. Each argument has the following format:

```
<tag>[%<value>%]
```

where <tag> is the name by which the argument is referenced within the CHAIN file, and <value> is the value assigned to that argument. As many arguments as fit on the command line can be specified. Multiple arguments must be separated by commas. Tags may be from one to thirty-two characters in length and can contain any displayable character except the period (.), the comma (,), the space ( ), or the percent sign (%). A tag's value can be any series of displayable characters with the exception of the percent sign. A tag is given a value by following the tag's name with the value enclosed in percent signs. If no percent sign follows a tag's name, it is assigned a null value. For example, the command line

```
CHAIN TFILE;LIST,DAY%17%,TIME%02:30%
```

defines three tags: LIST, DAY and TIME. The tag LIST is assigned a null value; the tag DAY is given the value 17; the tag TIME is given the value 02:30.

CHAIN allows two uses to be made of tags. First, tests can be performed within the CHAIN file to determine whether or not a specific tag has been specified on the CHAIN command line. Second, the value of a tag can be substituted for a tag's occurrence within the CHAIN file. Thus, using the above example, the CHAIN file could contain a test for the presence of the tag LIST to determine if the CHAIN process will produce output to a printer. The values of the tags DAY and TIME could be substituted in one of the heading lines that may be produced by the CHAIN process.

So far in the discussion, the value of a tag has not been used. The existence of a tag can be tested regardless of a tag's value. A tag's value is substituted for each occurrence of the tag's name contained between two delimiting percent signs. The following example will illustrate tag substitution. If a CHAIN file contains these statements:

```
RASM TESTPROG;H%OPTION%
PROGRAM ASSEMBLED ON %DATE%
EXBIN TESTPROG%START%
```

then the tags OPTION, DATE, and START will have their respective values put in place of their tag names and the delimiting percent signs before each line is written into the intermediate file. If no tags were specified for the above CHAIN at its invocation, then the following intermediate file would be compiled:

```
RASM TESTPROG;H
PROGRAM ASSEMBLED ON
EXBIN TESTPROG
```

If the tags were given initial values via the CHAIN command line as:

```
OPTION%XLG%,DATE%JANUARY 8, 1978%,START%;1000%
```

then the following intermediate file would be compiled:

```
RASM TESTPROG;HXLG
PROGRAM ASSEMBLED ON JANUARY 8, 1978
EXBIN TESTPROG;1000
```

Tag substitution is used here to specify the various options for the assembly process, a date for the heading line printed during the assembly, and the starting execution address for the converted object file. The use of tags and tag values, therefore, is of great importance in the creation of complicated and general purpose CHAIN files.

To pass tag values from one CHAIN file to another, a forcing character is used. The backslash character (\) is used to indicate that the next character of a line is not to be tested as a special character (i.e., to see if an operator follows, or a valid tag). Thus, passing a tag from one CHAIN file to another can be done with a series of statements like the following:

```
RASM TESTPROG;H%OPTION%
PROGAM ASSEMBLED ON %DATE%
CHAIN FILE2;START\%%START%\%
```

The first and last percent signs of the last line are not tag replacement indicators. When the above lines are compiled, the resultant intermediate file will not contain the backslash characters. If the value "XLG" is given to OPTION, "01.8.78" to DATE, and ";1000" to START, then the compiled CHAIN file would appear as

```
RASM TESTPROG;HXLG
PROGRAM ASSEMBLED ON 01.8.78
CHAIN FILE2;START%;1000%
```

The value of START would be passed from the first CHAIN file to the second CHAIN file. The second CHAIN process can only be invoked from the last line of the intermediate file.

6.3 Compilation Operators

Two types of CHAIN operators exist which can be used to modify the procedure that is performed through the CHAIN process: Compilation Operators and Execution Operators. Execution Operators are described in section 6.4. Compilation Operators permit the operator to parameterize a CHAIN file to perform many different procedures. For example, a CHAIN file may contain the MDOS command lines to assemble an entire system of programs. Based on the CHAIN arguments specified on the CHAIN command line, all or part of the system of programs may be assembled. The options for the assembly process can also be supplied via a CHAIN argument (see example in section 6.7).

All Compilation Operators are included in the CHAIN file along with any other statements. Compilation Operators are denoted by a slash (/) appearing in the first column of a line. Any number of intervening spaces (including none) can be placed between the slash and the operator. If an operator is found which is not defined, the CHAIN process will be aborted. The following Compilation Operators are defined:

Operator	Function
*	Comment
IFS	Conditional "if set" test
IFC	Conditional "if clear" test
XIF	End conditional
ELSE	Conditional alternative
ABORT	Unconditional CHAIN abort

6.3.1 Compilation Comments

If the character following a slash is an asterisk (*), then a Compilation Comment is indicated. The remainder of the line following the asterisk contains the comment, which can include any displayable characters. Compilation Comments are not written into the intermediate file. They are, however, displayed on the console immediately after they are read from the CHAIN file. Compilation Comments are useful in communicating to the operator what intermediate file is being compiled for execution. The comment lines are only displayed if the part of the file containing the comments is being compiled into the intermediate file (see next section).

6.3.2 IF operator

If the characters following a slash are "IF", an IF operator is denoted. There may be any number of intervening spaces between the slash and the IF operator. This feature allows a structured type of CHAIN file to be constructed that will show by its physical appearance the range of the conditional operators. The IF operator allows a test to be made for the existence of one or more tags on the CHAIN command line. If the test proves positive, or true, then the lines from the CHAIN file following the IF operator will be included in the intermediate file (written to the CHAIN.SY file). If, however, the test proves negative, or false, then the subsequent lines will not be included in the intermediate file. The lines from the CHAIN file will be included or excluded following the IF operator until an ELSE or XIF operator (explained below) is encountered.

The IF operator has two forms: IFS and IFC, which stand for "if set" and "if clear", respectively. The IFS operator proves positive if any of the tags listed as its operand have been specified on the CHAIN command line. For example,

```
/IFS LIST
```

will prove positive if the tag LIST was mentioned on the CHAIN command line. The same test will prove negative if LIST did not appear. Likewise, the IF operator

```
/IFC DAY
```

will prove positive if the tag DAY was not specified on the CHAIN command line. The test will prove negative if DAY did appear. Multiple IF operators can appear in sequence to see if all tags of a certain group were specified. Thus,

```
/IFS FLAG1
/IFS FLAG2
/IFS FLAG3
```

will prove positive only if tags FLAG1, FLAG2, and FLAG3 were specified on the CHAIN command line.

More than one tag can appear in the operand field of an IF operator. A comma separating tag names on an IF line will perform an "inclusive or" function. A period separating tag names, on the other hand, will perform an "and" function. The "and" function has precedence over the "or" function. That is, the commas (or) can be thought of as grouping the periods (and). For example, the IF operator line

```
/IFS FLAG1.FLAG2.FLAG3
```

is equivalent to the previous example of three successive IF operators. The following line,

```
/IFS F1.F2,FLAG3,TAG1.TAG2.LIST
```

which can be thought of as being evaluated by the following grouping,

```
(F1 and F2) or (FLAG3) or (TAG1 and TAG2 and LIST)
```

will prove positive if the tags F1 and F2 are specified, or if FLAG3 is specified, or if tags TAG1 and TAG2 and LIST are specified.

If one IF operator has proven negative, then the subsequent lines from the CHAIN file, including other IF operators, will be ignored until either a corresponding ELSE or XIF operator is found. In this way, the IF operator is used to modify the resultant intermediate file.

6.3.3 XIF and ELSE operators

Two Compilation Operators can cause the range of an IF operator to be ended. The XIF operator marks the end of a series of conditionally compiled statements. The ELSE operator reverses the sense of the IF's test condition, and is used to indicate what is compiled if the test condition is not met. The conditional IF operators can be nested to a depth of sixteen levels. The following example shows the use of XIF and ELSE:

```
/IFS LIST
LIST TESTFILE;LH
TEST PROGRAM HEADING LINE
/ELSE
LIST TESTFILE
/XIF
```

In this example, the file TESTFILE will be listed on the printer only if the tag LIST is specified on the CHAIN command line. A heading line is also provided within the CHAIN file if the LIST tag is used. If, however, LIST is not specified, then the ELSE portion of the conditional operator will be compiled, causing TESTFILE to be shown on the system console instead.

If the above example were to be written without the ELSE operator, one additional IF and XIF operator pair would have to be used, as shown:

```
/IFS LIST
LIST TESTFILE;LH
TEST PROGRAM HEADING LINE
/XIF
/IFC LIST
LIST TESTFILE
/XIF
```

It can be seen that the use of the ELSE operator makes the CHAIN file easier to understand.

Each IF operator must have a corresponding XIF operator. The ELSE operator is available at the option of the user. The following example shows how nested IF operators might appear in a CHAIN file:

```
/IFS F1
ASM TESTPROG
/   IFS F2
EXBIN TESTPROG
/   XIF
/XIF
```

In this case, the tag F1 governs whether or not the file TESTPROG will be assembled. If F1 is specified, then the assembly will be performed. Then, if in addition F2 is specified on the CHAIN command line, the object file conversion will also take place. The CHAIN file can be used, therefore, to perform only the assembly, or the assembly and the object file conversion, but not the object file conversion by itself.

If, through the use of the conditional operators, a null (empty) intermediate file is generated, then the Execution Phase of the CHAIN command will be skipped. Control will be given to the MDOS command interpreter, as if no CHAIN had ever been executed.

6.3.4 ABORT operator

The ABORT operator provides a way of instantly returning to MDOS during a CHAIN file's compilation. No messages will be displayed as a result of encountering the ABORT operator. It is the user's responsibility to include an explanation for the ABORT through the use of Compilation Comments.

The ABORT operator is typically employed in terminating a CHAIN compilation if one or more critical tags have been omitted from the CHAIN command line. For example, the following CHAIN file will be aborted during the compilation phase if both of the tags OPT and FILE are missing. The Compilation Comments will indicate the reason for the termination:

```
/IFS OPT.FILE
/* GOING TO ASSEMBLE %FILE%
RASM %FILE%;%OPT%
/ELSE
/* BOTH "FILE" AND "OPT" MUST BE SPECIFIED
/* CHAIN TERMINATED
/ABORT
/XIF
```

6.4 Execution Operators

Execution Operators can be used for the dynamic adjustment of a CHAIN process while it is being executed. Through the use of these operators, the user can set values in an error status word maintained by MDOS, test the word, and, depending upon the results of the test, skip a portion of the procedure. The error status word is accessed by all MDOS commands to indicate whether or not they completed their function without error.

All CHAIN Execution Operators are denoted by the commercial at-sign (@) as the first character of a line. Any number of intervening spaces (including none) can be placed between the at-sign and the operator. If an operator is found which is not defined, the CHAIN process will be aborted. The following Execution Operators are defined:

Operator	Function
*	Comment
	Operator breakpoint
SET	Set error status word
TST	Test error status word
JMP	Continue sequential processing at label
LBL	Define a label
CMD	Change state of CHAIN input echo

6.4.1 Execution Comments

If the character following the at-sign is an asterisk (*), then an Execution Comment is indicated. The remainder of the line following the asterisk contains the comment, which can include any displayable characters. Execution Comments are compiled into the intermediate file and are not displayed until they are encountered during the execution phase. Execution Comments are used to relay information to the operator during the actual execution of the intermediate file. In conjunction with the Operator Breakpoint (next section), these comments also serve as a means of passing instructions to the operator for mounting paper into the printer, swapping diskettes in drives one, two, or three, loading a cassette, etc.

6.4.2 Operator Breakpoints

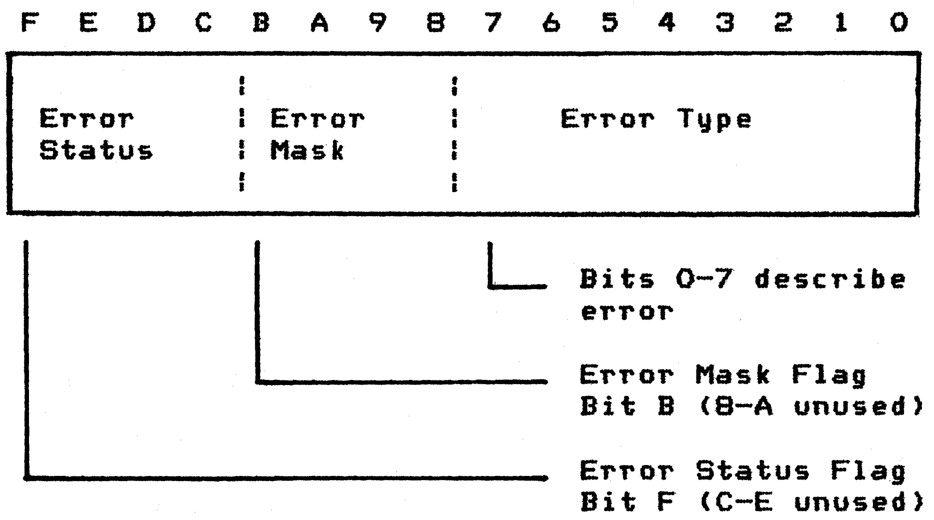
A variation of the Execution Comment is the Operator Breakpoint. If a period (.) is used instead of an asterisk for the Execution Comment, then the normal Execution Comment is displayed; however, instead of continuing with the processing of the next line of the intermediate file, the BEL ($07) character is sent to the console to alert the operator. The CHAIN process then waits for any key on the keyboard to be depressed before continuing. For example, the following compiled CHAIN file:

```
@* GOING TO ASSEMBLE PROGRAM
@. TURN ON PRINTER
RASM TESTPROG;LXG
```

would display the two comments during the execution of the CHAIN process. Prior to starting the assembly, however, the CHAIN process would pause allowing the operator time to ready the printer. Execution would not resume until after the operator had depressed any key on the system console.

6.4.3 Error status word

Among the operating system's resident variables is a two-byte error status word. Each MDOS command will set or clear a bit within this status word to indicate the status of the command's completion. The error status word has the following format:


F E D C B A 9 8 7 6 5 4 3 2 1 0
Error Status
Error Mask
Error Type
Bits 0-7 describe error
Error Mask Flag Bit B (8-A unused)
Error Status Flag Bit F (C-E unused)

Normally, after the completion of each command, all bits of the Error Status and the Error Type are cleared (= 0). The Error Mask is not affected by MDOS commands. If an error occurred during the command, the Error Status Flag (bit F) will be set by the command. In addition, an Error Type will be set into the lower half of the status word (bits 0-7). The Error Type is used to indicate which error was detected by the command.

Usually, the CHAIN process will abort anytime the Error Status Flag is set by one of the commands invoked from the intermediate file. The Error Mask can be used to inhibit CHAIN process aborting due to command errors by setting the Error Mask Flag (bit B) to a 1.

The Execution Operators can affect certain parts of the status word. The following symbols are used to refer to the various parts of the status word:

Word Designator	Error Status Word Part
W	Whole word (bits 0-F)
T	Error Type (bits 0-7)
M	Error Mask (bits 8-B)
S	Error Status (bits C-F)

6.4.4 SET operator

The SET operator can be used to place a certain bit pattern into the system error status word. In particular, the SET operator is the only way that the Error Mask Flag can be set to inhibit CHAIN process abortions. The MDOS commands will only set the Error Status and the Error Type. The SET operator has the following format:

```
@SET[,<j>] [<value>]
```

where <j> is the status word designator (explained above) and <value> is a hexadecimal number that is to be placed into the designated word part. The size of <value> must not be greater than the size of the word part into which the it is to be placed. If the status word designator is not specified, then W, the whole word part, will be assumed. If <value> is not specified, then zero will be assumed. As an example of the SET operator, the following will set the Error Mask Flag (bit B) to inhibit CHAIN process aborting due to command execution errors:

```
@SET,M 8
@SET,W 800
@SET 800
```

All three forms will set bit B of the error status word; however, the last two forms will, in addition, set to zero all other parts of the error status word.

6.4.5 TST operator

The TST operator is used to examine the error status word for a particular condition. This operator has the following format:

```
@TST[,<j>] <condition> [,<value>]
```

where <j> is the status word designator, <condition> is the test condition to be performed, and <value> is a hexadecimal number that is used as part of the test.

Use of the TST operator results in a true or false condition based on the test performed. If the result of the test is true, then the next sequential line in the intermediate file will be skipped. If the result of the test is false, however, then the next sequential line in the intermediate file will be processed. In other words, a false condition has the same effect as if the TST operator was not processed at all.

If the status word designator is not specified, then W, the whole word part, will be assumed. The following test conditions can be used in the <condition> field of the TST operator:

<condition>	Test performed on word part
EQ	Equal to <value>
NE	Not equal to <value>
GT	Greater than <value>
LT	Less than <value>
GE	Greater than or equal to <value>
LE	Less than or equal to <value>
BS	Bit set (=1)
BC	Bit clear (=0)

The first six tests are the standard relational tests for equality, etc., that can be performed with the <value> and the designated word part. The last two tests (BS and BC) allow specific bits in the designated word part to be tested for being set (BS) or clear (BC). The bits to be tested are indicated by the one bits from <value>.

The <value> part of the TST operator is a hexadecimal number in the range 0-FFFF. The size of <value> must not be greater than the size of the word part that is being tested. No signed numbers can be used. That is, all comparisons and tests are made with positive integers. If <value> is not specified, then the default of zero will be used.

6.4.6 JMP operator

The JMP operator allows skipping lines in the intermediate file during its execution. Used in conjunction with the TST operator, the JMP operator can be turned into a conditional jump around critical steps if certain conditions are detected during the execution of the CHAIN process.

The JMP operator has the following format:

```
@JMP <label>
```

where <label> must be defined via the label operator LBL. Jumps can only be made in a forward direction. That is, once a line has been executed from the intermediate file, it cannot be jumped to with the JMP operator, even if it has a defined label. Jumps to undefined labels or backward jumps will cause the CHAIN process to be aborted.

6.4.7 LBL operator

The LBL operator is used to define a label within the CHAIN file. All labels referenced by the JMP operator must be defined with the LBL operator. The format of the LBL operator is:

```
@LBL <label>
```

where <label> follows the same restrictions placed on tag names (section 6.2). Labels that are multiply defined, undefined, or backward references will be flagged as errors during the CHAIN compilation phase. Such errors will cause the CHAIN process to be aborted.

6.4.8 CMD operator

Normally, during the execution phase, as commands are processed from the intermediate file, each command line is displayed on the console. Likewise, all input requested by the command that is supplied from the intermediate file will be displayed on the console. The CMD operator can be used to suppress console display of all input that originates from the intermediate file. The CMD operator has the following format:

```
@CMD {ON or OFF}
```

where either ON or OFF must be specified. The CMD operator can be used as many times as needed within the intermediate file. Initially during the execution phase, the ON form of the CMD operator is in effect.

6.5 Messages

The following messages can be displayed by the CHAIN command. The standard error messages that can be displayed by all commands are not listed here. The messages are broken up into two sections: those that can be displayed during the compilation phase, and those that can be displayed during the execution phase.

The following error messages can be displayed during the compilation phase:

ILLEGAL NESTING OF CHAIN COMMANDS

A CHAIN command was found in the intermediate file that did not coincide with the last record of the file. CHAIN processes can only invoke another CHAIN command from the last line of the intermediate file.

SOURCE SYNTAX ERROR

One of the source lines of the CHAIN file contained a backslash (\) as the last character of the record, or an illegal tag reference was encountered.

ILLEGAL OPERATOR

The operator following a slash (/) was not a valid Compilation Operator, or the operator following an at-sign (@) was not a valid Execution Operator.

INVALID CONDITIONAL EXPRESSION

An invalid tag reference or invalid tag separator (other than period or comma) was used on a conditional Compilation Operator statement.

INVALID NESTING OF CONDITIONALS

More than sixteen levels of conditionals were used, an unequal number of IFs and XIFs exist, or an ELSE operator was used illegally.

EXECUTION OPERATOR OPERAND ERROR

The operand of an execution operator was invalid.

VALUE TOO LARGE FOR FIELD

A value was specified for the Execution Operators SET or TST that was larger than the status word part designator allowed.

END OF FILE REACHED BEFORE LAST XIF FOUND

The end of the CHAIN file was encountered while searching for an XIF operator. Usually this indicates an unbalanced number of IFs and XIFs.

UNDEFINED LABELS FOUND

A JMP operator referenced a label which was never defined with a LBL operator.

OUTPUT RECORD BUFFER OVERFLOW

A line from the CHAIN file was encountered which, after the substitution of all tag values, exceeded eighty characters in length.

** 48 CHAIN OVERLAY DOES NOT EXIST

The MDOS system CHAIN overlay does not have an entry in the directory. The REPAIR command (Chapter 22) should be used to check the diskette for other errors.

The following messages can be displayed during the execution phase:

END CHAIN

This message is displayed upon the successful termination of a CHAIN process. The next console input request will be obtained from the system console again. The intermediate file, CHAIN.SY:0, will have been deleted.

** 01 COMMAND SYNTAX ERROR

An Execution Operator was encountered that had an illegal operand field.

** 08 CHAIN ABORTED BY BREAK KEY

The operator depressed the BREAK key during the execution phase causing the CHAIN process to be aborted.

** 09 CHAIN ABORTED BY SYSTEM ERROR STATUS WORD

The last executed program set an error status into the system error status word which was not masked by the SET operator. If no SET operators are used in a CHAIN file, any error status word change will cause the CHAIN process to be aborted.

** 22 BUFFER OVERFLOW

The response obtained from the intermediate file to an input request exceeded the maximum number of characters that were acceptable to the input request.

** 49 CHAIN ABORTED BY ILLEGAL OPERATOR

An illegal Execution Operator was encountered in the intermediate file.

** 50 CHAIN ABORTED BY UNDEFINED LABEL

A JMP operator was encountered which referenced a label that did not exist (Backward references are treated as undefined labels).

** 51 CHAIN ABORTED BY PREMATURE END OF FILE

An access to the intermediate file returned an end-of-file condition when an input request was made by a program that was invoked by the CHAIN process. All input that is expected by the program must be in the intermediate file.

6.6 Resuming an Aborted CHAIN Process

If a CHAIN process is aborted during the execution phase for any reason, the CHAIN process can still be restarted. Since the intermediate file is not deleted until the CHAIN process has been successfully completed, this capability eliminates the need to recompile the original CHAIN file.

The special CHAIN command line:

```
CHAIN *
```

will restart the execution phase with the line last fetched from the intermediate file (the line that caused the error). For example, if an assembly has been invoked by the CHAIN process for which a duplicate object file exists, the CHAIN process will normally be aborted. The operator could then manually delete the duplicate file name and restart the CHAIN process with the above special form of the command line.

If the failing command can never succeed, the current line of the intermediate file can be bypassed, and the next one used to resume the aborted CHAIN process by using the following special command line:

```
CHAIN N*
```

If the next line of the intermediate file has been intended as a keyin response for the program (which just failed), then the process will generally abort again immediately. By using the "N*" form of the special command line several times, the invalid step can usually be bypassed and the CHAIN process resumed at a valid MDOS command line.

The Error Status Mask and the current state of the CMD operator are lost when a CHAIN is aborted. These values cannot be restored when an aborted CHAIN process is restarted.

6.7 Examples

The following example shows a fairly complex CHAIN file that incorporates most of the features described in this chapter. This CHAIN file is used to assemble and create loadable files of a system of program files that resides on multiple diskettes. The primary assumption made is that an MDOS system diskette is on drive zero and that the source programs will be on drive one (although not all at the same time).

In this example the CHAIN process will display messages to the operator if no parameters are supplied. It will also display messages that indicate what path the compilation phase is taking, based on the passed CHAIN tags.

```
/IFC ASM.LOAD
/*
/* THIS CHAIN REQUIRES AT LEAST ONE OF THE FOLLOWING
/* PARAMETERS:
/*
/*    ASM -- CHAIN FOR ASSEMBLIES
/*    LOAD -- CHAIN FOR PRODUCING MEMORY-IMAGE FILE
/*
/* AND ONE OR MORE OF THESE PARAMETERS:
/*
/* D1, D2 -- DISK 1 and DISK 2
/* ALL -- ALL FILES ON ALL DISKS
/* <name> -- NAME OF FILE
/*
/* THE FOLLOWING ARE OPTIONAL PARAMETERS
/*
/*    OPT -- ASSEMBLER OPTIONS
/*
/ABORT
/ELSE
/ IFS ASM
/* CHAIN FOR ASSEMBLING  PROGRAMS
/ XIF
/ IFS LOAD
/* CHAIN FOR MEMORY-FILE CREATION
/ XIF
/XIF
@SET,M 8
/IFS ALL,D1,PROG1,PROG2
@. INSERT DISK 1 INTO DRIVE 1 -- DEPRESS ANY KEY WHEN READY
/ IFS ALL,D1,PROG1
/* PROGRAM PROG1
```

```
/  IFS ASM
DEL PROG1.RO:1
RASM NOL,EQU,LIS,PROG1:1;R%OPT%O=PROG1:1
/  XIF
/  IFS LOAD
@TST,S EQ
@JMP SKIPPGM1
DEL PROG1.LO:1
RLOAD
IDON;BASE;CURP=\\$100;LOAD=PROG1:1
OBJA=PROG1:1
CURP=\\$100;LOAD=PROG1:1
MO=#LP;MAPF
EXIT
@LBL SKIPPGM1
/  XIF
/ XIF
/ IFS ALL,D1,PROG2
/* PROGRAM PROG2
/  IFS ASM
DEL PROG2.RO:1
RASM NOL,EQU,LIS,PROG2:1;R%OPT%O=PROG2:1
/  XIF
/  IFS LOAD
@TST,S EQ
@JMP END1
DEL PROG2.LO:1
RLOAD
IDON;BASE;CURP=\\$100;LOAD=PROG2:1
OBJA=PROG2:1
CURP=\\$100;LOAD=PROG2:1
MO=#LP;MAPF
EXIT
@LBL END1
/  XIF
/ XIF
/XIF
/IFS ALL,D2,PROG3,PROG4
@. INSERT DISK 2 INTO DRIVE 1 -- DEPRESS ANY KEY WHEN READY
/ IFS ALL,D2,PROG3
/* PROGRAM PROG3
/  IFS ASM
DEL PROG3.LX:1
RASM PROG3:1;%OPT%
/   XIF
/   IFS LOAD
@TST,S EQ
@JMP SKIPPGM3
DEL PROG3.LO:1
EXBIN PROG3:1
@LBL SKIPPGM3
/  XIF
/ XIF
/ IFS ALL,D2,PROG4
```

```
/* PROGRAM PROG4
/  IFS ASM
DEL PROG4.LX:1
RASM PROG4:1;%OPT%
/   XIF
/   IFS LOAD
@TST,S EQ
@JMP END2
DEL PROG4.LO:1
EXBIN PROG4:1
@LBL END2
/  XIF
/ XIF
/XIF
```

The tags ALL, D1, D2, PROG1, PROG2, PROG3, and PROG4 are used to identify which programs from the system of programs are to be selected by the CHAIN process. All programs from all diskettes can be selected by specifying ALL. A specific program can be selected by specifying its name: either PROG1, PROG2, PROG3, or PROG4. All programs on a specific diskette can be selected by specifying D1 or D2.

The tags ASM and LOAD are used to select what process the programs will go through. ASM specifies the programs will be assembled. LOAD specifies link/loading or object file conversion via EXBIN.

It should be noted that nested IFs have been indented (spaces between slash and IF) to indicate their level of nesting. This is optional, but makes the CHAIN file easier to understand. Prior to the assembly and link/load or object file conversion processes, a DEL command has been placed to ensure that the output file from the process does not exist. The first time that the CHAIN file is used, the DEL command will cause an error to occur; however, the SET operator has been used to inhibit CHAIN process aborting.

The TST operator is used after each assembly process to check for errors. If an error occurred, then the error status word will be non-zero in the portion indicated by the "S" designator. Thus, the test condition for being equal to zero will be false, causing the JMP to be executed. Therefore, if assembly errors occur, the link/load or object file conversion process will be bypassed since it would only generate an unusable file.

It should also be noted that the backslash character is used in the RLOAD command CURP. Thus, the CHAIN forcing character, which is also a backslash, must be entered.

The Operator Breakpoint is used to pause the CHAIN process. This allows the operator time to insert the proper diskette into drive one. Otherwise, if all programs from all diskettes were to be assembled, there might not be sufficient time for the operator to swap diskettes.

The following example illustrates what is displayed on the system console when the CHAIN is invoked without any parameters. Since this would produce an empty intermediate file, the condition is tested for and an appropriate message displayed. The name of the CHAIN file in the directory is SYSGEN.CF.

```
=CHAIN SYSGEN
 THIS CHAIN REQUIRES AT LEAST ONE OF THE FOLLOWING
 PARAMETERS:

    ASM -- CHAIN FOR ASSEMBLIES
    LOAD -- CHAIN FOR PRODUCING MEMORY-IMAGE FILE

 AND ONE OR MORE OF THESE PARAMETERS:

 D1, D2 -- DISK 1 and DISK 2
 ALL -- ALL FILES ON ALL DISKS
 <name> -- NAME OF FILE

 THE FOLLOWING ARE OPTIONAL PARAMETERS

    OPT -- ASSEMBLER OPTIONS

=
```

The next example uses the same CHAIN file again; however, this time the parameters for assembling (ASM), memory-image file creation (LOAD), and processing all files in the system (ALL) are specified. In addition, the options field of the assembler will be initialized with the value "LX" to produce a listing and a cross reference table on the line printer.

```
=CHAIN SYSGEN;ASM,LOAD,ALL,OPT%LX%
 CHAIN FOR ASSEMBLING  PROGRAMS
 CHAIN FOR MEMORY-FILE CREATION
 PROGRAM PROG1
 PROGRAM PROG2
 PROGRAM PROG3
 PROGRAM PROG4
@SET FOFF 0800
@. INSERT DISK 1 INTO DRIVE 1 -- DEPRESS ANY KEY WHEN READY
DEL PROG1.RO:1
PROG1   .RO:1 DELETED
RASM NOL,EQU,LIS,PROG1:1;RLXO=PROG1:1
MDOS MACROASSEMBLER 03.00
COPYRIGHT BY MOTOROLA 1977

@TST,F000 0000 0027
@JMP 2F29
```

```
DEL PROG2.RO:1
PROG2   .RO:1 DELETED
RASM NOL,EQU,LIS,PROG2:1;RLXO=PROG2:1
MDOS MACROASSEMBLER 03.00
COPYRIGHT BY MOTOROLA 1977

@TST,F000 0000 0027
@JMP 2F33
@. INSERT DISK 2 INTO DRIVE 1 -- DEPRESS ANY KEY WHEN READY
DEL PROG3.LX:1
PROG3   .LX:1 DELETED
RASM PROG3:1;LX
MDOS MACROASSEMBLER 03.00
COPYRIGHT BY MOTOROLA 1977

@TST,F000 0000 0027
@JMP 2F41
DEL PROG4.LX:1
PROG4   .LX:1 DELETED
RASM PROG4:1;LX
MDOS MACROASSEMBLER 03.00
COPYRIGHT BY MOTOROLA 1977

@TST,F000 0000 0027
@JMP 2F4B
END CHAIN
=
```

From the example above, it can be seen that even though the LOAD parameter was entered on the CHAIN command line, the process to create memory-image files was not performed. This resulted from the fact that the assembly process generated errors in each program. Had no errors occurred, the memory-image files would have been created. The operands of the Execution Operators have been converted into hexadecimal codes during the compilation to make it easier for the execution phase overlay to process the intermediate file.

The last example uses the same CHAIN file again; however, this time only a single program is processed, PROG3. The operator does not need to know on which diskette this program resides. The Operator Breakpoint is used to notify the operator when a diskette is to be inserted into drive one. In this example, no errors occurred during the assembly process since the memory-image file is created.

```
=CHAIN SYSGEN;ASM,LOAD,PROG3,OPT%LN=120%
 CHAIN FOR ASSEMBLING  PROGRAMS
 CHAIN FOR MEMORY-FILE CREATION
 PROGRAM PROG3
@SET FOFF 0800
@. INSERT DISK 2 INTO DRIVE 1 -- DEPRESS ANY KEY WHEN READY
DEL PROG3.LX:1
PROG3   .LX:1 DELETED
RASM PROG3:1;LN=120
MDOS MACROASSEMBLER 03.00
COPYRIGHT BY MOTOROLA 1977

@TST,F000 0000 0027
DEL PROG3.LO:1
PROG3   .LO:1 DELETED
EXBIN PROG3:1
@LBL 2F29
END CHAIN
=
```

CHAPTER 7

7. COPY COMMAND

The COPY command allows files to be copied from one diskette to another, from a diskette to another device, or from another device to a diskette. It is not possible to copy files between two non-diskette devices with the COPY command. Options exist for copy verification and for the use of non-standard devices.

7.1 Use

The COPY command is invoked with the following command line:

```
COPY <name 1>[,<name 2>] [;<options>]
```

where <name 1> is the name of a source file or source device, <name 2> is the name of a destination file or destination device, and <options> may specify the type of copying that is to be performed. The following options are valid. Their use is described explicitly in the next sections:

Option	Function
B	Perform both the copy and the verify processes when copying between two diskette files.
C	Use binary record conversion during the copy to a non-diskette device.
D=<name 3>[,]	Use a user-defined device driver instead of a standard MDOS-supported device driver during the copy or verify process. The driver is located in a diskette file <name 3>.
L	List errors on the line printer during file verification.
M	Go to debug monitor after loading user-defined device driver file.
N	Use non-file format mode for the non-diskette device.

V	Verify source and destination files. No copy is performed.
W	Use automatic overwrite if destination file already exists on diskette.

7.1.1 Diskette-to-diskette copying

In order to copy one diskette file into another, both <name 1> and <name 2> must be specified. The source file name specification, <name 1>, will be supplied with the default suffix "SA" and the default logical unit number zero if those quantities are not explicitly given. The destination file name specification, <name 2>, need only be specified with a file name, a suffix, or a logical unit number (or any combination thereof); however, at least one part of <name 2>'s file name specification must be entered. The unspecified parts of <name 2> will be supplied from the respective parts of <name 1>. Thus, if TESTPROG.SA:0 is to be copied to the diskette on drive one, then only the logical unit number need be specified for <name 2>, since the file name and suffix will be supplied from <name 1>:

```
COPY TESTPROG,:1
```

In this example the default values were first supplied for <name 1>, and then the default values supplied for <name 2>. There is no restriction in file format when copying from one diskette file into another.

Only the "B", "L", "V" and the "W" options are valid when copying between two diskette files. The "V" and "B" options, as well as the "V" and "W" options, are mutually exclusive. The "L" option is valid only valid with "V" or "B". The "W" option is used to allow the destination diskette file to be overwritten if its file name already exists. If, in the above example, the file name TESTPROG.SA:1 already existed, then COPY would have displayed the message

```
TESTPROG.SA:1 EXISTS.  OVERWRITE?
```

and await a response from the operator. A "Y" response would allow the COPY process to continue, and the file on drive 1 would be overwritten. Any other response would cause the COPY command to be terminated, and the destination file would be unaffected. The "W" option's presence will force the COPY command to attempt the copy if the destination file name exists, without prompting the operator.

The other options are explained in subsequent sections.

7.1.2 Diskette-to-device copying

If a diskette file is to be copied to another device, both <name 1> and <name 2> must be specified on the command line. The default assumptions for the source file are the same as in diskette-to-diskette copying; however, <name 2> must now indicate a destination device rather than a file. The following are valid device specifications that can be used for <name 2>:

Device Name	Associated Physical Device
#CN	Console printer
#CP	Console punch (record) device
#LP	Line printer
#UD	User-defined device

Unlike diskette-to-diskette copying, where <name 1> could be the name of any diskette file, <name 1> can only be an ASCII or binary record file (see section 24.3). Thus, not every diskette file can be copied to a non-diskette device. If memory-image files are to be copied to a non-diskette device, then they must first be converted via the BINEX command (Chapter 4).

There are two modes for copying files to a non-diskette device: file format mode and non-file format mode. The file format mode is the default mode that the COPY command uses. The file format mode will write one extra record to the device before any data records are copied from the file. This special record is called the File Descriptor Record (FDR) and serves the same purpose as a directory entry for diskette files: the FDR contains the diskette file's name, suffix and file format (see section 24.3). The "N" option inhibits the writing of the FDR to the output device, and is used to indicate the non-file format mode. Thus, if an FDR is to be written to the output device, the "N" option should be omitted; if an FDR should not be written, the "N" should be specified.

The output devices #CN and #LP can be used as the destination device in the diskette-to-device copy mode. However, the presence of the "N" option on the command line when copying to these devices has no effect. The #CN and #LP devices are not "file" devices since no FDR could ever be read from them. Thus, the COPY command will automatically force the non-file format mode to be in effect and suppress the writing of the FDR.

Some output devices cannot support eight-bit binary data. In such instances, the "C" option must be used when binary record files are being copied. The "C" option will cause the binary data to be converted into seven-bit ASCII data (see section 24.3) which can be handled by the device. The following table shows what the destination file format will be, based on the file format of the source file and the options specified:

Source File	Destination File
ASCII	ASCII.
Binary, no "C"	Binary, if supported by device; else ASCII-converted-binary.
Binary, "C"	ASCII-converted-binary.

In the non-file format mode ("N" option specified), only ASCII record files can be copied.

The "V" and "L" options are valid in this copy mode. The "W" and "B" options are invalid since no diskette file is being written to. The "D" and "M" options can be used, but only if the device #UD is specified for <name 2> (see section 7.2).

7.1.3 Device-to-diskette copying

If a file is to be copied from another device to the diskette, then <name 1> is required; however, depending on the copy mode chosen (file format or non-file format) <name 2> is optional. If the file format mode is to be used (no "N" option specified), then <name 2> can be omitted. In such cases, the file name to be used for the diskette file is taken out of the FDR; however, if <name 2> is specified (still no "N" option), the source device will be read until an FDR is found that matches <name 2> before the copy takes place. In other words, in the file format mode, <name 2> indicates the name of the file on the device which will be copied to diskette. The name of the file can only be changed with the NAME command (Chapter 20) after the file has been copied to diskette.

If the "N" option is specified, then no FDR processing will be performed. Therefore, <name 2> must indicate the diskette file that is to be written to.

In either case ("N" option or no "N" option), <name 1> will specify the source device, and <name 2> will specify the destination diskette file. The default values "SA" and zero will be supplied for <name 2>'s suffix and logical unit number, respectively, if they are not explicitly entered by the operator. The valid device specifications that can be used for <name 1> are:

Device Name	Associated Physical Device
#CR	Console reader device
#UD	User-defined device
#HR	EXORtape (see section 7.6)

Only ASCII record files can be copied using the "N" option. If paper tapes or cassettes have been generated in a non-MDOS environment, they must conform to the MDOS format for ASCII record files (section 24.3). Most important is the record termination sequence. Each record must end with a carriage return, line feed, and null character combination. Otherwise, leading data characters from the subsequent record can be dropped. Next in importance is the end-of-file indicator. The tape should contain the ASCII end-of-file record (section 24.3) or generate a timeout condition (section of erased or blank tape) to cause the console reader to stop.

If binary records are to be copied, then the file format mode must be used. The binary record copied to diskette will always be in the binary format, never in the ASCII-converted-binary format. The FDR contains the format of the file on the device. Thus, the conversion from ASCII-converted-binary to binary is performed automatically. The "C" option, therefore, is invalid with this form of the COPY command.

The "W" option can be specified to automatically overwrite the diskette file (<name 2>) if it already exists. The "D" and "M" options are only valid if <name 1> is the #UD device. The "B" option is invalid, but the "V" and "L" options are valid. The "L" option can only be specified if "V" is specified.

7.1.4 Verification

The "V" option can be used to compare two files against each other. No file copying will take place if this option is specified. The "V" option is valid with all three modes of the COPY command: diskette-to-diskette, diskette-to-device, and device-to-diskette. If, however, a device specification is being used for either <name 1> or <name 2>, it must be a device that supports input. For example, even though a file from diskette can be copied to the line printer or the console punch, the "V" option is invalid for those specific devices.

The verification process will display the message

VERIFY IN PROGRESS

while the verification is taking place. If the files being compared are both diskette files, then the parts of the files that do not compare will be displayed in the following format:

```
SECTOR nnnn
  OFFSET xx  SRC-yy  DST-zz
```

where "nnnn" is the logical sector number of the file, "xx" is the offset into the sector, "yy" is the source file's byte (<name 1>), and "zz" is the destination file's byte (<name 2>). All values are displayed in hexadecimal.

If memory-image files are being compared, then the files' RIBs will also be included in the verify process to ensure that the load information matches.

In the event that only a sector number is displayed during the verify process (no byte discrepancies shown), then the two files are of different lengths. The files are identical through the end-of-file of the shorter file. The sector number displayed is one sector beyond the end-of-file of the shorter file.

When verifying a diskette file with a non-diskette file, the mis-comparisons between the two files are displayed in a slightly different format as shown below:

```
RECORD mmmmm
  OFFSET kkk  SRC-yy  DST-zz
```

where "mmmmm" is the physical record number in the diskette file (in decimal), "kkk" is the offset within the record (also in decimal), and "yy" and "zz" are the same as described above. If the two files being compared are of different lengths, and if they are identical through the end-of-file of the shorter file, then the offset portion of the error message will not be printed.

The "L" option can be used in conjunction with the "V" option to cause the mis-comparisons between the two files to be printed on the line printer instead of the console.

7.1.5 Automatic verification

The "B" option can be used when copying from one diskette file to another to automatically cause the two files to be verified after the copy has taken place. Section 7.1.1 describes the copy process between two diskette files. Section 7.1.4 describes the verification process.

For example, the following command line:

```
COPY TESTPROG,:1;B
```

performs exactly the same function as the following two command lines:

```
COPY TESTPROG,:1
COPY TESTPROG,:1;V
```

The "L" option can be specified along with the "B" option to cause any errors during the verification process to be printed on the line printer instead of the console.

7.2 User-Defined Devices

The COPY command allows the user to specify his own device drivers. Such device drivers must follow the specifications described in this section. The device name #UD is used on the COPY command line to indicate that a user-defined device driver is specified in the options field. The "D" option is used to pass the file name of the device driver to the COPY command. The "D" option has the following format:

```
D=<name 3>[,]
```

where the terminating comma is optional. If the "D" option is the last option specified, then the comma need not be supplied; however, if other options follow the "D" option, then the comma must be present to serve as a terminator for the file name specification of the device driver.

The device driver must be in a file that has the memory-image format. <name 3> is a complete file name specification. The default values of "LO" and zero will be supplied for the suffix and for the logical unit number. The device driver must meet the requirements set forth in section 26.2 for entry points, for calling sequences, and for return conditions. In addition, the following criteria must be satisfied:

1. The first twelve bytes of the device driver must contain the Controller Descriptor Block (CDB) for the device (Chapter 26).

2. The device driver must not overlay the COPY command. It is suggested that the device driver load as close to the end of the COPY command as possible. This address should be $3000.

It may be necessary to set breakpoints in the user-defined device driver to ensure that it is working properly. The "M" option will cause the COPY command to enter the debug monitor after the device driver has been loaded into memory. This feature is especially useful during the initial testing of the device driver.

The "M" option cannot be used without the "D" option. If the "M" option is present, the debug monitor will display one of the following messages depending on the version of the EXbug firmware. The first message is displayed by EXbug 1, the second by EXbug 2:

```
BKPT ERROR
P-2126 X-2161 A-0D B-80 C-C0 S-226F
*

SWI P-2126 X-2161 A-0D B-80 C-C0 S-226F
*E
```

These messages indicate that the user-defined device driver has just been loaded into memory. The actual numbers in the pseudo-registers may differ and are inconsequential. The purpose of going to the debug monitor is to allow the user to set breakpoints at critical places in the device driver to verify that it is working properly. After the breakpoints are set, control is returned to the COPY command by entering the EXbug command

```
;P
```

Then, when the user-defined device driver is accessed by the COPY command, the set breakpoints will allow the user to check the device driver's functions.

7.3 COPY Mode Summary

The following table summarizes the requirements for the three COPY command modes. The following symbols are used in the table:

Symbol	Meaning
DK-DK	Diskette-to-diskette copying
DK-DV	Diskette-to-device copying
DV-DK	Device-to-diskette copying
R	Required
O	Optional
F	File name
D	Device name

COPY Mode	Valid Options	<name 1>	<name 2>	Restrictions
DK-DK	B,L,V,W	R,F	R,F	V and W options are mutually exclusive. V and B options are mutually exclusive. L is only valid with V or B.
DK-DV	C,D,L,M,N,V	R,F	R,D	N option implies ASCII record format. C option implies binary record format. D option implies #UD device name. V option implies input device. L option is only valid with V.
DV-DK	D,L,M,N,V,W	R,D	O,F	D option implies #UD device name. V option implies input device. W and V options are mutually exclusive. N option requires <name 2>. <name 2> causes search for FDR on device if no N option. L option is only valid with V.

7.4 Messages

The following messages can be displayed by the COPY command. Not all messages are error messages, although error messages are included in the list. The standard error messages that can be displayed by all commands are not listed here.

<name> EXISTS. OVERWRITE?

The file named by <name> already exists in the directory. Before overwriting the file, the operator must respond with a "Y". Any other response will terminate the COPY command.

VERIFY IN PROGRESS

The "V" or "B" option was specified on the command line. The two files are being compared.

SECTOR nnnn

Two diskette files did not compare during the verify process. "nnnn" indicates the logical sector number (hexadecimal) of the failure.

RECORD mmmmm

Two files did not compare during the verify process. One file is on diskette, the other file is not. "mmmmm" indicates the physical record number (decimal) in the diskette file where the failure occurred. The LIST command (Chapter 17) can be used to display the records in a file with their physical record numbers.

OFFSET {xx or kkk} SRC-yy DST-zz

This message indicates which bytes within a logical sector or within a physical record of the two files being compared do not match. The offset "xx" is hexadecimal if comparing diskette files. The offset "kkk" is decimal if comparing a diskette file with a non-diskette file. The byte in the source file is shown as "yy". The byte in the destination file is shown as "zz".

7.5 Examples

The following examples have been separated into the three COPY modes as illustrated in the table of section 7.3.

7.5.1 Diskette-to-diskette example

The following command line

```
COPY PROGS.RO:2,.RN:1
```

will copy the file PROGS.RO from drive two into the file PROGS.RN on drive one. A user response is required to continue the copy if the file on drive one already exists. The user response can be suppressed, regardless of whether the file on drive one exists, by adding the "W" option as shown:

```
COPY PROGS.RO:2,.RN:1;W
```

No error results if the file on drive one does not exist. In either case, if the logical unit number had been omitted from the <name 2> specification, the file would have been created on drive two.

The next example illustrates the display of the bytes which do not compare when two files are compared with the "V" option.

```
=COPY BLAKJACK:1,:0;V
VERIFY IN PROGRESS
SECTOR 0000
  OFFSET 10  SRC-31  DST-02
  OFFSET 11  SRC-34  DST-03
  OFFSET 12  SRC-2B  DST-04
  OFFSET 13  SRC-54  DST-05
  OFFSET 14  SRC-53  DST-06
  OFFSET 15  SRC-31  DST-07
  OFFSET 16  SRC-38  DST-08
  OFFSET 17  SRC-0D  DST-09
  OFFSET 18  SRC-2B  DST-00
  OFFSET 76  SRC-45  DST-55
  OFFSET 77  SRC-4C  DST-66
  OFFSET 78  SRC-53  DST-77
  OFFSET 79  SRC-45  DST-88
=
```

7.5.2 Diskette-to-device example

The following command line

```
COPY TEXT,#CP
```

will copy the file TEXT.SA from drive zero to the console punch (record) device. The punch device must be ready to receive data before the command line is entered. Since no "N" option was specified, an FDR record will be written before any data records are copied.

Most frequently, however, the user will copy object files to the console punch for loading via the EXbug LOAD command. In such cases, the FDR should not be written to the punch device. Then, the following command line should be used:

```
COPY TESTPROG.LX,#CP;N
```

where TESTPROG.LX is the output file from an assembly process (in the EXbug-loadable format). The "N" option suppresses the writing of the FDR. If the TESTPROG.LX file had a non-ASCII file format, then an error message would have been displayed.

The next example illustrates how source listings that have been directed to diskette by the assembler (RASM) can be printed on the line printer. Since the file already contains page formatting, the LIST command would cause the printed copy to look strange since LIST imposes its own page formatting. Thus, the COPY command should be used to print source listings from diskette:

```
COPY TESTPROG.AL,#LP
```

The console printer, #CN, could be used instead of #LP just as well. The "N" option is not used in this example because the printer (either #LP or #CN) is not a "file" device. Copying to a "non-file" device will automatically set the non-file format mode. If the "N" option were specified in such a case, no error would result. It would only be a redundant request.

The last example illustrates how the command line would appear if a user-defined device driver is used:

```
COPY TESTPROG.LX,#UD;ND=TAPE
```

The user device is indicated via the #UD. The "D" option must be present. Otherwise, an error would result. The file TAPE.LO on drive zero will be used as the device driver file for the user device.

7.5.3 Device-to-diskette example

Once a file has been copied to the console punch with an FDR, it can be verified or copied back to diskette without having to specify its name. The following command line:

```
COPY #CR
```

will cause COPY to search for the first FDR on the console reader device. Once it is found, the file name contained in the FDR will be used for <name 2>. If the file name does not exist in the directory, it will be created before receiving the data records from the console reader. If the file name already exists in the directory, a message will be displayed by the COPY command asking the operator if the file should be overwritten.

The command line

```
COPY #CR,TESTPROG.LX;VL
```

on the other hand, will search the console reader device for an FDR that contains the file name TESTPROG.LX. The same file name must also exist in the directory of the diskette in drive zero so that the verification can take place. Any mis-comparisons between the two files will be printed on the line printer.

If the user has files in a format that can be read by the console reader device, but which have no FDR, the "N" option must be used to copy those files to diskette:

```
COPY #CR,FILE1;N
```

In this example, the file indicated by <name 2> will receive the data from the console reader. No search is performed for an FDR. If the file is on paper tape, then it must be in a format that is compatible with the MDOS ASCII records (section 24.3). That is, a carriage return, line feed, null sequence must terminate each record. Otherwise, one or two data characters from the subsequent records may be lost. This results from the fact that the detection of a carriage return forces the device driver to turn off the reader. In the amount of time it takes to turn the reader off, one or two frames (characters) may have passed by the read head.

The following example illustrates how a user would set breakpoints in his device driver to verify that it is performing the functions of a driver as specified in section 26.2. The example shows EXbug 1 as the debug monitor:

```
=COPY #UD,TEST;NMD=DRIVER
BKPT ERROR
P-2126 X-2161 A-0D B-80 C-C0 S-226F
*3056;V
*3064;V
*3082;V
*;P
```

The EXbug monitor is given control after the user's driver file, DRIVER.LO:0, has been loaded into memory by the COPY command. The user then sets three breakpoints (the addresses for the breakpoints are, of course, meaningless in this example -- they serve only to illustrate that breakpoints are set). The ";P" command then returns control to the COPY command. When one of the breakpoints is reached during the execution of the COPY command, the normal breakpoint display will be seen. At that point, the user can examine registers, memory, etc., to ensure that his driver is functioning properly.

7.6 COPY with EXORtape Reader

The COPY command will provide users with EXORtape paper tape readers an additional device type. Users with paper tape readers that are similar to the EXORtape can also use the COPY command without the requirement of a user-defined device driver.

The EXORtape reader interfaces through a PIA on the EXORdisk I interface module. The following steps must be followed to permit the EXORdisk I Interface Module to be accessed by the COPY command.

1. No boards may reside in the EXORciser that respond to addresses at locations $E000-E7FF, inclusive.

2. The M68IFC's base address must be changed via the five-position microswitch so that:

   S5 is closed,
   S4 is closed,
   S3 is closed,
   S2 is open,
   S1 is open.

3. The M68IFC must be inserted into the EXORciser's card cage with power off on the entire system.

4. The EXORtape should then be connected via its cable to P3 of the Interface Module. The COPY command can now use the EXORtape reader as an input device through the device name

   #HR

   in all instances that an input device is valid.

For users without the M68IFC but with a compatible paper tape reader (see "M68R680 EXORtape User's Guide"), a standard PIA interface can be used if the PIA is configured to the address $E404.

CHAPTER 8

8. DEL COMMAND

The DEL command is used to remove MDOS file names from a directory and to deallocate all space that belongs to the deleted entry. A single file name, a list of file names, or a family of file names may be deleted with a single command.

8.1 Use

The DEL command is invoked with the following command line:

```
DEL [<name 1> [,.....,<name n>]] [;<options>]
```

where each <name i> (i = 1 to n) can specify a specific file name or a family of file names. The <options> field can be one or both of the following option letters:

Option	Function
S	When family name specifications are used include entries in the directory with the "system" attribute.
Y	Automatically delete all file names of a family. Do not ask the operator if each member of the family should be deleted.

The list of file names specified on the command line is processed from left to right. As the list is processed, the file names are searched for in the directory specified by the logical unit numbers. If no logical unit number is explicitly entered by the operator, zero will be supplied as a default. No default suffix is supplied.

File names which are deleted by accident via the DEL command may be restored if no other commands that affect the directory or the allocation table have been run after the deletion. The REPAIR command description (Chapter 22) contains an example of the procedure that must be followed to restore such file names. It is recommended, however, that files be configured with delete protection or that adequate backup copies be kept as an alternative to restoring file names in this manner, especially since this restoration will only work if the error is detected immediately after the file name is deleted.

8.1.1 Single file name deletion

A single file name is deleted by specifying its name as the only parameter on the command line. Both the file's name and suffix must be supplied by the operator. If the file name is not found in directory of the indicated (or default) drive, the message

```
<name> DOES NOT EXIST
```

will be displayed. If the file name is found in the directory and if the file is unprotected, the message

```
<name> DELETED
```

will be displayed to verify that the file name has been deleted. If the file is protected, the message

```
<name> IS PROTECTED
```

will be shown. In this case, the file name is not deleted.

8.1.2 Multiple file name deletion

Multiple file names can be deleted by specifying more than one name on the command line. Multiple file names must be separated by commas or some other valid delimiter. Like single file name deletion, multiple file name deletion will cause one message to be displayed for each file name entered on the command line to indicate whether it was deleted, whether it did not exist, or whether it was protected and could not be deleted. As many file names as can be accommodated on the command line can be deleted at one time.

8.1.3 Family deletion

In either the single or the multiple file name modes, a file name specification can contain the family indicator. The family of file names specified by such a designation will then be considered for deletion. Unlike the single and multiple file name modes, the operator will be prompted with the message

```
DELETE <name> ?
```

for each file name that belongs to the family. This permits the operator to see all family members before they are deleted. A "Y" response to the above prompt will cause the file name to be deleted. Any other response will inhibit deletion of that family member. Protected file names within the family will be displayed with the standard protection message indicating that they cannot be deleted.

Without the presence of any options, only file names lacking the "system" attribute will be considered as eligible for deletion in the family mode.

A special case of the family mode is the absence of any file name specification. In this case, the DEL command processes the command line as if the following file name specification had been given

.:0

which will make all non-system file names on drive zero eligible for deletion.

A logical unit number may be entered on the command line as the only part of the file name specification. In this case, the family *.* will be eligible for deletion. Instead of the default drive, however, the operator entered logical unit number will be used.

8.2 Options

The "S" option is used to include file names with the system attribute in the family mode of deletion. Normally, the family mode excludes such file names. The "S" option has no effect in the single or multiple file name modes.

The "Y" option will inhibit the DEL command's prompt asking if each family member is be deleted. The effect of specifying the "Y" option is to give an automatic "Y" response to the prompt; however, neither the prompt nor the automatic response are displayed. The deletion messages indicating which members of the family were deleted or protected will still be shown.

The "Y" and "S" options can be used concurrently.

8.3 Messages

The following messages can be displayed by the DEL command. Not all messages are error messages; however, error messages are included in the list. The standard error messages that can be displayed by all commands are not shown here.

<name> DOES NOT EXIST

This message is displayed for each file name on the command line that is not found in a directory.

<name> DELETED

This message is displayed for each file name that is deleted. It is displayed in single, multiple, or family file name modes.

DELETE <name> ?

This prompt is displayed whenever a family of file names containing at least one member has been specified on the command line, and the "Y" option is not present. The operator must respond with a "Y" to delete each member of the family.

<name> IS PROTECTED

This message is displayed for each file name that cannot be deleted due to its protection attributes. The message is displayed in single, multiple, or family file name modes.

8.4 Examples

To delete a single file name called TESTPROG.SA on drive zero, the following command line would be entered:

```
DEL TESTPROG.SA
```

The DEL command would then display the message

```
TESTPROG.SA:0 DELETED
```

after it has deleted the file name. To delete the three file names: SCRATCH.SA on drive one, TEST.LX on drive two, and PROG.RO on drive zero, the following command line would be used. The system's responses are also shown:

```
=DEL SCRATCH.SA:1,TEST.LX:2,PROG.RO
SCRATCH .SA:1 DELETED
TEST    .LX:2 DELETED
PROG    .RO:0 DELETED
=
```

The following command line

```
DEL *.SA,*.SA:1
```

will search for all file names without the system attribute and with the suffix "SA" on drives zero and one. After a file name is found, its complete name will be displayed along with the prompt asking if the file is to be deleted. The operator has complete control over the deletion of any member of the family since a response is required for every member.

To delete all unprotected file names on drive three without having to respond "Y" to each prompt, the following command line could be used:

```
DEL :3;YS or DEL *.*:3;YS
```

In this case, unprotected file names with and without the system attribute will be deleted.

CHAPTER 9

9. DIR COMMAND

The DIR command displays MDOS file names from the directory. The entire directory or selective parts of it may be displayed. Options exist for displaying an entire directory entry, its allocation information, and for directing the output to the printer.

9.1 Use

The DIR command is invoked with the following command line:

```
DIR [<name>] [;<options>]
```

where <name> can specify a specific file name or a family of file names. The <options> field can be one or more of the following option letters:

Option	Function
L	Direct output to line printer.
S	Include file names with the "system" attribute when displaying a family.
E	Display the entire directory entry for each file name.
A	Display the associated allocation information along with the entire directory entry.

Whenever the DIR command is invoked, regardless of options or file name specifications, the drive number and the ID from the diskette in the specified or default drive will be displayed as a heading. This heading will serve to identify the subsequent output. The heading has the following format:

```
DRIVE : i   DISK I.D. : xxxxxxxx
```

where "i" will be the logical unit number zero, one, two, or three, and "xxxxxxxx" will be the eight-character ID that was assigned to the diskette via the DOSGEN command (Chapter 10) or the BACKUP command (Chapter 3).

Normally, without the presence of any options, the directory entry specified by <name> will be searched for and its name and suffix displayed on the system console. The following sections explain the various options that can be specified on the command line.

9.1.1 Families

If <name> contains a family indicator in either the suffix or the file name portion of the file name specification, the entire family of file names will be searched for in the directory and displayed. If no <name> is specified at all, the default family "*.*:0" will be used. If only a logical unit number is specified, the family "*.*" on the indicated logical unit will be used. If the "S" option has not been specified, only file names without the "system" attribute will be included in the display. This eliminates the display of all MDOS system files and commands.

When <name> contains a family indicator (explicitly or by default), the file names are displayed in the order in which they are found in the directory. A file name's position in the directory is a function of its name and suffix. Appendix G describes in more detail how names are placed into the directory; however, it should be noted here that when a file's name or suffix is changed, its position in the directory may also change. Thus, when the directory is shown at different times, the order of the displayed names may differ.

9.1.2 System files

File names with the "system" attribute will be included in the output of the DIR command if the "S" option is specified on the command line. If a specific file name is being searched for (<name> does not contain the family indicator), then the "S" option has no effect.

The effect of the "S" option is identical to its effect with the DEL command (Chapter 8). Thus, the same family of file names displayed by the DIR command will be affected by the DEL command (if invoked with similar command line parameters). This feature allows an operator to see ahead of time what family of file names will be affected by the DEL command.

9.1.3 Entire directory entry

Normally, DIR will only display a file's name and suffix. The "E" option can be used to cause the entire directory entry to be displayed. The presence of the "E" option will cause each displayed line from the DIR command to look like:

```
FFFFFFFF.SS  WDSCN# RRRR ZZZZ DD
```

where the symbols take on the following meanings:

Symbol	Meaning
FFFFFFFF	File name
SS	Suffix
WDSCN#	Attributes
RRRR	RIB address
ZZZZ	File size
DD	Directory entry number

The file name and suffix are, of course, obvious. The file attributes are always displayed as a six-character field. The presence of a letter or number in a specific position of the attribute field indicates that the particular attribute applies to the file. A period in a position of the attribute field indicates that the particular attribute does not apply. The following letters (and positions) are defined in the attribute field:

```
W D S C N #
: : : : : :
: : : : : :.. File format (0=user defined,
: : : : :                  2=memory-image,
: : : : :                  3=binary record,
: : : : :                  5=ASCII record,
: : : : :                  7=ASCII-converted-
: : : : :                    binary record)
: : : : :.... Non-compressed spaces
: : : :...... Contiguous space allocation
: : :........ System file
: :.......... Delete protection
:............ Write protection
```

Thus, if the "W" is displayed, the file is write protected. If no "W" is displayed, the file is not write protected; if the "C" is displayed, the file is allocated contiguous space; if no "C" is displayed, the file is segmented; etc.

The remainder of the fields of the directory entry contain only hexadecimal numbers. The RRRR field contains the physical sector number of the first sector of the file. This sector is known as the file's Retrieval Information Block (RIB). It is described in detail in Chapter 24. The RIB contains the allocation information that describes where the remainder of the file is located on diskette.

The ZZZZ field contains the file's size in sectors. Due to the allocation scheme used by MDOS, this field will always be a multiple of the basic unit of allocation (see Chapter 24). The size is, therefore, the physical size of the file. The logical file size, or the number of sectors from the beginning to the end-of-file indicator, may be smaller.

The DD field is an eight-bit coded field that describes the directory entry's physical position within the directory. It is interpreted as follows:

```
  7   6   5   4   3   2   1   0
 -------------------------------
 !                  !          !
 -------------------------------
            :              :
            :              :..... Position within sector
            :                     (0-7)
            :
            :.................... Physical sector number
                                  ($3-$16)
```

9.1.4 Segment descriptors

If the "A" option is specified on the command line, then in addition to having the entire directory entry displayed for each file name, the file's allocation information will also be shown. The allocation information is contained in the file's RIB and describes where each segment of the file is located on the diskette. This information is displayed following the complete directory entry. One line is shown for each segment of the file. The format of the allocation information is

```
ss pppp zzz
```

where "ss" is the number of the segment (0-56, decimal), "pppp" is the physical sector number of the sector that starts the segment (hexadecimal), and "zzz" is the size of the segment in sectors (hexadecimal). For example, a directory entry could appear as follows:

```
FORLB    .RO    .DS..3 0490 0088 75    00 0490 080
                                       01 0510 008
```

The file FORLB.RO consists of two segments. The first segment starts in physical sector $490 and is $80 sectors long. The second segment starts in physical sector $510 and is 8 sectors long. The file's physical size is $88 sectors.

9.1.5 Other options

Normally, the output from the DIR command is displayed on the system console. The "L" option can be used to direct the output to the line printer. The format of the display is the same. Like other MDOS commands that direct output to the line printer, the paging will be preserved by the DIR command. Thus, once the paper in the printer has been aligned, it will remain aligned after a directory has been printed.

9.2 Messages

The following messages can be displayed by the DIR command. The standard error messages that can be displayed by all commands are not listed here.

DRIVE : i DISK I.D. : xxxxxxxx

This is the directory command's heading line that is displayed each time the command is invoked. "i" is the logical unit number. "xxxxxxxx" is the diskette's ID that was assigned to it when it was generated.

TOTAL NUMBER OF SECTORS : dddd/$hhh

This message is displayed if either the "E" or the "A" option was specified on the command line, and if one or more directory entries were found. It gives the total number of sectors that is allocated to the files whose names are displayed. "dddd" is the decimal value of the total. "hhh" is the hexadecimal value of the total. This message is displayed after all file names have been printed.

TOTAL DIRECTORY ENTRIES SHOWN ddd/$hh

This message is shown at the end of each directory search that found at least one file name. It gives the total number of directory entries included in the display. "ddd" gives the decimal value of the total. "hh" gives the hexadecimal value of the total.

NO DIRECTORY ENTRY FOUND

This message is displayed if the <name> specified on the command line does not result in any matches with directory entries on the diskette. If <name> contains a family indicator, the message means that no members of that family were found on the diskette.

NO SDW'S

This message will only be displayed if the "A" option is in effect and if an invalid RIB is found for a file. The message is displayed in place of the segment descriptor information that appears to the right of the entire directory entry. When such a message is seen, it indicates that the file has probably been damaged. Since no segment descriptors are found in the RIB, the file will not be accessible any longer. The REPAIR command (Chapter 22) should be used to check the remainder of the diskette, as well as to remove the erroneous file.

NO TERMINATOR FOUND IN FILE'S R.I.B.

This message can only be displayed if the "A" option was specified on the command line. Like the previous message, this one indicates that a file's RIB has been damaged. It indicates that the terminator was missing from the RIB. The allocation information displayed for the file is meaningless since 56 segment descriptors have been displayed. The file's content is no longer accessible. The REPAIR command (Chapter 22) should be used to check the remainder of the diskette, as well as to remove the erroneous file.

9.3 Examples

When the DIR command is invoked without any options on a newly received system diskette, this is what will be seen on the system console:

```
=DIR
DRIVE : 0   DISK I.D. : MDOS0300
NO DIRECTORY ENTRY FOUND
=
```

A new system diskette has only file names with the "system" attribute. Those file names will be excluded from the directory search unless the "S" option is specified. Thus, the default family *.*:0 (since no <name> was specified) contains no members. Using the "S" option on the above example would result in the following display:

```
=DIR ;S
DRIVE : 0   DISK I.D. : MDOS0300
BINEX   .CM
LIST    .CM
MDOSOV0 .SY
DIR     .CM
MERGE   .CM
MDOSOV4 .SY
MDOS    .SY
MDOSOV6 .SY
FREE    .CM
EQU     .SA
ROLLOUT .CM
DUMP    .CM
EXBIN   .CM
NAME    .CM
MDOSOV1 .SY
PATCH   .CM
BLOKEDIT.CM
LOAD    .CM
MDOSOV3 .SY
MDOSER  .SY
DEL     .CM
ECHO    .CM
CHAIN   .CM
BACKUP  .CM
REPAIR  .CM
MDOSOV5 .SY
DOSGEN  .CM
EMCOPY  .CM
COPY    .CM
FORMAT  .CM
TOTAL NUMBER OF ENTRIES SHOWN : 030/$1E
=
```

No file attributes or file sizes are displayed since neither the "E" nor the "A" option was specified.

If a diskette is in drive one which contains MDOS-Supported software products (see Appendix H), the following shows how the directory entries with suffix "CM" on that drive can be displayed:

```
=DIR *.CM:1;AS
DRIVE : 1   DISK I.D. : EDIT0300
ASM     .CM   .DSC.2 00B0 002C 70   00 00B0 02C
EDIT    .CM   .DSC.2 0230 0018 72   00 0230 018
TOTAL NUMBER OF SECTORS : 0068/$044
TOTAL DIRECTORY ENTRIES SHOWN : 002/$02
=
```

Both the EDIT and ASM commands reside on drive one. From their attributes it can be seen that those files are not write protected, are delete protected, are system files, are contigously allocated on diskette, and are of file format 2 (memory-image). The ASM command is located starting at physical sector $B0 and is $2C sectors long. The EDIT command is located starting at sector $230 and is $18 sectors long. Both files have only one segment descriptor. The ASM command's file name is the first directory entry in physical sector $E (found by shifting its directory entry number to the right three bit positions). The EDIT command's directory entry is in the same sector, but is the third entry in that sector.

In all of the above examples, the "L" option could have been used in addition to any other options to direct the output from the DIR command to the line printer.

It is recommended that a copy of the directory printout containing the entire directory entry and the allocation information be kept with each diskette. Since files can dynamically expand and contract, their location on diskette may change. If something happens to the diskette to damage the directory, there is no way to recover any information from it if a prior printout has not been saved. Normally, the printout will never be needed, but as a precaution it is indispensable.

CHAPTER 10

10. DOSGEN COMMAND

The DOSGEN command allows specialized MDOS diskettes to be prepared. Diskettes that have bad sectors can have those sectors locked out so that the diskette can be used in an MDOS environment. DOSGEN will also create all system tables and files on the generated diskette. The DOSGEN command can be used to generate system diskettes on either single-sided or appropriately formatted double-sided diskettes.

10.1 Use

New single-sided diskettes, or single-sided diskettes never before used on an MDOS system, must first be prepared for use with MDOS. One way to generate a new MDOS diskette is by invoking the BACKUP command (Chapter 3); however, the BACKUP command does not perform the write/read test that can be invoked via DOSGEN; nor is there the guarantee that all system files are copied to the destination diskette since the operator can selectively prevent files from being copied. Another way to generate a new MDOS diskette is by invoking the DOSGEN command from an already up-and-running MDOS system.

DOSGEN does not create the sector addressing information. Single-sided diskettes usually come pre-formatted in an IBM-3740-similar format with the established sector addressing information. Double-sided diskettes, however, must be formatted with the FORMAT command (Chapter 15), since the double-sided format required by an EXORdisk III is a non-standard single-density format. In either case, whether single- or double-sided, other information must be written on a new diskette in order to make it recognizable by MDOS. DOSGEN creates the system tables required by MDOS (see Chapter 24). These tables include a skeleton directory; a bit map showing which sectors of the diskette are available for space allocation; a lockout map showing which sectors of the diskette are bad or locked out by the user; and an identification sector containing a name to identify the diskette, the generation date, and the MDOS version number. The DOSGEN command also copies across the required MDOS family of system files which must be present on any diskette used in the MDOS environment. These files and tables must not be moved or changed in any way other than through the DOSGEN command and two other commands: BACKUP (Chapter 3) and REPAIR (Chapter 22). Optionally, the MDOS commands may be copied to the diskette.

The DOSGEN command is invoked with the following command line:

```
DOSGEN [:<unit>] [;<options>]
```

where <unit> is the logical unit number (1-3) of the drive containing the diskette to be DOSGENed, and <options> can be one or both of the option letters described below:

Option	Function
T	Perform write/read test.
U	Generate minimum system (user diskette).

If <unit> is not specified, logical unit one will be used as a default. Logical unit zero cannot be DOSGENed.

The diskette to be DOSGENed must be placed in the logical unit specified on the command line (logical unit one, if no <unit> was specified). DOSGEN will respond with the following question asking if <unit> contains a diskette that can be written to:

```
DOSGEN DRIVE <unit> ?
```

The response should be the letter "Y", if the diskette in the indicated <unit> is to be DOSGENed. Any other response will terminate the DOSGEN command and return control to MDOS. In this case, the diskette in <unit> is not affected.

If a "Y" is given as a response, certain information for the diskette's identification sector must be supplied by the operator. This information is entered in response to the following DOSGEN prompts:

Prompt	Operator Input
DISK NAME:	An alphanumeric name, a maximum of 8 characters in length, which will appear on subsequent heading lines from the DIR and FREE commands. The name must begin with an alphabetic character.
DATE (MMDDYY):	The date of generation in six-digit, numeric form as indicated by the parenthetical inset.
USER NAME:	A maximum of twenty displayable characters used for descriptive information only.

The version and revision numbers of MDOS will be automatically supplied by the DOSGEN command.

The operator is then given a chance to lock out an area of the diskette. This area will not be accessed by any MDOS command or function since it is an allocated block without a RIB. This permits the operator to set aside a part of the diskette for his own use. All MDOS information must be in files in order to be accessed by MDOS. The message

```
LOCKOUT ADDITIONAL SECTORS?
```

is displayed to allow sector lockout. An "N" response will cause DOSGEN to continue with the next step; no sectors will be locked out, leaving as much diskette space as possible for conventional file use. A "Y" response will cause the following messages to be shown:

```
ENTER STARTING SECTOR (HHH):
ENTER ENDING SECTOR (HHH):
```

The operator can respond with only a carriage return, which will casue the lockout request to be bypassed. Otherwise, the response must be a valid hexadecimal sector number for each prompt. The sector numbers entered must meet the following criteria in order to cause the specified diskette area to be locked out:

1. The sector numbers must be hexadecimal.

2. The starting sector number must be the physical sector number of the first cluster to be locked out. The ending sector number must be the physical sector number of the last cluster to be locked out.

3. The starting sector number must be less than or equal to the ending sector number. If the two numbers are equal, only one cluster will be locked out.

4. Both sector numbers must be greater than $18 and less than $7D0 if generating a single-sided diskette, or greater than $18 and less than $FA4 if generating a double-sided diskette. In either case, the locked out area should be located such that the largest block of free space resides in sectors with numbers less than that of the start of the locked out area.

DOSGEN will then write the ID sector, an initialized allocation table, a lockout table, an empty directory, and a Bootblock to the destination diskette. Normally, DOSGEN will then copy all files that have the "system" attribute from the diskette in drive zero to the destination diskette. When DOSGEN is finished, a complete MDOS system will have been generated on the destination diskette.

10.2 Diskette Surface Test

If DOSGEN is invoked with the "T" option, a write/read test will be performed to ensure that the sectors on the destination drive are good. Any sectors which fail the write/read test will be flagged with the deleted data mark. If sectors cannot be flagged in this manner, the diskette cannot be generated. Such diskettes may be made usable again by using the FORMAT command (Chapter 15). If a sector can be marked as bad, then the cluster to which the bad sector belongs will be automatically locked out from MDOS usage. This individual cluster lockout is independent of the area of diskette that can be locked out by the operator. It will allow diskettes with bad spots to be generated and made usable as MDOS system diskettes.

Diskettes that have such bad sectors can be used as normal diskettes with the following exception. The BACKUP command should not be invoked without a Main Option (unless the "D" option is used) to make a complete copy of the allocated space. Without the "D" option, the complete copy process will abort if a fatal read error occurs. Since the complete copy is based on the allocation table, it is inevitable that the bad sectors locked out via DOSGEN will be read. Thus, the resultant copy of the diskette will always be incomplete. Therefore, BACKUP should always be run with the "R" option to force file reorganization. In this manner, the bad sectors will never be read since they are not a part of any allocated file.

Diskettes which have had bad sectors locked out should not be used as the destination diskette with BACKUP.

If sectors get locked out into which the MDOS system files normally are copied (in the first several cylinders) the DOSGEN process will fail. Such diskettes cannot be used as MDOS system diskettes unless the FORMAT command (Chapter 15) can be used to correctly rewrite the bad sectors.

10.3 Minimum System Generation

If the DOSGEN command is invoked with the "U" option, the resultant diskette will not contain any of the MDOS commands from drive zero. Only the MDOS family of system files that must reside on every diskette used in an MDOS environment will be copied. The "U" option is useful in generating user diskettes which are to contain only data files and will almost always be used in drives other than zero.

10.4 Messages

The following messages can be displayed by the DOSGEN command. Not all messages are error messages, although error messages are included in the list. The standard error messages that can be displayed by all commands are not listed here.

DOSGEN DRIVE <unit> ?

> This message permits the operator to exit the DOSGEN command or allows him time to insert a scratch diskette before continuing. A "Y" response will cause DOSGEN to continue. Any other response will cause control to be returned to MDOS.

DISK NAME:

> This prompt is used to obtain the eight character ID field that is subsequently displayed by all DIR and FREE commands when used on the generated diskette. The ID field has the same format as an MDOS file name.

DATE (MMDDYY):

> This prompt is used to obtain the date of diskette generation. The date must be six numeric characters.

USER NAME:

> This prompt is used to obtain the descriptive information for the ID sector. Up to twenty displayable characters may be entered.

LOCKOUT ADDITIONAL SECTORS?

> This message allows the user to specify whether or not he wishes to reserve a block of the diskette for his own use. The block will be excluded from use by MDOS. A "Y" response will cause the next two prompts to be issued. Any other response will cause the lockout request to be bypassed.

ENTER STARTING SECTOR (HHH):

This prompt is used to obtain the starting hexadecimal sector number of the first cluster that is to be locked out.

ENTER ENDING SECTOR (HHH):

This prompt is used to obtain the starting hexadecimal sector number of the last cluster that is to be locked out.

ABOVE SECTORS HAVE BEEN LOCKED OUT

This message will be displayed if valid starting and ending sector numbers have been specified for the area to be locked out.

INVALID SECTOR NUMBER

This message is displayed if either the starting or ending sector number does not meet the criteria set forth in section 10.1. The operator is given another chance to enter the sector number range.

SECTOR nnnn LOCKED OUT

When a bad sector is detected during the write/read test ("T" option), this message is displayed to indicate which sector failed the test. The "nnnn" is the hexadecimal, physical sector number. The cluster in which the sector resides will be automatically locked out.

COPYING FILE <name>

This message is displayed for each system file as it is being copied to the destination diskette. It serves only to monitor the DOSGEN operation.

MDOS.SY DOES NOT START AT SECTOR $18

This message indicates that the destination diskette cannot be generated. Either the operator or the write/read test locked out sectors which prevented the resident operating system file MDOS.SY from residing at the specified physical location. If the operator locked out those sectors, the diskette should be regenerated with a different range locked out. If the write/read test locked out those sectors, the diskette is unusable as a system diskette. Chapter 15 should be consulted for making such a diskette usable again.

10.5 Examples

The following example shows the operator-system interaction during a DOSGEN process:

```
=DOSGEN ;TU
DOSGEN DRIVE 1? Y
DISK NAME: USER0001
DATE (MMDDYY): 072578
USER NAME: SYSTEM DEVELOPMENT 1
LOCKOUT ADDITIONAL SECTORS? N
COPYING FILE MDOS    .SY
COPYING FILE MDOSOV0 .SY
COPYING FILE MDOSOV1 .SY
COPYING FILE MDOSOV2 .SY
COPYING FILE MDOSOV3 .SY
COPYING FILE MDOSOV4 .SY
COPYING FILE MDOSOV5 .SY
COPYING FILE MDOSOV6 .SY
COPYING FILE MDOSER  .SY
=
```

The diskette to be generated was tested with the write/read test ("T" option) to ensure that all sectors were good. A minimum system was generated ("U" option). The new ID, USER0001, the generation date, July 25, 1978, and the descriptive information, SYSTEM DEVELOPMENT 1, were placed into the ID sector. Since no additional sectors were locked out, DOSGEN proceeded to copy the MDOS family of system files that must reside on each diskette.

The following example shows what might happen if a bad diskette is used in the generation process:

```
=DOSGEN :2;T
DOSGEN DRIVE 2? Y
DISK NAME: USER0002
DATE (MMDDYY): 072578
USER NAME: TEST SYSTEM
SECTOR 0030 LOCKED OUT
SECTOR 0031 LOCKED OUT
SECTOR 0056 LOCKET OUT
LOCKOUT ADDITIONAL SECTORS? N
COPYING FILE MDOS    .SY
MDOS.SY DOES NOT START AT SECTOR $18
=
```

Three bad sectors were found during the write/read test. When the MDOS family of files was copied, it was detected that the locked out sectors prevented the resident operating system file MDOS.SY from residing at the specified physical location. If the operator locked out those sectors, the diskette should be regenerated with a different range locked out. If the write/read test locked out those sectors, the diskette is unusable as a system diskette. Chapter 15 should be consulted for making such a diskette usable again.

CHAPTER 11

11. DUMP COMMAND

The DUMP command allows the user to examine the entire contents of any physical sector on the diskette. The sector can be displayed on either the system console or the printer. The display contains both the hexadecimal and the ASCII equivalent of every byte in the sector. The DUMP command allows the opening of files so that they can be examined using logical sector numbers. Sectors can also be moved into a temporary buffer where changes can be applied before they are written back to diskette.

11.1 Use

The DUMP command is invoked with the following command line:

```
DUMP [<name>]
```

where the presence of the optional file name determines the initial mode of operation. The DUMP command is an interactive program that has its own command structure. Once DUMP is running, it will display a colon (:) as an input prompt whenever it is ready to accept a command from the operator. Commands exist for selecting logical units, for opening and closing files, for displaying sectors, for modifying single sectors, and for displaying the directory and cluster allocation table.

11.1.1 Physical Mode of operation

If no <name> is specified on the command line, or if <name> only consists of a logical unit number, then DUMP will be in the "Physical Mode" when it displays its input prompt. The heading

```
PHYSICAL MODE
```

will be displayed prior to the prompt the first time that DUMP is activated. From that point on, it is the operator's task to keep track of which mode of operation DUMP is in. The Physical Mode of operation means that all subsequent commands referring to sector numbers will be interpreted as physical sector numbers. The Physical Mode of operation remains active as long as no files are opened.

If no <name> is specified on the command line, DUMP will default to logical unit zero for all subsequent commands. The unit will remain selected until another unit selection command is issued by the operator. To override the default unit selected, the operator can specify only a logical unit number on the command line in place of <name>. In this case, the initial unit selected will be the logical unit number entered on the command line (0-3). The logical unit number must be preceded by a colon, the logical unit number delimiter.

When a logical unit number is specified on the command line, the diskette to be inspected with DUMP should already be in the indicated drive. If no diskette is in the specified drive, the message

```
**PROM I/O ERROR-STATUS=33 AT h DRIVE i-PSN j
```

is displayed, indicating that the drive is not ready. The "U" command (section 11.2.2) must be used to restore the diskette drive after the diskette has been inserted.

11.1.2 Logical Mode of operation

If a <name> which exists in the directory is specified on the command line, then DUMP will be in the "Logical Mode" of operation when it displays the input prompt. <name> must contain an explicit suffix. No default suffix is supplied by the DUMP command. The logical unit number, however, is given a default value of zero if it is not specified on the command line.

If the <name> cannot be found in the directory, a standard error message will be displayed indicating that the file name does not exist. In that case, the Physical Mode of operation will be entered; however, the physical mode message will not be displayed since the error message has already indicated that the file could not be opened.

The Logical Mode of operation means that all subsequent references to sector numbers will be interpreted as logical sector numbers of the file <name>. A special convention is used when referring to the RIB of a file. The logical sector number of the RIB is FFFF. Since logical sector number zero is the first data sector of the file, the RIB has a logical sector number of minus one (FFFF). DUMP will remain in the Logical Mode of operation until the file is closed or until another unit is selected.

11.1.3 Sector change buffer

Certain commands can reference a temporary sector buffer known as the "sector change buffer". This buffer is large enough to accommodate one sector from diskette. The sector change buffer can be used in either mode of operation. The contents of the sector change buffer will not be destroyed or altered unless the operator issues a command to do so.

Associated with the sector change buffer is a "sector address validity flag". This flag indicates whether or not a critical command has been executed between the time the sector change buffer was read into and the time that the sector change buffer is written back to diskette. When the sector change buffer is read into, a sector address is specified. This address is retained so that if the sector is to be written back to diskette, the address need not be specified again; however, certain actions, described under the separate command descriptions that follow, can cause the sector address to be invalidated. Then, the writing of the sector change buffer requires a respecification of the sector address into which the buffer is to be written.

The sector change buffer is very useful in modifying sectors. Most frequently, the sector change buffer is used in conjunction with the REPAIR command (Chapter 22) to fix critical system tables which have been found in error. Of course, this procedure is not recommended unless the operator has detailed knowledge of the system table structure. Situations do arise when critical file information can only be recovered through the manual reconstruction of certain system tables. The DUMP command's sector change buffer provides the ideal means for doing this.

11.2 DUMP Command Set

Each command to DUMP must be entered by the operator after the input prompt (:) is displayed on the system console. Like all MDOS input, all DUMP commands must be terminated by a carriage return. In the following command descriptions these symbols are used:

Symbol	Meaning
m,n	Both "m" and "n" are one to four digit hexadecimal numbers used for specifying a sector number or a cluster number.
i	"i" is a one digit number used for referring to the logical unit number.
b	"b" is a one or two digit hexadecimal number used as an offset into the sector change buffer.

c	"c" is a one or two digit hexadecimal number.
a	"a" is an ASCII character.
<str>	"<str>" is a string of elements separated by commas. Each element can be a "c" or a group of "a"s enclosed in double quotes.
<cr>	"<cr>" is a carriage return.

11.2.1 Quit -- Q

The Q command is used to terminate DUMP and return control to MDOS. The format of the Q command is simply the letter "Q". Any information in the sector change buffer is lost. The Q command is valid in either mode of operation. If a file is open, it is unaffected by the execution of the Q command.

11.2.2 Select logical unit -- U

The U command is used to select the logical unit number. The format of the U command is

```
U i
```

where "i" can be any of the digits 0-3. The U command is valid in either mode of operation; however, if the current mode of operation is the Logical Mode, then the file that is open will be automatically closed. After the U command is executed, the Physical Mode of operation will be in effect. The sector address associated with the sector change buffer is invalidated by the U command.

If DUMP was invoked with only a logical unit number on the command line, and if a diskette was not in the drive at the time DUMP was invoked, then the U command must be used to restore the diskette drive after a diskette has been inserted into the drive. If this procedure is not followed, timeout errors may occur on that drive since the head may not have been properly positioned to cylinder zero.

11.2.3 Open diskette file -- O

The O command is used to open a file and thereby enter the Logical Mode of operation. The format of the O command is

```
O <name>
```

where <name> consists of at least a file name and a suffix. If no logical unit number is specified for <name>, the last logical unit selected via the U command will be used as a default. If a logical unit number is specified for <name>, then it will become the selected unit number even if the Physical Mode of operation is entered later. If a file is currently open, it will be automatically closed when the O command is executed. If the file <name> is not found, then the Physical Mode of operation will be in effect after an error message is displayed. The sector address associated with the sector change buffer is invalidated by the O command.

11.2.4 Close diskette file -- C

The C command is used to close the file that is currently open. The format of the close command is simply the letter "C". If the current mode of operation is already the Physical Mode, then no action results from the execution of the C command. If a file is open, then the Physical Mode of operation will be entered after the file is closed. The sector address associated with the sector change buffer is invalidated by the C command.

11.2.5 Show sector -- S

The S command is used to display a sector's contents on the system console. There are several forms of the S command.

Command	Effect
S	Display the contents of the sector change buffer.
SB	Display the contents of the Cluster Allocation Table. The SB command is only valid in the Physical Mode of operation.
S m[,n]	Display the contents of sector "m" or the contents of sectors "m" through "n". The values of "m" and "n" are either physical or logical sector numbers depending on the current mode of operation.
SD [m[,n]]	Display the contents of the directory sectors. The entire directory will be displayed if no "m" and no "n" are given. Otherwise, the logical sector "m" or the logical sectors "m" through "n" of the directory will be displayed. The SD

	command is only valid in the Physical Mode of operation.
SC m[,n]	Display the contents of cluster "m" or the contents of clusters "m" through "n". In this case, "m" and "n" are physical cluster numbers rather than physical sector numbers. The SC command is only valid in the Physical Mode of operation. For each cluster, four sectors will be displayed.

The format of a displayed sector is shown in section 11.4.

11.2.6 Print sector -- L

The L command is used to print a sector's contents on the line printer. There are several forms of the L command.

Command	Effect
L	Print the contents of the sector change buffer.
LB	Print the contents of the Cluster Allocation Table. The LB command is only valid in the Physical Mode of operation.
L m[,n]	Print the contents of sector "m" or the contents of sectors "m" through "n". The values of "m" and "n" are either physical or logical sector numbers depending on the current mode of operation.
LD [m[,n]]	Print the contents of the directory sectors. The entire directory will be printed if no "m" and no "n" are given. Otherwise, the logical sector "m" or the logical sectors "m" through "n" of the directory will be printed. The LD command is only valid in the Physical Mode of operation.
LC m[,n]	Print the contents of cluster "m" or the contents of clusters "m" through "n". In this case, "m" and "n" are physical cluster numbers rather than physical sector numbers. The LC command is only valid in the Physical Mode of operation. For each cluster, four sectors will be printed.

The format of a printed sector is shown in section 11.4.

11.2.7 Read sector into change buffer -- R

The R command is used to read a specified sector into the sector change buffer. Once the sector is in the change buffer, changes can be applied to it. The sector change buffer can then be written back to diskette. The R command has several forms. Each form of the R command will initialize the sector address validity flag associated with the sector change buffer. This flag allows the change buffer to be re-written to the same sector from which it was read without specifying the sector address again.

Command	Effect
RB	Read the Cluster Allocation Table into the sector change buffer. The RB command is only valid in the Physical Mode of operation.
RD m	Read the specified logical sector of the directory into the change buffer. The RD command is only valid in the Physical Mode of operation.
R m	Read the specified sector into the change buffer. The current mode of operation will determine whether "m" is a logical or a physical sector number.

11.2.8 Write change buffer into sector -- W

The W command is used to write the contents of the sector change buffer into a sector. The W command has several forms.

Command	Effect
W	Write the change buffer back into the sector from which it was originally read. This form of the W command is only valid if the U, O, C, or F commands have not been used since the sector change buffer was read into.

CAUTION: THE FOLLOWING FORMS OF THE W COMMAND CAN DESTROY SYSTEM TABLES OR USER DATA IF USED INDISCRIMINATELY. USE OF THE FOLLOWING FORMS SHOULD BE RESTRICTED TO DISKETTE REPAIR

FUNCTIONS.

WB	Write the contents of the sector change buffer into the Cluster Allocation Table. The WB command is only valid in the Physical Mode of operation.
WD m	Write the contents of the sector change buffer into logical sector "m" of the directory. The WD command is only valid in the Physical Mode of operation.
W m	Write the contents of the sector change buffer into sector "m". The current mode of operation will determine whether "m" is a logical or a physical sector number. If the current mode of operation is the Logical Mode, then writing past the end-of-file sector will cause the CAT and the file's RIB to be updated in the event that additional diskette space is allocated.

11.2.9 Fill change buffer -- F

The F command is used to fill the sector change buffer with a certain bit pattern or a certain ASCII character. The format of the F command is:

F c or F "a"

where the first form will fill the buffer with the hexadecimal bit pattern "c", and the second form will fill the buffer with the character "a". The sector address associated with the sector change buffer is invalidated by the F command.

11.2.10 Examine/change sector buffer

A special command is used for examining/changing the individual bytes of the sector change buffer. In order to gain access to a specific byte of the sector change buffer, the offset must be specified in the following manner:

b/<cr>

where "b" is a hexadecimal number ($00-7F). The slash character causes the location at offset "b" to be "opened" and its contents displayed. After a particular location has been opened in this manner, the change buffer can be examined or changed a byte at a time by using the following commands:

```
[<str>]<cr>

or

[<str>]^<cr>

or

[<str>]/<cr>
```

The element string <str> will cause successive bytes of the change buffer to be changed to the respective values of <str>. If <str> is not specified, no changes will be applied to the change buffer. The <cr> only will cause the next offset of the change buffer to be opened and displayed. The "^<cr>" will cause the previous location of the change buffer to be opened and displayed. The "/<cr>" will cause the current location to be closed and the examine/change mode to be terminated.

The initial command used to enter the examine/change mode can also take on the following forms:

```
b/<str><cr>
```

which will cause the locations of the change buffer starting at offset "b" to be changed according to the string <str>. Then the location after the last one changed will be displayed. The operator can then enter other examine/change commands. If the initial command has the form:

```
b/<str>/<cr>
```

then the same function will be performed as in the previous command; however, instead of remaining in the examine/change mode, the normal command mode is entered.

11.3 Messages

The following messages can be displayed by the DUMP command. Not all messages are error messages; however, error messages are included in the list. The standard error messages that can be displayed by all commands are not listed here.

WHAT?

The command issued in response to the DUMP input prompt was not recognized. A new input prompt is displayed.

SYNTAX ERROR

The command issued in response to the DUMP input prompt was recognized; however, it was parameterized illegally. A new input prompt is displayed. The command has not been processed.

MODE ERROR

The B, C, or D qualifier was used with the S, L, R, or W command while in the Logical Mode of operation. These forms of the commands are only valid in the Physical Mode.

BOUNDARY ERROR

The offset "b" in the examine/change command was outside the range of the sector change buffer ($00-7F), or a subsequent location was to be displayed which was outside the range of the sector change buffer. The examine/change mode is terminated.

INVALID SECTOR ADDRESS

The sector address associated with the sector change buffer has been invalidated. In this case, the W command cannot be used without specifying a sector address.

PHYSICAL MODE

This message is displayed initially when the DUMP command is entered and the mode of operation is the Physical Mode. If the message is not displayed and if no error messages are shown, the Logical Mode of operation is initially in effect. Subsequent mode changes must be kept track of by the operator.

** 21 END OF FILE

This message indicates that a logical sector beyond the logical end-of-file was to be read with one of the DUMP commands. In the Logical Mode of operation only sectors allocated to the file can be read.

**PROM I/O ERROR-STATUS=36 AT h DRIVE i-PSN j

This message indicates that a physical sector beyond the end of the diskette was to be accessed with one of the DUMP commands. In the Physical Mode of operation, only sectors 0-$7D1 (single-sided) or sectors 0-$FA3 (double-sided) can be accessed. A memory address (only meaningful for system diagnostics) is substituted for the letter "h"; the logical unit number is substituted for the letter "i"; and the physical sector number (PSN) at which the error occurred is substituted for the letter "j".

The display format of a sector's contents is shown in section 11.4. The messages associated with that display are explained here. The sector display will contain headings to identify what sector is being displayed.

"UNIT" will always specify the currently selected logical unit number.

The heading "CHANGE BUFFER" will be displayed if the sector change buffer is being shown.

The heading "CLUSTER ALLOCATION MAP" indicates that the B qualifier was used with either the S or L command. Likewise, the heading "DIRECTORY" indicates that the D qualifier was used with either the S or L command.

The heading "FILE=xxxxxxxx.xx" indicates that the Logical Mode of operation is in effect. The file's name and suffix are displayed to the right of the equal sign.

"PSN" gives the displayed sector's physical sector number, regardless of the mode of operation. "LSN", or logical sector number, is only shown if the directory is being displayed or if the current mode of operation is the Logical Mode.

The digits 00-70 down the left edge of the display are the hexadecimal offsets into the sector. The contents of the sector are shown both in hexadecimal and in displayable ASCII. Non-displayable characters are printed as periods (.).

If sectors are displayed on the line printer, they will appear five sectors per page. The unit number and associated heading will be automatically printed at the top of each page. The paper alignment will be restored once the Q command is issued.

11.4 Examples

The following example shows how the Cluster Allocation Table is displayed with the DUMP command (a single-sided diskette is used).

```
=DUMP
PHYSICAL MODE
: SB
  UNIT=0   CLUSTER ALLOCATION MAP


    PSN=0001
00  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  ................
10  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  ................
20  FF FF FF FF FF FF FF FF  FF FF FF FF FF F0 00 00  ................
30  00 00 00 03 FF FF FF FF  FF 00 00 00 00 00 0F FF  ................
40  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  ................
50  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  ................
60  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  ................
70  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  ................
: Q
=
```

The next example illustrates how the logical sectors zero through three of the directory are displayed.

```
=DUMP
PHYSICAL MODE
: SD 0,3
  UNIT=0   DIRECTORY


    PSN=0003                 LSN=0000
00  42 49 4E 45 58 20 20 20  43 4D 01 4C 72 00 00 00  BINEX   CM.Lr...
10  42 55 49 4C 44 20 20 20  43 4D 01 6C 72 00 00 00  BUILD   CM.lr...
20  4C 49 53 54 20 20 20 20  43 4D 02 F8 72 00 00 00  LIST    CM..r...
30  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
40  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
50  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
60  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
70  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................

    PSN=0004                 LSN=0001
00  4D 44 4F 53 4F 56 30 20  53 59 00 5C 72 00 00 00  MDOSOVO SY.\r...
10  46 4F 52 54 20 20 20 20  43 4D 02 74 72 00 00 00  FORT    CM.tr...
20  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
30  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
40  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
50  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
60  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
70  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
```

```
    PSN=0005                        LSN=0002
00  44 49 52 20 20 20 20 20   43 4D 01 B8 72 00 00 00   DIR     CM..r...
10  4D 45 52 47 45 20 20 20   43 4D 03 28 72 00 00 00   MERGE   CM.(r...
20  52 4C 4F 41 44 20 20 20   43 4D 04 1C 72 00 00 00   RLOAD   CM..r...
30  00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00   ................
40  00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00   ................
50  00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00   ................
60  00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00   ................
70  00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00   ................

    PSN=0006                        LSN=0003
00  4D 44 4F 53 4F 56 34 20   53 59 00 88 72 00 00 00   MDOSOV4 SY..r...
10  00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00   ................
20  00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00   ................
30  00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00   ................
40  00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00   ................
50  00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00   ................
60  00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00   ................
70  00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00   ................
: Q
=
```

In the following example, the DUMP command is invoked with a file name on the command line; however, the file name does not exist as it is specified (i.e., a suffix of spaces). The Physical Mode of operation is entered automatically. Then the O command is used to open the file. Subsequently, two sectors of the file are displayed. The logical sector numbers allow a user to examine the file's contents without knowing where the file is physically located on the diskette.

```
=DUMP MDOSER
** 04 FILE NAME NOT FOUND
: O MDOSER.SY
: S 1,2
 UNIT=0  FILE=MDOSER  .SY


    PSN=00A6                        LSN=0001
00  81 30 36 81 44 55 50 4C   49 43 41 54 45 81 46 49   .06.DUPLICATE.FI
10  4C 45 81 4E 41 4D 45 0D   30 44 81 30 37 81 4F 50   LE.NAME.0D.07.OP
20  54 49 4F 4E 81 43 4F 4E   46 4C 49 43 54 0D 33 30   TION.CONFLICT.30
30  81 30 38 81 43 48 41 49   4E 81 41 42 4F 52 54 45   .08.CHAIN.ABORTE
40  44 81 42 59 81 42 52 45   41 4B 81 4B 45 59 0D 33   D.BY.BREAK.KEY.3
50  31 81 30 39 81 43 48 41   49 4E 81 41 42 4F 52 54   1.09.CHAIN.ABORT
60  45 44 81 42 59 81 53 59   53 54 45 4D 81 45 52 52   ED.BY.SYSTEM.ERR
70  4F 52 81 53 54 41 54 55   53 81 57 4F 52 44 0D 31   OR.STATUS.WORD.1

    PSN=00A7                        LSN=0002
00  43 81 31 30 81 46 49 4C   45 81 49 53 81 44 45 4C   C.10.FILE.IS.DEL
10  45 54 45 81 50 52 4F 54   45 43 54 45 44 0D 32 34   ETE.PROTECTED.24
20  81 31 31 81 44 45 56 49   43 45 81 4E 4F 54 81 52   .11.DEVICE.NOT.R
30  45 41 44 59 0D 30 45 81   31 32 81 49 4E 56 41 4C   EADY.0E.12.INVAL
40  49 44 81 54 59 50 45 81   4F 46 81 4F 42 4A 45 43   ID.TYPE.OF.OBJEC
```

```
50  54 81 46 49 4C 45 0D 30  46 81 31 33 81 49 4E 56  T.FILE.OF.13.INV
60  41 4C 49 44 81 4C 4F 41  44 81 41 44 44 52 45 53  ALID.LOAD.ADDRES
70  53 0D 31 33 81 31 34 81  49 4E 56 41 4C 49 44 81  S.13.14.INVALID.
: Q
=
```

The following example illustrates how the DUMP command can be used to "fix" part of the MDOS system tables that were found to be in error by the REPAIR command (Chapter 22). No discussion is given here of the REPAIR command; however, the example does show what the REPAIR command displayed insofar as diagnostic messages are concerned. These messages contain the required information needed by the operator so that the DUMP command can be used to "fix" the bad sector. The REPAIR command could show the following on the system console:

```
=REPAIR
DISK ID:   MDOS0300
VERSION:   03
REVISION:  00
DATE:      072578
USER:      SYS DEVELOPMENT DRV0
06  03 01  TESTPROG.SA  05BC  0581 0000
ILLEGAL ATTRIBUTE OR UNUSED BYTES.  DELETE? N
33 GOOD FILES  00 FILES WITH BAD RIBS
RECONSTRUCTED C.A.T. MATCHES DISK
=
```

The first few lines show the contents of the ID sector. The line that begins with "06 03 01" shows the contents of a directory entry that has been found in error. The subsequent line shows the error that REPAIR detected. The error is in the attribute bytes of the directory entry. Chapter 22 describes the format of the displayed directory entry. With that information, the operator knows that the attribute field is displayed as "0581". The error is in the least significant byte of this field. It should be zero, not "81" as shown. From the other information displayed, it can be seen that this directory entry is the second entry (01) in the third sector (03) of the directory. With that information the DUMP command is used to read the sector containing the bad directory entry into the sector change buffer. The buffer is modified so that the "81" becomes a "00". In the following example, the sector change buffer is displayed both before and after the modification.

Such repair functions must be performed with extreme caution. The REPAIR command should always be run again after a system sector has been changed in this way to ensure that the change was made correctly.

```
=DUMP
PHYSICAL MODE
: RD 3
: S
  CHANGE BUFFER

    PSN=0006
00   4D 44 4F 53 4F 56 34 20  53 59 00 88 72 00 00 00  MDOSOV4 SY..r...
10   54 45 53 54 50 52 4F 47  53 41 05 BC 05 81 00 00  TESTPROGSA......
20   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
30   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
40   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
50   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
60   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
70   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
: 18/
18 53
19 41
1A 05
1B BC
1C 05
1D 81 00/
: S
  CHANGE BUFFER

    PSN=0006
00   4D 44 4F 53 4F 56 34 20  53 59 00 88 72 00 00 00  MDOSOV4 SY..r...
10   54 45 53 54 50 52 4F 47  53 41 05 BC 05 00 00 00  TESTPROGSA......
20   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
30   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
40   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
50   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
60   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
70   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
: W
: Q
=
```

CHAPTER 12

12. ECHO COMMAND

The ECHO command can only be used on an EXORciser II system. ECHO causes all subsequent input/output that is directed to the system console to also be printed on the line printer. The ECHO command is also used to stop echoing console I/O on the printer.

12.1 Use

The ECHO command is invoked with the following command line:

```
ECHO [;<options>]
```

where <options> can be the letter "N". If the ECHO command is invoked without any options, then all subsequent input and output to the system console via the MDOS console driver or the EXbug entry points will be duplicated on the line printer. The line printer will continue to receive a copy of all console I/O until the ECHO command is invoked with the "N" option.

The "N" option will turn off the echo feature. No paging is performed. Thus, if paper alignment is critical, it will have to be manually reset after the echo feature is disabled.

12.2 Messages

The following messages can be displayed by the ECHO command.

ECHO NOT AVAILABLE WITH EXBUG 1

> The ECHO command was invoked on an EXORciser I system. The command has no effect on such systems.

** 11 DEVICE NOT READY

> The printer was not ready when the ECHO command was invoked. The command has had no effect on the system. The printer must be readied and the ECHO command invoked again if the echo feature is to be enabled.

CHAPTER 13

13. EMCOPY COMMAND

The EMCOPY command allows files from a user's EDOS 2 system diskette to be copied to and catalogued on an MDOS diskette. Options exist for copying the entire diskette, selected files, or single files.

13.1 Use

The EMCOPY command is invoked with the following command line:

```
EMCOPY [<name 1>][,<name 2>] [;<options>]
```

where <name 1> is the name of an EDOS file, <name 2> can be a new name that is to be used for <name 1> on the MDOS diskette, and <options> can be one or more of the option letters defined below. If neither of the two file names is entered on the command line, then an <options> specification must be present. The following option letters are available. They are described in detail in the following sections.

Option	Function
A	File is of the ASCII record format.
R	File is of the binary record format as created by the Macro Assembler with the OPT REL option.
D	Set the delete protection on the MDOS file.
C	Create the MDOS file with contiguous space allocation.
S	Copy selected files from the EDOS diskette.
E	Copy the entire EDOS diskette.

For each of the different ways that EMCOPY can be used, the EDOS diskette must always be in drive one and the MDOS diskette in drive zero, regardless of whether a two-drive or a four-drive system is being used.

13.1.1 Single file copy

If a single EDOS file is to be copied to the MDOS diskette, its name must be specified as <name 1>. Only EDOS file names that meet the MDOS criteria for valid file names can be copied (see section 2.7.1). Since EDOS file names are only five characters long and have no suffixes, <name 1> is not specified with a suffix. Only the first five characters of <name 1> will be used to search the EDOS directory. A logical unit number should not be specified for <name 1>. The options "E" or "S" cannot be specified on the command line if only the single file <name 1> is to be copied. An error will be displayed if <name 1> cannot be found in the EDOS directory.

If no <name 2> is given on the command line, then an MDOS file with the name of <name 1> and the default suffix "ED" will be used as the destination file on drive zero. The default suffix can be overridden by specifying only a suffix for <name 2>. The default name can also be overridden by specifying a file name for <name 2>.

In either case, whether an explicit or a default <name 2> is used, a file with that name must not already exist on the MDOS diskette. A standard error message will be displayed if <name 2> already exists.

If no option or if the "A" option is specified on the command line, EMCOPY will assume that the EDOS file is in the ASCII record format. The "R" option can be used to copy EDOS files that were created by the EDOS Macro Assembler with the relocatable option (OPT REL). Obviously, "R" and "A" cannot be specified at the same time. If the EDOS file is found with the "Permanent Attribute" set, then the MDOS file will be automatically created with delete protection. The delete protection can be explicitly set for the MDOS file by using the "D" option on the command line.

13.1.2 Entire diskette copy

To copy all valid EDOS files from drive one to the MDOS diskette in drive zero, no file name specification must be given for <name 1>, no file name must be given for <name 2> (however, a suffix can be specified), and the "E" option must be specified.

The EDOS diskette will have its entire directory searched, one entry at a time, for valid (MDOS compatible) file names. When a valid name is found, it will be given the default suffix "ED" or the explicit suffix specified by <name 2>, and copied to the MDOS diskette. Of course, a file with that name cannot already exist on the MDOS diskette. This process is repeated until all entries in the EDOS directory have been examined.

As file names are processed from the EDOS directory one of the following two messages will be displayed for each file name. The message

```
COPYING FILE: <name>
```

indicates that the EDOS file identified by <name> is being transferred to the MDOS diskette. The message

```
<name>
** 25 INVALID FILE NAME
```

indicates that the file <name> does not have a valid MDOS file name and cannot be copied. If the file is to be copied, it must first be renamed on an EDOS system using the RENAM command.

The "C", "D", "R", or "A" options can be specified on the command line. These options, explained in the previous section, can be used to assign attributes to all files copied from the EDOS diskette. If no options are specified, then the MDOS files will use segmented allocation and be of the ASCII record format. The delete protection will automatically be set for files with the "Permanent Attribute" on the EDOS diskette.

The "S" option cannot be specified at the same time as the "E" option.

13.1.3 Selected file copy

To copy only selected files from the EDOS diskette, the "S" option must be specified on the command line. Nothing can be specified for <name 1> or <name 2> if the "S" option is used. Basically, the selected file copy mode works like the entire diskette copy mode; however, the operator can assign different attributes and suffixes to each file, as well as deciding whether or not a particular file is to be copied at all. During the selected file copy mode, as valid file names are found in the EDOS directory, the message

```
COPY <name> ?
```

will be displayed. The operator must respond with a "Y" if the file is to be copied to the MDOS diskette. Any other response will cause that file to be bypassed and not copied. The next valid file name will then be searched for.

If a "Y" response is given to the above prompt, EMCOPY will display two additional prompts:

```
SUFFIX?
ATTRIBUTES?
```

The operator can assign an explicit suffix by entering it after the "SUFFIX?" prompt, and he can assign explicit attributes by entering the appropirate attribute letters (A, C, D, R) after the "ATTRIBUTES?" prompt. The default suffix "ED" and the default attribute "A" can be assigned by responding with only a carriage return. If an invalid attribute is entered by the operator, the "ATTRIBUTES?" prompt will be redisplayed, forcing the operator to enter new attributes. This procedure will continue until all entries from the EDOS directory have been processed. At that time the message

```
NO MORE FILES
```

will be displayed and control returned to MDOS.

13.2 File Differences Between EDOS and MDOS

Both EDOS and MDOS systems support the ASCII and the binary record format. The ASCII record format is primarily used for source program files and object program files in the EXbug-loadable format. The binary record format is used primarily for the relocatable object output files created by the Macro Assembler, RASM.

The EMCOPY command will transfer either type of file on a sector-by-sector basis. Thus, after a file is copied to the MDOS diskette, its sectors are still in the same internal format; however, when an ASCII record file is processed by the MDOS editor, it will be altered. Multiple spaces will be compressed into a single byte, and the carriage return, line feed, null sequence that terminates all ASCII records on EDOS files will be replaced by a single carriage return. Thus, the resultant MDOS file will be significantly smaller than its original EDOS form.

Space compression is, of course, not performed on the binary record files; however, were the same object file to be produced by the MDOS Macro Assembler, it would not be identical to its EDOS counterpart. The carriage return, line feed, null sequence would have been replaced by a single carriage return.

13.3 Messages

The following messages can be displayed by the EMCOPY command. Not all messages are error messages, although error messages are included in the list. The standard error messages that can be displayed by all commands are not listed here.

COPYING FILE: <name>

During the entire diskette copy mode, this message monitors which files are being copied to the MDOS diskette.

COPY <name> ?

During the selected file copy mode, this prompt allows the operator to choose which files get copied. A "Y" response will cause <name> to be copied. Any other response will cause <name> to be bypassed.

SUFFIX?

This prompt allows the user to specify an explicit two-character suffix during the selected file copy mode. A response of carriage return only will cause the default suffix "ED" to be used.

ATTRIBUTES?

This prompt allows the user to specify explicit attributes during the selected file copy mode. The attribute letters "A", "C", "D", or "R" can be entered. A response of carriage return only will cause the "A" attribute to be used.

NO MORE FILES

The EDOS directory has been exhausted during the selected file copy mode.

13.4 Examples

The following example illustrates how the single file TESTP from an EDOS diskette would be copied into the file TESTPROG.SA on an MDOS diskette.

```
EMCOPY TESTP,TESTPROG.SA
```

The MDOS file will be allocated segmented space. It will be in the ASCII record format. The file may be delete protected if the EDOS file had the "Permanent Attribute" set.

The following example shows how an entire EDOS diskette is copied. The first two files are not copied since their file names are not valid MDOS file names. It should be noted that <name 1> is not specified. Thus, in order to specify a suffix for <name 2>, the comma had to be entered to indicate that <name 1> is null, or missing. The <name 2> suffix "SA" will be used instead of the default suffix "ED" for all files copied to the MDOS diskette. Since no other options were given, all files will be created in the ASCII record format.

```
=EMCOPY ,.SA;E
$DOS
** 25 INVALID FILE NAME
$DIR
** 25 INVALID FILE NAME
COPYING FILE: PRNTX
COPYING FILE: 0120
COPYING FILE: OMEX
COPYING FILE: OXRF
COPYING FILE: ONOL

** 06 DUPLICATE FILE NAME
COPYING FILE: OLIS
COPYING FILE: ONMC
COPYING FILE: DASM
COPYING FILE: DUP05
COPYING FILE: OO1K
COPYING FILE: OOP1
COPYING FILE: TITLE
COPYING FILE: PAGE
COPYING FILE: PCHO
COPYING FILE: RSMB

** 41 INSUFFICIENT DISK SPACE
=
```

The file ONOL was not copied because the MDOS file ONOL.SA already existed. The file RSMB was partially copied. The MDOS diskette lacked sufficient space for that EDOS file. The EMCOPY is stopped at that point since subsequent files would probably not have room either. Files like RSMB, RLOAD, ASMB, and EDIT on EDOS diskettes should not be copied to MDOS diskettes, since those programs make assumptions about the diskette structure, and will fail to work if copied and executed (after EXBIN conversion).

The last example shows how the selected file copy mode is used. In this example, not all files have the same record format. Thus, if they were copied with the "E" option, some would be created with the wrong file format. The file PRNTX is a binary record file. It is given the suffix "RO" (suffix for relocatable object files created by the Macro Assembler). The file ONOL, on the other hand, is an ASCII record file. It is given the default suffix "ED" (from the above example, ONOL.SA already existed on the MDOS diskette). The invalid file names from the EDOS diskette are displayed, but they are not copied. A single carriage return is used in this example to respond to the "COPY?" prompt to indicate a negative response.

```
=EMCOPY ;S
$DOS
** 25 INVALID FILE NAME
$DIR
** 25 INVALID FILE NAME
COPY PRNTX?
Y
SUFFIX?
RO
ATTRIBUTES?
R
COPY 0120 ?

COPY OMEX ?

COPY OXRF ?

COPY ONOL ?
Y
SUFFIX?

ATTRIBUTES?

COPY OLIS ?

COPY ONMC ?

COPY DASM ?

COPY DUP05?

COPY OO1K ?

COPY OOP1 ?

COPY TITLE?

COPY PAGE ?

COPY PCHO ?

COPY RSMB ?

NO MORE FILES
=
```

CHAPTER 14

14. EXBIN COMMAND

The EXBIN command is used to convert files in the EXbug-loadable format (e.g., object output from the assembly process without the OPT REL or OPT ABS directive) into files that can be loaded into memory for execution. The EXBIN command performs the inverse operation of the BINEX command.

14.1 Use

The EXBIN command is invoked with the following command line:

```
EXBIN <name 1>[,<name 2>] [;<options>]
```

where <name 1> is the file specification of an EXbug-loadable file that is to be converted, and <name 2> is the file specification of a file that is to receive the results of the conversion. Only <name 1> is required to be entered on the command line. The default suffix "LX" and the default logical unit number zero will be supplied for <name 1> if those quantities are not explicitly given. The output file specification, <name 2>, is optional. If <name 2> is entered, it may be a partial file specification consisting of only a file name, a suffix, or a logical unit number (or any combination thereof). The unspecified parts of <name 2> will be supplied from the respective parts of <name 1>, with the exception of the suffix. The default suffix for <name 2> is "LO" to indicate its memory-image format. If no file specification is given for <name 2>, the output file will be created with the same file name as <name 1> but with the suffix "LO". If only a suffix is given for <name 2>, that suffix will be used instead of the default "LO". If no logical unit number is given for <name 2>, the output file will be created on the same drive as given for <name 1>. In any case, <name 2> must be a file specification for which no entry already exists in the directory.

Standard error messages will be displayed if <name 2> already exists, if <name 1> does not exist, or if <name 1> is of the wrong file format.

The <options> field can be used to specify a starting execution address for the memory-image file. If no <options> field is given, EXBIN will use the address contained in the S9 record for the starting execution address.

EXBIN will ignore the S0, or name record, as well as any null records from <name 1>. Null records consist of a carriage return only. The content of the S1 records will be converted to its binary equivalent and written into <name 2>.

Since the EXbug-loadable files can contain S1 records that would be loaded into non-adjacent blocks of memory based on their address fields, the resulting memory-image file may be larger (occupy more diskette space) than <name 1>. This results from the fact that <name 2> is a memory-image file. All parts of memory which are not directly referenced by the S1 records, but which are included between the lowest and the highest address contained in all S1 records, will be a part of the memory-image in the file <name2> (initialized to binary zeroes).

The EXbug-loadable file, <name 1>, is unaffected by the entire EXBIN conversion process. The output file, <name 2>, can then be loaded into memory directly from diskette using the LOAD command (see Chapter 18).

14.2 Execution Address Specification

A starting execution address for the memory-image file can be specified by entering a valid hexadecimal number in the <options> field. The number must be in the range $0000-FFFF (entered in the <options> field without the dollar sign). In addition, the execution address must fall within the range of addresses spanned by the file. That is, the starting execution address cannot be less than the lowest address found in an S1 record, and it cannot be greater than the highest address. If an execution address is specified in the <options> field, it will override any value contained in the S9 record of <name 1>.

14.3 Error Messages

The following error messages can be displayed by the EXBIN command. The standard error messages that can be displayed by all commands are not listed here.

CHECKSUM ERROR

One of the S records from <name 1> contained an invalid checksum.

RECORD FORMAT ERROR

One of the records from <name 1> was not in the EXbug-loadable format. Exceptions to this are null records, or records which consist of only a carriage return. Null records are simply dropped and will produce no errors. Otherwise, only records beginning with S0, S1, or S9 are acceptable. If all records do begin with these characters when this error occurs, then something else is wrong with their format. The "M6800 EXORciser User's Guide" contains a complete description of the S record format.

SOURCE FILE NOT ASCII

The file <name 1> is not in the ASCII record format. EXbug-loadable files must be ASCII.

START ADDRESS OUT-OF-RANGE

The starting execution address specified in the <options> field or the address contained in the S9 record is not within the range of memory addresses spanned by the file.

** 30 INVALID EXECUTION ADDRESS

Normally, this standard error message has a slightly different meaning. During the EXBIN process, however, this error indicates that the starting execution address given in the <options> field was not a valid hexadecimal number.

14.4 Examples

Most frequently, the default suffixes and logical unit numbers suffice for the EXBIN operation. The following command line

```
EXBIN TESTPROG
```

will produce the file TESTPROG.LO on logical unit zero from the EXbug-loadable file TESTPROG.LX, also on logical unit zero. The starting execution address from the S9 record will be used.

The following command line

```
EXBIN TESTPROG,:2;2100
```

will create the same file as in the previous example. In this case, however, the file is created on logical unit two.

The starting execution address $2100 will be assigned to the output file, regardless of what is contained in the S9 record.

CHAPTER 15

15. FORMAT COMMAND

The FORMAT command attempts to rewrite the sector addressing information on diskettes. The FORMAT command can be used to reformat either single-sided or double-sided diskettes; however, double-sided diskettes must be formatted with this command before they can be used with MDOS. Single-sided diskettes usually come pre-formatted in a compatible format. The FORMAT command will only work on systems that are operating at one of the standard clock frequencies of 1 MHz, 1.5 MHz, or 2 MHz.

15.1 Use

The FORMAT command is invoked with the following command line:

```
FORMAT [:<unit>]
```

where <unit> is an optional logical unit number. If specified, <unit> can take on the values 1-3. If <unit> is not specified, logical unit number one will be used as a default.

If a user has a dual-drive EXORdisk II system, there is no need for him to specify a <unit> on the command line. If he does, caution must be used since the specification of logical unit number 2 on a EXORdisk II system will cause logical unit number zero to be formatted due to the way the disk controller works!

Since the FORMAT command will destroy all information on the diskette in the specified drive, the prompt

```
FORMAT DRIVE <unit>?
```

will be displayed, where <unit> indicates the logical unit number containing the diskette to be formatted. <unit> is either the number entered on the command line, or the default value supplied by the command itself. Any response other than "Y" will cause the FORMAT command to be terminated and control returned to MDOS. In this case, the diskette in the specified drive is unaffected. If the "Y" response is entered, the operator should have placed a diskette that needs to be formatted into the specified logical unit.

FORMAT will then proceed to:

1. Rewrite the soft sector addressing information on each cylinder (Appendix F contains a description of the diskette format),

2. Initialize every byte of each sector to the hexadecimal value $E5,

3. Re-read each cylinder to verify that the CRC's are good and that the diskette is readable.

The above process terminates when the diskette is completely formatted or when a diskette controller error occurs repeatedly. In the former case, control is returned to MDOS. In the latter case, the FORMAT command will display the diskette controller error with the standard "PROM I/O" error message. The diskette is not necessarily unusable if such errors occur. The FORMAT command should be re-run after having noted the physical sector number at which the error occurred. If the same error occurs at the same physical sector number after three attempts at running the FORMAT command, then the oxide on the diskette is probably damaged. The diskette is unusable in such cases. If the unusable diskette is inspected carefully by manually turning the diskette within its protective envelope, a mark or indentation can usually be found on its surface.

The FORMAT command can be used to format single-sided diskettes on the single- and double-sided Calcomp EXORdisk II/III systems or on the single-sided Pertec EXORdisk II systems; however, double-sided diskettes can only be formatted on the double-sided Calcomp EXORdisk III systems.

15.2 Messages

The only messages that the FORMAT command can display are the prompt shown above, asking if the diskette in the specified <unit> is to be formatted, and the standard PROM I/O error message, indicating that a diskette controller error was encountered during the formatting process.

15.3 Example

The following example shows the FORMAT command being used repeatedly after an error is detected. Since the physical sector number of the error keeps increasing, it indicates that the FORMAT command is able to rewrite more and more of the diskette; however, at one point, the physical sector number is always the same. At that time the FORMAT command is not used any longer since the diskette in drive one is unusable.

```
=FORMAT
FORMAT DRIVE 1?
Y
**PROM I/O ERROR-STATUS=38 AT 2006 ON DRIVE 1-PSN 01D8
=FORMAT
FORMAT DRIVE 1?
Y
**PROM I/O ERROR-STATUS=38 AT 2006 ON DRIVE 1-PSN 01F2
=FORMAT
FORMAT DRIVE 1?
Y
**PROM I/O ERROR-STATUS=38 AT 2006 ON DRIVE 1-PSN 0226
=FORMAT
FORMAT DRIVE 1?
Y
**PROM I/O ERROR-STATUS=31 AT 2006 ON DRIVE 1-PSN 0226
=FORMAT
FORMAT DRIVE 1?
Y
**PROM I/O ERROR-STATUS=31 AT 2006 ON DRIVE 1-PSN 0226
=
```

CHAPTER 16

16. FREE COMMAND

The FREE command displays the number of unallocated sectors and the number of empty directory entries remaining on a diskette.

16.1 Use

The FREE command program is invoked with the following command line:

```
FREE [:<unit>] [;<options>]
```

where <unit> can be the logical unit number 0, 1, 2, or 3, and <options> can be the letter "L". If the <unit> is not specified on the command line, the default value zero will be used.

The FREE command normally displays its summary data on the system console. The option "L", however, can be used to direct this data to the line printer instead. After the FREE command has determined the available space on the diskette, the data will be displayed in the following format:

```
DRIVE i :   xxxxxxxx
      aaaa/$bbb SECTORS  ccc/$dd FILES
      eeee/$fff LARGEST CONTIGUOUS BLOCK
```

The symbols have the following meanings:

Symbol	Meaning
i	Logical unit number selected.
xxxxxxxx	Eight character diskette ID.
aaaa	Available sectors in decimal.
$bbb	Available sectors in hexadecimal.
ccc	Available directory entries in decimal.
$dd	Available directory entires in hexadecimal.
eeee	Size of largest, available block of contiguous sectors in decimal.
$fff	Size of largest, available block of contiguous sectors in hexadecimal.

16.2 Example

The following example shows the output from the FREE command as displayed on the system console (a double-sided diskette is used).

```
=FREE :3
DRIVE 3 :  MDOS0300
      3004/$BBC SECTORS  124/$7C FILES
      0212/$0D4 LARGEST CONTIGUOUS BLOCK
=
```

The last example uses a single-sided diskette. No <unit> is entered on the command line, so the default of zero is used.

```
=FREE
DRIVE 0 :  MDOS0300
      0820/$334 SECTORS  140/$8C FILES
      0064/$040 LARGEST CONTIGUOUS BLOCK
=
```

CHAPTER 17

17. LIST COMMAND

The LIST command is used to print any ASCII file on either the system console or the printer. Options exist for numbering lines, specifying page formats, printing headings, and indicating starting and ending points. In addition, files can be accessed by their logical sector numbers for rapid access to any portion of a file.

17.1 Use

The LIST command is invoked with the following command line:

```
LIST <name>[,[<start>][,<end>]] [;<options>]
```

where <name> is the file specification of an ASCII file that is to be displayed, <start> and <end> are the optional starting and ending points of the display, and <options> can be one or more of the option letters described below.

Option	Function
L	Display file on line printer.
H	Get heading information from system console.
N	Display physical line numbers for each line.
F	Use a non-standard page format.

The <name> parameter must be specified with the LIST command. If no suffix is given, the default value "SA" will be supplied. The default logical unit number is zero.

The following sections describe each of the options in detail. The "L" option can be used with any other options to specify that the output from the LIST command is to be directed to the line printer. If the "L" option is missing, the system console will be used instead.

If the ASCII file contains any non-displayable characters, the LIST command will convert them into a percent sign (%) so that they will be visible. If records are contained in the file that are longer than the selected page format, they will be truncated on the right before they are displayed.

17.1.1 Start/end specifications

The default starting point for the display is the first physical line of <name>. The default ending point is the last physical line. The <start> specification can be used to start the display of the file at a specific physical line number or at a specific logical sector number. If the <start> specification is present on the command line it must be in one of the following two formats:

Lnnnnn

or

Smmm

The "Lnnnnn" form is used to specify a starting physical line number. The value "nnnnn" must be a 1-5 digit decimal number in the range 1-65535, inclusive. The "Smmm" form is used to specify a starting logical sector number. The value "mmm" must be a 1-3 digit hexadecimal number in the range $0-FFF, inclusive. The default <start> specification is "L1".

The <end> specification can be used to specify where the display of the file is to stop. The <end> specification has the same two forms as the <start> specification. If no <start> specification is entered on the command line, then the <end> specification can be of either form; however, if the <start> specification is entered, then the <end> specification must be of the same form. For example, it is invalid to specify a <start> specification of logical sector five and an <end> specification of physical line 216. The <end> specification must be larger than the <start> specification. The default <end> specification is the logical end of the file.

17.1.2 Physical line numbers

Normally, the displayed file will not be shown with physical line numbers. Only the actual data of the lines in the file will be shown. The "N" option can be used to cause physical line numbers to be generated by the LIST command and displayed with each line of data from the file. The physical line numbers will be printed as five digit decimal numbers. If the standard page format is used, each data line that is longer than the eighty characters will be displayed with eight fewer data characters, truncated from the right. The physical line numbers are useful when using the BLOKEDIT command (Chapter 5) or when trying to find verify errors from the COPY command (Chapter 7) between a diskette file and a tape file.

The physical line number option "N" is fairly meaningless if the logical sector form of the <start> specification is used. Since no count is available for the number of lines between the beginning of the file and the specified logical sector, the physical line numbers (if printed) would only be relative to the part of the file that was displayed. A partial line will usually be seen as the first line since the records randomly cross sector boundaries.

17.1.3 User-supplied heading

Normally, the LIST command will print a page number and the file name specification of the file being listed as a heading. The "H" option can be used to cause additional information to be displayed on the heading line. The "H" option will cause the following prompt to be shown on the system console before the file is listed:

```
ENTER HEADING:
```

The operator can then respond with a line of text that is to be used as the heading. The maximum length of the entered heading is 100 (decimal) characters. The heading line containing the page number, file name specification, and user-supplied text will automatically be printed on the second line of each page.

17.1.4 Non-standard page formats

Normally, the LIST command will display a maximum of eighty characters per line and sixty-six lines per page. The "F" option can be used to override the standard page format. The format of the "F" option is as follows:

```
F[ccc].[pp]
```

where at least one of the two parameters must be present. The "ccc" parameter is used to specify the number of columns to be printed per line. It must be a decimal number in the range 1-132, inclusive. The "pp" parameter is used to specify the number of lines per page. It, too, must be a decimal number, but in the range 10-99, inclusive. An error message will be displayed if an illegal page format is given. Either the line length or the page length can be specified without the other (e.g., "F20." or "F.58", respectively). Only the line length need be specified if longer lines are to be printed on a standard length page.

17.2 Messages

The following messages can be displayed by the LIST command. Not all messages are error messages; however, error messages are included in the list. The standard error messages that can be displayed by all commands are not listed here.

PAGE ddd <name>

This is the standard heading supplied by the LIST command. "ddd" is the decimal page number and <name> is the file name specification of the file being printed.

ENTER HEADING:

This message is displayed when the "H" option is used to print additional heading text on each page. A maximum of 100 (decimal) characters can be entered.

** 24 LOGICAL SECTOR NUMBER OUT OF RANGE

This error is caused when a <start> specification references a logical sector number that is greater than the logical sector number of the end of file.

** 34 INVALID START/END SPECIFICATIONS

The <start> and <end> specifications on the command line were not both of the same form ("L" or "S"), or the <end> specification had a value that was less than the value of the <start> specification. This error can also be caused if the <start> or <end> specifications begin with letters other than "L" or "S".

** 35 INVALID PAGE FORMAT

The parameters of the "F" option did not meet the criteria explained in section 17.1.4.

** 36 FILE EXHAUSTED BEFORE LINE FOUND

The <start> specification on the command line specified a physical line number whose value was larger than the total number of lines in the file.

17.3 Examples

The MDOS equate file is used in all of the following examples. The following example shows what is probably the most commonly used form of the LIST command. No options are used. The default values for suffix, logical unit number, <start> and <end> specifications, page format, and output device are used. It is assumed that the BREAK key was depressed to terminate the LIST command and return control to MDOS in this example.

```
=LIST EQU

PAGE 001   EQU      .SA:0

*
* TURN OFF THE LISTING
*
 OPT NOL
 PAGE
*
* MDOS VERSION 03.00 -- SYSTEM EQUATE FILE -- JULY 25,1978
*
 SPC 3
*
=
```

The following example uses the <end> specification to stop on the tenth line of the file. Since the default value for the <start> specification is to be used, a null parameter must be specified for it. This is done by entering the two adjacent commas. The "N" option causes the display of the physical line numbers.

```
=LIST EQU,,L10;N

PAGE 001   EQU      .SA:0

00001    *
00002    * TURN OFF THE LISTING
00003    *
00004     OPT NOL
00005     PAGE
00006    *
00007    * MDOS VERSION 03.00 -- SYSTEM EQUATE FILE -- JULY 25,1978
00008    *
00009     SPC 3
00010    *
=
```

The following example uses both <start> and <end> specifications to cause the display of physical lines 30 through 40, inclusive.

```
=LIST EQU,L30,L40

PAGE 001  EQU      .SA:0

* THE SAME CONCEPT AS THE "SKIP2" MACRO IS USED, EXCEPT THAT
* A "BIT TEST ACCUMULATOR A IMMEDIATE" OP CODE IS GENERATED.
*
SKIP1 MACR
 FCB $85
 ENDM
*
* S C A L L     M A C R O    (SYSTEM FUNCTION CALL)
*
SCALL MACR
 IFEQ NARG-1
=
```

The following example illustrates how the logical sector number can be used to rapidly access any part of a file. When the <start> and <end> specifications refer to physical line numbers, the file must be read from the beginning, a record at a time, in order to find the correct lines; however, the logical sector form of the <start> specification permits the LIST command to go directly to the sector. The physical line number option "N" is fairly meaningless if the logical sector form of the <start> specification is used. Since no count is available for the number of lines between the beginning of the file and the specified logical sector, the physical line numbers (if printed) would only be relative to the part of the file that was displayed. A partial line will usually be seen as the first line since the records randomly cross sector boundaries. The BREAK key was used in this example to terminate the display of the file.

```
    =LIST EQU,S5

    PAGE 001  EQU      .SA:0

    TE" OP CODE IS GENERATED.
    *
    SKIP1 MACR
     FCB $85
     ENDM
    *
    * S C A L L     M A C R O    (SYSTEM FUNCTION CALL)
    *
    SCALL MACR
     IFEQ NARG-1
    =
```

The following example displays the MDOS equate file using a non-standard line length specification. Only the first twenty characters of each line will be shown. Notice that this format also applies to the printed heading. The BREAK key was used to terminate the display.

```
=LIST EQU;F20

PAGE 001  EQU     .S

*
* TURN OFF THE LISTI
*
 OPT NOL
 PAGE
*
* MDOS VERSION 03.00
*
=
```

The last example lists the first nine lines of the MDOS equate file. In addition to the previously shown features, the "H" option is used to specify a heading. This heading would be printed at the top of each page if multiple pages were printed.

```
=LIST EQU,,L9;HN
ENTER HEADING: THIS IS THE MDOS SYSTEM EQUATE FILE

PAGE 001  EQU     .SA:0  THIS IS THE MDOS SYSTEM EQUATE FILE

00001    *
00002    * TURN OFF THE LISTING
00003    *
00004     OPT NOL
00005     PAGE
00006    *
00007    * MDOS VERSION 03.00 -- SYSTEM EQUATE FILE -- JULY 25,1978
00008    *
00009     SPC 3
=
```

CHAPTER 18

18. LOAD COMMAND

The LOAD command is used to load a program from a memory-image file on the diskette into memory. Options exist for entering the debug monitor after loading a program, for automatically executing a program, for loading a program into the User Memory Map of EXORciser II systems, and for loading a program over the resident operating system.

18.1 Use

The LOAD command is most frequently used to load a program into memory for testing; however, certain types of programs, specifically those that overlay MDOS, that load outside range of contiguous memory knonw to MDOS, or that execute in the User Memory Map of an EXORciser II system with the dual memory map configured, can only be executed via the LOAD command and one of its options (G). The LOAD command is invoked with the following command line:

```
LOAD [<name>] [;<options>]
```

where <name> is the file name specification of a file from which the program is to be loaded into memory, and <options> specifies how to load the program. If <name> is specified, it must be the name of a file that has the memory-image format. The default suffix "LO" will be supplied if no explicit suffix is given. The default logical unit number is zero.

The <options> are divided into "Main Options" and "Other Options". Main Options are mutually exclusive. That is, only one Main Option can be specified on the command line at a time. The Other Options can be included with any one of the Main Options. The following tables show both Main and Other Options.

Main Option	Function
none	Load program into contiguous memory above MDOS; keep MDOS SWI vector to allow system function access.
U	Load program into User Memory Map of an EXORciser II system with a dual memory map configuration.
V	Allow program to load over MDOS or anywhere else in memory; disable MDOS's SWI vector.

Other Option	Function
none	Enter debug monitor after loading program.
G	Execute program after loading.
(<str>)	Initialize MDOS command line buffer with the character string <str> as indicated in the enclosed parentheses.

The <options> are discussed in detail in the following sections.

The LOAD command does not verify that memory exists for the areas into which a program gets loaded. Command-interpreter-loadable programs (section 18.1.1) are guaranteed that memory exists since the memory was sized at initialization time; however, programs loading into discontiguous areas of memory or into the User Memory Map of a dual memory map configuration are not guaranteed that memory exists. The operator is responsible for knowing where memory is configured in his system and where his programs are loaded. Also, due to the nature of the diskette controller, it is not possible for the LOAD command to compare what is read from the file with what is stored into memory. Only diskette controller read errors can be detected.

Programs brought into memory from the diskette will be loaded in multiples of eight bytes. This fact must be considered when programs are loaded into adjacent blocks of memory close to other programs, or if programs are loaded into the upper end of a block of memory.

18.1.1 Command-interpreter-loadable programs

Programs that can be loaded by the MDOS command interpreter are usually loaded for testing by not specifying anything in the <options> field. The "G" option can be used to load and execute the program in one step; however, for such programs this is awkward. They are usually loaded and executed directly by the MDOS command interpreter by entering their file names as the first file name specification on an MDOS command line. The command line

```
LOAD TESTPROG
```

would attempt to load the file TESTPROG.LO from logical unit zero above the resident operating system (the program must have already been assembled at, or link/loaded and assigned memory locations at the proper addresses so it loads above MDOS). After the file was loaded, control would be given to the debug monitor.

The following command lines

```
TESTPROG.LO
```

or

```
LOAD TESTPROG;G
```

would load the program from TESTPROG.LO from logical unit zero and execute the program. It should be noted that these two command lines will accomplish the same function. Since the first form of the command line is shorter, especially if the suffix were change to "CM", the second form is seldomly used.

Command-interpreter-loadable programs must meet the following requirements:

1. The program must load above the resident operating system; it must be origined to load above hexadecimal location $1FFF. The program can access the direct addressing area below hexadecimal address $100 (BSCT) during execution; however, that area of the memory cannot be loaded into. Thus, variables in BSCT cannot be initialized during loading. In addition, if a program is going to use diskette I/O, none of the locations below address $20 can be used by the program for its own variables.

2. The program must load within the range of contiguous memory that was established during MDOS initialization. Such programs require an

additional eight bytes of memory beyond their highest loaded address to allow room for a stack when the debug monitor is entered. These eight bytes must be within the contiguous memory block known to MDOS.

If either of these criteria is not met, the standard error message will be displayed indicating that the program has an invalid load address.

After the program is loaded (without any options), the debug monitor will be entered (as seen by the input prompt of the resident monitor). The pseudo registers of the debug monitor will have been initialized by the LOAD command to the following values:

Pseudo register	Contents
P	Starting execution address
X	Lowest address loaded into
S	Highest address loaded into (eight bytes greater than the highest actual program location)
A, B, C	Indeterminate
Y	Indeterminate (MDOS09)
U=S	MDOS09 only
DP=0	MDOS09 only

Normally, command-interpreter-loadable programs take advantage of the fact that the stack pointer is initialized to the end of the program area by using that part of memory for the actual stack during execution. Such stacks must be a minimum of 80 (decimal) bytes in size.

In addition to setting up the pseudo registers, the LOAD command will change the MDOS variable ENDUS$ (Chapter 24) to contain the last address loaded into by the program. This allows the program to dynamically allocate additional contiguous memory for buffers, etc., via the ".ALUSM" function (Chapter 27).

Caution must be exercised when loading a program and entering the debug monitor. If MDOS is to be reinitialized, the ABORT or RESTART pushbuttons must first be depressed before the debug command "E800;G" or "MDOS" is executed.

18.1.2 Non-command-interpreter-loadable programs

Programs are not loadable by the MDOS command interpreter must be loaded into memory for either testing or execution via the LOAD command. Normally, such programs will overlay the resident operating system or will load into areas outside of the contiguous memory known to MDOS. Such programs cannot be executed directly via the MDOS command interpreter.

The "V" option will inhibit the memory boundary tests explained in the previous section. A program loaded with the "V" option, however, must still meet the following requirements:

1. The program must load above the RAM variables required by the diskette controller. That is, the program must be assembled to load above hexadecimal location $1F. The program can access the direct addressing area below hexadecimal location $20 during execution; however, that area of memory cannot be loaded into. Thus, variables in the direct addressing area cannot be initialized during loading if their addresses are between $0000 and $001F, inclusive.

2. The program's ending load address, as calculated from the parameters in the RIB, must not be greater than $FFFF. Specifically, the starting load address plus the number of sectors to load minus one (expressed in numbers of bytes), plus the number of bytes to load from the last sector minus one, must be less than or equal to $FFFF (see section 24.2).

If either of these criteria is not met, the standard error messages will be displayed indicating that the program has an invalid load address.

If the program is to be loaded for testing, only the "V" option should be specified. Thus, the command line

```
LOAD TESTPROG;V
```

will cause the debug monitor to be entered after the program is loaded from the file TESTPROG.LO from logical unit zero. The pseudo registers will contain the following values:

Pseudo register	Contents
P	Starting execution address
X	Lowest address loaded into
S	EXbug stack address
A,B,C	Indeterminate
Y	Indeterminate (MDOS09)
U=S	MDOS09 only
DP=0	MDOS09 only

Since the memory boundary check is bypassed with the "V" option, the program can be assembled to load anywhere above location $1F; however, no check is made to verify that memory exists where the program is loaded.

Once programs have been tested, they can be executed via the LOAD command by specifying the additional option "G", as in the following command line:

```
LOAD TESTPROG;VG
```

The "G" option will bypass entering the debug monitor and cause control to be passed directly to the loaded program. The stack pointer is still configured as explained above.

If the "V" option is used (with or without the "G" option), the SWI vector will be restored to its original value that points back to the debug monitor. Thus, programs loaded with the "V" option cannot use the resident MDOS functions.

18.1.3 Programs in the User Memory Map

By using the "U" option as shown in the following command line, the LOAD command can be used to load a program into the User Memory Map of an EXORciser II system that has the dual memory map configured:

```
LOAD TESTPROG;U
```

If the dual memory map is not configured, an error message will be displayed.

The only requirement placed on programs loading into the User Memory Map is that the ending load address not be greater than $FFFF. Otherwise, any memory locations ($0000-FFFF) can be loaded into; however, no check is made to ensure that memory exists where the program is loaded. If the "G" option omitted, the debug monitor will be entered after the program is loaded. The debug monitor will display the User Memory Map prompt, not the Executive Memory Map prompt. The pseudo registers will contain the following values:

Pseudo register	Contents
P	Starting execution address
X	Lowest address loaded into
S	Highest address loaded into
A, B, C	Indeterminate
Y	Indeterminate (MDOS09)
U=S	MDOS09 only
DP=0	MDOS09 only

Caution must be exercised in starting execution of programs loaded in this manner. Since the stack pointer contains the address of the last loaded program location, use of the debug monitor commands ";P" or ";N" will cause seven locations of the program to be destroyed. This may alter program data or instructions. It is recommended that the stack pointer first be changed via the ";S" command; that the "nnnn;G" command be used to initiate execution; or that area for the stack be provided at the end of the program.

The LOAD command's "G" option can be used in addition to the "U" option to give control to the program immediately after it has been loaded:

```
LOAD TESTPROG;UG
```

The "M6800 EXORciser II User's Guide" should be consulted for a complete discussion of the User Memory Map.

If the "U" option is used (with or without the "G" option), the SWI vector will be restored to its original value that points back to the debug monitor. Thus, programs loaded with the "U" option cannot use the resident MDOS functions.

18.1.4 MDOS command line initialization

The Other Option (<str>) is used while testing command-interpreter-loadable programs (section 18.1.1). Such programs usually obtain parameters via the initial command line that activated the program. When testing such programs, however, the command line buffer will contain the command line that invoked the LOAD command. Thus, the (<str>) option is used to allow testing of the loaded program as if it had been invoked from the command line directly, simulating its execution-time environment. The quantity <str> will be placed into the MDOS command line buffer. The command line buffer pointer, CBUFP$ (Chapter 24), will be adjusted to point to a null character which precedes the string (a valid terminator for the .PFNAM function, Chapter 27). Any displayable characters, except the right parenthesis ")", can be included in the string <str>. The string will be terminated with a carriage return after it is placed into the command line buffer. Thus, the use of the null string "()", will cause a single carriage return to be placed into the buffer.

The (<str>) option can be used with any of the Main Options; however, it only makes sense when no Main Option is used (command-interpreter-loadable programs).

18.1.5 Entering the debug monitor

The LOAD command can be invoked without entering a file specification. For example, the command line

```
                    LOAD
```

will cause the debug monitor to be entered directly. For MDOS, the message

```
BKPT ERROR
P-2131 X-2170 A-0D B-80 C-C0 S-227F
*
```

or the message

```
SWI P-2131 X-2170 A-0D B-80 C-C0 S-227F
E*
```

will be displayed depending on whether EXbug 1 or EXbug 2, respectively, is in the system. The actual contents of the pseudo registers may differ.

For MDOS09, the message

```
SWI P-2131 U-227F Y-FF34  X-2170  DP-00  A-0D  B-80  C-C0
S-227F
```

will be displayed.

If the LOAD command is invoked in this way, then at no time should MDOS be reinitialized via the "E800;G" or "MDOS" command without first depressing either the ABORT or RESTART pushbuttons on the front panel of the EXORciser. If the LOAD command was entered as shown in the example above, MDOS can be reentered without reinitialization by using the debug monitor command ";P". The LOAD command has configured itself so that the ";P" command will cause a normal return to the MDOS command interpreter.

If the "V" option was used without a file name specified on the command line, the ";P" command will cause MDOS to reinitialize as if an "E800;G" or "MDOS" command had been given to the debug monitor. The "V" option has the same effect as using the ABORT or RESTART pushbuttons insofar as the SWI vector configuration is concerned.

The "U" option is invalid with this form of the LOAD command.

The Other Options "G" and "(<str>)" are invalid when the LOAD command is invoked without a file name specification on the command line.

18.2 Error Messages

The LOAD command displays error messages from the standard error message set; however, since some of these messages have special significance to the LOAD command only, they are listed here.

** 07 OPTION CONFLICT

This error message can be displayed for the following reasons: More than one Main Option was specified at the same time; the LOAD command was invoked without a file name with the "U" option; or the "U" option was used on an EXORciser I system or on an EXORciser II system without the dual memory map configured.

Earlier versions of MDOS supported the "P" and "M" options which were used as defaults if no options were entered. The "P" option had same effect as the null Main Option. The "M" option had the same effect as the null Other Option. If "P" was used with any of the Main Options, or if "M" was used with the "G" option, then this message would also be displayed.

** 12 INVALID TYPE OF OBJECT FILE

This error message is displayed if the file specified on the command line was not a memory-image file. In odd cases, this message is also be displayed if the Retrieval Information Block of the file has been damaged. If this is the suspected cause, then the REPAIR command (Chapter 22) should be run to verify that the RIB is in error.

** 13 INVALID LOAD ADDRESS

If the LOAD command was invoked with the null Main Option, the program cannot be loaded for one of the following reasons:

1. It loads over the resident operating system. That is, it loads below hexadecimal location $2000.

2. It loads beyond the range of contiguous memory known to MDOS (established at initialization time).

If the LOAD command was invoked with the Main Option "V", the program cannot be loaded because it loads below hexadecimal location $20, or the program's ending load address is greater than $FFFF.

If the LOAD command was invoked with the Main Option "U", ending load address is greater than $FFFF.

In the cases where the ending load address exceeds $FFFF, the RIB of the file has been invalidly created. Usually, this occurs when a program loads into the highest memory location ($FFFF) but does not start loading at an address that is a multiple of eight. Since the only information available to the LOAD command is the starting load address and the program's size (a multiple of eight bytes), the ending load address may exceed $FFFF (diskette controller forces the multiple of eight byte criterion). Then, the program should be re-assembled or re-link/loaded so that the starting load address is a multiple of eight. If this is not the case, the REPAIR command (Chapter 22) should be invoked to check for other files that may also be in error.

** 30 INVALID EXECUTION ADDRESS

The the file from which a program is to be loaded has an invalid RIB which must be fixed with REPAIR. The starting execution address lies outside of the block of memory that would be loaded by the program.

18.3 Examples

The following command line:

```
LOAD TESTPROG:1;(FILE1,FILE2;S=1000)
```

will load the program from the file TESTPROG.LO from logical unit one into memory. The program must be origined to load above the resident MDOS and below the end of contiguous memory. The MDOS command line buffer will be initialized with the string

```
FILE1,FILE2;S=1000
```

to allow the program to be tested as if it had been invoked from the command line directly. After the program is loaded, control is given to the debug monitor.

The next example illustrates how user-written programs are executed from diskette directly. The program can load anywhere in memory except below hexadecimal location $20. The program cannot use any of the resident MDOS functions:

```
LOAD BLAKJACK;VG
```

The next example illustrates how the PROM Programmer I program can be used for making PROMs of programs that load above resident MDOS and the area required by the command interpreter and LOAD command. It is assumed that the program in the file TPROM.LO loads above $2300. Since the contents of memory are not destroyed during the initialization procedure, MDOS can be reinitialized after loading the program TPROM without losing the content of those memory locations. Then, the LOAD command is used again to load and execute a version of the Prom Programmer I program (origined to load at location $20).

```
=LOAD TPROM;V
*E800;G
MDOS 03.00
=LOAD PPLO;VG
?
```

The command "E800;G" can be validly used since the program in the file TPROM.LO was loaded with the "V" option. If no Main Options are used, the ABORT or RESTART pushbuttons would have to be depressed first.

CHAPTER 19

19. MERGE COMMAND

The MERGE command allows one or more files to be concatenated into a new file. This command is useful in combining several smaller program files into one large file, or in building relocatable libraries to be used in conjunction with the M6800 Linking Loader (RLOAD).

19.1 Use

The MERGE command is invoked with the following command line:

```
 MERGE <name 1>[,<name 2>,...,<name n>],<dname>[;<options>]
```

where <name i> (i=1 to n) are the names of the files to be merged together, <dname> is the name of the destination file, and <options> can be one or both of the options listed below. A maximum of 38 (decimal) file names can be accommodated by the MERGE command.

Option	Function
W	Use automatic overwrite if destination file already exists on diskette.
<addr>	Use hexadecimal <addr> as starting execution address of destination file.

The <options> are described in detail in the following sections.

Only <name 1> and <dname> are required. All file name specifications on the MERGE command line must contain at least a file name. For all <name i>, the default suffix "SA" and the default logical unit number zero will be used if none are explicitly given. The default suffix and logical unit number for <dname> are taken from <name 1>.

MERGE will perform two different functions depending on whether <dname> is the same as <name 1> or not. If <dname> is different from <name 1>, then all of the files specified by <name i> will be combined into the destination file <dname>. Each of the <name i> files will remain unaffected. If <dname> is the same as <name 1>, however, then MERGE will append the files specified by <name 2> through <name n> to the end of the file <name 1>. In this case, the file <name

1> will be changed.

The file names <name 2> through <name n> are optional. If they are specified, they must be of the same file format and have similar allocation and space compression attributes as <name 1>. In addition, their names cannot be the same as that of <dname> unless <dname> is the same as <name 1>. If file names <name 2> through <name n> are not specified, the MERGE command performs the same function as the COPY command. That is,

```
MERGE <name 1>,<dname>
```

is identical to the command line

```
COPY <name 1>,<dname>
```

assuming that <name 1> is not the same as <dname>.

Only four types of files can be processed by the MERGE command. The files specified by <name i> must have one of the following formats:

File format as shown by DIR	File format
0	User-defined
2	Memory-image
3	Binary record
5	ASCII record

Memory-image files can be merged together. The file <dname>, however, cannot exist in such cases because MERGE must ensure that the destination file is allocated contiguous space to accommodate the memory-images of all <name i> files. If <dname> already exists, MERGE cannot ensure such allocation. For all other file formats that <name i> can assume, <dname> can already exist. In such cases where <dname> is different from <name 1> and already exists in the directory (and no "W" option on command line), the message

```
<dname> EXISTS.  OVERWRITE?
```

will be displayed. The operator must respond with a "Y" if MERGE is to perform the merge operation. Any other response will terminate the MERGE command and return control to MDOS.

19.1.1 Merging non-memory-image files

If the files specified by <name i> are all of the user-defined format, the binary record format, or the ASCII record format, then the destination file <dname> will be a direct concatenation of all of the source files. For example, if five ASCII record files are merged, the destination file can be represented by:

```
                      Destination File

------------------------------------------------------------
:          :             :       :        :                 :
:  File 1  :  File 2     : File 3: File 4 :     File 5      :
:          :             :       :        :                 :
------------------------------------------------------------
:                                                           :
:..... start of file                          end of file....:
```

The same type of concatenation would take place if the file format was either user-defined or binary record. The MERGE command can be used in this manner to create one large data or source program file from smaller files, or a library file of relocatable object programs.

19.1.2 Merging memory-image files

If all of the files specified by <name i> are memory-image format files, then the destination file <dname> will be a memory-image file also; however, it will span all memory locations between the lowest and the highest address spanned by the <name i> files. If the files to be merged occupy overlapping areas in memory, then the destination file will contain the contents of the last file to be merged that occupies those common locations. The MERGE command produces a file that is the memory image of files 1-n as if they were loaded into memory in the sequence in which they appear on the command line. Regions of memory spanned by <dname> that are not "loaded" into by the <name i> files will contain binary zeroes.

For example, if three memory-image files as described in the following table were merged together,

<name i> file	Lowest address	Highest address
1	600	FFF
2	100	7FF
3	1200	13FF

then the resulting destination file can be represented by:

```
Memory                                            1      1
Location  1              6       8      F         2      3
          0              0       0      F         0      F
          0              0       0      F         0      F
         --------------------------------------------------
         |2222222222222222222222211111111         33333333|
         |2222222222222222222222211111111         33333333|
         |2222222222222222222222211111111         33333333|
         --------------------------------------------------
          :                 :                             :
          :                 :                             :
          :                 :...Overlayed <name 1>        :
          :                                               :
          :......Start of <dname>       End of <dname>....:
```

The numbers in the body of the rectangle above indicate the data of the respective <name i> file. Thus, "2" indicates the data of <name 2>, etc. Between locations $600 and $7FF, the data of <name 2> is seen. It overlayed any information put into <dname> by <name 1>. Since none of the <name i> files spanned the addresses from $1000 to $11FF, inclusive, that part of <dname> is initialized to binary zeroes.

It should be noted that programs from memory-image files loaded into memory are always a multiple of eight bytes in length. This is a function of the diskette controller. Regardless of the actual data of a file, a multiple of eight bytes will always be loaded. This fact must be kept in mind when merging files which span memory locations that are close together.

Memory-image files have associated with their load information a starting execution address. If no <options> field is specified on the MERGE command line, <dname> will have the starting execution address of <name 1> assigned to it; however, as can be seen from the above example, this default execution address can be meaningless. An explicit starting execution address can be specified in the <options> field as a one to four digit hexadecimal number. The address must lie within the range of memory addresses spanned by <dname>.

19.1.3 Other options

The "W" option is used to allow the destination file to be overwritten if its file name already exists; the "OVERWRITE" prompt is not displayed and MERGE performs its expected function. If the "W" option is not used, the MERGE command will prompt the operator before overwriting the destination file. The "W" option is not valid if <name 1> is a memory-image file because the destination file cannot exist in that case.

19.2 Messages

The following messages can be displayed by the MERGE command. Not all messages are error messages, although error messages are included in the list. The standard error messages that can be displayed by all commands are not listed here.

<name> EXISTS. OVERWRITE?

> The specified file name already exists in the directory. The operator is prompted before the file is overwritten. A "Y" response will cause the merge to take place. Any other response will cause control be to returned to MDOS.

** 15 <name> HAS INVALID FILE TYPE

> The file indicated by <name> is not of the proper format (i.e., ASCII record, binary record, memory-image, or user-defined), or the RIB of the file is damaged. A memory-image file's RIB is considered to be damaged if the number of sectors to load is zero, the number of bytes to load from the last sector is zero, or if the ending load address is larger than $FFFF. If a damaged RIB is suspected, the REPAIR command (Chapter 22) should be invoked to correct the error.

** 16 CONFLICTING FILE TYPES

> The files specified by <name i> have different file formats. They must all be the same format. Even if the format (ASCII record, etc.) is the same, the contiguous allocation attribute and the space compression attribute must also agree between all <name i>. This error can also occur if <dname> (not the same as <name 1>) exists and has a different file format than <name 1>.

** 33 TOO MANY SOURCE FILES

> More than 38 (decimal) file names were specified for <name i>.

19.3 Examples

The following example combines the first four files specified on the command line into a new file (the last name on the command line). The first four files all have the same attributes. The last name is the name of a new file since the OVERWRITE prompt was not displayed.

```
MERGE PART1,PART2:3,PART3:1,PART4:2,BOOK
```

The default suffix "SA" was used for each file name. The destination file BOOK is created on the default logical unit number used for PART1, unit zero.

The next example illustrates how a relocatable library file can be constructed from various smaller files. The library file already exists. It will have the files appended to its end.

```
MERGE LIB.RO,DSKIO.RO,CNSIO.RO,FLOT.RO,LIB.RO
```

The last example illustrates how a patch file can be attached to a test program file. A new starting execution address is specified as $1F20.

```
MERGE TESTPROG.LO,PATCH1.LO,NEWTEST.LO;1F20
```

The file name NEWTEST.LO must not already exist. Both of the other two files must be memory-image in format.

CHAPTER 20

20. NAME COMMAND

The NAME command allows the names, suffixes and/or attributes of a file to be changed in the directory. A single file name or a family of file names can be affected. The contents of a file remain unchanged.

20.1 Use

The NAME command is invoked with the following command line:

```
NAME <name 1> [,<name 2>] [;<options>]
```

where <name 1> is the file name specification of an existing file, <name 2> is the new name the file is to be given, and <options> can be one or more of the option letters listed below.

Option	Function
D	Set delete protection
W	Set write protection
X	Remove protection
S	Set system attribute
N	Remove system attribute

The <options> are discussed in detail in the following sections.

20.1.1 Changing file names

If <name 2> is specified on the command line, the NAME command will attempt to change the name and/or suffix of <name 1>. <name 1> must always be specified. The default suffix "SA" and the default logical unit number zero are supplied if none are explicitly given for <name 1>.

If only a file name is specified for <name 2>, then only <name 1>'s file name will be changed; its suffix will remain the same. For example, the following command line

```
NAME TESTPROG,BLAKJACK
```

will change the file name TESTPROG.SA:0 to the new name BLAKJACK.SA. The default suffix and logical unit number were applied to <name 1> before performing the name change. Likewise, if only a suffix is supplied for <name 2>, then <name 1>'s file name will not be changed; only its suffix will be affected. Thus, the following command line

```
NAME TESTPROG.LX:1,.EY
```

will change the suffix of the file name TESTPROG.LX on drive one to "EY".

A logical unit number should not be specified for <name 2> since the file <name 1> cannot be moved from one logical unit to another when its name is being changed; however, if a logical unit number is specified for <name 2>, it must agree with the logical unit number of <name 1>.

When changing file names, the family indicator can be used in either the file name portion or in the suffix portion of <name 1>. The family indicator cannot appear in both places. The family indicator can be used to change the names or the suffixes of an entire family of file names. For example, the command line

```
NAME *.ED,.SA
```

would change all file names on drive zero that had the suffix "ED" (as would be created by the EMCOPY command when it uses the default suffix) so that they had the new suffix "SA". Similarly, the command line

```
NAME TESTPROG.*:2,BLAKJACK
```

would change all files named TESTPROG (any suffix) on drive two to have the new name BLAKJACK. The suffixes would remain the same, preserving the identity of source, EXbug-loadable object, and memory-image files as designated by their respective suffixes.

Regardless of how the NAME command is invoked to change a file's name and/or suffix, the new name must not already exist in the directory. Similarly, the old name specified by <name 1> must exist in the directory. If either one of these two conditions is not true, one of the standard error messages will be displayed.

20.1.2 Changing file attributes

In addition to changing a file's name and/or suffix, the NAME command can be used to change a file's attributes. The way in which the attributes are to be changed is specified in the <options> field. Thus, it is possible to change both a file's name and/or suffix and its attributes with the same invocation of the NAME command.

The inherent attributes of a file that define its physical format on the diskette (contiguous allocation, space compression, memory-image, etc.) cannot be changed. These attributes remain with a file from the time it is created until the time it is deleted; however, the protection attributes and the system attribute can be changed at any time.

The protection attributes of a file are changed by specifying the letter "X" (remove protection), "W" (set write protection), or "D" (set delete protection) in the <options> field. The system attribute is changed by specifying the letter "S" (set system attribute) or "N" (remove system attribute). A maximum of five option letters can be specified at one time. The option letters are processed from left to right. For example, if a file with write protection set is to have only delete protection set, the command line

```
NAME TESTPROG;XD
```

could be used. If the "X" and "D" options were reversed, the file would be unprotected.

If no <name 2> is specified, then an <options> field must be present. In such cases, the family indicator can be used for both the file name and the suffix of <name 1>. Thus, a diskette can have all of its files protected or unprotected with a single invocation of the NAME command.

20.2 Error Messages

The following error messages can be displayed by the NAME command. The standard error messages that can be displayed by all commands are not listed here.

** 25 INVALID FILE NAME

This error message is displayed for the following reasons: both <name 1> and <name 2> were specified on the command line and the family indicator was present in both the file name and the suffix portion of <name 1>; both <name 1> and <name 2> were entered with the family indicator; or a device name was used for <name 1> or <name 2>.

20.3 Examples

The following command line

```
NAME *.*:1;X
```

will remove both delete and write protection from every file named in the directory of drive one.

The next command line shows how files' names and their attributes can be changed at the same time.

```
NAME *.ED,.LX;X
```

This example will take all file names with the suffix "ED", change it to "LX", and remove any protection that may be present.

The last example illustrates how a user-written program can be incorporated as a system command file.

```
NAME TESTPROG.LO:3,SURFACE.CM;SD
```

This command line changes both file name and suffix. In addition, the system attribute and delete protection are set. Thus, the program file named SURFACE.CM will now be treated as a system file by the DIR, DEL, and DOSGEN programs.

CHAPTER 21

21. PATCH COMMAND

The PATCH command allows changes to be made to memory-image files. An object file can be "fixed" due to minor bugs or assembly errors without having to re-edit and re-assemble its corresponding source file. The "fixes" can be entered using M6800 assembly language mnemonics or the equivalent hexadecimal operation codes.

21.1 Use

The PATCH command is invoked with the following command line:

```
PATCH <name>
```

where <name> is the file specification of a memory-image file. The default suffix "LO" and the default logical unit number zero will be supplied if none are explicitly given for <name>. One of the standard error messages will be displayed if the file <name> does not exist or if it is of the wrong file format.

The PATCH command is an interactive program that has its own command structure. Once PATCH is running, it will display a greater-than sign (>) as an input prompt to indicate that a command must be entered by the operator. Commands exist to assign an offset used as a base address for accessing the file, to calculate the relative addresses for branches, to dis-assemble opcodes, to search the file for eight- or sixteen-bit patterns, to display and change locations in the file, and to change the starting execution address of the file.

If the file <name> exists and is of the proper format, the PATCH command will display the following:

```
nnnn cc
>
```

The "nnnn" is the absolute hexadecimal address of the lowest location of the memory-image file and is used as the initial offset (section 21.2.2). The "cc" is the hexadecimal content of that location. The second line is the PATCH input prompt. The following sections describe the various commands that comprise the PATCH command set.

21.2 PATCH Command Set

Each command to PATCH must be entered by the operator after the input prompt (>) is displayed on the system console. Like all MDOS input, all commands must be terminated by a carriage return. In the following command descriptions these symbols are used:

Symbol	Meaning
m,n	Both "m" and "n" are one to four digit hexadecimal numbers.
c	"c" is a one or two digit hexadecimal number.
a	"a" is an ASCII character.
<str>	"<str>" is a string of elements separated by commas. Each element can be a "c" or a group of "a"s enclosed in double quotes.
i	"i" is a valid M6800 assembly language mnemonic (M6809 assembly language mnemonic if using MDOS09).
	The period symbol represents the current position within the file <name>. It takes on the value of the current absolute address minus the current offset.
*	The asterisk represents the assembler location counter when used in the operand field of instructions.
<cr>	"<cr>" is a carriage return.

21.2.1 Quit -- Q

The Q command is used to terminate PATCH and return control to MDOS. The format of the Q command is simply the letter "Q". Any changes to the file which are still in memory will be written into the file before PATCH is terminated.

21.2.2 Set/display offset -- O

The O command is used to display and/or change the value of the current offset. The offset is used as a base address to which the location parameters of the other PATCH commands are added to arrive at an absolute address within the file. The format of the O command is

```
[m[,n]]O
```

If the parameters "m" and "n" are not specified, the O command will display the current value of the offset. For example,

```
>O
OFFSET=2000
```

If either of the parameters "m" or "n" are specified, the current value of the offset will be changed to either the single value "m", if only "m" is specified, or to the value "m plus n", if both parameters are present. The following sequence of commands illustrates both forms of the O command:

```
>A01FO
>O
OFFSET=A01F
>1234,5678O
>O
OFFSET=68AC
```

21.2.3 Display single location

The command to display the contents of a single location within the file has the following format

```
[m[,n]]<cr>
```

If both "m" and "n" are omitted, only a single carriage return is entered. This form of the command will cause the next sequential location of the file to be displayed. Since PATCH initializes the current location to the first location of the file when first invoked, the carriage return by itself can be used to step through the file showing a byte at a time, as in the following example.

```
=PATCH TESTPROG
2000 30
>
2001 32
>
2002 30
>
2003 30
>
2004 0E
>Q
=
```

If either "m" or "n" are entered prior to the carriage return, the effect of the command will be to display the contents of location "m plus the current offset" or the contents of location "m plus n". For example,

```
=PATCH TESTPROG
2000 30
>O
OFFSET=2000
>10
2010 2D
>100
2100 0D
>200,2000
2200 A6
>1000,1000
2000 30
>Q
=
```

21.2.4 Display lowest address -- L

The L command is used to change the current location to the lowest address of the file. The contents of the lowest address will also be displayed. The format of the L command is simply the letter "L".

Initially, when the PATCH command is started, the lowest address is shown automatically. The L command can be used to return to this point of the file at any time. Locations at addresses numerically less than "L" cannot be accessed since they do not correspond to any diskette space allocated to the file.

21.2.5 Display highest address -- H

The H command is used to change the current location to the highest address of the file. The contents of the highest address will also be displayed. The format of the H command is simply the letter "H". Locations at addresses numerically greater than "H" cannot be accessed since they do not correspond to any diskette space allocated to the file.

21.2.6 Calculate relative address -- R

The R command is used to calculate the relative address between any two locations in the file. The format of the R command is

m[,n]R

The R command will calculate the relative address between the current location in the file and the address "m plus the current offset" or the address "m plus n". The following example illustrates the use of the R command. It is assumed that the locations used in the example are the second bytes of branch instructions.

```
=PATCH LOG.CM
8200 00
>BA
82BA 05
>C0R
REL ADDR=0005
>119
8319 F9
>113R
REL ADDR=FFF9
>Q
=
```

The first relative address is in the forward direction. The second relative address is in the backward direction. The relative address is shown as a sixteen-bit number, even though only eight bits are required for the operand of the M6800 branch instructions.

21.2.7 Dis-assemble operation code -- I

The I command is used to convert a one-byte operation code into its equivalent M6800 or M6809 assembly language mnemonic. The format of the I command is

cI

where "c" is the one-byte hexadecimal operation code for MDOS. For MDOS09, "c" may be a one- or two-byte hexadecimal operation code. If two bytes, the first byte must be 00, 10, or 11. The contents of the file are not affected by the I command. For MDOS, the format of the assembly language mnemonic that is displayed is the following:

```
MMM [[A or B] [#]{HH or HHHH or RR} [,X]]
```

For MDOS09, the format of the assembly language mnemonic that is displayed is the following:

```
MMM [[A or B] [#]{HH or HHHH or RR or RRRR or RL or R,R}
[,R]]
```

The symbols take on the following meanings:

Symbol	Meaning
MMM	The three-character mnemonic or base mnemonic.
A or B	The accumulator specification for accumulator instruction types.
#	The immediate addressing mode operand qualifier (cannot appear concurrently with ",X", "RR", "RRRR", "RL", "R,R", or ",R").
HH	A one-byte hexadecimal operand.
HHHH	A two-byte hexadecimal operand.
RR	A one-byte hexadecimal operand indicating relative addressing mode (cannot appear concurrently with "#", ",X", or ",R").
,X	The indexed addressing mode operand qualifier (cannot appear concurrently with "#", "HHHH", or "RR").
RRRR	A two-byte hexadecimal operand indicating relative addressing mode (cannot appear concurrently with "#" or ",R").
RL	The operand is a register list (cannot appear concurrently with "#" or ",R").
R,R	The operand is a register pair (cannot appear concurrently with "#" or ",R").
,R	The indexed addressing mode operand qualifier (cannot appear concurrently with "#", "RR" or "RRRR").

The following example for the M6800 illustrates the different types of displays that can be generated by the I command.

```
                    =PATCH TESTPROG
                    2000 30
                    >8BI
                    ADDA #HH
                    >9BI
                    ADDA HH
                    >ABI
                    ADDA HH,X
                    >BBI
                    ADDA HHHH
                    >53I
                    COMB
                    >8DI
                    BSR RR
                    >BDI
                    JSR HHHH
                    >29I
                    BVS RR
                    >Q
                    =
```

21.2.8 Set search mask and pattern -- M

The M command is used to initialize a sixteen-bit search pattern and a sixteen-bit search mask for subsequent byte or word searches (sections 21.2.9-21.2.12). The format of the M command is

[m][,n]M

where "m" is the search pattern and "n" is the search mask. Initially, both the search pattern and the search mask are set to zero. The M command can be used to set both pattern and mask or to set either independently of the other. For example,

E5E5M

will set only the search pattern to the hexadecimal number $E5E5. The search mask is unaffected; however, the command

,FFFFM

will set only the search mask to the hexadecimal number $FFFF. The search pattern is unaffected. The command

E5E5,FFFFM

will set both the search pattern and the search mask.

21.2.9 Search for byte -- S

The S command is used to search the file for a specific eight-bit pattern. The format of the S command is

m,nS

where "m" and "n" represent the starting and ending addresses of the search. The addresses are both modified by the current value of the offset. The pattern to be searched for must have been specified via the M command (section 21.2.8). Only the least significant bytes of the search pattern and the search mask are used by the S command. The S command will display all addresses that contain patterns which meet the search criteria. The locations of the file included in the search is from address "m plus offset" to "n plus offset", inclusive. A match is indicated if a byte in the file meets the following condition:

contents of address & search mask = search pattern

where the "&" indicates the logical "and" function. The following example illustrates the use of the S command:

```
=PATCH TESTPROG
8200 30
>00EE,FFFFM
>0,1D7S
82A7 EE
82AD EE
82AF EE
>Q
=
```

21.2.10 Search for word -- W

The W command is similar to the S command; however, instead of searching for only a single byte, a double byte, or word, is searched for. The format of the W command is

m,nW

The address range searched with the W command is from "m plus offset" to "n plus one plus offset", inclusive. Thus, "n" cannot be the highest address of the file, since "n+1" would be an illegal address. Otherwise, the W command functions identically to the S command.

21.2.11 Search for non-matching byte -- N

The N command is similar in format and function to the S command; however, instead of displaying all bytes that meet the search criteria, all bytes that do not meet the search criteria are shown. This makes it easy to search through a buffer of all zeroes, for example, to find any non-zero locations.

21.2.12 Search for non-matching word -- X

The X command is similar in format and function to the W command; however, instead of displaying all double bytes that meet the search criteria, all double bytes that do not meet the search criteria are shown.

21.2.13 Display range of locations -- P

The P command prints the contents of a range of locations on the system console. The format of the P command is

```
m,nP
```

where locations "m plus offset" through "n plus offset", inclusive, are the locations to be shown. The format of the display is illustrated in the following example:

```
=PATCH TESTPROG
8200 30
>95,DOP
8290 0090 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
82A0 00A0 00 00 00 00 00 3F 32 EE 04 FF 80 04 30 EE 00 EE .....?2.....0...
82B0 00B0 06 FF 80 06 CE 80 00 3F 05 24 05 5F 3F 20 3F 1A .......?.$._? ?.
82C0 00C0 3F 33 3F 05 24 03 7E 03 D3 7E 04 32 30 31 30 30 ?3?.$.....20100
82D0 00D0 00 00 00 00 43 4F 4E 53 4F 4C 45 20 4C 4F 47 20 ....CONSOLE LOG
>Q
=
```

The contents of the locations are shown in both hexadecimal and the equivalent displayable ASCII. If a location contains a non-displayable character, it is shown as a period (.). The first four-digit number contains the absolute address while the second four-digit number contains the relative address of the locations (relative to the beginning of the file). Even though the starting location requested was $95, the displayed locations start at location $90. A full sixteen locations are displayed for each line, regardless of the requested starting and ending points of the range.

21.2.14 Set/display execution address -- G

The G command is used to display and/or change the value of the file's starting execution address. The format of the G command is

```
[m[,n]]G
```

If the parameters "m" and "n" are not specified, the G command will display the current value of the execution address. The following example illustrates this use of the G command:

```
=PATCH TESTPROG
8200 30
>G
EXEC ADR=8259
>Q
=
```

If either of the parameters "m" or "n" are specified, the current value of the execution address will be changed to "m plus offset" or "m plus n". The execution address must be within the range of addresses spanned by the file (between addresses shown with L and H commands). The following example shows how the G command is used to change the starting execution address:

```
=PATCH TESTPROG
8200 30
>G
EXEC ADR=8259
>2G
>G
EXEC ADR=8202
>Q
=
```

21.2.15 Change locations

Two commands exist that will open a specified location within the file and allow the contents of that and subsequent locations to be examined or changed. The format of these commands is

```
m[,n]{/ or \}[<str>]
```

where the slash (/) and backslash (\) characters are used to distinguish between the two commands. Both commands will open the specified location ("m plus offset" or "m plus n"). The slash command will set the "increment" mode. The backslash command will set the "decrement" mode. The parameter <str> contains any changes that are to be applied to the specified locations. If the "increment" mode is set (slash command), any changes specified in <str> will be applied to the opened location and each subsequent higher location, one increment being applied for each element of the string. If the "decrement" mode is set (backslash command), any changes specified in <str> will be applied to the opened location and each preceding lower location, one decrement being applied for each element of the string. If any of the elements of the string are null, an increment (or decrement) will still be applied for those elements. Thus, if the entire string is null (one null element), one increment (or decrement) will be applied. The "increment" or "decrement" modes will remain in effect until changed by another slash, backslash, or parenthesis command (section 21.2.16).

The string <str> can contain either hexadecimal elements or ASCII string elements, in any combination. For example, the command

```
1500,0/AA,1,2E,"AABBCC"
```

will change the following locations to the indicated values:

Absolute Address	New value
1500	$AA
1501	$01
1502	$2E
1503	$41
1504	$41
1505	$42
1506	$42
1507	$43
1508	$43

If the backslash command had been used instead, locations $14FF, $14FE, etc., would have received the values $01, $2E, etc.

An element of the string can be null (indicated by successive commas). Null elements will not affect the location that corresponds to that part of the string.

If an error is encountered in the string of elements <str>, the entire command will be ignored and no changes will be applied. An error message is printed to indicate that the command was not parameterized properly.

21.2.16 Instruction mnemonic decode mode

The instruction mnemonic decode mode is similar to the slash command explained above. Instead of using a slash, however, the open-parenthesis character (() is used. This command allows changes to be applied to a series of locations in the file using M6800 or M6809 assembly language mnemonics instead of the hexadecimal operation codes. The format of the command is

```
m[,n]([i][)]
```

where "m" and "n" specify the starting location (either "m plus offset" or "m plus n"), the open-parenthesis character signifies the start of the instruction mnemonic decode mode, "i" can be any valid M6800 assembly language mnemonic (M6809 assembly language mnemonic for MDOS09), and the close-parenthesis character indicates the end of the instruction decode mode. Since the close-parenthesis is optional, the user can remain in the instruction mnemonic decode mode to enter several lines of instructions until a close-parenthesis character is entered.

Once the open-parenthesis command has been issued, all other PATCH commands are invalid until the close-parenthesis command is issued, or until an error is encountered.

The format of the commands following the open-parenthesis command is shown below:

```
<blanks> ) <any> <cr>

or

<blanks> <opcode> [<blanks> <operand>] [<any> ) <any>] <cr>
```

The syntactic elements are described as follows:

Element	Meaning
<blanks>	Any number of spaces, including zero.
<any>	Any character besides a carriage return or a close-parenthesis.
<cr>	Carriage return.
<opcode>	Any valid assembly language mnemonic as specified in the "M6800/M6801/M6805/M6809 Macro Assemblers Reference Manual"; no space is allowed between the mnemonic and the accumulator designator (e.g., LDAA is valid, LDA A is not).
<operand>	Only valid if the instruction requires an operand. If no operand is required, the <operand> is treated as <any>.

The <operand> field, when required, has the following format:

```
[#]<arg>[{+ or -}<arg>]

          or

[<arg>[{+ or -}<arg>],]X
```

where the "#" indicates immediate addressing mode and ",X" indicates the indexed addressing mode. The "+" or "-" allows simple expressions to be used in the operand field. Each of the arguments <arg> can be one of the following kinds of elements:

Element	Meaning
'A	A one-character ASCII literal.
$HHHH	A one to four digit hexadecimal number.
DD...D	A decimal number, any number of digits in length; only the least significant 8 or 16 bits of the converted number will be used.
%BB...B	A binary number, any number of digits in length; only the least significant 8 or 16 bits of the converted number will be used.
*	The value of the current location counter (identical to the "*" used by M6800 assembler).
O	The value of the current offset.

For MDOS09, the <operand> field is expanded to allow register lists, indirect, auto-increment, auto-decrement, and forced direct/extended. PATCH automatically generates direct mode instructions only when the most significant byte of the expression is zero. In all other cases, the direct mode must be forced by the user. Reference "M6800/M6801/M6805/M6809 Macro Assemblers Manual."

This format allows the operator to enter assembly language mnemonics with comments after the operand field for documenting the patch. The instruction mnemonic decode mode automatically puts the PATCH command into the "increment" mode.

As long as a close-parenthesis character is not encountered, PATCH will remain in the instruction mnemonic decode mode. A different input prompt is displayed to distinguish the two different PATCH input modes; the normal input prompt (>) is replaced the by the instruction mnemonic decode mode prompt (=>).

The following M6800 example illustrates how the instruction mnemonic decode mode is used to insert a patch into a file:

```
Line   Console Display
----   ---------------

 01    =PATCH TESTPROG
 02    8200 30
 03    >O
 04    OFFSET=8200
 05    >F7
 06    82F7 CE
 07    >.(JMP $8317 GO TO THE PATCH AREA OF PROGRAM)
 08    >8317,O(LDX #O+$A THE LDX OVERLAYED BY THE JMP
 09    =>STX O+$D2
 10    =>SWI THIS IS A SYSTEM FUNCTION CALL)
 11    ./1D
 12    .(BEQ *+5 IF NO ERRORS, CONTINUE
 13    =>JMP O+$113 GO PROCESS ERROR
 14    =>LDX X PICK UP THE POINTER
 15    =>LDAA 0,X GET A CHARACTER
 16    =>CMPA #'1 IS IT UNIT 1?
 17    =>BNE *-10 GO PROCESS ERROR
 18    =>JMP $82FD RETURN TO MAIN CODE
 19    =>)
 20    >Q
 21    =
```

In the above example, line 03 was used to display the value of the current offset. Line 05 was used to display the contents of location $F7, relative to the beginning of the file. Line 07 was used to enter the instruction mnemonic decode mode to modify the current location (offset + $F7). Three locations were changed as a result of entering line 07. Line 08 was used to reenter the instruction mnemonic decode mode; however, this time absolute location $8317 was the address where a patch was to be placed. Line 11 was used to insert a hexadecimal constant into the location following the previously entered SWI instruction. Line 12 was used to return to the instruction mnemonic decode mode at the location following the hexadecimal constant inserted using line 11. Line 19 was used to finally exit the instruction mnemonic decode mode. Line 20 was used to exit the PATCH command and return control to MDOS. Comments were used throughout the instruction mnemonic decode mode to document what the patch does.

21.3 Special Considerations

The period symbol (.) can be used with any PATCH command that requires an address as an argument. The value associated with the period symbol is the absolute address of the current location minus the value of the current offset. Since the offset is automatically added to most of the command parameters, the resulting value for the period symbol will be the absolute address of the current location.

For example, the following uses of the period can save time and eliminate remembering the address of the current location:

Command	Function
.,nO	Sets the offset to the current location if "n" is the value of the offset before the command is entered.
.<cr>	Displays the contents and the address of the current location.
./<str>	Opens the current location and applies the changes from the string <str>. It is not necessary for the operator to count the number of elements in <str> if the next command is to apply more changes. Long strings are usually changed by initially using the "m,n/" form of the change command. Then, subsequent changes use "./". The same holds true for the backslash and open-parenthesis commands used with the period symbol.
.,nS	Search from the current location to the address "n plus offset".
m,.P	Display locations "m plus offset" to the current location.

21.4 Error Messages

The following messages can be displayed by the PATCH command. The standard error messages that can be displayed by all commands are not listed here.

WHAT?

The command issued in response to the PATCH input prompt (>) was not recognized. A new input prompt is displayed.

SYNTAX ERROR

The command issued in response to the PATCH input prompt (>) was recognized; however, it was parameterized illegally. A new input prompt is displayed. The command has not been processed.

ILLEGAL ADDRESS

An address was specified which referenced a location that was outside of the range of addresses spanned by the file. Only addresses between the lowest (L command) and the highest address (H command) can be referenced by PATCH. If new program area is to be allocated for additional patch space, a merge process, reassembly process, or link/load process must be used to create the new space.

ILLEGAL OP CODE

The instruction mnemonic decoder did not recognize a valid M6800 assembly language mnemonic. The instruction mnemonic decode mode is terminated. The current instruction was not used to change the file. This error can also occur if an invalid M6800 operation code is given as the operand of the "I" command.

ILLEGAL OPERAND

An illegal operand was used in the operand field of the instruction. The instruction mnemonic decode mode is terminated. The current instruction was not used to change the file.

INITIALIZATION ERROR

This error indicates some sort of internal system malfunction. Errors of this type indicate a hardware failure or damaged program files on the diskette.

CHAPTER 22

22. REPAIR COMMAND

The REPAIR command allows the user to check and repair a malfunctioning or a non-functioning MDOS diskette. Errors in the system tables can be found, identified, and corrected with this command. Since MDOS performance is directly related to the correctness of these system tables, the REPAIR command is a useful diagnostic utility. The REPAIR command works with either single-sided or double-sided MDOS diskettes.

22.1 Use

The REPAIR command is invoked with the following command line:

```
REPAIR [:<unit>]
```

where <unit> is the logical unit number on which a diskette that is to be "repaired" resides. If no <unit> is given, logical unit number zero will be used as a default.

The REPAIR command runs through five different phases:

1. ID , LCAT, CAT, and Bootblock sector check phase,

2. Directory sector check phase,

3. Retrieval Information Block check phase,

4. CAT regeneration phase, and

5. CAT replacement phase.

Each of the different phases is described in detail in the following sections.

REPAIR progresses from each phase to the next carrying along information that was obtained during a prior phase. If errors are discovered, the operator will be notified via the system console. If REPAIR can fix the error, the operator will also be asked if the error should be corrected on the diskette. Thus, the operator has complete control over any changes that are made to the diskette. The operator can suppress any action that may be suggested by the REPAIR command as the means for correcting an error.

The amount of knowledge about the MDOS tables that is required by the operator depends upon two things: the amount of actual damage on the diskette and the amount of information the operator wants to recover from the damaged tables.

If the operator merely permits REPAIR to perform every suggested action to correct every error, then the resulting diskette is guaranteed to have error free system tables. In this case, the amount of systems knowledge required is insignificant.

On the other hand, if the operator takes notes during the REPAIR command on what tables are damaged, and if the operator does not choose to delete those files that are invalid, then a great deal about the the MDOS file structure and system tables must be known to reconstruct the tables. Chapter 24 describes the system structure in detail. It is required reading for a complete understanding of all the functions and the errors that the REPAIR command can perform and detect.

The REPAIR command must be invoked from a working MDOS diskette. Thus, if a given diskette cannot be used for initialization, it must be placed into drives one, two, or three, and another working diskette (of the same MDOS version as the dammaged diskette) placed into drive zero before the REPAIR command can be used.

REPAIR does not attempt to find errors within data files. It only attempts to find errors within the system tables.

It is suggested that REPAIR be used for the following reasons:

1. As a regular diskette checking utility. It never hurts to run REPAIR as a preventative maintenance tool to catch errors as they may be developing, before serious malfunctions are noticed. If nothing is wrong with a diskette, no operator interaction is required. REPAIR will simply return to MDOS after having displayed some monitoring information.

2. If strange things start happening or if system error messages are displayed without apparent reason. If files or records within files disappear or get scrambled, the system tables may have been damaged.

3. If MDOS will not run at all.

4. After the ABORT or RESTART pushbuttons were depressed to stop the system while diskette transfers were in progress.

5. After a power failure occurred while diskette transfers were in progress. Power failures include those caused by inadvertently switching off the EXORciser or EXORdisk II as well as those that affect an entire installation.

6. After a diskette has had its system tables repaired manually with the DUMP command. This ensures that the tables were corrected properly.

22.2 ID, LCAT, CAT, Bootblock Sector Check

Phase 1 of REPAIR begins by checking the ID sector for readability. If an error occurs during the read attempt, REPAIR will display the following:

```
**PROM I/O ERROR-STATUS=31 AT 2C4C ON DRIVE 1-PSN 0000
ID SECTOR READ ERROR
WRITE TO DISK TO ATTEMPT TO CLEAR ERROR?
```

The actual error status, address, and drive number of the first line will vary depending on the type of read error that was detected, the version of REPAIR being used, and the drive in which the diskette resides. The same is true for all of the PROM I/O error messages given in the examples of this chapter. A response of either "N" or "Y" must be made by the operator. The "N" response will cause the message

```
ID SECTOR CANNOT BE CHECKED
```

to be displayed. Since the other system tables could still be accessed, REPAIR will continue. If a "Y" response is given, the ID sector will be re-written in an attempt to clear the error. If an error develops during the write, the ID sector is considered unfixable; however, in this case, the other system tables could still be accessed, so REPAIR will continue.

If the ID sector can be read initially without error, or if the ID sector can be rewritten without error, the contents of the ID sector will be displayed as follows:

```
DISK ID:   MDOS0300
VERSION:   03
REVISION:  00
DATE:      072578
USER:      SYS DEVELOPMENT DISK
```

Each field within the ID sector is checked by the REPAIR command. The following table shows what tests are made for the respective fields:

Field	Test Performed
DISK ID	MDOS file name format
VERSION	Same as MDOS.SY
REVISION	Same as MDOS.SY
DATE	ASCII numeric
USER	Displayable ASCII
Remainder	Binary zero, excluding MDOS RIB address area

If the fields in the ID sector fail to meet the above criteria, the field's name will be displayed as a prompt to the operator to enter a correct value. If only a carriage return is entered in response to such a prompt, the ID sector field will not be changed. Otherwise, the entered field will be checked for correctness and then stored into the ID sector.

The version and revision numbers in the ID sector are compared against those of the resident operating system file on diskette. If the numbers are not identical, REPAIR will use the version/revision numbers from the MDOS file since the diskette cannot be initialized if they are not the same. The message:

```
 VERSION AND REVISION NUMBERS IN ID SECTOR AND RESIDENT MDOS
                       FILE ARE DIFFERENT
     THE NUMBERS IN THE ID SECTOR ARE CHANGED TO:  vv.rr
```

to indicate the correction. The numbers "vv" and "rr" are the version and revision numbers of the resident operating system file, respectively. The operator has no control over what the version/revision numbers are in the ID sector. Thus, those two fields cannot be supplied by the operator. In the event that a diskette controller error occurs when trying to read the correct version/revision numbers from the MDOS file, the message

```
**PROM I/O ERROR-STATUS=31 AT 2E8A ON DRIVE 1-PSN 0019
RESIDENT MDOS CANNOT BE LOADED -- SECTOR READ ERROR
```

will be displayed. The diskette being repaired cannot be used in drive zero since the operating system cannot be read; however, REPAIR will continue to check the remaining system tables.

If the unused area of the ID sector has been damaged, the message

```
ID UNUSED AREA NOT ZERO.  ZERO IT?
```

The operator must respond with either a "Y" or an "N". The "Y" response will cause the ID sector's unused area to be filled with binary zeroes, as it is supposed to be. The "N" response will cause REPAIR to leave the ID sector alone.

After the ID sector has been checked, REPAIR will examine the Lockout Cluster Allocation Table (LCAT) for readability. If the LCAT sector cannot be read, REPAIR will display the following messages:

```
**PROM I/O ERROR-STATUS=31 AT 2E8A ON DRIVE 1-PSN 0002
LOCKOUT C.A.T. READ ERROR
WRITE TO DISK TO ATTEMPT TO CLEAR ERROR?
```

The operator must respond with either a "Y" or "N" to the last question. If an "N" is entered, REPAIR cannot continue to check other system tables since subsequent checking is based on the validity of the LCAT. Thus, the message

```
DISK IS NOT FIXABLE
```

is displayed and control returned to MDOS. If a "Y" response is given, REPAIR will attempt to rewrite the LCAT sector. If an error develops during the write, the sector will be considered unfixable (as will the diskette). The message shown above will be displayed and MDOS given control.

If the LCAT sector is readable, or if rewriting the sector clears the error, REPAIR will proceed to check the contents of the LCAT. The LCAT must show that the diskette's system tables in the first cylinder are locked out (unavailable for allocation by a file), and all regions of the diskette that correspond to non-physical locations (beyond the highest physical sector number) must be locked out.

If either of these two criteria is not satisfied, the LCAT will be considered destroyed. REPAIR will display the message

```
LOCKOUT C.A.T. IN ERROR - RECONSTRUCT?
```

and await a response from the operator. An "N" response will make the LCAT unfixable. REPAIR will display a message to that effect and return to MDOS. A "Y" response will cause a new LCAT to be rebuilt by REPAIR. In order to build a new LCAT, the entire diskette is read in an attempt to find any deleted data marks. The deleted data marks signify bad clusters found by the DOSGEN surface test (Chapter 10). All clusters containing deleted data marks will be locked out again automatically by this process. In addition, the operator can lock out an additional area of the diskette (for the same reasons as specified in Chapter 10). After the diskette's surface has been completely read, REPAIR will display the message

```
WHICH SECTOR RANGE IS TO BE LOCKED OUT?
```

The operator can respond with a carriage return to indicate that no additional sectors are to be locked out. Otherwise, the operator can respond with a range of sector numbers entered in the format

```
mmm-nnn
```

where "mmm" and "nnn" are hexadecimal numbers of sectors that start on a cluster boundary (sector number is evenly divisible by four). If an illegal sector number is entered, or if the starting number is greater than the ending number, the above message will be redisplayed until the operator enters a valid range or a single carriage return. Only one contiguous range of sectors can be locked out. The same cautions described in Chapter 10 regarding user-locked out sectors apply here; however, in this case, since files already reside on the disk with allocated space, the locked out sectors must not conflict with any files. If a diskette did not have user-locked out sectors before, then sectors must not be locked out during the REPAIR process since they could conflict with sectors already allocated. The REPAIR command is not intended to be used for the normal lockout procedure; that is the function of the DOSGEN command (Chapter 10). If a diskette did have sectors locked out, then the identical sectors must be locked out by the operator again here.

After the LCAT has been rebuilt, or if it was good to begin with, the Cluster Allocation Table (CAT) will be checked. If the CAT sector cannot be read, the following message will be displayed:

```
**PROM I/O ERROR-STATUS=31 AT 2E8A ON DRIVE 1-PSN 0001
C.A.T. READ ERROR
WRITE TO DISK TO ATTEMPT TO CLEAR ERROR?
```

The operator must respond with either a "Y" or an "N" to the last question. If an "N" is entered, REPAIR cannot continue to check the other system tables since subsequent checking is based on the validiy of the CAT. Thus the message,

```
DISK IS NOT FIXABLE
```

is displayed and control returned to MDOS. If a "Y" response is given, REPAIR will attempt to rewrite the CAT sector. If an error develops during the write, the sector will be considered unfixable (as will the diskette). The message shown above will be displayed and MDOS given control.

If the CAT sector is readable, or if rewriting the sector cleared the error, REPAIR will proceed to check the contents of the CAT. The CAT must show that all parts of the diskette locked out by the LCAT are flagged as allocated (see above for LCAT validity criteria). If the CAT contains an error at this point, REPAIR will display the message

```
C.A.T. IN ERROR - RECONSTRUCT?
```

and await a response from the operator. An "N" response will result in an unfixable diskette. REPAIR will show the message

```
DISK IS NOT FIXABLE
```

and return control to MDOS. A "Y" response will cause a new CAT to be reconstructed from the information gathered in Phases 2 through 4.

After checking the CAT, REPAIR will attempt to read the Bootblock sector. If the Bootblock sector cannot be read, REPAIR will display the following message:

```
**PROM I/O ERROR-STATUS=31 AT 2EDC ON DRIVE 1-PSN 0017
BOOT BLOCK SECTOR READ ERROR
WRITE TO DISK TO ATTEMPT TO CLEAR ERROR?
```

The operator must respond with either a "Y" or "N" to the last question. If an "N" is entered, REPAIR will display the message

```
BOOT BLOCK SECTOR CANNOT BE CHECKED
```

before continuing. Since the Bootblock is not affected by other system tables, REPAIR will continue to check the remainder of the diskette; however, a diskette with a damaged Bootblock sector cannot be used as an MDOS diskette in drive zero. If a "Y" is entered, REPAIR will attempt to rewrite the sector in an attempt to clear the error. If an error develops during the write, the sector is unfixable and the diskette can never be used to initialize the system from drive zero.

If the Bootblock sector is readable or if the error is cleared by rewriting the sector, REPAIR will verify that the sector contains a valid copy of the Bootblock program. If the data is in error, the message

```
BOOT BLOCK SECTOR HAS BEEN DESTROYED
WRITE TO DISK TO ATTEMPT TO CLEAR ERROR?
```

will be displayed. An "N" response wil leave the Bootblock sector unchanged. A "Y" response will cause a new Bootblock to be written to the diskette. The REPAIR command will then begin Phase 2.

22.3 Directory Sector Check

Phase 2 of REPAIR deals entirely with the MDOS directory sectors. Each of the directory sectors is first checked for readability. If a read error is found, the operator is informed and given the choice of trying to clear the read error via the following display:

```
**PROM I/O ERROR-STATUS=31 AT 2F38 ON DRIVE 1-PSN 0013
DIRECTORY SECTOR READ ERROR
WRITE TO DISK TO ATTEMPT TO CLEAR ERROR?
```

The actual numbers in the error message will depend on the actual sector that is in error. If the operator responds with an "N", or if the rewrite attempt ("Y" response) fails to clear the error, the message

```
DISK IS NOT FIXABLE
```

will be displayed and control returned to MDOS. If the sectors are all readable, or if the rewrite attempt succeeded, each directory sector is examined again. This time, each directory entry within each sector is tested against the following criteria.

1. If the first byte of the directory entry is zero (unused entry), then the remaining bytes of the entry must be zero also.

2. If the first byte of the directory entry is the hexadecimal number $FF (deleted entry), then the second byte of the entry must be $FF also. If the second byte is not $FF, and if the remainder of the entry is valid, then the entry is the result of an incomplete name change. It was probably caused by a power failure or interrupt (ABORT or RESTART pushbuttons) during the time that the old name was deleted and the new name was added to the directory. REPAIR will allow the operator to delete the directory entry entirely or to reassign a name to the partially deleted entry. The name assigned must be the same as the original one. Otherwise, the name will probably be improperly placed in the directory (criterion 5).

3. The physical sector number of the Retrieval Information Block must the first sector of a cluster, must not be the sector number of one of the system tables checked in Phase 1 or 2, and must not be greater than the highest valid physical sector number.

4. The directory entry's attribute field must have the least significant byte (unused) set to zero. In addition, the two unused bytes at the end of a directory entry must be set to zero.

5. The calculated hash index for the file name and suffix must locate the directory entry where it currently resides. An error in the hash index means that the directory entry is inaccessible. Appendix G contains a detailed description of the hashing method.

6. The system file MDOS.SY must have a Retrieval Information Block in a specific physical sector. In addition, the other files in the family MDOS*.SY must be present in the directory.

If any directory entry fails to meet one of the first five criteria, REPAIR will display the entry in error as well as a message identifying the problem. The directory entry is displayed in the following format:

```
PSN LSN EN NAME SUF RIB ATTR NU [HEXNAM HEXSUF]
```

where the symbols have the following meanings:

Symbol	Meaning
PSN	Directory sector's physical sector number
LSN	Directory sector's logical sector number
EN	Entry number within sector
NAME	File name
SUF	File suffix
RIB	Physical sector number of RIB
ATTR	Attributes
NU	Not used portion of directory entry
HEXNAM	File name in hexadecimal
HEXSUF	Suffix in hexadecimal

All of the fields are displayed as hexadecimal numbers with the exception of the file name and suffix. If non-displayable characters appear in either the file's name or suffix, they will be shown as percent signs (%). In such cases, the hexadecimal forms of the file name and suffix are shown to the right of the directory entry.

In the following examples, the same directory entry is used so that the changes from one to the other can be more easily detected. The first line always shows the directory entry. The second line contains the error message and a prompt to the user. If a "Y" is entered, the entry will be removed from the directory (and later the space associated with that directory entry will be deallocated). An "N" response will leave the directory entry unchanged.

The following message is shown for directory entries that fail to meet criterion 1. Not all bytes of the entry are zero if first byte is zero.

```
03 00 00  %INEX   .CM  014C  7200 0000  00494E4558202020434D
DIRECTORY ENTRY IN ERROR.  DELETE?
```

The following message is shown for directory entries that fail to meet criterion 2. The directory entry is the result of an incomplete name change. Instead of asking the operator if the file name should be deleted, REPAIR allows the original name to be reassigned. If no name is entered in response to the prompt (carriage return only), the directory entry will fail criterion 2, so the entry will be redisplayed as in the above example. If the original name is supplied, the file's directory entry will be recreated in the directory. The content of the file is unaffected; however, if a name is assigned other than the original, criterion 5 will probably not be satisfied. The directory entry would then be displayed again, with the corresponding error message.

```
03 00 00  %INEX   .CM  014C 7200  0000  FF494E4558202020434D
POSSIBLE INCOMPLETE NAME CHANGE
NEW NAME:
```

The following example illustrates a directory entry that fails to meet criterion 3. The RIB address is of the directory entry is invalid. In this case, the RIB address is a sector that is not on a cluster boundary.

```
        03  00 00  BINEX   .CM  014D  7200 0000
        INVALID RIB SECTOR NUMBER. DELETE?
```

The next example shows a directory entry that fails to meet criterion 4. The directory entry's attribute field has a non-zero unused byte.

```
        03  00 00  BINEX   .CM  014C  72FF 0000
        ILLEGAL ATTRIBUTE OR UNUSED BYTES.  DELETE?
```

The last example illustrates a directory entry that fails to meet criterion 5. The hash index for the file name and suffix places the directory entry into a different directory sector than the one in which it appears (file's original name is BINEX.CM).

```
        03  00 00  AINEX   .CM  014C  7200 0000
        HASH OR NAME DUPLICATION ERROR.  DELETE?
```

Criterion 6 does not deal with directory entries in general. Rather, the specific names of the system files are searched for in the directory to ensure they exist. The absence of any one of the system files is noted by the display of one of the following messages:

```
MDOS    .SY DOES NOT EXIST
MDOSER  .SY DOES NOT EXIST
MDOSOV0 .SY DOES NOT EXIST
MDOSOV1 .SY DOES NOT EXIST
MDOSOV2 .SY DOES NOT EXIST
MDOSOV3 .SY DOES NOT EXIST
MDOSOV4 .SY DOES NOT EXIST
MDOSOV5 .SY DOES NOT EXIST
MDOSOV6 .SY DOES NOT EXIST
```

In addition, if the resident operating system file does not have a RIB in the proper physical sector, the diskette could not be used for system initialization in drive zero. Thus, the message

```
MDOS.SY DOES NOT START AT SECTOR $18
```

is displayed in such cases.

Since errors in the directory entries are not fatal insofar as REPAIR is concerned (they can be if the diskette is to be used for initialization or to run any programs), Phase 3 is started after these checks have been completed.

22.4 Retrieval Information Block Check

Phase 3 of REPAIR checks the Retrieval Information Blocks (RIBs) of all directory entries that have a valid RIB address. If a RIB address is invalid in a directory entry, then the RIB cannot be found. The RIBs are checked in the order in which they are referenced in the directory. If a RIB sector cannot be read, the following message will be displayed:

```
**PROM I/O ERROR-STATUS=31 AT 30D8 ON DRIVE 1-PSN 0570
RIB READ ERROR
WRITE TO DISK TO ATTEMPT TO CLEAR ERROR?
```

The operator must respond with either a "Y" or an "N" to the last question. If a "Y" is entered, REPAIR will attempt to rewrite the RIB. If the error is cleared, REPAIR will continue. If an error occurs during the rewriting of the RIB, or if an "N" was entered, REPAIR cannot check the RIB any further. Thus, a message of the form

```
03  00 00  BINEX   .CM  014C  5200 0000
RIB IN ERROR - DELETE FILE?
```

is displayed to allow the operator to delete the file completely so it is not allocated space in Phase 4. The first line shows the directory entry that belongs to the file. It is in the same format as the directory entry explained in the previous section. If the file is not deleted ("N" response), it will not be affected, nor will the allocation table be updated. If the file is deleted ("Y" response), then whatever space was allocated to it will be marked as available for allocation in the reconstructed allocation table. If a RIB is in error, the content of the file is usually unaccessible unless the error is corrected by the user. If this cannot be done, the file should be deleted by responding with a "Y" to the above prompt.

If the RIB can be properly read, or if the RIB was properly rewritten, then REPAIR will continue to check the RIB for the following criteria. If the RIB fails to satisfy the criteria, an error message will be shown, followed by the directory entry and a prompt that allows the file to be deleted:

```
<cause of error>
03  00 00  BINEX   .CM  014C  5200 0000
RIB IN ERROR - DELETE FILE?
```

The actual content of the directory entry, however, will vary. The following messages can appear in place of the <cause of error> field.

FIRST SDW IN ERROR

This error message will be displayed if the first Segment Descriptor Word (SDW) does not contain the cluster number of the RIB as its starting cluster number. Since a RIB is the first physical sector of a file, it will always be in the file's first cluster. This message will also be displayed if the first SDW has the terminator bit set to one.

SDW BOUNDS ERROR

This error message will be displayed if an SDW has an invalid starting cluster number. Invalid cluster numbers are those that include the system table area of the diskette as well as areas beyond the maximum physical sector number.

If an SDW describes a segment which doesn't lie entirely within the boundaries of the diskette, this message will also be shown. That is, the contiguous clusters adjacent to the starting cluster of an SDW must also have valid cluster numbers.

RIB CLUSTER ALLOCATION DUPLICATION

This error message will be displayed if two SDWs describe the same physical cluseter. All SDWs must span unique segments of the diskette.

ILLEGAL SDW TERMINATOR

This error message will be displayed if the SDW that acts as the terminator for the other segment descriptors does not exist or if it contains a logical sector number (used for monitoring the logical end-of-file) that is not a part of the allocated file.

NON-CONTIGUOUS SDW ERROR

This error message will be displayed if files with the contiguous allocation attribute have SDWs that describe a segmented area of the diskette.

NON-0 BYTES AFTER SDW TERMINATOR

This error message will be displayed if bytes following the terminating SDW are not zero. Only files in the memory-image format can have non-zero bytes in the RIB following the terminator, and then only beginning with the 117th (decimal) byte of the sector (117 is relative to zero; zero being the first byte in the RIB).

BINARY LOAD FILE RIB ERROR

This error message can be displayed for a variety of reasons. The RIB of memory-image files contains special load information in the last eleven bytes of the sector. If those bytes do not meet the following specifications, this error message will be displayed. The offsets used to refer to the various bytes are relative to zero (zero being the first byte of the RIB sector). All offsets are given in decimal.

1. Byte 117, the number of bytes to load from the last sector, must be non-zero, a multiple of 8, and less than or equal to 128 ($80).

2. Bytes 118-119, the number of sectors to load, must contain a number that is non-zero, less than the total number of sectors allocated to the file, and less than or equal to 512 ($200).

3. Bytes 120-121, the starting load address, are not checked. For programs loading in an EXORciser I system, in the User Memory Map of an EXORciser II system with the single memory map configured, or in the Executive Memory Map of an EXORciser II system with the dual memory map configured, this value must be greater than hexadecimal location $1F if the program is to be loaded via the MDOS loader. EXORciser II systems with the dual memory map configured can have programs loaded into the User Memory Map starting at location zero.

4. The ending load address is calculated from bytes 117-121 in the following manner:

   $$EL = (NSL - 1) * 128 + NBLS + SL - 1$$

   where EL is the ending load address, NSL is the number of sectors to load (bytes 118-119), NBLS is the number of bytes in the last sector (byte 117), and SL is the starting load address (bytes 120-121). The ending load address must be less than 65536.

5. Bytes 122-123, the starting execution address, must lie within the range of addresses spanned by the program (greater than or equal to the starting load address, and less than or equal to the ending load address).

6. Bytes 124-127 are not used and must be zero.

Because of the complexity of the errors that can occur in a RIB, the REPAIR command will make no attempt to "fix" a RIB. If a RIB error is detected, REPAIR will give the operator a choice of deleting the file (thereby removing the RIB and fixing the problem) or leaving the RIB alone.

No space can be allocated to files with directory entries that have invalid RIB addresses or to files that have RIBs with detectable errors (since the allocation information is contained in the RIB). Thus, when REPAIR goes through the Phase 4, it will exclude all files with bad RIBs; however, the REPAIR command will not update the allocation table on diskette if files with bad RIBs are left undeleted. Thus, the files with bad RIBs should be deleted when REPAIR gives the operator the option to do so (the DEL command must not be used!), or they should be manually repaired via the DUMP command (Chapter 11) before the diskette is used. The DUMP command can be used to examine the damaged RIB and, if necessary, to examine where a file's sectors actually are on the diskette. DUMP's sector read, sector change, and sector write commands can be used to reconstruct a valid RIB. Sometimes, it will require less effort to recreate a file's RIB (if the allocation map has been recently printed via the DIR command) than to recreate the file itself.

After a RIB has been reconstructed, REPAIR should be run again to ensure that there is no dual allocation with another file.

After all of the RIBs have been checked, a summary is displayed to monitor REPAIR's progress. The summary information takes on the following format:

```
xx GOOD FILES  yy FILES WITH BAD RIBS
```

where "xx" and "yy" are both hexadecimal numbers. The display of this message indicates the end of Phase 3.

22.5 CAT Regeneration Phase

Phase 4 of the REPAIR command reconstructs a cluster allocation table in memory from the RIBs of those files that have no errors ("xx" in the Phase 3 summary message). Phase 4 consists of three passes.

Pass 1 of Phase 4 reads all valid RIBs. All clusters that are allocated are retained in memory in a table called Table 1. A second table, Table 2, also in memory, will contain all clusters which have been allocated to more than one file. If no dual allocation has occurred, Table 2 should be empty at the end of Pass 1. If it is, the rest of Phase 4 is skipped.

If Pass 1 has determined that dual allocation occurred, then Pass 2 of Phase 4 will read all RIBs a second time. This time, the files which have clusters allocated in Table 2 are flagged so the file's names and conflicts can be shown in Pass 3.

A summary message is displayed at the end of Pass 2 that gives totals of the number of files with and without dual allocation. The format of the summary message is

```
xx GOOD FILES  yy FILES WITH DUPLICATIONS
       zzzz ALLOCATION DUPLICATIONS
```

where "xx", "yy", and "zzzz" are all hexadecimal numbers. The totals "xx" and "yy" refer to numbers of files. The number "zzzz", however, refers to the number of clusters that are common to the "yy" files. The actual message is displayed on a single line.

Pass 3 of Phase 4 will perform an analysis of all files that have allocation conflicts with each other. The files are analyzed two at a time. The result of the analysis will be displayed in the following format:

```
09  06 00  RASM    .CM  031C  7200 0000
       SIZE: 001F   CONFLICTS: 001F CLUSTERS
     10  0D 01  FORLB   .RO  05D0  6300 0000
             SIZE: 0041   CONFLICTS: 001F CLUSTERS
     031C 0320 0324 0328 032C 0330 0334 0338 033C 0340 0344 0348
     034C 0350 0354 0358 035C 0360 0364 0368 036C 0370 0374 0378
     037C 0380 0384 0388 038C 0390 0394
```

The names of the files and the numeric data displayed differ, of course, depending on the exact files involved.

The first line of the display contains the directory entry of a file with which other files have duplicate allocation. The format of the directory entry is the same as during Phase 2 (section 22.3). Since this line is extended to the left further than the other lines, this file is referred to in the following description as the "Outer file". The second line of the display contains the total size of the Outer file in clusters (SIZE) and the total number of clusters that cause allocation conflicts (CONFLICTS). When the total size is compared to the part of the file that is in conflict, a relative indication can be obtained of the fraction of the file that may be in error. The CONFLICTS total for the Outer file includes the allocation conflicts with all "Inner files" (described below).

The third and fourth lines of the display are of the same format as the first two lines; however, these lines describe an "Inner file" that has allocation conflicts with the Outer file. Since more than one Inner file can be shown, the CONFLICTS total for each Inner file contains only the number of clusters in that file that cause allocation conflicts with the Outer file.

Following the two-line description of the Inner file will be a list of clusters (belonging to the Inner file) that conflict with the Outer file. The starting physical sector number is given for each cluster.

After the Outer file and one Inner file have been displayed in this format, REPAIR will issue the following prompt (data supplied to go along with the above example):

```
DELETE: NEITHER(1), BOTH(2), FORLB   .RO(3), RASM    .CM(4):
```

The above prompt allows the user to select the action that REPAIR is to take by entering a number from 1 to 4. Number 1 will cause neither the Inner nor the Outer file to be deleted. Number 2 will cause both files to be deleted. Number 3 will cause the Inner file to be deleted. Number 4 will cause the Outer file to be deleted.

As long as the Outer file is not deleted, all of the files that have conflicts with it will be displayed as Inner files. When all Inner files conflicting with the Outer file have been displayed in this fashion, REPAIR will take the next file in its list of files with allocation conflicts and use it as an Outer file. This process continues until all files with allocation conflicts have been dealt with.

Conflicting pairs of files will be printed only once. An Inner file may subsequently be displayed as an Outer file if it has additional conflicts with other files. As files are deleted, other files that were originally in conflict with it may no longer have allocation conflicts.

Usually, the REPAIR command will be used more than once if files happen to have allocation conflicts. The first time, the operator will pick the "NEITHER" selection from the above prompt. In this way, he can accumulate the information required to decide which files should be deleted and which should be retained. The DUMP command may be used to examine the conflicting clusters to see which file they actually belong to. Then, REPAIR is run a second time to actually delete the files in error. The files must not be deleted with the DEL command since it deallocates the files' space in addition to deleting the directory entries.

For files with allocation conflicts, one of the following statements may be true:

1. The Outer file may have a correct RIB and contain all valid data. Thus, the error is caused by the Inner files that have allocation conflicts with the Outer file.

2. The Outer file may have an incorrect or overwritten RIB. In this case, the Inner files having allocation conflicts with the Outer file are all correct.

3. Some of the Outer file's existing space may have been erroneously allocated to, and possibly overwritten by, an Inner file. In that case, since the Inner file was written to last, the Inner file contains valid data and has a valid RIB even though its space was allocated by error.

4. Some of the Inner file's existing space may have been erroneously allocated to, and possibly overwritten by, an Outer file. In that case, since the Outer file was written to last, the Outer file contains valid data and has a valid RIB even though its space was allocated by error.

5. A combination of 2, 3, and 4 may have occurred.

It is necessary to be knowledgeable of the MDOS file structure before allocation conflicts can be wisely resolved. It should be noted that although space is allocated to a file, the space may not necessarily have been written into.

If only an Outer file is displayed with no Inner files at the beginning of Phase 4, then the user has locked out sectors which conflict with files that already have allocated space. REPAIR assumed that the correct sectors were specified by the user during the Phase 1; however, if that is not true, then this kind of a allocation conflict will be seen.

22.6 CAT Replacement Phase

Phase 5 of the REPAIR command compares the reconstructed allocation table in memory with the actual allocation table on the diskette. If the two tables are identical (normal case), REPAIR will display the message

```
RECONSTRUCTED C.A.T. MATCHES DISK
```

before terminating and returning control to MDOS.

If the reconstructed table does not match the one on diskette, and if no RIB errors remain, then the message

```
WRITE RECONSTRUCTED C.A.T. TO DISK?
```

will be displayed. The operator must respond with either a "Y" or an "N". The "Y" response will cause the new allocation table (the correct one) to be written to the diskette. The "N" response will leave the erroneous system table intact. MDOS will be given control after either response.

The allocation table that is written to the diskette is a combination of Table 1 which was built during Pass 1 of Phase 4, and the LCAT. If files with invalid RIBs were encountered during the REPAIR process which were not deleted, in all probability the allocation tables will differ. REPAIR will not update the diskette table until the files with invalid RIBs are fixed or deleted (but they must not be deleted with the DEL command -- only by the REPAIR command). In such cases, REPAIR will display the message

```
INVALID RIBS RESULTED IN RECONSTRUCTED C.A.T. NOT MATCHING
                              DISK
```

as a reminder that the allocation table and some RIBs contain errors. MDOS is given control after the message is displayed.

22.7 Messages

The following messages can be displayed by the REPAIR command. Only those messages not already covered in the preceding sections are listed.

Y(YES) OR N(NO):

> The REPAIR command's prompts usually accept only a "Y" or "N" response from the operator. If any other response is given, this message will be displayed, forcing a new response to be entered.

22.8 Examples

The following example illustrates how REPAIR is used on a working diskette in drive zero to verify that the system tables are correct:

```
=REPAIR
DISK ID:   MDOS0300
VERSION:   03
REVISION:  00
DATE:      072578
USER:      SYS DEVELOPMENT DISK
31 GOOD FILES  00 FILES WITH BAD RIBS
RECONSTRUCTED C.A.T. MATCHES DISK
=
```

The next example illustrates how REPAIR is used once just to gather information about what is wrong with the diskette. Then, DUMP is used to fix the directory, and REPAIR run a second time to verify that the error was corrected. The file LOG.CM is presumably a user-written program that functions as a command; however, the attribute area of the directory entry was created illegally or has been destroyed.

```
=REPAIR :2
DISK ID:   MDOS0300
VERSION:   03
REVISION:  00
DATE:      072578
USER:      SYS DEVELOPMENT DISK
0A  07 03  LOG      .CM  0570  FFFF 0000
ILLEGAL ATTRIBUTE OR UNUSED BYTES.  DELETE? N
NON-0 BYTES AFTER SDW TERMINATOR
0A  07 03  LOG      .CM  0570  FFFF 0000
RIB IN ERROR - DELETE FILE? N
2F GOOD FILES  01 FILES WITH BAD RIBS
INVALID RIBS RESULTED IN RECONSTRUCTED C.A.T.  NOT
  MATCHING DISK
=
```

The DUMP command (Chapter 11) can then be used to change the directory entry. Since LOG.CM is a memory-image file, the RIB contains load information after the terminator; however, the attribute part of the directory entry was destroyed. Thus, REPAIR could not detect the memory-image format.

From the information shown for the directory entry, it is determined that the directory entry for the file LOG.CM is in physical sector $0A or directory logical sector 7. The following sequence is used to "repair" the attribute field:

```
=DUMP :2
PHYSICAL MODE
:RD 7
:S
  CHANGE BUFFER

    PSN=000A
00  42 41 53 49 43 20 20 20  43 4D 01 00 72 00 00 00  BASIC   CM..r..
10  46 52 45 45 20 20 20 20  43 4D 02 B4 72 00 00 00  FREE    CM..r..
20  45 51 55 20 20 20 20 20  53 41 04 B0 65 00 00 00  EQU     SA..e..
30  4C 4F 47 20 20 20 20 20  43 4D 05 70 FF FF 00 00  LOG     CM.p...
40  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ...............
50  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ...............
60  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ...............
70  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ...............
:3C/
3C FF 52,00/
:W
:Q
=REPAIR :2
DISK ID:   MDOS0300
VERSION:   03
REVISION:  00
DATE:      072578
USER:      SYS DEVELOPMENT DISK
30 GOOD FILES  00 FILES WITH BAD RIBS
RECONSTRUCTED C.A.T. MATCHES DISK
=
```

The REPAIR command was then invoked a second time to ensure that the "fix" was correctly applied. Since REPAIR then recognized the file LOG.CM as a memory-image file, the RIB error disappeared automatically.

The same error could have been corrected without having the detailed systems knowledge that was used in the above example. If the file were deleted, the error would be fixed and the diskette would be a valid MDOS diskette. The following example shows the minimal-knowledge approach to fixing the diskette:

```
=REPAIR :2
DISK ID:   MDOS0300
VERSION:   03
REVISION:  00
DATE:      072578
USER:      SYS DEVELOPMENT DISK
0A  07 03  LOG      .CM  0570  FFFF 0000
ILLEGAL ATTRIBUTE OR UNUSED BYTES.  DELETE? Y
2F GOOD FILES  00 FILES WITH BAD RIBS
WRITE RECONSTRUCTED C.A.T. TO DISK? Y
=
```

Since the file was deleted, the reconstructed allocation table did not match the one on the diskette. Thus, a new one was written to make the allocation table correct.

The last example illustrates how a file just having been deleted by accident can be recreated if no other process is invoked that causes a directory entry to be created or space to be allocated or deallocated. Since the deletion only removes the name from the directory and frees the allocated space, all that needs to be done is to rebuild the directory entry using DUMP, and to recreate the allocation table using REPAIR. The following example shows the sequence of events from the file's deletion through its directory entry reconstruction. This example assumes that the operator knows the file's position in the directory (from DEN of a directory listing). Otherwise, the DUMP command "SD" would have to be used to display the entire directory, allowing the operator to search for the deleted entry visually.

```
=DEL TESTPROG.SA
TESTPROG.SA:0 DELETED
=DUMP
PHYSICAL MODE
:RD 3
:S
  CHANGE BUFFER

    PSN=0006
00  4D 44 4F 53 4F 56 34 20  53 59 00 88 72 00 00 00  MDOSOV4 SY..r..
10  FF FF 53 54 50 52 4F 47  53 41 05 FC 05 00 00 00  ..STPROGSA.....
20  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ...............
30  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ...............
40  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ...............
50  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ...............
60  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ...............
70  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ...............
:10/"TE"/
:S
  CHANGE BUFFER

    PSN=0006
00  4D 44 4F 53 4F 56 34 20  53 59 00 88 72 00 00 00  MDOSOV4 SY..r..
10  54 45 53 54 50 52 4F 47  53 41 05 FC 05 00 00 00  TESTPROGSA.....
20  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ...............
30  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ...............
40  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ...............
50  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ...............
60  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ...............
70  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ...............
:W
:Q
=REPAIR
DISK ID:   MDOS0300
VERSION:   03
```

```
REVISION: 00
DATE:     072578
USER:     SYS DEVELOPMENT DISK
33 GOOD FILES  00 FILES WITH BAD RIBS
WRITE RECONSTRUCTED C.A.T. TO DISK? Y
=DIR TESTPROG.SA;A
DRIVE : 0   DISK I.D. : MDOS0300
TESTPROG.SA   .....5 05FC 0004 31   00 05FC 004
TOTAL NUMBER OF SECTORS : 0004/$004
TOTAL DIRECTORY ENTRIES SHOWN : 001/$01
=
```

The above procedure should only be used as a last resort. It can be avoided completely if an adequate backup copy is kept of all files and if the protection attributes are set for those files which are not to be deleted.

CHAPTER 23

23. ROLLOUT COMMAND

The ROLLOUT command is used for writing the contents of memory to diskette. The ROLLOUT command supports the single and dual memory maps of EXORciser II as well as the single memory map of EXORciser I. Options exist for writing memory directly into a diskette file or for writing to a scratch diskette.

23.1 Use

The ROLLOUT command is invoked with the following command line:

```
ROLLOUT [<name>] [;<options>]
```

where <name> is the name of a diskette file and <options> is one of the options described below. The file name, if used, is given the default suffix "LO" and the default logical unit number zero. In some cases, it is invalid to have the file name specified with logical unit number one (see section 23.1.4). If a file name is specified on the command line, it must be the name of a file which does not already exist in the directory. Whenever the file is created, it will be in the memory-image format and allocated contiguously on the diskette.

There are four different ways in which the ROLLOUT command can be used. Each of the four uses of ROLLOUT is specified via the <options> field.

Option	Function
U	Write memory into a file from the User Memory Map of an EXORciser II system that has the dual memory map configured.
none	Write memory into a file. Only memory not overlayed by MDOS or ROLLOUT command can be accessed.
V	Write memory to scratch diskette (not to a file). Any memory block can be written out.
D	Copy the scratch diskette's data ("V" option) into a diskette file.

The ROLLOUT command cannot be invoked from within a CHAIN file (Chapter 6). Since most of the processing is done by a position-independent routine that must work without MDOS being resident, the resident MDOS I/O functions cannot be used. Therefore, the special keyboard keys CTL-X, CTL-D, CTL-W, BREAK, and RUBOUT are non-functional during the ROLLOUT command; however, each operator response must still be terminated with a carriage return.

Caution must be used when writing out blocks of memory that include the highest addressed memory location $FFFF. Since MDOS can only load programs in a multiple of eight bytes, the starting load address of such programs must be an address that is a multiple of eight. Otherwise, the ending load address will be greater than $FFFF.

23.1.1 User Memory Map

When the ROLLOUT command is invoked with the command line

```
ROLLOUT <name>;U
```

the memory from the User Memory Map of an EXORciser II system with the dual memory map configured will be written into the diskette file <name> on the specified logical unit. If the dual memory map is not configured, ROLLOUT will terminate after displaying the following message:

```
USER MEMORY MAP NOT CONFIGURED
```

If the dual memory map is configured, then ROLLOUT will continue and display the messages

```
START ADDRESS:
END ADDRESS:
```

The user responds by entering the starting and ending memory addresses in the User Memory Map which are to be written into the diskette file. The addresses must be input in hexadecimal ($0000-FFFF), and the starting address must be less than or equal to the ending address. If these two conditions are not met, the message

```
INVALID ADDRESS RANGE
```

will be displayed and the operator will be given another chance to enter the addresses. After having supplied the memory range to be written to diskette, the message

```
ARE YOU SURE (Y, N, Q)?
```

will be displayed. The operator must respond with a "Y" to have the memory written into the diskette file. The memory block is only written into the file if sufficient contiguous space can be allocated. ROLLOUT will then terminate and return control to MDOS.

The "N" response will cause the memory start and end address messages to be redisplayed in order to allow another set of addresses to be entered. The "Q" response will terminate the ROLLOUT command and return control to MDOS.

23.1.2 Non-overlayed memory

If the ROLLOUT command is invoked with the command line

```
ROLLOUT <name>
```

then any block of memory not overlayed by MDOS or the ROLLOUT command in either EXORciser I or II (single or Executive memory map) can be written to the diskette file specified by <name>. The file can be specified to reside on any logical unit number.

As described in section 23.1.1, the start/end address message prompts will be displayed; however, in addition to the criteria set forth in that section for valid addresses, the address range must not have been overlayed by MDOS or the ROLLOUT command. If an address range is specified that falls into the overlayed memory, the message

```
START ADDRESS MUST BE GREATER THAN $nnnn
```

will be displayed. The "nnnn" is the last address that has been used by MDOS or the ROLLOUT command. The operator is then given a chance to re-enter the addresses. Otherwise, the function of the ROLLOUT command is similar to the function described in the previous section.

23.1.3 Overlayed memory

If the ROLLOUT command has been invoked with the command line

```
ROLLOUT ;V
```

then any block of memory can be subsquently written to a scratch diskette. A position-independent routine will be moved into memory. This routine can subsequently be activated by the user from the debug monitor after loading his test program into memory. The routine will be used to write memory to a scratch diskette that has been placed into drive one.

No file name specification can be entered with the "V" option. The diskette that will be written to in drive one must not contain an MDOS system that is to be used again. The system tables on that diskette will be overwritten. The diskette will have to be regenerated in order to be used as an MDOS system diskette.

ROLLOUT will display the following message once it has been invoked with the "V" option:

```
LOAD ADDRESS:
```

to which the operator must respond with the starting hexadecimal address of a memory block into which the ROLLOUT command will attempt to move the position-independent routine. The address must be for memory above that required by MDOS and the ROLLOUT command. If the address entered is too low, ROLLOUT will display the message

```
LOAD ADDRESS MUST BE GREATER THAN $nnnn
```

and return control to MDOS. "nnnn" is the hexadecimal address of the last location in memory occupied by MDOS or the ROLLOUT command. If the entered address specified spans non-existent memory, ROLLOUT will display the standard error message

```
** 53 INSUFFICIENT MEMORY
```

and return to MDOS.

Caution must be used in locating the position-independent routine in memory. Since MDOS uses the upper end of memory when the command interpreter is running, the routine should not be loaded within 100 (decimal) bytes of the end of contiguous memory. Care must also be taken to ensure that the program being tested does not destroy the $200 locations occupied by the position-independent routine.

If the position-independent routine was successfully transferred, ROLLOUT will terminate and return control to MDOS. The user can then invoke the LOAD command to bring his test program into memory. Then, whenever the time is reached that memory is to be written to diskette, the user need only give control to the still resident position-independent routine at the address that was entered in response to the "LOAD ADDRESS" prompt discussed above. This is done via the EXbug command

```
nnnn;G
```

When the position-independent routine receives control in this manner, it will prompt the operator for the starting and ending addresses as described in section 23.1.1. After the address range has been entered and the "Y" response given to the "ARE YOU SURE?" question, the message

```
DRIVE 1 SCRATCH?
```

will be displayed. At this point, a scratch diskette must be placed into drive one. A "Y" response will then cause the block of memory to be written to the scratch diskette. Any other response will give control to the debug monitor.

The "N" response to the "ARE YOU SURE?" prompt will allow the address range to be reentered. The "Q" response, however, will return control to the debug monitor, rather than to MDOS. After the block of memory has been rolled out, the debug monitor will receive control again.

The ROLLOUT command can be subsequently used (see section 23.1.4) to copy the raw data from the scratch diskette into a file on drive zero.

23.1.4 Scratch diskette conversion

If the ROLLOUT command is invoked with the command line

```
ROLLOUT <name>;D
```

then the memory written to the scratch diskette with the "V" option will be copied into the file <name>. ROLLOUT will assume that a scratch diskette is in drive one that has been created via the ROLLOUT command with the "V" option. The <name> specified must be for logical unit zero. Since the diskette in drive one is scratch, no file can be created there.

The ROLLOUT command will display the following message once it has been invoked with the "D" option:

```
DOES DRIVE 1 CONTAIN A MEMORY ROLLOUT?
```

to which the operator must respond with a "Y" if the ROLLOUT command is to continue. Any other response will terminate the ROLLOUT command and return control to MDOS.

If the "Y" response is given to the above message, ROLLOUT will check that the diskette in drive one was generated with the "V" option. If an invalid diskette has been placed into drive one, the message

```
INVALID DISKETTE IN DRIVE 1
```

will be displayed and ROLLOUT will be terminated. If a valid diskette is found, then ROLLOUT will proceed to build a file on drive zero that contains the memory information from the scratch diskette.

23.2 Messages

The following messages can be displayed by the ROLLOUT command. Not all messages are error messages, although error messages are included in this list. The standard error messages that can be displayed by all commands are not listed here.

START ADDRESS:

The starting address of the block of memory to be written out must be entered.

END ADDRESS:

The ending address of the block of memory to be written out must be entered.

INVALID ADDRESS RANGE

The starting address was greater than the ending address, or one of the two addresses contained an invalid hexadecimal number.

ARE YOU SURE (Y, N, Q)?

This message allows the operator to verify that the starting/ending addresses entered are what he wants. The "Y" response will cause ROLLOUT to continue. The "N" response will allow a new address range to be entered. The "Q" response will terminate the ROLLOUT command.

DRIVE 1 SCRATCH?

This message is displayed by the position-independent routine to allow the operator a chance to insert a scratch diskette into drive one. A "Y" response will cause the memory to be written to the diskette. Any other response will return control to the debug monitor.

START ADDRESS MUST BE GREATER THAN $nnnn

The start/end addresses include memory occupied by MDOS and/or the ROLLOUT command. If this memory is to be written out, the ROLLOUT command should be invoked with the "V" option. Otherwise, the start/end addresses must be greater that "nnnn".

LOAD ADDRESS MUST BE GREATER THAN $nnnn

The address specified for locating the position-independent routine in memory includes memory occupied by MDOS and/or the ROLLOUT command. The address must be greater than $nnnn shown in the message.

USER MEMORY MAP NOT CONFIGURED

The "U" option has been specified on an EXORciser I system or on an EXORciser II system that has no dual memory map configured.

LOAD ADDRESS:

The operator must specify an address at which the position-independent routine will be located for subsequent access via the debug monitor. The load address entered will be the starting execution address that is used to activate the ROLLOUT routine from the debug monitor.

DOES DRIVE 1 CONTAIN A MEMORY ROLLOUT?

This message allows the operator time to insert the scratch diskette created via a previous ROLLOUT process with the "V" option into drive one before ROLLOUT will convert the data into a diskette file on drive zero. A "Y" response will cause ROLLOUT to continue. Any other response will cause control to be returned to MDOS.

INVALID DISKETTE IN DRIVE 1

This message indicates that the diskette in drive one was not created by the ROLLOUT command with the "V" option.

** 53 INSUFFICIENT MEMORY

The operator specified an address which started a block of memory that does not exist or that contains bad memory. This block is used to receive a copy of the position-independent routine that is given control from the debug monitor. $200 bytes of memory must be available starting at the address entered by the operator. The cautions listed in section 23.1.3 should also be reviewed.

23.3 Examples

The following example shows the operator-system dialogue for writing a block of memory to a file from the User Memory Map of an EXORciser II system with the dual memory map configured:

```
=ROLLOUT UMBLOCK;U
START ADDRESS: 100
END ADDRESS: 7FF
ARE YOU SURE (Y, N, Q)? Y
=
```

The file named UMBLOCK.LO will be created on drive zero. It will contain the block of memory from $100 to $7FF, inclusive, from the User Memory Map.

The following example illustrates how a copy of the diskette controller ROM can be written into a diskette file:

```
=ROLLOUT DISKROM:2
START ADDRESS: E800
END ADDRESS: EBFF
ARE YOU SURE (Y, N, Q)? Y
=
```

The file named DISKROM.LO will be created on drive two. This example is valid for either type of EXORciser system.

The following example shows how the ROLLOUT command is used to write memory to disk during a test session of a user program that overlays MDOS. A maximum contiguous memory range of 32K is assumed.

```
=ROLLOUT ;V
LOAD ADDRESS: 7F80
** 53 INSUFFICIENT MEMORY
=ROLLOUT ;V
LOAD ADDRESS: 7D00
=LOAD TESTPROG;V
* (User does testing here via EXbug)
*7D00;G
START ADDRESS: 100
END ADDRESS: 5FFF
ARE YOU SURE (Y, N, Q)? N
START ADDRESS: 100
END ADDRESS: 2FFF
ARE YOU SURE (Y, N, Q)? Y
DRIVE 1 SCRATCH? Y
*
```

In the above example, the operator initially specified a block of memory which was too small to receive the position-independent routine. $200 bytes are required to contain the routine; however, since the end of memory is used by the MDOS command interpreter, an additional block of memory is allowed for the MDOS stack. Thus, the ROLLOUT command had to be invoked again. Then, after loading and testing his program, the operator invoked the routine via the "7D00;G" EXbug command. After entering the end address, the user realized an error, and responded "N" to the "ARE YOU SURE?" question. Testing can be continued after the block of memory has been written to the diskette.

The last example illustrates how the scratch diskette generated above is converted into a file:

```
=ROLLOUT TESTROLL;D
DOES DRIVE 1 CONTAIN A MEMORY ROLLOUT? Y
=
```

The file named TESTROLL.LO will be created on drive zero.

CHAPTER 24

24. SYSTEM DESCRIPTION

This chapter contains the detailed descriptions of the structure of an MDOS diskette, the structure of MDOS files and their formats, the system overlays, the memory map, the command interpreter, interrupt handlers, the system function handler, and the MDOS equate file. The subsequent three chapters contain the detailed descriptions of the individual system functions and how they are parameterized.

24.1 Diskette Structure

MDOS is based on a single- or double-sided, single-density flexible disk, or diskette. The diskette is compact in size, portable, fairly durable, and easily inserted into and removed from the diskette drives. Due to the diskette's portability and interchangeability, each diskette is treated by MDOS as a complete, self-contained entity. Each diskette has its own system tables, operating system, and files.

Information on an MDOS diskette is stored in sectors 128 (decimal) bytes in size. As the diskette turns, the read/write head in a stationary position will pass over 26 (decimal) sectors each revolution. The area accessible to the stationary head on one side of the diskette is called a track. The area accessible to the stationary head on both sides of the diskette is called a cylinder. The head can be positioned over 77 (decimal) discrete cylinders. Thus, there are a total of 2002 (decimal) sectors on each surface of a diskette. A single-sided diskette only has one surface that can be read from and written to. A double-sided diskette has two surfaces.

In order to minimize access time and yet provide for a dynamic allocation scheme, all diskette space allocation is done in terms of clusters, rather than sectors. MDOS clusters consist of four, physically sequential sectors. A cluster is the smallest structural unit of information on the diskette. Thus, the smallest possible size that a file can have is one cluster.

The following table summarizes these diskette statistics.

	Single-sided		Double-sided	
Quantity	Decimal	Hex	Decimal	Hex
Surfaces/diskette	1	1	2	2
Bytes/sector	128	80	128	80
Sectors/track	26	1A	26	1A
Tracks/cylinder	1	1	2	2
Sectors/cylinder	26	1A	52	34
Cylinders/diskette	77	4D	77	4D
Sectors/surface	2002	7D2	2002	7D2
Sectors/diskette	2002	7D2	4004	FA4
Sectors/cluster	4	4	4	4
Clusters/diskette	500	1F4	1001	3E9

MDOS accesses sectors on the diskette via a physical sector number (PSN). The diskette controller decodes the PSN into the appropriate cylinder/sector position. To avoid confusion, all sector numbers given in this section will refer to physical sector numbers. If a need should arise to convert between cylinder/sector and physical sector numbers, Appendix A has been provided. It contains the physical sector numbers of the first sector of each cylinder on each surface.

A portion of each diskette is reserved for some special system tables. These tables reside in the outermost cylinder of the diskette, cylinder zero. Each table, with the exception of the directory, occupies a single sector. The following table summarizes the location of the system tables:

System table	PSN
Diskette Identification Block	$00
Cluster Allocation Table	$01
Lockout Cluster Allocation Table	$02
Directory	$03-16
Bootblock, MDOS RIB	$17,18

24.1.1 Diskette Identification Block

The Diskette Identification Block is created during system generation. It contains an ID, the version and revision number of the resident operating system, the date the diskette was generated, a user name identification area, and a dynamic area for the MDOS overlay RIB addresses. The ID is displayed by the DIR, FREE, and REPAIR commands. The

Diskette Identification Block has the following format:

Bytes	Size	Contents
0-7	8	Diskette ID
8-9	2	Version number
$A-B	2	Revision number
$C-11	6	Generation date
$12-25	$14	User name
$26-39	$14	MDOS overlay RIB addresses
$3A-$7F	$46	Zeroes

24.1.2 Cluster Allocation Table

The Cluster Allocation Table (CAT) contains a bit map of the areas on the diskette that are available for new space allocation. Each bit in the CAT represents a physical cluster of diskette storage. The first bit of the first byte of the CAT (bit 7 of byte 0) represents cluster 0. The subsequent bits represent subsequent clusters. A bit set to one indicates that the cluster is allocated. If a bit is set to zero, it indicates that the corresponding cluster is available for allocation. Since not all 128 bytes of the CAT correspond to physical clusters, the parts of the CAT that represent clusters beyond the physical end of the diskette are marked as allocated so that they cannot be used by any MDOS functions.

On single-sided diskettes, bytes 0-$3E of the CAT correspond to the physical locations on the diskette; however, in byte $3E, bits 0-3 are set to one since no physical sectors correspond to those cluster numbers. Bytes $3F-7F are set to all ones. The cluster division for allocation only includes 2000 (decimal) sectors. Since there are 2002 sectors, the last two physical sectors of a single-sided diskette are not available for allocation ($7D0-7D1).

On double-sided diskettes, bytes 0-$7D correspond to the physical locations on the diskette; however, in byte $7D, bits 0-6 are set to one since no physical sectors correspond to those cluster numbers. Bytes $7E and $7F are set to all ones. The cluster division for allocation includes all physical sectors (4004, decimal). There are no unused sectors on a double-sided diskette.

24.1.3 Lockout Cluster Allocation Table

The Lockout Cluster Allocation Table, or LCAT, is similar to the CAT in structure; however, it is only used during the DOSGEN and REPAIR processes. The LCAT provides a map of which areas of the diskette have been flagged as bad during the DOSGEN write/read test. In addition, the LCAT is configured so that those sectors of the diskette occupied by the system tables in cylinder zero and any user locked out areas (see Chapter 10, DOSGEN command) are flagged as unavailable for normal allocation.

24.1.4 Directory

The directory occupies twenty sectors. Each directory sector contains eight entries of sixteen bytes each. Each entry contains a file name, a suffix, the address of the file's first cluster, the file's attributes, and some room for expansion.

A file is one or more clusters containing related information. This information may be ASCII source programs, binary object records, user-generated data, etc. Each file must reside wholly on a single diskette. Files are identified to the system by their names, suffixes, and logical unit numbers.

The name as stored in the directory consists of ten bytes; however the MDOS command interpreter deals with an eight-byte name. and a two-byte suffix. This is merely a convention of the command interpreter and has no significance in relation to the internal format of the directory. System routines and functions dealing with file names as a parameter use a ten-byte block which is always dealt with as a monolithic item.

File names assigned by the user must be from one to eight alphanumeric characters in length. The first character must be alphabetic. A file's suffix is used to further identify the file. The suffix is primarily used to identify the format of the file content; however, this is purely a convention; the attribute field of the directory entry describes the file's physical format. Suffixes are considered as an extension of the file name. They can be one or two alphanumeric characters in length. The first character of the suffix must be alphabetic. Both the file name and the suffix, if shorter than their maximum allowable lengths, are left justified and space-filled in the directory entry.

In most cases, the MDOS commands make certain default assumptions about a file's suffix if it is not explicitly specified by the operator; however, explicit suffixes can be used whenever the default is to be overridden. The standard MDOS default suffixes are:

Suffix	Implied meaning
AL	Assembly listing file
CF	Chain procedural file
CM	Command file file
ED	EDOS-converted file
LO	Loadable, memory-image file
LX	EXbug-loadable file
RO	Relocatable object file
SA	ASCII source file
SY	Internally-used system file

Logical unit numbers identify the drive that contains the file. Since each diskette carries with it its own directory, different files with identical names and suffixes can reside on different diskettes.

The standard format for specifying file names, suffixes and logical unit numbers is:

```
<file name>.<suffix>:<logical unit number>
```

where the period (.) and colon (:) serve to delimit the start of the suffix and the logical unit number fields, respectively.

In addition to a name, each directory entry contains a set of attributes which characterize the file's content. A file's attributes include inherent attributes and assignable attributes. The inherent attributes of a file describe its allocation scheme (contiguous or segmented), the file format (ASCII record, binary record, memory-image, or user-defined), and whether space compression is used for ASCII records. The file formats are described in section 24.3.

The assignable attributes include write protection, delete protection, and the system file attribute. If a file is write protected, it cannot be written into or deleted. If a file is delete protected, it cannot be deleted. If a file has the system attribute, it will be included in the system generation process (DOSGEN) and is handled differently by the DEL and DIR commands.

The format of a directory entry is described in the following table:

Bytes	Size	Contents
$0-7	8	File name
$8-9	2	Suffix
$A-B	2	PSN of first cluster
$C-D	2	Attributes
$E-F	2	Zeroes

The attribute field of a directory entry has the following format:

```
   F   E   D   C   B   A   9   8   7   6   5   4   3   2   1   0
-----------------------------------------------------------------
 !   !   !   !   !   !           !                              !
-----------------------------------------------------------------
   :   :   :   :   :       :     <------- Not Used (=0) -------->
   :   :   :   :   :       :
   :   :   :   :   :       :... File format (0=user-defined,
   :   :   :   :   :                         2=memory-image,
   :   :   :   :   :                         3=binary record,
   :   :   :   :   :                         5=ASCII record,
   :   :   :   :   :                         7=ASCII-converted-
   :   :   :   :   :                           binary record)
   :   :   :   :   :
   :   :   :   :   :............ Non-compressed space bit
   :   :   :   :................ Contiguous allocation bit
   :   :   :.................... System file bit
   :   :........................ Delete protection bit
   :............................ Write protection bit
```

Associated with each directory entry is an eight-bit number, the directory entry number (DEN), which is a function of the physical location of the entry within the directory. The DEN is not found anywhere in the directory. It is a calculated quantity and is interpreted as follows:

```
   7   6   5   4   3   2   1   0
  -------------------------------
  !                  !          !
  -------------------------------
           :               :
           :               :.... Position within sector
           :                     (0-7)
           :
           :.................... Physical sector number
                                ($3-$16)
```

24.1.5 Bootblock

The Bootblock is a small loader program that is brought into memory along with the next physical sector by the diskette controller during system initialization. The second sector that is loaded contains information regarding the size of the resident operating system. From this information, the Bootblock program configures the diskette controller to load into memory the actual resident operating system.

24.2 File Structure

While the contents of a file can be thought of as a logically contiguous block of information, the actual diskette area allocated to the file may or may not be physically contiguous. Space can be allocated to one or more groups of physically contiguous clusters on the diskette. Each contiguous group of clusters is called a segment. This segmentation allows the dynamic allocation and deallocation of space to occur without having to move any of the information contained in the file or in other files.

Each file must, therefore, have a table that describes which segments are allocated to the file. This table is kept in the first physical sector of each file and is called the Retrieval Information Block (RIB). It is the address of the RIB that is contained in the directory entry of a file.

MDOS accesses sectors within a file by logical sector number (LSN). Since the first physical sector of a file is not really a data sector, the RIB is given an LSN of minus one ($FFFF). Therefore, logical sector zero of a file (the first data sector) is actually the second physical sector of the file. Logical sector numbers for data sectors are numbered sequentially beginning with zero. Thus, even though a file may be segmented (not physically contiguous on the diskette), it is treated as a logically contiguous collection of sectors when accessed by logical sector number. The system I/O functions decode the LSN into the actual PSN.

24.2.1 Retrieval Information Block

For all files, the RIB contains a series of two-byte entries called segment descriptor words (SDWs). A special SDW is used as a terminator to indicate the end of the segment descriptors within the RIB. Each SDW (other than the terminator) contains two pieces of information: the cluster number of the first cluster in the segment, and the length of the segment. Since each segment consists of physically contiguous clusters, this information is all that is needed to describe where a segment of the file is located on the diskette. A RIB can contain a maximum of 57 (decimal) SDWs and one terminator.

The RIB of a memory-image file contains some additional information that describes where the contents of the file are to be loaded in memory. This information includes the starting load address, the number of sectors to load, the number of bytes in the last sector, and the starting execution address.

The memory-image file load information is described in the following paragraphs. Both the content and the location of each field are described. The offsets used to refer to the various bytes are relative to zero (zero being the first byte of the RIB sector). All offsets are given in decimal.

1. Byte 117, the number of bytes to load from the last sector, must be non-zero, a multiple of 8, and less than or equal to 128 ($80).

2. Bytes 118-119, the number of sectors to load, must contain a number that is non-zero, less than the total number of sectors allocated to the file, and less than or equal to 512 ($200).

3. Bytes 120-121, the starting load address, are not checked. For programs loading in an EXORciser I system, in the User Memory Map of an EXORcier II system with the single memory map configured, or in the Executive Memory Map of an EXORciser II system with the dual memory map configured, this value must be greater than hexadecimal location $1F if the program is to be loaded via the MDOS loader. EXORciser II systems can have programs loaded into the User Memory Map of the dual memory map configuration starting at location zero.

4. The ending load address is calculated from bytes 117-121 in the following manner:

   $$EL = (NSL - 1) * 128 + NBLS + SL - 1$$

   where EL is the ending load address, NSL is the number of sectors to load (bytes 118-119), NBLS is the number of bytes in the last sector (byte 117), and SL is the starting load address (bytes 120-121). The ending load address must be less than 65536.

5. Bytes 122-123, the starting execution address, must lie within the range of addresses spanned by the file (greater than or equal to the starting load address, and less than or equal to the ending load address).

6. Bytes 124-127 are not used and must be zero.

The following diagrams illustrate the format of a segment descriptor word and the terminator.

SEGMENT DESCRIPTOR WORD

```
F   E   D   C   B   A   9   8   7   6   5   4   3   2   1   0
---------------------------------------------------------------
!  !                 !                                        !
---------------------------------------------------------------
 :           :         <------- Starting cluster number ------>
 :           :
 :           :.........  Number of contiguous clusters - 1
 :
 :.....................  Zero (Non-terminator bit)
```

TERMINATOR

```
F   E   D   C   B   A   9   8   7   6   5   4   3   2   1   0
---------------------------------------------------------------
!  !                                                          !
---------------------------------------------------------------
 : <------- Logical sector number of logical end-of-file ----->
 :
 :....... One (Terminator bit)
```

The SDW terminator is used to monitor the logical end-of-file. It contains the logical sector number of the end-of-file. The sector which is the end of a file may be partially filled with null characters. Thus, no actual end-of-file record will be found within a file. This feature allows files to be merged together without having to read through the entire file looking for an end-of-file record.

The actual format of a RIB is shown in the following diagram. For non-memory-image files, the bytes following the terminator must all be zero. Only memory-image files can have non-zero bytes following the terminator, and then those bytes must meet the six criteria listed above.

```
       F  E  D  C  B  A  9  8  7  6  5  4  3  2  1  0
      -------------------------------------------------
 00   !                   SDW 0                       !
      -------------------------------------------------
 02   !                   SDW 1                       !
      -------------------------------------------------
 04   !                                               !
      ~                                               ~
      ~                 Other SDWs                    ~
      !                                               !
      -------------------------------------------------
      !                 TERMINATOR                    !
      -------------------------------------------------
      !                                               !
      ~                                               ~
      ~                   Zeroes                      ~
      !                                               !
      !                       -------------------------
 74   !                       ! BYTES IN LAST SECTOR  !
      -------------------------------------------------
 76   !          NUMBER OF SECTORS TO LOAD            !
      -------------------------------------------------
 78   !            STARTING LOAD ADDRESS              !
      -------------------------------------------------
 7A   !          STARTING EXECUTION ADDRESS           !
      -------------------------------------------------
 7C   !                   ZERO                        !
      -------------------------------------------------
 7E   !                   ZERO                        !
      -------------------------------------------------
```

24.2.2 File formats

MDOS deals with four types of file formats on diskette: user-defined, memory-image, binary record, and ASCII record.

User-defined files are dealt with by MDOS at the sector level. MDOS will keep track of where the file is and will only allow access to the file by logical sector number. The user has the responsibility of formatting the data within the sectors in the manner suited to his application.

Memory-image files include all files whose contents are to be loaded into memory directly from the diskette by the MDOS loader. Memory-image files are allocated contiguous space on the diskette. The only information retained about where the content is to be loaded is kept in the file's RIB. The data within the sectors of the file contain no load or record information. It is merely an image of a block of memory to be loaded into. Due to the nature of the diskette controller, MDOS programs can only be loaded in multiples of eight bytes. A further restriction placed on memory-image files is that their content cannot load below memory location $20 if they are to reside in the single memory map of an EXORciser I or EXORciser II system.

Binary record files are used primarily for the relocatable object data produced by the Macro Assembler and the relocatable FORTRAN compiler; however, the user can create data files using this binary record format as well.

ASCII record files are used to contain all other MDOS-supported data. Such files can be in either space-compressed or non-space-compressed form. Normally, MDOS will always create ASCII files with the space-compression attribute to conserve diskette space.

The non-memory-image files can be allocated in either contiguous or segmented fashion. Normally, MDOS will create such files in a segmented manner to take advantage of the dynamic allocation scheme. If files are segmented, they can expand to the full capacity of the diskette when they need to grow in size; however, if files have contiguously allocated space, then they can only be expanded if they are allocated space that is contiguous to the originally allocated space. Normally, contiguous files are created with the maximum space that they will ever need.

24.3 Record Structure

This section describes in detail the two record types supported for diskette files. In addition, a special record type used for copying binary files to a non-diskette device is also discussed. The actual use of such records is fully discussed in Chapter 25 which describes the supported I/O functions. All records supported by MDOS are terminated by a carriage return, line feed, and null sequence; however, on the diskette, only the carriage return character is retained in order to conserve diskette space. When diskette files are copied to a non-diskette device, the other two characters are automatically supplied by MDOS.

24.3.1 Binary records

Binary records are used primarily as output from the Macro Assembler and the FORTRAN compiler, and for input to the Linking Loader. Binary records contain a special record header, a byte count, and a checksum. The checksum is a two's-complemented sum of all bytes in the record from the byte count through the last data byte, inclusive. A maximum of 254 (decimal) data bytes can be contained in each binary record.

The format of a binary record can be illustrated as follows:

```
-------------------------------/ /------------
! D ! BC !          DATA            ! CK ! CR !
-------------------------------/ /------------
```

The symbols take on the following meanings:

Symbol	Meaning
D	The binary record header character "D" ($44).
BC	A one byte "byte count" that contains the number of data bytes in the record plus one (for the checksum byte).
DATA	A maximum of 254 (decimal) data bytes. Any eight-bit values are valid for the data bytes.
CK	The two's-complemented sum of the byte count and all data bytes. CK is a one byte field.
CR	The terminating carriage return. For non-diskette devices this will actually be a carriage return, line feed, and null sequence.

Since diskette files contain the logical end-of-file indicator in the RIB, the binary EOF record only will be seen on non-diskette devices. The binary EOF record has the following format:

```
--------------------
! E ! BC ! CK ! CR !
--------------------
```

The symbol "E" is the end-of-file record header which is the letter "E" ($45). The other symbols are the same as in the above table. The EOF record has no data bytes. Thus, the byte count will be equal to one.

24.3.2 ASCII records

ASCII records are used primarily for source files on the diskette; however, EXbug-loadable format files are ASCII even though they are object files output from the assemblers or Linking Loader.

ASCII records contain no record headers, byte counts, or checksum fields. The first ASCII record in a file begins with the first data character of a file and is terminated by the first carriage return. All other ASCII records in the file begin with the first data character following a carriage return. When ASCII records are copied to non-diskette devices, the terminating carriage return is actually a combination of three control characters: carriage return, line feed, and null. ASCII records should contain only displayable characters.

When MDOS writes ASCII records to diskette, they normally contain space compression characters to conserve diskette space. A space compression character is indicated by a data byte having the sign bit (bit 7) set to a one. The remaining bits (0-6) contain a binary number representing the number of spaces ($20) to be inserted in place of the compressed character. MDOS automatically expands these characters into spaces when such files are read. MDOS will also automatically create these compressed characters when such files are written.

Since MDOS maintains the logical end-of-file indicator in a file's RIB, no ASCII EOF record will be seen in a diskette file; however, when ASCII record files are written to a non-diskette device, the following EOF record will be supplied:

```
 -----------
 | 1A | CR |
 -----------
```

where the "1A" symbol represents the end-of-file indicator. It is the hexadecimal value $1A or SUB control character (CTL-Z). The CR symbol is the carriage return, line feed, and null sequence.

If ASCII record files generated on another system are to be processed by MDOS, it is important that the carriage return, line feed, and null sequence be present at the end of each record. Otherwise, it is possible for each data record to lose one or two characters from its beginning.

24.3.3 ASCII-converted-binary records

A special form of the binary record exists when copying to a non-diskette device that can only accept seven-bit data. This record format is usually never kept in a diskette file. The format of the ASCII-converted-binary record is identical to the binary record; however, each byte, with the exception of the special header character and the terminating carriage return, line feed, and null sequence, is converted into two eight-bit bytes with bit seven set to zero. This is accomplished by taking each half of the original byte and adding the bit mask %00110000 to the half-byte. The result is a displayable two-byte sequence. For example, the hexadecimal data byte $85 would be converted into the two byte sequence $38 and $35.

24.3.4 File descriptor records

MDOS I/O operations with non-diskette devices can be in one of two modes: file format or non-file format. The non-file format mode requires no special processing and uses only the ASCII record format.

The file format mode allows MDOS to treat the data on certain non-diskette devices as a "file", similar to a file on diskette. The File Descriptor Record (FDR) is employed to serve the same function as a directory entry for a diskette file. The FDR contains a file name, suffix, and a file format descriptor. Thus, MDOS can search for a named file on a cassette or paper tape, if it was originally created using the file format mode.

All FDRs are identical in format, regardless of the record format of the data file. Since the FDR must be acceptable to any device, it is written in the ASCII-converted-binary form, even if the remaining data of the file is in binary or ASCII. The FDR format is shown in the following diagram:

```
-----------------------------------------------------------
! H ! BC ! NAME ! SUFX ! NU ! FDF ! NU ! CK ! CR !
-----------------------------------------------------------
```

The symbols take on the following meanings:

Symbol	Meaning
H	The FDR header character "H" ($48).
BC	A one-byte "byte count" that contains the number of bytes in all fields from NAME through CK, inclusive. This number is fixed for FDR records at 17 (decimal). This number reflects the real data bytes in the unconverted binary form, not the bytes written in the ASCII-converted-binary form.
NAME	The eight-character file name.
SUFX	The two-character suffix.
NU	A two-byte field which is not used. It contains zeroes.
FDF	A two-byte field similar in format to the attribute field of a directory entry. Only bits $8-$A are used to describe the file format.
CK	The two's-complemented sum of the byte count and all other data bytes. CK is a one byte field.
CR	The terminating character sequence of carriage return, line feed, and null.

The length of all fields of the FDR (except H and CR) is doubled when written (ASCII-converted-binary format). Thus, if the CR field is counted as three characters (carriage return, line feed, null), then the physical length of an FDR in the ASCII-converted-binary format is 36 (decimal) bytes.

24.4 System Files

On every MDOS diskette there are nine files which comprise the operating system. These files contain the resident operating system, a series of overlays to reduce the main memory requirements of the system, and standard error messages. The resident operating system file MDOS.SY must reside in a fixed place on the diskette if the Bootblock program is to work after being activated by the diskette controller. The other system files must remain in fixed positions after MDOS has been initialized since they are referenced by their physical sector numbers.

24.4.1 System overlays

The system overlay files are loaded into memory into one of the four overlay regions discussed in the subsequent section. The overlay handler only brings an overlay into memory if it is not already in memory at the time a specific function is required. If an overlay remains in memory, access to its function is faster than if it has be to loaded from the diskette. The functions contained in the seven overlay files are shown in the following table:

Overlay	Function
MDOSOV0.SY	Diskette space allocation and deallocation.
MDOSOV1.SY	Processing standard file names, allocating contiguous memory, reserving a device, releasing a device, writing standard records, writing FDRs, writing end-of-file records, console reader/punch device handling.
MDOSOV2.SY	Reading standard records, reading FDRs.
MDOSOV3.SY	Closing a file/device, rewinding diskette files, changing file names and attributes.
MDOSOV4.SY	Opening a file/device.
MDOSOV5.SY	CHAIN file execution.
MDOSOV6.SY	Command line interpretation.

When MDOS is initialized, the directory is searched for the seven overlays by name. The physical diskette addresses are then retained so that a subsequent reference to an overlay function does not involve another directory search. Thus, MDOS must be reinitialized each time the diskette in drive zero is changed so that the overlays can be located again.

Overlays MDOSOV0 and MDOSOV1 use overlay region one. Overlays MDOSOV2 and MDOSOV3 use overlay region two. Overlays MDOSOV4 and MDOSOV5 use overlay region three, and overlay MDOSOV6 uses the User Program Area into which the MDOS commands also are loaded. The overlay regions are shown in the memory map diagram of section 24.5.

24.4.2 System error message file

In an attempt to use English language descriptions for the various error conditions that may arise, all standard error messages are kept in the system file MDOSER.SY. This file is accessed by the error message function .MDERR (section 27.4). The error messages are placed in this file so that the most frequently used messages are near the beginning.

If the error message file cannot be read or accessed, the error message function will display a message indicating that an invalid error message has been requested.

24.5 Memory Map

The memory mapping of MDOS within the EXORciser system is illustrated in the following diagram:

```
           ---------------------------------
0000       | DISKETTE CONTROLLER VARIABLES |
           ---------------------------------
0020       ~   UNUSED DIRECT ADDRESSING    ~
           ~            AREA               ~
           ---------------------------------
00AE       |      COMMAND LINE BUFFER      |
           ---------------------------------
00FE       |  COMMAND LINE BUFFER POINTER  |
           ---------------------------------
0100       |        MDOS VARIABLES,        |
           |   IOCBs and SYSTEM BUFFERS    |
           ---------------------------------
           |          SWI HANDLER          |
           |    KERNEL SYSTEM FUNCTIONS    |
           ---------------------------------
           | CONTROLLER DESCRIPTOR BLOCKS  |
           ---------------------------------
           |   SUPPORTED DEVICE DRIVERS    |
           ---------------------------------
           |   RESIDENT SYSTEM FUNCTIONS   |
           ---------------------------------
           |       OVERLAY HANDLER         |
           ---------------------------------
           |       OVERLAY REGION 1        |
           ---------------------------------
           |       OVERLAY REGION 2        |
           ---------------------------------
           |       OVERLAY REGION 3        |
           ---------------------------------
2000       |       OVERLAY REGION 4        |
           |             and               |
           |       USER PROGRAM AREA       |
           ~                               ~
           ~                               ~
3FFF       | END OF MINIMUM SYSTEM MEMORY  |
           ~                               ~
           ~                               ~
           |   END OF CONTIGUOUS MEMORY    |
           ---------------------------------
                   RAM Discontinuity
           ---------------------------------
           ~         NON-MDOS RAM          ~
           ~                               ~
           ---------------------------------
E800       |   DISKETTE CONTROLLER PROM    |
           ---------------------------------
EC00       |             PIAs              |
           ---------------------------------
F000       |         EXbug MONITOR         |
           ---------------------------------
FFF8       |       INTERRUPT VECTORS       |
           ---------------------------------
```

Locations $0000-001F, inclusive, are reserved for the variables of the diskette controller. These locations cannot be initialized by a program loading from the diskette. In addition, if a program requires the use of the diskette functions (either directly through the diskette controller or through the MDOS functions), then these locations cannot be used by the program for storage. Locations $00AE-00FD, inclusive, contain the MDOS command line as it was entered by the operator. Command-interpreter-loadable programs must load above location $1FFF. They can use the direct addressing area for variable storage; however, this area cannot be initialized while the program is being loaded into memory. Programs that do not make use of MDOS system functions can load anywhere in memory above location $001F. If such programs do not use the diskette controller entry points (Appendix D), the direct addressing area below location $0020 can be used, but only after the program is resident in memory.

The MDOS variables (locations $FE and higher) contain pointers to several areas in memory that might be required by a user program. The absolute addresses of these pointers should be obtained from the MDOS equate file. The pointers most often required are:

Pointer Name	Content
CBUFP$	The address in the command line buffer to the terminator of the command being executed. Parameters following the command name should be scanned for by using the contents of this variable.
ENDOS$	The address of the last location of resident MDOS. The value of this address plus one is the first location that a command-interpreter-loadable program can load into.
ENDUS$	The address of the last location loaded into by the current program. The program can allocate additional memory (between the last loaded location and the end of contiguous memory) via one of the system functions.
ENDSY$	The address of the last byte of contiguous memory (RAM).

SWI$UV	The address of a user-defined SWI handler. This vector must be initialized by a user program if it is using SWIs other than those defined for MDOS system functions. This vector is set to point to an RTI instruction each time the MDOS command interpreter is given control.
IRQ$UV	The address of a user-defined IRQ handler. This vector must be initialized by a user program if it is using IRQs. This vector is set to point to an RTI instruction each time the MDOS command interpreter is given control. This vector is not availabale with MDOS09.

24.6 MDOS Command Interpreter

The MDOS command interpreter is one of the MDOS overlays that gets control whenever MDOS has been initialized or whenever a command has completed and returned control to MDOS. This overlay will cause the standard command line input prompt (=) to be displayed whenever it is activated.

Once in control, the interpreter waits for operator input. After a line has been entered, it is scanned for the first valid file name specification. If no valid file name is recognized, the standard message

WHAT?

will be displayed and a new input prompt shown. If the first encountered file name specification contains a valid file name, it will be used to search the directory. The default suffix "CM" and the default logical unit number zero will be supplied by the MDOS command interpreter if none are explicitly entered by the operator. If the file name is not found in the directory specified by the logical unit number, the "WHAT?" message shown above will be displayed and another input prompt shown. If the file name is found, it must be the name of a file that contains a command-interpreter-loadable program. That is, the file must be in the memory-image format and must have a starting load address that is greater than the value contained in the MDOS variable ENDOS$ (greater that $1FFF). If the file passes these tests, its contents are automatically loaded into memory and given control at the starting execution address contained in the file's RIB.

The loaded program can then extract parameters from the MDOS command line buffer. The pointer into the buffer

(CBUFP$) was left pointing to the terminator that stopped the scan for the first valid file name specification when the MDOS command interpreter processed the input buffer. After completing its function, the command can return to MDOS through one of the system functions (.MDENT) which will pass control back to the MDOS command interpreter, repeating the cycle.

It should be noted here that commands invoked via the MDOS command interpreter do not necessarily have to have the suffix "CM" or reside on drive zero. If a user program with an "LO" suffix is being tested, it can be loaded and executed directly from the command line (if it meets the requirements for command-interpreter-loadable programs) by explicitly entering the suffix after the file name. Similarly, if a required command does not happen to reside on drive zero, its name can be followed with a logical unit number to cause it to be looked for and loaded from the specified unit. For example, the command line

```
DIR:2
```

will invoke the directory command from drive two to display the directory of the diskette in drive zero.

Whenever the MDOS command interpreter regains control after a command terminates, it checks that the diskette in drive zero still has the same parameters (version number, overlay RIB addresses) as the diskette used during the last MDOS initialization. If these parameters differ, one of the standard error messages EI, ER, EU, EV (Chapter 28) will be displayed and control given to the debug monitor. MDOS will then have to be reinitialized before the MDOS command interpreter will accept further commands.

In addition, the following parameters are reinitialized each time the MDOS command interpreter is given control. The user-defined SWI and IRQ vectors (SWI$UV and IRQ$UV) are reset to point to an RTI instruction. (Only SWI$UV is reset for MDOS09.) Since the user program is no longer resident, the interrupt handlers are deactivated. The stack pointer is reset to the end of contiguous memory for the duration of the command interpreter's execution. The Error Status and Error Type parts of the system error status word are set or cleared depending on whether or not a valid command name was entered on the command line.

24.7 Interrupt Handling

When MDOS initializes, it saves the contents of the SWI vector required by the debug monitor. The SWI and IRQ vectors are then changed to point into the MDOS function handler. Both vectors are required to allow the operator to make full use of the debug facilities of the debug monitor while using MDOS system functions. Some versions of the M6800 MPU will give control to the address in the IRQ vector if an NMI occurs while an SWI is in progress. Since the debug facilities of the debug monitor use NMI, continuing from a system function call will result in passing control to the address in the IRQ vector. Thus, MDOS must intercept both SWI and IRQ interrupts; however, MDOS can distinguish the difference between this "pseudo-IRQ" and a real IRQ even though both give control to the same address. Since MDOS does not have any devices in the system that generate IRQ, there is no true IRQ interrrupt handler. User programs, however, can configure the MDOS variable IRQ$UV so that if a real IRQ occurs, the routine specified by the user will be given control.

Such user-defined IRQ handlers are accessible as long as the MDOS command interpreter is not re-entered. Whenever control is returned to the MDOS command interpreter, the user-defined IRQ vector will be changed to point back into MDOS. Thus, IRQs cannot occur after the user program has terminated. Otherwise, MDOS will hang up in a loop. This is to be expected, since MDOS has no way of knowing what device generated the IRQ, where the device is, or how to respond to the IRQ. An IRQ must not be pending or occur when the MDOS command interpreter is given control.

Since the M6809 MPU does not give control to the address in the IRQ vector if an NMI occurs while an SWI is in progress, MDOS09 handles IRQs in a slightly different manner. During initialization, the IRQ vector is set up so that if an IRQ occurs, control is returned to EXbug after printing an "EQ" error message. If the user wishes to incorporate his own IRQ handler, he should save off the current value in the IRQ vector location (the one set up by MDOS) nad then insert the address of his IRQ handler. Only then is it safer to allow IRQs. MDOS does not reset the IRQ vector if control is returned to it. Thus, the user must take responsibility for restoring the original value upon completion of its interrupt processing.

For MDOS09, the FIRQ, SWI2 and SWI3 vectors are handled in a similar manner. MDOS09 sets up these vectors so that the respective messages "EF", "ES" and "EW" are printed if an interrupt occurs prior to the user having set up his own interrupt handler.

Certain precautions must be remembered if a user program is to process IRQs and use the MDOS system functions. Not all MDOS system functions are re-entrant, thus, SCALLs should not be used in an IRQ handler. Also, the MDOS diskette controller runs with interrupts inhibited for the duration of any diskette access. Thus, regardless of whether a single sector or multiple sectors are being processed, interrupts are inhibited throughout. Therefore, an IRQ cannot always be serviced within a definite time window if diskette accesses can be in progress. The time is dependent on the length of the diskette access.

Another potential problem exists if NMI is to be used while diskette functions are in progress. The NMI vector is taken over by the diskette controller while the diskette access is in progress. The NMI is used as a timeout condition. Thus, if a user's system is generating NMIs while diskette accesses are going on, a timeout condition will result and the user will not be able to process the NMI. It is for this reason that no user-defined NMI vector is provided by MDOS.

The system functions provided by MDOS are accessible through use of the software interrupt or SWI instruction. A full explanation regarding the MDOS SWIs is given in the next section; however, like the user-defined IRQ vector, MDOS allows a user-defined SWI vector to be configured through the variable SWI$UV. Like the user-defined IRQ handler, the user-defined SWI handler is only accessible as long as the MDOS command interpreter is not reentered. Whenever control is returned to the MDOS command interpreter, the user-defined SWI vector will be changed to point back into MDOS. Thus, user-defined SWIs cannot be processed after the user program has terminated. This is to be expected, since MDOS commands and user programs all load into one area of memory. Thus, the user-defined SWI handler is not resident after the MDOS command interpreter regains control.

24.8 System Function Calls

All of the system functions that MDOS commands use are also available to the user and can be incorporated into his program development. All MDOS system functions are accessed via the software interrupt or SWI instruction. Each SWI must be followed by a byte that contains the number of the function to be executed. MDOS's resident software interrupt handler can access up to 128 (decimal) functions; however, not all of these functions are defined. An error message will be printed if the software interrupt handler is activated and the function number is not defined.

A special convention is used to allow the user to define a maximum of 128 functions also (to be processed by the user's software interrupt handler that is configured via SWI$UV). If the sign bit of the function number byte (bit 7) is set to one, a user-defined software interrupt is indicated. All MDOS software interrupts have function number bytes with the sign bit set to zero. The user-defined SWI handler gets control with the registers on the stack as if it intercepted the SWI directly. The B accumulator will have the value of the function number (with the sign bit set to zero) to facilitate indexing into the user's function table.

Since MDOS assumes control of the SWI vector which is normally used by EXbug, certain precautions must be observed when debugging programs using the debug monitor.

1. MDOS must not be initialized via the debug monitor command "E800;G" or "MDOS" without first having depressed the ABORT or RESTART pushbuttons on the EXORciser's front panel. These two pushbuttons will restore EXbug's SWI vector.

2. The normal breakpoints can be used while testing a program, regardless of whether MDOS system functions are used or not; however, breakpoints set by simply placing an SWI instruction into memory via the memory change function will cause a system function to be executed rather than a breakpoint to occur. Breakpoints must only be set or cleared via the debug monitor commands.

3. Breakpoints can be set on an SWI instruction that is an MDOS system function call; however, before continuing from that particular breakpoint with the ";P" or ";N" commands, the breakpoint should be cleared (this is only true for the newer versions of the M6800 MPU which do not give control to the IRQ vector when an NMI occurs while an SWI is executing).

4. MDOS system functions cannot be traced or single-stepped through with the EXbug commands ";N" or ";T". Since these debug monitor functions utilize the stack, parts of MDOS will be overwritten due to the internal use of the stack pointer within the system function handler.

MDOS system function calls or user-defined function calls are programmed by using the SWI instruction mnemonic and the FCB assembler directive. If programs are assembled with the MDOS equate file (next section), the provided macro definitions with the names SCALL and UCALL can be used to generate the code for MDOS system functions and user-defined functions, respectively. The macros require an argument to be passed. This argument is the name or value of the function to be executed. The names of MDOS functions are assigned symbols in the MDOS equate file so that the use of absolute numbers is not necessary. Use of the SCALL or UCALL macro makes the program a bit easier to read, especially if names are used for the macro arguments.

MDOS system functions receive their parameters in the registers or in tables that are pointed to by the registers.

Chapters 25 and 27 contain the detailed entry parameters and exit conditions for all MDOS system functions.

Some system functions may not be able to perform their expected action. These functions will return an indication of whether a normal return or an abnormal return is being made. This condition is always passed back in the processor status (condition code) register. In addition, a status byte may be returned in one of the parameter tables or registers.

Some of the more complex system functions involving input or output can encounter fatal error conditions as well as non-fatal error conditions. Fatal errors suggest that the program is hopelessly confused. In these cases, the only logical action is to display what the problem appears to be and to re-enter the MDOS command interpreter. Non-fatal errors can include such things as illegal record formats, checksum errors, file protection violation, lack of space on the diskette, etc. Such conditions are noted and returned to the calling program. In these instances, it is the responsibility of the calling program to identify the source of the error and decide what the course of action should be.

24.9 MDOS Equate File

With each MDOS system diskette comes a file, EQU.SA, known as the MDOS equate file. The MDOS equate file contains the definitions of all symbols that are required by the resident MDOS and all of the MDOS commands. Not all of these symbols will be required by the user; however, the file is left as is to make it as useful as possible.

The MDOS equate file contains the following definitions. The sequence of the descriptions more or less follows the sequence of the file from beginning to end. Four macro definitions are found at the beginning of the MDOS equate file that are useful to the user.

Macro Name	Function
SKIP2	To be used as an instruction. The effect of the instruction is to execute a branch to location *+3. The "*" refers to the address of the branch instruction. The condition codes are changed as in a CPX instruction; however, this branch instruction requires only one byte of memory.
SKIP1	To be used as an instruction. The effect of the instruction is to

	execute a branch to location *+2. The condition codes are changed as in a BITA instruction; however, the branch instruction requires only one byte of memory.
SCALL	To be used with a single argument to execute a software interrupt (SWI) to the MDOS system function handler. This macro ensures that the sign bit of the function byte is set to zero. The symbols for the system functions are defined later in the MDOS equate file.
UCALL	To be used with a single argument to execute a software interrupt (SWI) to the user-defined function handler. This macro ensures that the sign bit of the function byte is set to one. The UCALL macro only makes sense if the user has configured an SWI handler.

All other macro definitions in the MDOS equate file are for internal use.

Following the macro definitions is a list of names that identifies all of the system functions accessible via the SCALL macro (or an SWI instruction followed by a function byte). These equates are defined using a macro that allows the labels to sequence themselves. Thus, if one label is removed from the list, the numbers assigned to the labels will still be consecutive, ascending integers. The first function is given the value of zero. Subsequent functions are assigned a number one higher than the previous function. If the SCALL macro is used in writing programs, it is suggested that the system symbols for the system functions also be used.

After the definitions of the system function symbols is a set of equates for all of the ASCII control characters including space and rubout characters. These symbols are followed by equates for the special MDOS delimiters used for suffixes, options, logical unit numbers, device names, and family indicators.

Next is a list of MDOS sector equates that defines where the various system tables are located. In addition, the sector size and the sectors/cylinder, etc., are defined.

Then, offsets into the various system tables are defined. These equates are followed by the definitions of the fields in the I/O control block (IOCB), which, in turn, are followed by another series of self-sequencing definitions for the various I/O function error statuses.

Following the error statuses, the locations of all of the MDOS internal variables are defined. These include the locations of the variables needed by the user for accessing the command buffer, the memory sizes established at initialization, and the user-defined interrupt vectors.

After the variables is a series of equates that defines the various bit positions of the IOCB, the offsets into the controller descriptor block (CDB), bit definitions within the CDB, and the offsets to the entry points of the device drivers.

Lastly, the diskette controller variables, entry points, and error statuses are equated to symbols. These equates are followed by a partial list of the locations in EXbug required by MDOS. The EXbug equate list is not complete. Thus, users requiring other entry points into EXbug must provide them within their programs.

If programs are being written that use the resident MDOS functions, it is suggested that the MDOS equate file be included as a part of the assembly (requires M6800 Macro Assembler). Symbols within the MDOS equate file may have their values changed by Motorola in subsequent versions of MDOS; however, all attempts will be made to ensure a minimal number of such changes. Therefore, the MDOS equate file should not be copied from one version of MDOS to another. Like the resident system and command files that comprise the operating system, the MDOS equate file is associated with a specific version and revision of the operating system.

A listing of the MDOS equate file is contained in Appendix I.

CHAPTER 25

25. INPUT/OUTPUT FUNCTIONS FOR SUPPORTED DEVICES

In the following description of the I/O functions for supported devices these symbols will be used:

Symbol	Meaning
A	A accumulator
B	B accumulator
X	Index register
CC	Condition code register
Z	Zero flag of condition code register (bit 2)
C	Carry flag of condition code register (bit 0)
CR	Carriage return

It is assumed that the reader is familiar with what system functions are, how they are invoked, what precautions must be taken when testing programs using system functions, and how errors are handled by system functions (see section 24.8).

25.1 Supported Devices

MDOS provides input and output functions to access the following supported devices:

MDOS Name	Physical Device
CN	Console keyboard and/or display
CP	Console punch
CR	Console reader
DK	Diskette drive
LP	Line printer

The following sections describe the system functions that are available for accessing these devices.

25.2 Device Dependent I/O Functions

MDOS provides system functions for directly accessing the console keyboard, display, line printer, and diskette drives. All of the functions are accessed by executing an SWI instruction followed by a function byte. The value of the function byte indicates the function to be executed and can be obtained from the MDOS equate file. All system functions that perform input/output operations require a stack in the user program area. The size of the stack must be at least 80 bytes (decimal). Each system function call pushes seven bytes on the stack. Since function calls may be nested within MDOS, a large stack is required. It should be noted that EXbug does not have sufficient stack space available; the stack area must be provided by the user elsewhere.

The device dependent functions for the console and the line printer use the device independent functions (section 25.3) via parameter tables held in the MDOS variable section of memory. Any error conditions detected by these system functions will cause the calling program to be aborted, a standard system error message to be displayed, and control to be given to the MDOS command interpreter. Since MDOS manages these parameter tables (reserving, opening, etc.), any error except "Buffer Overflow" during a console input will be a fatal error.

If, while accessing the console or the line printer, the errors are to be handled by the calling program, the device independent I/O functions (section 25.3) must be used instead.

25.2.1 Console input -- .KEYIN

The .KEYIN function inputs a specified number of characters from the system console keyboard. All characters entered (with the following exceptions) are stored into an input buffer. The function does not return until a terminating carriage return is supplied from the keyboard.

The following characters are treated as special control characters when encountered by the .KEYIN function:

Character	Value	Function
RUBOUT or DEL	$7F	Removes last character entered into buffer unless buffer is empty. The removed character is displayed on the system console to indicate that it has been removed from the buffer. No action occurs if the buffer is empty.

CTL-X or CAN	$18	Deletes all characters from the input buffer. A carriage return/line feed is displayed on the console to indicate that a new input line must be entered.
CTL-D or EOT	$04	Displays the current contents of the input buffer from the first character to the last character entered. The input is not terminated. This feature offers a means of displaying a "clean" line after many characters have been backed out via the RUBOUT key.
CTL-M or CR	$0D	Terminates the input. The carriage return is the last character placed into the input buffer. A carriage return/line feed is displayed on the console.

All characters are normally echoed on the console display mechanism to indicate that they have been entered into the input buffer; however, the following characters are echoed but are not placed into the input buffer:

Character	Value
Null	$00
Line feed	$0A
DC1	$11
DC2	$12
DC3	$13
DC4	$14

ENTRY PARAMETERS: B = The maximum number of characters to be input from the keyboard (not including the terminating CR). Characters entered after the maximum has already been input will not be echoed on the console, nor will they be placed into the input buffer. If B = 0, then only a CR will be accepted from the keyboard. The function does not return until a CR is entered.

X = The address of the input buffer that is to receive the data obtained from the console keyboard. The buffer must be large enough to accommodate one more character than is specified in B. This extra space must be provided for the terminating carriage return which is placed into the buffer. If X happens to contain the address of the MDOS command line buffer, then a special test is made to ensure that B is less than 80 (decimal). If B is greater than 79, it will be automatically changed to 79 to prevent the resident MDOS from being overwritten with keyboard data.

EXIT CONDITIONS: A is indeterminate.

B = The number of characters input (not including the terminating CR). If B = 0, then only a CR was entered.

X is unchanged.

CC is indeterminate.

The input buffer contains the entered data, including the terminating carriage return.

25.2.2 Check for BREAK key -- .CKBRK

The .CKBRK function examines the system ACIA for a framing error status, indicating that the BREAK key has been depressed since the last character was input from the console keyboard. This function also checks to see if the CTL-W key has been depressed. If the CTL-W is detected, the .CKBRK function will enter a loop waiting for any other character on the keyboard to be entered before returning to the calling program.

ENTRY PARAMETERS: None.

EXIT CONDITIONS: A, B, and X registers are unchanged.

C = 0, Z = 1 if no framing error (no BREAK key) is detected. The remainder of CC is indeterminate.

C = 1, Z = 0 if a framing error (BREAK key) is detected. The remainder of CC is indeterminate.

No indication is returned concerning the CTL-W key.

This feature merely allows the operator at the console to pause the system.

The framing error cannot be cleared from the ACIA by this function. The framing error can only be cleared upon subsequent reception of another character from the console keyboard. Thus, if the .CKBRK function is called more than once without the ACIA having received any characters between successive calls, the framing error status is detected in each case (even though the BREAK key was depressed only once). As a result, the BREAK key status is not detected if the BREAK key is depressed during an input request from the system console, since it is the reception of another character that clears the framing error status (and each input request must be terminated with a CR).

25.2.3 Console output -- .DSPLY, .DSPLX, .DSPLZ

The .DSPLY, .DSPLX, and .DSPLZ functions are all used to display a specified character string on the system console. The function .DSPLY displays a string that is terminated by a carriage return character. The functions .DSPLX and .DSPLZ display strings that are terminated by an EOT character, facilitating the use of embedded carriage returns within the string to output multiple-line messages with one function call. Both .DSPLY and .DSPLX will send a carriage return/line feed sequence to the console so that subsequent input or output is performed on a new line. The .DSPLZ function does not send the terminating carriage return/line feed sequence so that subsequent input or output can be performed on the same line as the displayed string.

ENTRY PARAMETERS: X = The address of a displayable ASCII string. The string must be terminated by a carriage return ($0D) if using .DSPLY. Otherwise, the string must be terminated by an EOT ($04). The functions .DSPLX and .DSPLZ will convert embedded carriage return characters into carriage return/line feed sequences automatically.

EXIT CONDITIONS: A and B registers are unchanged.

X = The address of the string's terminating character.

CC is indeterminate.

25.2.3.1 Example of console I/O

The following example illustrates the use of the .KEYIN and .DSPLY system functions. The example initially displays a message on the console to prompt the operator for input. The entered string is then displayed back on the console, but all characters have been reversed (the last character input is the first character output, etc.). If only a carriage return is entered, MDOS is given control via the system function .MDENT. The system function .ADBX is used to add the contents of the B accumulator to the X register. Both of these functions are described in Chapter 27. A maximum string length of ten is allowed. The example has been assembled with the MDOS equate file.

It is assumed in this example that the program is origined above location $1FFF since it is using the resident MDOS functions. The program can either be loaded with the LOAD command or invoked from the MDOS command interpreter directly. At the time the program is loaded, the stack pointer is automatically initialized to the last-loaded program location. In this example, this location is used as the top of the stack.

```
START  LDX    #PROMPT  .
       SCALL  .DSPLY   . SHOW INPUT PROMPT
*
* INPUT THE STRING FROM CONSOLE
*
INPUT  LDAB   #10      . MAX 10 CHAR
       LDX    #IBUFF   .
       SCALL  .KEYIN   . GET INPUT STRING
       TSTB            . CHECK FOR ZERO INPUT
       BNE    SWAP     .
       SCALL  .MDENT   . EXIT IF NO INPUT
*
* INVERT STRING INTO OBUFF
*
SWAP   LDX    #OBUFF   .
       SCALL  .ADBX    . POINT TO END OF OBUFF
       LDAA   #CR      . STORE TERMINATOR
       STAA   X        .
       DEX             .
       STS    STKSAV   . SAVE STACK POINTER
       LDS    #IBUFF-1 .
LOOP   PULA            . GET CHAR
       STAA   X        . STORE CHAR
       DEX             . BUMP POINTER
       DECB            .
       BNE    LOOP     . LOOP UNTIL ZERO
       LDS    STKSAV   . RESTORE STACK
       LDX    #OBUFF   .
       SCALL  .DSPLY   . SHOW INVERTED STRING
       BRA    INPUT    .
*
* WORKING STORAGE
*
IBUFF  BSZ    10+1     . INPUT BUFFER
OBUFF  BSZ    10+1     . OUTPUT BUFFER
PROMPT FCC    "ENTER STRINGS < 11 CHARACTERS"
       FCB    CR       .
STKSAV FDB    0        . SAVE AREA
       BSZ    80       . STACK SET HERE BY LOAD
*
       END    START    . BEGIN EXECUTION AT THIS LABEL
```

25.2.4 Printer output -- .PRINT, .PRINX

The .PRINT and .PRINX functions are both used to print a specified character string on the line printer. The function .PRINT prints a string that is terminated by a carriage return character. The function .PRINX prints a string that is terminated by an EOT character, facilitating the use of embedded carriage returns within the string to print multiple-line messages with one function call. Both functions will send a carriage return/line feed sequence to the printer at the end of each string. The .PRINX function will, in addition, send a carriage return/line feed sequence for each embedded carriage return character.

ENTRY PARAMETERS:	X = The address of a displayable ASCII string. The string must be terminated by a carriage return ($0D) if using .PRINT. Otherwise, the string must be terminated by an EOT ($04). The .PRINX function will convert embedded carriage return characters into carriage return/line feed sequences automatically.
EXIT CONDITIONS:	A and B registers are unchanged.
	X = The address of the string's terminating character.
	CC is indeterminate.

25.2.4.1 Example of printer output

The following example illustrates the use of the .PRINT system function. The example will print strings of eighty identical characters, beginning with spaces ($20) and proceeding through the entire displayable ASCII character set. The system function .STCHR is used to fill a buffer with the character contained in the A accumulator. The system function .MDENT is used to return control to MDOS. Both of these functions are described in Chapter 27. The example was assembled with the MDOS equate file.

It is assumed in this example that the program is origined above location $1FFF since it is using the resident MDOS functions. The program can either be loaded with the LOAD command or invoked from the MDOS command interpreter directly. At the time the program is loaded, the stack pointer is automatically initialized to the last-loaded program location. In this example, this location is used as the top of the stack.

```
START  LDAA   #SPACE   . INITIAL CHAR
LOOP   LDX    #OBUFF   .
       LDAB   #80      .
       SCALL  .STCHR   . FILL BUFFER
       SCALL  .PRINT   . PRINT THE STRING
       INCA            . BUMP CHARACTER
       CMPA   #RUBOUT  . END OF DISPLAYABLE SEQUENCE
       BNE    LOOP     .
       SCALL  .MDENT   . EXIT TO MDOS
*
* WORKING STORAGE
*
OBUFF  BSZ    80       . OUTPUT BUFFER
       FCB    CR       .
       BSZ    80       . STACK SET HERE BY LOAD
*
       END    START    . BEGIN EXECUTION AT THIS LABEL
```

25.2.5 Physical sector input -- .DREAD, .EREAD

The .DREAD and .EREAD functions are both used to read a single physical sector from the diskette into a specified buffer. For multiple physical sector input the functions in section 25.2.7 should be used. The .DREAD function will only return to the calling program if no diskette controller errors are detected during the read attempt. The .EREAD function, on the other hand, will return to the calling program whether an error occurred or not. The .EREAD function will return the error status that was detected by the diskette controller.

In either case, if a diskette error occurred that was retryable (CRC, deleted data mark, data address mark, or address mark CRC errors), the following steps were taken in an attempt to recover from the error:

1. The sector was reread five times without repositioning the read head.

2. The read head was stepped outward (towards cylinder zero) a maximum of five cylinders, repositioned over the cylinder in which the sector to be read resides, and another five read attempts were performed.

3. The read head was stepped inward (towards cylinder 76) a maximum of five cylinders, repositioned over the cylinder in which the sector to be read resides, and another five read attempts were performed.

If an error occurs during the .DREAD function, the standard "PROM I/O" error message will be displayed giving the status of the error and the sector number that was being accessed. Control will then be given to the MDOS command interpreter. If an error occurs during the .EREAD function, the EXIT CONDITIONS described below apply (for C = 1).

If either of these two functions is to access a diskette in a drive which as not had the read head restored (via functions .DIRSM, .OPEN, .LOAD or .CHANG, or via an MDOS command), then the diskette controller firmware must be invoked to restore the head. The RESTOR entry point is described in Appendix D. If the head is not restored properly, it is possible to receive timeout errors.

The diskette controller variables below location $0020 will be changed by these functions.

ENTRY PARAMETERS: B = The logical unit number. Bits 2-7 are ignored.

X = The address of a five-byte I/O parameter packet. The packet has the following format:

```
     ---------------------
  0  !   Return status   !
     ---------------------
  1  !  Physical sector  !
     --      number     --
  2  !    to be read     !
     ---------------------
  3  !  Address of 128   !
     --       byte      --
  4  !   sector buffer   !
     ---------------------
```

EXIT CONDITIONS: C = 0 if no errors occurred. The remainder of the CC is indeterminate.

The A register is indeterminate.

The X register is unchanged.

The B register contains the return status returned in the packet ($30).

The first byte of the parameter packet (Return Status) is set to $30 (ASCII zero). The remainder of the parameter packet is unchanged.

The sector buffer contains the 128 bytes read from the specified physical sector.

C = 1 if an error occurred (.EREAD only). The remainder of the CC is indeterminate.

The A register is indeterminate.

The X register is unchanged.

The B register contains the return status returned in the first byte of the parameter packet.

The first byte of the parameter packet contains the diskette controller error ($31-$39). Section 28.1 has a complete description of the diskette controller errors.

The contents of the 128 byte sector buffer are indeterminate.

25.2.6 Physical sector output -- .DWRIT, .EWRIT

The .DWRIT and .EWRIT functions are both used to write a single physical sector to the diskette from a specified buffer. For multiple physical sector output the functions described in section 25.2.8 should be used. The .DWRIT function will only return to the calling program if no diskette controller errors are detected during the write attempt. The .EWRIT function, on the other hand, will return to the calling program whether an error occurred or not. The .EWRIT function will return the error status that was detected by the diskette controller.

If an error occurred, the same type of recovery procedure described in section 25.2.5 (.DREAD, .EREAD) was attempted. In addition, the same precautions described for those functions regarding the restoring of the read head apply to the .DWRIT and .EWRIT functions.

ENTRY PARAMETERS:	Same as for .DREAD and .EREAD; however, the sector buffer must contain the 128 bytes that are to be written to the diskette.
EXIT CONDITIONS:	Same as for .DREAD and .EREAD; however, the the contents of the sector buffer are unchanged after returning to the calling program.

25.2.7 Multiple sector input -- .MREAD, .MERED

The .MREAD and .MERED functions are both used to read a multiple number of physically contiguous sectors from the diskette into a specified buffer. The .MREAD function will only return to the calling program if no diskette controller errors are detected during the read attempt. The .MERED function, on the other hand, will return to the calling program whether an error occurred or not. The .MERED function will return the error status that was detected by the diskette controller.

If an error occurred, the same type of recovery procedure described in section 25.2.5 (.DREAD, .EREAD) was attempted. In addition, the same precautions regarding the restoring of the read head described in that section apply to the .MREAD and .MERED functions.

ENTRY PARAMETERS: B = The logical unit number. Bits 2-7 are ignored.

X = The address of a seven-byte I/O parameter packet. The parameter packet has the following format:

```
     ---------------------
 0   !   Return status   !
     ---------------------
 1   ! Starting physical !
     --  sector number  --
 2   !    to be read     !
     ---------------------
 3   !    Address of     !
     --    multiple     --
 4   !   sector buffer   !
     ---------------------
 5   !     Number of     !
     --     sectors     --
 6   !    to be read     !
     ---------------------
```

The sector buffer must be an integral number of sectors in size, and must be large enough to accommodate the number of sectors specified in bytes 5 and 6 of the parameter packet.

EXIT CONDITIONS: Same as for .DREAD and .EREAD; however, the sector buffer contains data from the number of sectors specified in bytes 5 and 6 of the parameter packet (only if no error occurred).

25.2.8 Multiple sector output -- .MWRIT, .MEWRT

The .MWRIT and .MEWRT functions are both used to write a multiple number of physically contiguous sectors from a specified buffer to the diskette. The .MWRIT function will only return to the calling program if no diskette controller errors are detected during the write attempt. The .MEWRT function, on the other hand, will return to the calling program whether an error occurred or not. The .MEWRT function will return the error status that was detected by the diskette controller.

If an error occurred, the same type of recovery procedure described in section 25.2.5 (.DREAD, .EREAD) was attempted. In addition, the same precautions regarding the restoring of the read head described in that section apply to the .MWRIT and .MEWRT functions.

ENTRY PARAMETERS:	Same as for .MREAD and .MERED; however, the sector buffer must contain the bytes that are to be written to the diskette.
EXIT CONDITIONS:	Same as for .MREAD and .MERED; however, the contents of the sector buffer are unchanged after returning to the calling program.

25.2.9 Diskette controller entry points

The diskette controller has various entry points that allow the diskette to be accessed on a physical sector basis; however, since these entry points are independent of MDOS, they are described in a separate section (Appendix D). That appendix also describes some entry points for accessing the line printer on an MDOS-independent basis.

25.3 Device Independent I/O Functions

The following sections describe functions which facilitate writing software for input/output operations independent of the physical hardware device. In addition, these functions are used to access files on the diskette without having to perform physical sector I/O.

Through the use of a single parameter table, the I/O Control Block or IOCB, a common set of functions can be accessed independently of the I/O device. Thus, the same function would be called for writing a record to a diskette file or for writing a record to a line printer. The only difference is in the initial parameterization of the IOCB.

The normal sequence for calling the I/O functions, regardless of the device being used, is:

.RESRV	Reserve a device
.OPEN	Open a file
.GETRC	Read a record
.PUTRC	Write a record
.CLOSE	Close a file
.RELES	Release a device

The reading/writing of records, of course, may not necessarily be used for the same device. Once the file is open, the record I/O functions can be called as many times as required.

Use of the device independent I/O functions will cause the diskette controller variables below location $0020 to be changed, regardless of whether or not a diskette device is being used for a given I/O process.

In order to fully describe each device independent I/O function, the structure of the IOCB must first be described. In the description of the errors that can be returned by each function, the names of the system symbols from the MDOS equate file are used. These are noted in the description of the status byte of the IOCB, section 25.3.1.1. A summary of all possible input parameters that are required by the twelve different modes in which an IOCB can be used is contained in Appendix K.

25.3.1 I/O Control Block -- IOCB

The device independent I/O functions are parameterized through the IOCB. The I/O functions, in turn, interface to a device driver through another table, the Controller Descriptor Block or CDB (see section 26.2). It is only the device driver which interfaces directly to the device.

The IOCB is a table of flags, buffer pointers, and other information which is maintained by the calling program for the duration of the I/O accesses that are to be performed. Some of the entries in the IOCB must be initialized by the program before calling an I/O function. Other entries of the IOCB are initialized and changed by the I/O functions themselves. The entries of the IOCB must not be changed between I/O accesses unless specifically indicated in the ENTRY PARAMETERS section of each I/O function's description. The IOCB has the following format:

```
Byte
 !    7   6   5   4   3   2   1   0    <-- Bit position
 V   ---------------------------------
 00  !         Error status          !   IOCSTA
     ---------------------------------
 01  !  IO   ! S ! O ! T ! F !   M   !   IOCDTT - Data transfer
     ---------------------------------            type
 02  !          Data buffer          !
     --           pointer           --   IOCDBP
 03  !                               !
     ---------------------------------
 04  !          Data buffer          !
     --        start address        --   IOCDBS
 05  !                               !
     ---------------------------------
 06  !          Data buffer          !
     --         end address         --   IOCDBE
 07  !                               !
     ---------------------------------
 08  !      Generic device word      !
     --             or              --   IOCGDW
 09  !          CDB address          !
     ---------------------------------
 0A  !   ! R !          LUN          !   IOCLUN -- Logical unit
     ---------------------------------             number
 0B  !           File name           !
     --             or              --   IOCNAM / IOCMLS
 0C  !      Maximum LSN referenced   !
     ---------------------------------
 0D  !       File name continued     !
     --             or              --   IOCSDW
 0E  !Current segment descriptor word!
     ---------------------------------
 0F  !       File name continued     !
     --             or              --   IOCSLS
 10  !       Starting LSN of SDW     !
     ---------------------------------
 11  !       File name continued     !
     --             or              --   IOCLSN
 12  !    Next logical sector number !
     ---------------------------------
 13  !            Suffix             !
     --             or              --   IOCSUF / IOCEOF
 14  ! Logical sector number of EOF  !
     ---------------------------------
 15  !     Physical sector number    !
     --        of file's RIB        --   IOCRIB
 16  !                               !
     ---------------------------------
 17  ! W ! D ! S ! C ! N !    FMT    !   IOCFDF - File descrip-
     --                             --            tor flags
 18  !        (reserved; =0)         !
     ---------------------------------
```

```
          7   6   5   4   3   2   1   0
         ---------------------------------
   19    !                               !
         --        (reserved; =0)       --
   1A    !                               !
         ---------------------------------
   1B    !        PSN         !    EN    !  IOCDEN - Directory
         --                             --           entry number
   1C    !         (reserved; =0)        !
         ---------------------------------
   1D    !    Initial new file size      !
         --             or              --  IOCSBP
   1E    !    Sector buffer pointer      !
         ---------------------------------
   1F    !         Sector buffer         !
         --        start address        --  IOCSBS
   20    !                               !
         ---------------------------------
   21    !         Sector buffer         !
         --         end address         --  IOCSBE
   22    !                               !
         ---------------------------------
   23    !         Sector buffer         !
         --      internal pointer       --  IOCSBI
   24    !                               !
         ---------------------------------
```

IOCB FLAG DESCRIPTION SUMMARY

Field	Name	Bit	Content
IOCDTT	IO	6-7	I/O transfer flag
			Bit 6: 1 => Output transfer
			Bit 7: 1 => Input transfer
	S	5	Sector/record flag
			0 => Record I/O
			1 => Sector I/O
	O	4	Open/closed flag
			0 => File open
			1 => File closed
	T	3	Truncate flag
			0 => Ignore truncate action
			1 => Truncate file upon closing
	F	2	Non-file format flag
			0 => File format mode
			1 => Non-file format mode
	M	0-1	Mode flag
			00 => Update mode, existing file
			01 => Input mode, existing file
			10 => Output mode, new file
			11 => Update mode, any file
IOCLUN	-	7	Not used (=0)
	R	6	Reserved flag
			0 => IOCB released
			1 => IOCB reserved
	LUN	0-5	Logical unit number ($30-$39)
IOCFDF	W	F	Write protection bit
			0 => No write protection
			1 => Write protected
	D	E	Delete protection bit
			0 => No delete protection
			1 => Delete protected
	S	D	System file bit
			0 => Non-system file
			1 => System file
	C	C	Contiguous allocation bit
			0 => Segmented allocation
			1 => Contiguous allocation
	N	B	Non-compressed space bit
			0 => Spaces compressed
			1 => Spaces non-compressed

IOCB FLAG DESCRIPTION SUMMARY continued

Field	Name	Bit	Content
IOCFDF	FMT	8-A	File format 000 => User-defined format 001 => Use device's default format for binary records 010 => Memory-image format 011 => Binary record format 100 => Undefined format 101 => ASCII record format 110 => Undefined format 111 => ASCII-converted-binary record format
	-	0-7	Not used (=0)
IOCDEN	PSN	B-F	Physical sector number ($03-16)
	EN	8-A	Entry number within sector (0-7)
	-	0-7	Not used (=0)

25.3.1.1 IOCSTA -- Error status

The IOCSTA byte contains the return status from an I/O function. A zero in this byte indicates that an I/O function completed normally without any errors. A non-zero value indicates that an I/O function encountered some sort of an error. The following table contains all of the currently defined values that can be returned in the IOCSTA. Along with each value the system symbol equated to the value (MDOS equate file), and the standard error message that would be displayed if the error message function were invoked to show a message are given. The two-digit reference number displayed along with the error message should be used to locate the error message's description in Chapter 28. It should be noted that in order to decode the IOCSTA byte into the proper error message, the error message function, .MDERR, must be called with the B accumulator equal to zero. Section 27.4 describes the error message handler.

IOCSTA Value	System Symbol	Standard Error Message Displayed by .MDERR (B=0, X=IOCB address)
00	I$NOER	Normal return, no error
01	I$NODV	** 28 DEVICE NAME NOT FOUND
02	I$RESV	** 18 DEVICE ALREADY RESERVED
03	I$NORV	** 19 DEVICE NOT RESERVED
04	I$NRDY	** 11 DEVICE NOT READY
05	I$IVDV	** 31 INVALID DEVICE
06	I$DUPE	** 06 DUPLICATE FILE NAME
07	I$NONM	** 04 FILE NAME NOT FOUND
08	I$CLOS	** 20 INVALID OPEN/CLOSED FLAG
09	I$EOF	** 21 END OF FILE
0A	I$FTYP	** 14 INVALID FILE TYPE
0B	I$DTYP	** 17 INVALID DATA TRANSFER TYPE
0C	I$EOM	** 37 END OF MEDIA
0D	I$BUFO	** 22 BUFFER OVERFLOW
0E	I$CKSM	** 23 CHECKSUM ERROR
0F	I$WRIT	** 26 FILE IS WRITE PROTECTED
10	I$DELT	** 10 FILE IS DELETE PROTECTED
11	I$RANG	** 24 LOGICAL SECTOR NUMBER OUT OF RANGE
12	I$FSPC	** 41 INSUFFICIENT DISK SPACE
13	I$DSPC	** 40 DIRECTORY SPACE FULL
14	I$SSPC	** 42 SEGMENT DESCRIPTOR SPACE FULL
15	I$IDEN	** 43 INVALID DIRECTORY ENTRY NO. AT nnnn
16	I$RIB	** 32 INVALID RIB
17	I$DEAL	** 44 CANNOT DEALLOCATE ALL SPACE, DIRECTORY ENTRY EXISTS AT nnnn
18	I$RECL	** 45 RECORD LENGTH TOO LARGE
19	I$SECB	** 52 SECTOR BUFFER SIZE ERROR

25.3.1.2 IOCDTT -- Data transfer type

The IOCDTT byte contains the basic information about an I/O access: whether an input or an output transfer is to take place, whether sector or record I/O is to be performed, whether the file is currently open or closed, whether a file (diskette only) should be truncated when it is closed, and whether the file or non-file format mode is to be used.

The format of the IOCDTT byte is shown below:

```
  7   6   5   4   3   2   1   0
 -------------------------------
 !  IO   ! S ! O ! T ! F !   M   !
 -------------------------------
    :      :   :   :   :      :..... Mode flag
    :      :   :   :   :........... Non-file format flag
    :      :   :   :............... Truncate flag
    :      :   :................... Open/closed flag
    :      :....................... Sector/record flag
    :.............................. I/O transfer flag
```

Regardless of the type of device being accessed, the non-file format flag (F) and the mode flag (M) are to be initialized by the user. If the device is a diskette drive, the user may also change the sector/record flag (S) or the truncate flag (T) between I/O function calls. If the flags are to be changed after the IOCDTT byte has been initialized, care must be taken so that none of the system supplied flags are destroyed. Flags must be "or-ed" into the IOCDTT to be set, and "and-ed" out of the IOCDTT to be cleared, once the IOCB has been reserved.

The properties controlled by the various bits of the IOCDTT are explained below.

IO (Bits 6-7) -- I/O transfer flag

These two bits are controlled exclusively by the I/O functions themselves. They should not be set or changed by the user in any case. If bit 6 is set to one, the device driver recognizes an output transfer. If bit 7 is set to one, the device driver recognizes an input transfer. The device driver will not be able to input or output a character if both of these bits are zero or one.

S (Bit 5) -- Sector/record flag

This bit controls whether sector or record processing is performed during an I/O function. For non-diskette devices, this bit must always be zero. For diskette devices, this bit can be in either state. A one implies that logical sector I/O will be performed. A zero implies that record I/O will be performed; however, care must be taken that the corresponding I/O function is called for the proper state of the bit. That is, the record I/O functions (.GETRC and .PUTRC) cannot be called if "S" is set to one. Likewise, the logical sector I/O functions (.GETLS and .PUTLS) cannot be called if "S" is set to zero.

O (Bit 4) -- Open/closed flag

This bit is supplied by the system I/O functions if they are properly called in their correct sequence. The "O" bit must not be changed once I/O transfers have been made. A one indicates that the file (or device) is closed. A zero, on the other hand, indicates that the file (or device) is open.

T (Bit 3) -- Truncate flag

The truncate flag is only applicable to I/O on a diskette device. Normally, the user will not have to set or change this bit; however, certain cases will arise where changing of the truncate flag by the user may be necessary (see .CLOSE function, section 25.3.6). The truncate flag is used as an indication that new space was allocated to a diskette file. If it is set to one, any unused parts of the newly allocated space (space beyond the maximum logical sector number referenced in IOCMLS) will be deallocated (returned to the available diskette space) when the file is closed. If the truncate flag is zero, no truncation will occur upon closing.

A special case exists if IOCMLS contains the value $FFFF when the truncate flag is set to one. In addition to having all of the file's space deallocated, the directory entry belonging to the file is removed from the directory. The file is, in effect, deleted.

F (Bit 2) -- Non-file format flag

If "F" is set to one, the non-file format mode is indicated. In this mode, all I/O must be to a non-diskette device. No FDR (File Descriptor Record) processing is performed. The only valid file format that can be supported in this mode is ASCII (FMT = 5 of IOCFDF).

If the "F" flag is set to zero, then the file format mode is indicated. In this mode, I/O can be either to a diskette or to a non-diskette device. If a non-diskette device is being used, FDR processing will be performed. That is, an FDR will be written to the device if opened for output, or an FDR will be searched for on the device if opened for input. The file format mode (F = 0) must be used for accessing the diskette.

M (Bits 0-1) -- Mode flag

The mode flag can take on one of four different values:

00 => Open an existing file (diskette only) for either input or output.

01 => Open an existing diskette file or open a device for input only.

10 => Create a new diskette file or open a device for output only.

11 => Open an existing file or create a new file (diskette only) for either input or output.

The update modes (M = 00 or 11) can only be used when accessing diskette files. The way in which the four different modes are used is described in the .OPEN function, section 25.3.3.

25.3.1.3 IOCDBP -- Data buffer pointer

This two-byte field of the IOCB is used as a working storage area by the record I/O functions. This entry should not be changed by the calling program once I/O functions have been called.

25.3.1.4 IOCDBS -- Data buffer start

This two-byte field of the IOCB must be initialized by the calling program before any record I/O functions are called. IOCDBS must be configured to contain the address of the first byte of a buffer into which a record is to be read, or from which a record is to be written. None of the I/O functions will alter IOCDBS. The data buffer may be used for FDR processing by the .OPEN function (section 25.3.3) when dealing with non-diskette devices.

25.3.1.5 IOCDBE -- Data buffer end

This two-byte field of the IOCB must be initialized by the calling program before any record I/O functions are called. IOCDBE must be configured to contain the address of the last byte of a buffer into which a record is to be read, or from which a record is to be written. During record input, IOCDBS and IOCDBE define the maximum size record that the buffer can accommodate. During record output, IOCDBS and IOCDBE describe the first and last byte of the record to be written. None of the I/O functions will alter IOCDBE. The data buffer may be used for FDR processing by the .OPEN function (section 25.3.3) when dealing with non-diskette devices.

25.3.1.6 IOCGDW -- Generic device word

This two-byte field of the IOCB serves a dual function. Before any I/O functions can be invoked, IOCGDW must contain the MDOS device name that is to be accessed (see section 25.1). The device name consists of two ASCII characters. Once the .RESRV function (section 25.3.2) has been called, IOCGDW will contain the address of the controller descriptor block (CDB, section 26.2.1) associated with that device. After the CDB address has been put into IOCGDW, the contents of this field must not be changed by the calling program. Section 26.2 contains a description of how to configure the IOCGDW field for non-supported devices.

25.3.1.7 IOCLUN -- Logical unit number

The IOCLUN byte contains two pieces of information. Initially, the calling program must store the logical unit number of the device to be accessed in this byte. The logical unit number identifies a specific device within a generic device family (e.g., drive zero of the family DK). If there is only one device in a generic device family, a logical unit number of zero must be placed in IOCLUN. Logical unit numbers should be ASCII numbers in the range $30-$39 (0-9). Bit "R" of IOCLUN indicates whether or not the IOCB has been reserved (.RESRV function). Initially, when the logical unit number is stored in IOCLUN, bit "R" will be set to zero. After the .RESRV function has been successfully invoked, bit "R" will be set to one, indicating that the IOCB has been reserved. The IOCLUN field must not be changed by the calling program after the .RESRV function has been called.

25.3.1.8 IOCNAM -- File name

These eight bytes of the IOCB serve a dual purpose. If the non-file format mode is being used (F = 1 of IOCDTT), IOCNAM is not used at all; however, in the file format mode, IOCNAM must contain the name of the file to be accessed. The file name must be in the valid MDOS file name format. Any unused parts of the name must be spaces ($20). The file name should be placed into IOCNAM before the .OPEN function is invoked. After a file has been opened, the eight bytes will be replaced with the four two-byte fields IOCMLS, IOCSDW, IOCSLS, and IOCLSN (only if the device is diskette).

When dealing with non-diskette devices in the file format mode, the IOCNAM entry can be configured so that the first byte is a binary zero. In this case, the .OPEN function will search for the first FDR on the non-diskette device, and place the found file name (and suffix) into IOCNAM (and IOCSUF).

25.3.1.9 IOCSUF -- Suffix

This two-byte field of the IOCB serves a dual purpose. If the non-file format mode is being used (F = 1 of IOCDTT), IOCSUF is not used at all; however, in the file format mode, IOCSUF must contain the suffix of the file to be accessed. The suffix must be in the valid MDOS suffix format. Any unused parts of the suffix must be spaces ($20). The suffix should be placed into IOCSUF before the .OPEN function is invoked (at the same time that the file name is placed into IOCNAM). After a file has been opened, IOCSUF will be replaced with the two-byte field IOCEOF (only if the device is diskette). If the device being accessed is the system console, the first character of the IOCSUF field may be changed by the user to a displayable ASCII character ($20-$5F). Then, whenever an input request is made on that device, the character will be displayed as an input prompt.

When dealing with non-diskette devices in the file format mode, the IOCNAM entry can be configured so that the first byte is a binary zero. In this case, the .OPEN function will search for the first FDR on the non-diskette device, and place the found file name (and suffix) into IOCNAM (and IOCSUF).

25.3.1.10 IOCMLS -- Maximum LSN referenced

This two-byte field of the IOCB overlays the first two bytes of the IOCNAM after the .OPEN function has been called (diskette I/O only). It is a system-maintained field that contains the maximum logical sector number ever referenced by any of the I/O functions. IOCMLS and the truncate flag (T of IOCDTT) are used in determining the amount of newly allocated diskette space that is to be deallocated from a file when it is closed. Space will only be deallocated if the truncate flag is set to a one. Since MDOS automatically sets the truncate flag to a one if new diskette space is allocated to a file, any unused space will always be returned to the available space pool.

Normally, the user never changes the IOCMLS or the truncate flag in the IOCDTT since the truncate flag is automatically set whenever additional space allocation is performed or whenever a new file is created. When accessing an existing file using both input and output (M = 00 or 11 of IOCDTT), however, the truncate flag may have to be set to one by the user if the file is to be shortened or if the end-of-file pointer in the RIB is to be updated. If an extant file does not grow in size, the truncate flag will be zero.

In addition, when files are to be deleted (upon a subsequent .CLOSE function call), the IOCMLS must be set to a value of $FFFF and the truncate flag must be set to one.

25.3.1.11 IOCSDW -- Current SDW

The IOCSDW field overlays the second two bytes of IOCNAM after the .OPEN function has been called (diskette I/O only). This field contains the segment descriptor word which identifies the current file segment that can be accessed. If another segment of the file is to be accessed, the disk driver will automatically reread the file's RIB and extract the appropriate SDW into IOCSDW. The contents of IOCSDW should never be changed by the calling program.

25.3.1.12 IOCSLS -- Starting LSN of SDW

The IOCSLS field overlays the third two bytes of IOCNAM after the .OPEN function has been called (diskette I/O only). This field contains the starting logical sector number of the current segment descriptor word. The contents of IOCSLS should never be changed by the calling program.

25.3.1.13 IOCLSN -- Next LSN

The IOCLSN field overlays the fourth two bytes of IOCNAM after the .OPEN function has been called (diskette I/O only). This field is never changed by the calling program if record I/O (S = 0 of IOCDTT) is being used. If logical sector I/O is being used (S = 1 of IOCDTT), then IOCLSN can be changed by the calling program to specify which logical sectors are to be read from or written to the file. This feature allows the calling program to randomly access the file (by logical sector number) without having to know physically where the file resides on the diskette. After an I/O access has been completed, IOCLSN will contain the logical sector number of the next sector on the diskette to be accessed. When using a multiple sector buffer, IOCLSN may have been incremented by more than one, depending on the number of sectors processed.

25.3.1.14 IOCEOF -- LSN of end-of-file

The IOCEOF field overlays IOCSUF after the .OPEN function has been called (diskette I/O only). IOCEOF is a system-maintained parameter that represents the logical sector number of the logical end-of-file. This value must not be changed by the calling program once the .OPEN function has been invoked.

25.3.1.15 IOCRIB -- PSN of RIB

This two-byte field of the IOCB is initialized with the physical sector number of the file's RIB after the .OPEN function has been called (diskette I/O only). The RIB is used to access the file via its SDWs to allocate additional space, to deallocate unused space, and to monitor the LSN of the file's logical end-of-file. The IOCRIB entry should never be changed by the calling program.

25.3.1.16 IOCFDF -- File descriptor flags

This two-byte field contains the flags that describe the inherent and the changeable attributes of a file. The format of the IOCFDF entry is shown below:

```
  F   E   D   C   B   A   9   8   7   6   5   4   3   2   1   0
-----------------------------------------------------------------
! W ! D ! S ! C ! N !   FMT     !                               !
-----------------------------------------------------------------
  :   :   :   :   :       :    <-------- Not Used (=0) -------->
  :   :   :   :   :       :
  :   :   :   :   :       :... File format bits
  :   :   :   :   :........... Non-compressed space bit
  :   :   :   :............... Contiguous allocation bit
  :   :   :................... System file bit
  :   :....................... Delete protection bit
  :........................... Write protection bit
```

The functions of the various bits are described below:

W (Bit F) -- Write protection bit

The "W" bit only applies to diskette files. If this bit is set to one, the file can only be accessed with input requests. Any I/O functions that attempt to write to a file with the "W" bit set will return an error. In addition, the file cannot be deleted. If the "W" bit is set to zero, the file can be read from, written to, or deleted (the "D" bit must be zero also). The "W" bit is one of the changeable attributes of a file.

D (Bit E) -- Delete protection bit

The "D" bit only applies to diskette files. If this bit is set to one, the file cannot be deleted. If the "D" bit is set to zero, the file can be deleted (the "W" bit must be zero also). The "D" bit is one of the changeable attributes of a file.

S (Bit D) -- System file bit

The "S" bit only applies to diskette files. If this bit is set to one, the file is considered to be a system file. System files are treated specially by the DIR, DEL, and DOSGEN commands. If the "S" bit is set to zero, the file is not a system file. The "S" bit is one of the changeable attributes of a file.

C (Bit C) -- Contiguous allocation bit

The "C" bit only applies to diskette files. If this bit is set to one, only contiguous diskette space can be allocated to the file. All files whose contents are to be loaded into memory directly from the diskette must be allocated contiguous space. If the "C" bit is set to zero, the file may be allocated segmented diskette space. The "C" bit is one of the inherent attributes of a file. It is specified at the time the file is created and cannot be changed thereafter.

N (Bit B) -- Non-compressed space bit

The "N" bit only applies to diskette files. If this bit is set to one, ASCII records written to the file will not have spaces compressed. If the "N" bit is set to zero, ASCII records written to the file will have spaces compressed into a byte of the following format:

```
  7   6   5   4   3   2   1   0
 ---------------------------------
 !   !                           !
 ---------------------------------
   :               :
   :               :.... Number of compressed spaces
   :.................... Compression flag (=1)
```

All MDOS commands create ASCII files with space compression (N = 0) in order to minimize the amount of diskette space consumed. The "N" bit is one of the inherent attributes of a file. It is specified at the time the file is created and cannot be changed thereafter. The space compression attribute is only meaningful if the file format is ASCII record (FMT = 5). For other formats, the space compression attribute is ignored.

FMT (Bits 8-A) -- File format bits

The file format bits describe the internal data structure of the file. The file format is one of the inherent attributes of a file. FMT is specified at the time the file is created and cannot be changed thereafter. The following table lists the values of FMT and their meanings:

FMT	File format
0	User-defined format. This format is only valid for diskette files. The record I/O functions cannot be used to access files with this format. Only logical sector I/O can be performed with this format. The calling program is responsible for extracting data from the sectors according to his data structure.
1	Use device's default format for binary records. Each device has associated with its CDB (section 26.2) a flag that indicates what the default binary record format is (either FMT = 3 or FMT = 7). Since some devices can only process seven-bit data while other devices can process both seven-bit and eight-bit data, this format (FMT = 1) allows a program to process binary records without knowing the specific format supported by a particular device. The program will always be dealing with eight-bit data in memory. The FMT field is automatically changed to either a "3" or "7" depending on the device by the .OPEN function.
2	Memory-image format. This format applies only to diskette files. Any file whose contents are to be loaded into memory directly from the diskette must be in the memory-image format. Due to the nature of the diskette controller, memory-image format files must be allocated contiguous diskette space (C = 1 of IOCFDF). Memory-image files have no record information within the data sectors. All information concerning the starting load address, number of bytes to load, etc., is contained in the file's RIB. The load information must be written into the RIB by the program that is creating the memory-image file; the information is not automatically supplied by any system function. The load information must meet the requirements defined in section 24.2. The record I/O functions cannot be used to access files with this format. Only logical sector I/O can be performed with this format.

3 Binary record format. This format applies to both diskette and non-diskette files; however, non-diskette files can only be accessed in the file format mode (F = 0 of IOCDTT) using this format.

4 This format is undefined and should not be used.

5 ASCII record format. This format applies to both diskette and non-diskette files. Non-diskette files of this format can be accessed in either the file format or the non-file format modes. ASCII record files can be space compressed, but only if they reside on diskette.

6 This format is undefined and should not be used.

7 ASCII-converted-binary record format. This format usually applies to non-diskette files. This format is intended to be used for writing binary record files from the diskette to a non-diskette device that can only accept seven-bit data bytes. Otherwise, this format is identical to FMT = 3.

NOT USED (Bits 0-7) -- Reserved area

The least significant byte of the IOCFDF field is reserved for future expansion. This byte must be zero for all files.

25.3.1.17 IOCDEN -- Directory entry number

Associated with each directory entry is a number, the directory entry number, which is a function of the physical location of the entry within the directory. The directory entry number is not found anywhere in the directory, rather it is a calculated quantity. The two-byte IOCDEN field is supplied by the system after the .OPEN function (section 25.3.3) has been called. It only applies to diskette files. The contents of IOCDEN should never be changed by the calling program. The IOCDEN field has the following format:

```
 F   E   D   C   B   A   9   8   7   6   5   4   3   2   1   0
----------------------------------------------------------------
!       PSN          !    EN     !                              !
----------------------------------------------------------------
         :                :       <--------- Not Used (=0) ------->
         :                :
         :                :..... Position within sector (0-7)
         :
         :...................... Physical sector number ($3-$16)
```

25.3.1.18 IOCSBP -- Sector buffer pointer

The IOCSBP field only applies to diskette I/O. This two-byte field of the IOCB serves a dual purpose. If an existing file is being opened, the initial value of IOCSBP is ignored. If a file is being created, this field must contain the initial number of sectors that are to be allocated to the file. If the value of zero is specified, MDOS will default the initial file size to a full segment descriptor (32 clusters) and no error will occur during the file's initial space allocation if fewer than 32 clusters are available. If a non-zero (non-default) initial size is specified, however, an error will occur if that initial size cannot be allocated. The .ALLOC system function description (section 27.4) contains a more detailed explanation of the allocation mechanism.

After a file has been opened, the IOCSBP contains a pointer into the sector buffer that is used by the record I/O functions. Therefore, the contents of IOCSBP must not be changed by the calling program once a file is open when using the record I/O functions. If the sector I/O functions are used, then IOCSBP can be altered by the calling program in any way after a file is open.

25.3.1.19 IOCSBS -- Sector buffer start

This two-byte field of the IOCB only applies to diskette I/O. It must be initialized by the calling program before any of the I/O functions are invoked. IOCSBS must be configured to contain the address of the first byte of a buffer into which one or more 128-byte sectors can be read. This sector buffer will be used for directory searches as well as for data transfers. IOCSBS will not be altered by any of the I/O functions.

25.3.1.20 IOCSBE -- Sector buffer end

This two-byte field of the IOCB only applies to diskette I/O. It must be initialized by the calling program before any of the I/O functions are invoked. IOCSBE must be configured to contain the address of the last byte of a sector buffer that is exactly large enough to accommodate an integral number of 128-byte sectors. An error will occur if the size of the sector buffer described by IOCSBS and IOCSBE is not correct. Specifically, the following relationship must be true:

```
        IOCSBE-IOCSBS+1
        --------------- = INTEGER (Maximum # of Sectors)
            128
```

IOCSBE will not be altered by any of the I/O functions.

25.3.1.21 IOCSBI -- Internal buffer pointer

This two-byte field of the IOCB applies only to diskette I/O. IOCSBI is used to indicate the end of valid data within sector buffers. Since partial buffers (an integral number of sectors less than or equal to the maximum sector buffer size) may be read or written, IOCSBI is used to locate the last valid data byte within a sector buffer.

IOCSBI is initialized and changed by the I/O functions. The contents of IOCSBI must not be changed by the calling program after a file has been opened when using the record I/O functions; however, when using logical sector I/O, the contents of IOCSBI may be changed. The value of IOCSBI will always be less than or equal to the value of IOCSBE. The following relationship must always be true:

```
        IOCSBI-IOCSBS+1
        --------------- = INTEGER (Actual # of Sectors)
            128
```

25.3.2 Reserve a device -- .RESRV

The .RESRV system function links the appropriate controller descriptor block (CDB) to the calling program's IOCB. The .RESRV function must be called before any other of the device independent I/O functions can be invoked. Section 26.2.4 should be consulted for a description of the impact on the .RESRV call and the IOCB when using non-standard devices.

ENTRY PARAMETERS:	X = The address of an IOCB.
	IOCGDW must contain one of the valid generic device names: CN, CP, CR, DK, or LP.
	IOCLUN must contain the logical unit number of the device to be reserved.

Bit "R" of IOCLUN must be set to zero (this will normally be the case when the ASCII logical unit number, $30-$39, is stored into IOCLUN).

All other entries of the IOCB need not be initialized.

EXIT CONDITIONS: A is indeterminate.

B = The contents of the IOCSTA entry. If no errors occurred, B will be zero. A non-zero value indicates that an error occurred.

X is unchanged.

C =0 and Z = 1 if no errors occurred (B = 0). The remainder of CC is indeterminate.

C = 1 and Z = 0 if an error occurred (B not zero). The remainder of CC is indeterminate.

The IOCB is affected in the following manner if an error occurred:

IOCSTA contains the error status. The following error statuses can be returned: I$IVDV, I$RESV, I$NODV.

The remainder of the IOCB is not changed.

The IOCB is affected in the following manner if no errors occurred:

IOCSTA = 0.

IOCDTT has the "IO" bits set to zero and the "O" bit set to one (file closed). The remainder of the IOCDTT is not changed.

IOCGDW contains the address of the CDB that is associated with the generic device. The original contents of IOCGDW are destroyed.

IOCLUN has the "R" bit set to one (IOCB reserved). The remainder of IOCLUN is not changed.

The remainder of the IOCB is not changed.

25.3.3 Open a file -- .OPEN

The .OPEN function prepares a file for subsequent access by the record or logical sector I/O functions. Data cannot be transferred between the file (or device) and the calling program until the .OPEN function has been invoked. The specific function performed by .OPEN depends on the device type and on the contents of the IOCDTT entry (specifically, the non-file format flag (F) and the mode flag (M)).

There are four modes in which a file can be opened. The input mode (M = 01 of IOCDTT) will allow only input requests to be issued to the file. The output mode (M = 10 of IOCDTT) will allow only output requests to be issued to the file, and the update modes (M = 00 or 11 of IOCDTT) will allow both types of requests to be issued to the file. The update modes are only valid if the device type is DK.

The non-file format flag also has an effect on what .OPEN does. If the file format mode is specified (F = 0 of IOCDTT), then FDR processing will be performed. FDR processing consists of searching for a file descriptor record or a directory entry if the file is being opened for input. FDR processing consists of creating a file descriptor record or a directory entry if the file is being opened for output. One form of update mode processing (M = 11 of IOCDTT) will be identical to the input mode processing if the file already exists in the directory; or, it will be identical to the output mode processing if the file does not exist in the directory. The other form of update mode processing (M = 00 of IOCDTT) will always be the same as the input mode processing since the file must exist for this mode.

If a memory-image file is being created, the load information must be written into the RIB by the program that is creating the file and must meet the requirements described in section 24.2. The RIB can be accessed using logical sector I/O. It has the logical sector number $FFFF.

If the non-file format mode is specified (F = 1 of IOCDTT), then no FDR processing is performed. The non-file format mode is invalid for diskette devices.

ENTRY PARAMETERS:	X = The address of an IOCB which has been properly reserved (i.e., no errors occurred) via the .RESRV function. Since the IOCB needs to be reserved only once per device of a given logical unit number, it is possible to open and close a file and then reopen another file using the same IOCB without issuing another .RESRV call. In these instances, the IOCB

must not contain information for an open file (i.e., the first file must have been properly closed). The .OPEN function does not force an already-open file to be closed.

IOCDTT must have the "M" bits set for input, output, or update modes. The update modes are only valid for diskette devices. In addition, the "F" bit must specify file or non-file format. The non-file format mode is invalid for diskette devices. The "S" bit must indicate the subsequent access method to be used. Sector I/O is invalid for non-diskette devices.

IOCDBS must contain a buffer start address unless diskette I/O (either record or logical sector) or the non-file format mode has been specified in the IOCDTT. The data buffer described by IOCDBS and IOCDBE is used for FDR processing with non-diskette devices. If used, it must be large enough to accommodate an FDR (section 24.3.4).

IOCDBE must contain a buffer end address unless diskette I/O (either record or logical sector) or the non-file format mode has been specified in the IOCDTT. The data buffer described by IOCDBS and IOCDBE is used for FDR processing with non-diskette devices. If used, it must be large enough to accommodate an FDR (section 24.3.4).

IOCNAM must contain a valid MDOS-formatted file name unless the non-file format mode has been specified in the IOCDTT or unless the first byte of file name is binary zero. In the file format mode on a non-diskette device being opened for input, the .OPEN function will cause a search to be performed for the first FDR if the first byte of IOCNAM is a binary zero. This file will then be used by the subsequent record input requests. Otherwise, the file name supplied in IOCLUN, IOCNAM, and IOCSUF is searched for or created (depending on M of IOCDTT).

IOCSUF must contain a valid MDOS-formatted suffix unless the non-file format mode has been specified in the IOCDTT or unless the first byte of IOCNAM contained a binary zero (see above).

IOCFDF must only be initialized to specify the file format (FMT bits) if the output mode (M = 10 of IOCDTT) or the update mode to a non-existing file (M = 11 of IOCDTT) is indicated. In addition, if the device type is DK, the other bits of IOCFDF must be specified for these two open modes. A special case exists if the non-file format mode is indicated in the IOCDTT. In this instance, the FMT bits of IOCFDF must be set to the ASCII record format (FMT = 5).

It is not recommended that diskette files be created with the protection attributes set, since they will prevent a file from being deleted upon closing if no information was written into the file. The protection attributes should be set via the .CHANG system function or via the NAME command.

IOCSBP must be initialized if the device type is DK and either the output mode (M = 10 of IOCDTT) or the update mode to a non-existing file (M=11 of IOCDTT) is specified. A value of zero will cause the default space to be initially allocated to the file. A non-zero value will cause that number of sectors to be used for the initial allocation.

A non-zero value in IOCSBP when opening an existing file will have no affect on the allocation of the file. Existing files only change in size when writing beyond the end-of-file or when closing them with the truncate flag set.

IOCSBS must contain the starting address of a sector buffer only if the device type is DK. The sector buffer must be an integral number of sectors in size (see section 25.3.1.20).

IOCSBE must contain the address of the last byte of a sector buffer only if the device type is DK. The sector buffer must be an integral number of sectors in size (see section 25.3.1.20).

EXIT CONDITIONS: A is indeterminate.

B = The contents of the IOCSTA entry. If no errors occurred, B will be zero. A non-zero value indicates that an error occurred.

X is unchanged.

C = 0 and Z = 1 if no errors occurred (B = 0). The remainder of CC is indeterminate.

C = 1 and Z = 0 if an error occurred (B not zero). The remainder of CC is indeterminate.

The IOCB is affected in the following manner if an error occurred:

IOCSTA contains the error status. The following error statuses can be returned: I$CKSM, I$CLOS, I$DSPC, I$DTYP, I$DUPE, I$EOF, I$FSPC, I$FTYP, IEOM, IIVDV, I$NONM, I$NORV, I$NRDY, I$RIB, I$WRIT.

The remainder of the IOCB and the contents of the data buffer (non-diskette device) and the sector buffer (diskette device) are indeterminate.

The IOCB is affected in the following manner if no errors occurred:

IOCSTA = 0.

IOCDTT has the "O" bit set to zero (file open). The "T" bit will have been set to one if a new file had to be created on the diskette. The "IO" bits are indeterminate. The remainder of IOCDTT is not changed.

IOCDBP is indeterminate.

IOCNAM is unchanged if the device type is not DK. If the device type is DK, then IOCNAM will have been replaced with the four entries IOCMLS, IOCSDW, IOCSLS and IOCLSN.

IOCMLS contains the value $FFFF if the device type is DK.

IOCSDW contains the first SDW from the file's RIB if the device type is DK.

IOCSLS contains the value $FFFF if the device type is DK.

IOCLSN contains the value zero if the device type is DK.

IOCSUF is unchanged if the device type is not DK. If the device type is DK, then IOCSUF will have been replaced with the IOCEOF entry.

IOCEOF contains the logical sector number of the logical end-of-file if the device type is DK.

IOCRIB contains the physical sector number of the file's RIB if the device type is DK.

IOCDEN contains the file's directory entry number if the device type is DK.

IOCFDF contains the FDF field from the directory entry or the FDR (if open mode is input or update to existing file). Otherwise, the IOCFDF field contains its initial value; however, if the initial FMT bits contained a "1", FMT will have been changed to either a "3" or a "7" as described in section 25.3.1.16.

IOCSBP contains the value of zero if the device type is DK.

IOCSBI contains the value in IOCSBE.

The remainder of the IOCB is unchanged.

The contents of the data buffer (non-diskette device) and the sector buffer (diskette device) are indeterminate.

25.3.4 Input a record -- .GETRC

The .GETRC function reads a record from an opened file or device into a data buffer. The specific processing performed by .GETRC depends on the FMT bits of IOCFDF and on the device type. The record input function will process three file formats: binary record (FMT = 3), ASCII record (FMT = 5), and ASCII-converted-binary record (FMT = 7).

Binary records will be stripped of their record header (see section 24.3), their byte count, and their checksums. Only the data characters between the byte count and checksum fields will be returned. If characters are encountered after the checksum field of one binary record but before the header field of the next record, they will be ignored.

ASCII records will be stripped of null characters, line feeds, rubouts, and the device control characters DC1-DC4. When reading records from the diskette, compressed spaces (bytes with bit 7 set to 1) will be automatically expanded into the appropriate number of spaces before being placed into the data buffer. This automatic space expansion occurs regardless of the compression bit in IOCFDF (bit "N"). A carriage return will be the last data character in the data buffer.

ASCII-converted-binary records are handled similarly to binary records; however, the conversion of two seven-bit data bytes into a single eight-bit data byte is automatically performed.

The .GETRC function treats the system console (CN) in a slightly different way than it does other devices, since the input from this device is usually in an interactive mode with the operator. In addition to the normal ASCII record processing, .GETRC will perform the following. First, if the first byte of the IOCSUF field contains a displayable character in the range $20-$5F, it will be automatically displayed as an input prompt each time the .GETRC function is invoked. Next, the special keyboard characters rubout ($7F), cancel (CTL-X, $18), and EOT (CTL-D, $04) will cause the standard MDOS keyboard functions to be performed (section 2.5). Rubout will delete the previously entered character, cancel will delete the entire input line entered thus far, and EOT will cause the input line entered thus far to be redisplayed on a new line of the console. Lastly, the carriage return character will cause a carriage return, line feed, and null sequence to be sent to the console. All other data characters will be echoed back to the console display mechanism as they are entered from the keyboard. This function is the same as for the .KEYIN system function described earlier in this chapter (section 25.2.1).

ENTRY PARAMETERS:

X = The address of an IOCB which has been properly reserved and opened (i.e., no errors occurred) via the .RESRV and .OPEN functions, respectively.

IOCDTT must have the "S" bit set to zero (record I/O). The mode flag (bit "M") must specify either the input or the update modes as configured prior to opening the file.

IOCDBS must contain the address where the first byte of the record is to be stored.

IOCDBE must contain the address where the last byte of the maximum size record is to be stored. The buffer described by IOCDBS and IOCDBE must be large enough to accommodate the largest possible record that may be encountered in the file.

IOCSUF may be configured by the calling program to contain a displayable character in its first byte if the input device is the system console. In this case, the character will be shown on the console as an input prompt each time the .GETRC function is invoked. IOCSUF must not be changed after opening a file when other devices are used.

IOCFDF must have been configured for a valid file format on a previous .OPEN call (FMT = 3, 5, or 7).

EXIT CONDITIONS:

A is indeterminate.

B = The contents of the IOCSTA entry. If no errors occurred, B will be zero. A non-zero value indicates that an error occurred.

X is unchanged.

C = O and Z = 1 if no errors occurred (B = O). The remainder of CC is indeterminate.

C = 1 and Z = 0 if an error occurred (B not zero). The remainder of CC is indeterminate.

The IOCB is affected in the following manner if an error occurred:

IOCSTA contains the error status. The following error statuses can be returned: I$BUFO, I$CKSM, I$CLOS, I$DTYP, IEOF, IFTYP, IEOM, INRDY, I$RANG, I$SECB.

IOCDBP is indeterminate.

IOCMLS, IOCSDW, IOCSLS, IOCLSN, IOCSBP, and IOCSBI are indeterminate if the device type is DK. Otherwise, IOCNAM, IOCSBP, and IOCSBI are unchanged.

The remainder of the IOCB is unchanged.

If a buffer overflow error occurred (IOCSTA = I$BUFO), then the last data character of the record (carriage return) will be the last character of the buffer. The first "n" characters (n being the size of the data buffer minus one) of the record are intact. Otherwise, the contents of the data buffer are indeterminate.

If the device type is DK, then the contents of the sector buffer are indeterminate.

The IOCB is affected in the following manner if no errors occurred:

IOCSTA = 0.

IOCDTT has the I/O transfer flag set to indicate input (IO = 10). The remainder of IOCDTT is unchanged.

IOCDBP contains the address of the last character read into the input buffer.

IOCMLS, IOCSDW, IOCSLS, IOCLSN, IOCEOF, IOCSBP, and IOCSBI contain the system-maintained parameters as described in section 25.3.1 if the device type is DK. They reflect the current diskette file pointers. IOCNAM, IOCSUF, IOCSBP, and IOCSBI are unchanged if the device is not DK.

The remainder of the IOCB is unchanged.

The data buffer contains the record.

The sector buffer contains data from the logical sectors read. This number is given by IOCLSN minus the valid buffer size in sectors ((IOCSBI-IOCSBS+1)/128) if the device is DK.

25.3.5 Output a record -- .PUTRC

The .PUTRC function writes a record from a data buffer to an opened file or device. The specific processing performed by .PUTRC depends on the FMT bits of IOCFDF and on the device type. The record output function will process three file formats: binary record (FMT = 3), ASCII record (FMT = 5), and ASCII-converted-binary record (FMT = 7).

Binary records will be automatically supplied with their record header (see section 24.3), a byte count, and a checksum. In addition, a terminating carriage return is supplied by the .PUTRC function. If the output device is a non-diskette device, the terminating carriage return will actually be a carriage return, line feed, null sequence. None of these automatically supplied fields are present in the data buffer described by the IOCB.

ASCII records will be automatically space compressed if the output device is diskette and if the "N" bit of IOCFDF is zero. Otherwise, spaces will not be compressed. A carriage return character will be automatically written to the output device after the last data character has been sent unless the last data character happens to be a carriage return. All carriage returns, those encountered within the data buffer as well as the automatically supplied terminating one, are converted into a carriage return, line feed, null sequence when being written to a non-diskette device. The line feed and null characters generated from embedded carriage returns will not be written to the diskette.

ASCII-converted-binary records are handled similarly to binary records; however, the conversion of one eight-bit data byte into two seven-bit data bytes is automatically performed.

If a record is being written into a diskette file, additional space may be allocated to accommodate the increased space requirements of the file. The file allocation is done automatically. The amount of secondary allocation will depend on the available file space; however, an attempt will be made to allocate the default number of clusters. If less space is available than the default, then the largest available block will be allocated.

ENTRY PARAMETERS:

X = The address of an IOCB which has been properly reserved and opened (i.e., no errors occurred) via the .RESRV and .OPEN functions, respectively.

IOCDTT must have the "S" bit set to zero (record I/O). The mode flag (bit "M") must specify either the output or the update modes as configured prior to opening the file.

IOCDBS must contain the address of the first byte of the record that is to be written.

IOCDBE must contain the address of the last byte of the record that is to be written. A terminating carriage return is not required in the data buffer.

IOCFDF must have been configured for a valid file format during the previous .OPEN call (FMT = 3, 5, or 7). The non-compressed space bit (bit "N") determines whether or not spaces are compressed (only applies to ASCII files being written to diskette).

EXIT CONDITIONS:

A is indeterminate.

B = The contents of the IOCSTA entry. If no errors occurred, B will be zero. A non-zero value indicates that an error occurred.

X is unchanged.

C = 0 and Z = 1 if no errors occurred (B = 0). The remainder of CC is indeterminate.

C = 1 and Z = 0 if an error occurred (B not zero). The remainder of CC is indeterminate.

The IOCB is affected in the following manner if an error occurred:

IOCSTA contains the error status. The following error statuses can be returned: I$CLOS, I$DTYP, I$FTYP, I$NRDY, I$RECL, I$RANG, I$SECB, I$RIB, I$FSPC, I$SSPC.

IOCDBP is indeterminate.

IOCMLS, IOCSDW, IOCSLS, IOCLSN, IOCEOF, IOCSBP, and IOCSBI are indeterminate if the device type is DK. IOCNAM, IOCSUF, IOCSBP, and IOCSBI are unchanged otherwise.

The remainder of the IOCB is unchanged.

The contents of the data buffer are unchanged.

The contents of the sector buffer are indeterminate.

The IOCB is affected in the following manner if no errors occurred:

IOCSTA = 0.

IOCDTT has the I/O transfer flag set to indicate output (IO = 01). If additional file space was allocated, the truncate flag (T) is set to one if it was not already one prior to the output transfer. The remainder of IOCDTT is unchanged.

IOCDBP contains the address of the last character in the data buffer (same as IOCDBE).

IOCMLS, IOCSDW, IOCSLS, IOCLSN, IOCEOF, IOCSBP, and IOCSBI contain the system-maintained parameters as described in section 25.3.1 if the device is DK. They reflect the current diskette file pointers. If .PUTRC has been called for the first time, and if IOCMLS contained the value $FFFF upon entry, IOCMLS will contain the value $0000 upon exiting the function. In this way, the file will not be deleted upon closing, even if only a single record has been written into the sector buffer.

IOCNAM, IOCSUF, IOCSBP, and IOCSBI are unchanged if the device is not DK.

The remainder of the IOCB is unchanged.

The contents of the data buffer are unchanged.

The sector buffer contains the data that are going to be written to diskette starting with the logical sector specified by IOCLSN. The sector buffer is not cleared after having been written. Thus, the parts of the sector buffer not affected by the .PUTRC call will still contain the data from the buffer last written.

25.3.6 Close a file -- .CLOSE

The .CLOSE function is used to signify completion of all I/O transfers to a file or device in the current open mode. Data cannot be transferred between the file (or device) and the calling program after the .CLOSE function has been invoked. The specific function performed by .CLOSE depends on the mode flag (M of IOCDTT), the I/O transfer flag (IO of IOCDTT), and the device type.

If the IOCB has been opened in the input mode (M = 01 of IOCDTT), then the .CLOSE function will simply change the IOCB to indicate that the file is closed.

If the IOCB has been opened in the output mode (M = 10 of IOCDTT), then .CLOSE will perform the following. For a device type of DK, .CLOSE will zero-fill any unused portions of the unwritten sector buffer to a sector boundary before writing the buffer to the diskette (only if record I/O is being performed; logical sector I/O will not cause the last sector buffer to be changed or written). All space that has been newly allocated but not written into (those logical sectors greater than IOCMLS) will normally be deallocated on a cluster boundary and returned to the free space pool (assumes that the truncate flag and IOCMLS have not been changed by the calling program). The end-of-file LSN will be adjusted in the RIB. If the device is not DK, then .CLOSE will cause an end-of-file record to be written to the device (file format mode only). In the non-file format mode, .CLOSE will only write an end-of-file record to the device if it is a file-type device (e.g., an end-of-file is written to CP but not to LP or CN). File-type devices are those which use a medium that can be re-read later.

If the IOCB has been opened in the update modes (M = 00 or 11 of IOCDTT), then .CLOSE will perform the same functions as in the input or the output mode depending on the last I/O transfer type. The .GETRC and .GETLS functions will set IO of IOCDTT to indicate an input transfer, while the .PUTRC and .PUTLS functions will set IO of IOCDTT to indicate an output transfer. In the latter case, space is only deallocated if the truncate flag (T of IOCDTT) is set to one (done automatically when new space is allocated, or done by user to indicate file shortening or updating of end-of-file pointer in RIB).

ENTRY PARAMETERS: X = The address of an IOCB which has been properly reserved and opened (i.e., no errors occurred) via the .RESRV and .OPEN functions, respectively.

Normally, no additional parameters are required; however, when dealing with diskette files in the update mode (M = 00 or 11 of IOCDTT), the truncate flag (T of IOCDTT) and the maximum referenced logical sector number (IOCMLS) can be configured by the calling program. Since the update modes only set the truncate flag to one if a new file is created during the open process or if additional space is allocated during the output process (file grows), space will not be deallocated or the end-of-file pointer updated from existing files unless the truncate flag and IOCMLS are explicitly set up by the calling program. When IOCMLS is set to the value $FFFF (value set up during .OPEN), then the file will have its directory entry deleted in addition to having all of its space deallocated (if truncate flag is set to one when .CLOSE is invoked).

IOCDBS and IOCDBE must describe a valid data buffer when dealing with non-diskette devices (output only) since an end-of-file record is written (file-type devices only).

EXIT CONDITIONS: A is indeterminate.

B = The contents of the IOCSTA entry. If no errors occurred, B will be zero. A non-zero value indicates that an error occurred.

X is unchanged.

C = 0 and Z = 1 if no errors occurred (B = 0). The remainder of CC is indeterminate.

C = 1 and Z = 0 if an error occurred (B not zero). The remainder of CC is indeterminate.

The IOCB is affected in the following manner if an error occurred:

IOCSTA contains the error status. The following error statuses can be returned: I$CLOS, I$DELT, I$IDEN, I$RANG, I$SECB, I$FSPC, I$SSPC, I$RIB, I$DEAL.

The remainder of the IOCB and the contents of the data buffer and the sector buffer are indeterminate.

The IOCB is affected in the following manner if no errors occurred:

IOCSTA = 0.

IOCDTT has the "O" bit set to one (file closed). The remainder of the IOCDTT is unchanged.

IOCRIB will be zero if the file was deleted from the diskette. Otherwise it will be unchanged.

IOCEOF will contain the LSN of the logical end-of-file if the device type is DK. IOCEOF will be unchanged if the truncate flag was zero upon entry.

The remainder of the IOCB is unchanged.

The contents of the data buffer and the sector buffer are indeterminate.

25.3.7 Release a device -- .RELES

The .RELES function breaks the link between the appropriate controller descriptor block and the calling program's IOCB. The .RELES function should be the last I/O function called after all I/O has been completed.

ENTRY PARAMETERS: X = The address of of an IOCB which has been properly reserved (i.e., no errors occurred) via the .RESRV function. If the .OPEN function has been invoked at any time after reserving the IOCB, the file (or device) must first be closed via the .CLOSE function before the IOCB can be released.

EXIT CONDITIONS: A is indeterminate.

B = The contents of the IOCSTA entry. If no errors occurred, B will be zero. A non-zero value indicates that an error occurred.

X is unchanged.

C = 0 and Z = 1 if no errors occurred (B = 0). The remainder of CC is indeterminate.

C = 1 and Z = 0 if an error occurred (B not zero). The remainder of CC is indeterminate.

The IOCB is affected in the following manner if an error occurred:

IOCSTA contains the error status. The following error statuses can be returned: I$NORV, I$CLOS.

The remainder of the IOCB and the contents of the data buffer and the sector buffer are unchanged.

The IOCB is affected in the following manner if no errors occurred:

IOCSTA = 0.

IOCGDW = 0.

IOCLUN has the "R" bit set to zero (IOCB released). The remainder of IOCLUN is unchanged.

The remainder of the IOCB and the contents of the data buffer and the sector buffer are unchanged.

25.3.8 Example of device independent I/O

The following example uses the device independent I/O functions described thus far. The IOCB shown below is used in the example as the control block for writing to a diskette file. The initial values set up in this IOCB are typical for most output operations. A four-sector buffer is used to allow a maximum of four sectors to be written to the diskette each time it is accessed. The larger a sector buffer is, the fewer will be the number of diskette accesses. The logical unit number, file name, and suffix are going to be initialized from an operator-supplied parameter on the command line. The system symbols from the MDOS equate file are used throughout this example.

```
     OUTPUT EQU     *           . START OF OUTPUT IOCB
            FCB     0           . IOCSTA
            FCB     DT$OPO+DT$CLS . IOCDTT
            FDB     0           . IOCDBP
            FDB     RBUFF       . IOCDBS
            FDB     RBUFFE      . IOCDBE
            FCC     2,DK        . IOCGDW
            FCB     '0+0        . IOCLUN -- DEFAULT = 0
            FCC     8,           . IOCNAM
            FCC     2,SA        . IOCSUF -- DEFAULT = SA
            FDB     0           . IOCRIB
            FDB     FD$FMA!<8 . IOCFDF --  ASCII
            FDB     0           . RESERVED
            FDB     0           . IOCDEN
            FDB     0           . IOCSBP
            FDB     SCTBUF      . IOCSBS
            FDB     SCTBUF+(SC$SIZ*4)-1 . IOCSBE
            FDB     0           . IOCSBI
    *
    SCTBUF  BSZ     SC$SIZ*4 . SECTOR BUFFER (4 SECTORS)
    RBUFF   BSZ     80          . RECORD BUFFER
    RBUFFE  EQU     *-1         .
```

The code that is shown below performs the following functions. First, a file name specification which has been entered on the MDOS command line is extracted from the command line buffer and placed into the IOCB. This is accomplished with the .PFNAM system function described in Chapter 27. Then, the IOCB is reserved and opened. Next, an input prompt is displayed on the system console and an line of text is accepted from the keyboard. If the entered line consisted of only a carriage return, the IOCB is closed, released, and control returned to the MDOS command interpreter (via the function .MDENT). Otherwise, the entered line is written into the diskette file. The input process is repeated until only a carriage return is entered.

The error message function, .MDERR, is used to display standard error messages if an invalid file name specification is entered, if a file name is missing, or if one of the I/O functions returns an error condition (e.g., if the file name already exists in the directory, or if insufficient diskette space is available). The function .ADBX is used to add the contents of the B accumulator to the index register. Both of these functions are discussed in detail in Chapter 27.

In this example, the assumption is made that the program is invoked from the MDOS command line. Thus, it must be origined to load above location $1FFF. The stack pointer is automatically initialized through the loading process to point to the last-loaded program location. The stack area has been set up so that the default value of the stack pointer can be used without having to execute a load stack pointer instruction.

```
*
* DEFINE SOME WORKING STORAGE
*
PFNPAK FDB    0,0        . PROCESS FILE NAME PACKET
PROMPT FCB    ':,EOT     . INPUT PROMPT
*
* EXTRACT THE FILE NAME FROM THE COMMAND LINE
*
START  LDX    #OUTPUT+IOCLUN .
       STX    PFNPAK+2 . DESTINATION OF FILE NAME
       LDX    CBUFP$   . POINTER INTO CMD BUFFER
       STX    PFNPAK   . SOURCE OF FILE NAME
       LDX    #PFNPAK  .
       SCALL  .PFNAM   . FORMAT STANDARD FILE NAME
       TSTB            . CHECK FOR ERRORS
       BEQ    STARTA   . EQ => GOOD NAME
       ASLB            .
       BCS    ERR1     . CS => NAME MISSING
       LDAB   #7       . ILLEGAL NAME MSG NUMBER
       BRA    ERR2     .
*
ERR1   LDAB   #5       . NAME REQUIRED MSG NUMBER
ERR2   SCALL  .MDERR   . DISPLAY STD ERROR MSG
       BRA    MDOS     . EXIT THE PROGRAM
*
ERR3   CLRB            . I/O ERR MSG NUMBER; DECODED
       BRA    ERR2     . FROM IOCSTA
*
* OPEN AND RESERVE THE IOCB -- CREATE THE OUTPUT FILE
*
```

```
STARTA LDX     #OUTPUT  .
       SCALL   .RESRV   .
       BCS     ERR3     . CS => ERROR
       SCALL   .OPEN    .
       BCS     ERR3     . CS => ERROR
*
* GET LINE FROM CONSOLE
*
LOOP   LDX     #PROMPT  . DISPLAY THE INPUT PROMPT, NO CR/LF
       SCALL   .DSPLZ   .
       LDX     #RBUFF   . GET THE INPUT LINE
       LDAB    #RBUFFE-RBUFF
       SCALL   .KEYIN   .
       LDAA    X        . GET 1ST CHAR IN BUFFER
       CMPA    #CR      . CHECK FOR TERMINATOR
       BEQ     EXIT     . EQ => THIS IS THE TERMINATING LINE
       STX     OUTPUT+IOCDBS . SETUP START RECORD POINTER
       DEX              . CALC END OF RECORD BUFFER
       SCALL   .ADBX    . B = NUMB CHARS INPUT
       STX     OUTPUT+IOCDBE . SETUP END RECORD POINTER
       LDX     #OUTPUT  .
       SCALL   .PUTRC   . WRITE THE RECORD
       BCC     LOOP     . CC => NO ERRORS
       BRA     ERR3     .
*
* CLOSE AND RELEASE THE IOCB, RETURN TO MDOS
*
EXIT   LDX     #OUTPUT  . POINT TO THE IOCB
       SCALL   .CLOSE   .
       BCS     ERR3     . CS => ERROR
       SCALL   .RELES   .
       BCS     ERR3     . CS => ERROR
MDOS   SCALL   .MDENT   . RETURN TO MDOS
*
* LEAVE SOME ROOM FOR STACK
*
       BSZ     80       . STACK SET HERE BY LOAD
       END     START    .
```

25.3.9 Specialized diskette I/O functions

Three additional I/O functions exist that also use the IOCB as a parameter table; however, they are dependent on the device type being DK. An error will be returned if any other device type is specified.

25.3.9.1 Input logical sectors -- .GETLS

The .GETLS function reads one or more logical sectors from an opened file into a sector buffer.

ENTRY PARAMETERS: X = The address of an IOCB which has been properly reserved and opened (i.e., no errors occurred) via the .RESRV and .OPEN functions, respectively.

IOCDTT must have the "S" bit set to one (sector I/O). The mode flag (bit "M") must specify either the input or the update modes as configured prior to opening the file.

IOCLSN must contain the logical sector number that is to be read. The actual number of sectors read depends on the size of the sector buffer (see below). The data sectors of the file begin with logical sector zero. If the RIB is to be accessed via the .GETLS function, then IOCLSN must contain the value $FFFF.

IOCSBS must contain the starting address of a sector buffer. The sector buffer must be an integral number of sectors in size (see section 25.3.1.20). This buffer does not necessarily have to be the same one used to open the file. The sector buffer can be in a different location for each .GETLS call; however, if the sector buffer is to be moved after a file has been opened, then IOCSBS, IOCSBE, and IOCSBI must be changed by the calling program.

IOCSBE must contain the address of the last byte of a sector buffer. The sector buffer must be an integral number of sectors in size (see section 25.3.1.20). The buffer described by IOCSBS and IOCSBE indicates the maximum number of sectors that can be processed starting with the logical sector whose number is in IOCLSN.

EXIT CONDITIONS: A is indeterminate.

B = The contents of the IOCSTA entry. If no errors occurred, B will be zero. A non-zero value indicates that an error occurred.

X is unchanged.

C = 0 and Z = 1 if no errors occurred (B = 0). The remainder of CC is indeterminate.

C = 1 and Z = 0 if an error occurred (B not zero). The remainder of CC is indeterminate.

The IOCB is affected in the following manner if an error occurred:

IOCSTA contains the error status. The following error statuses can be returned: I$CLOS, I$DTYP, IEOF, ISECB, I$RANG.

IOCMLS, IOCSDW, IOCSLS, IOCLSN, IOCSBP, and IOCSBI are indeterminate.

The remainder of the IOCB is unchanged.

The contents of the sector buffer are indeterminate.

The IOCB is affected in the following manner if no errors occurred:

IOCSTA = 0.

IOCMLS, IOCSDW, and IOCSLS contain the system-maintained parameters as described in section 25.3.1. They reflect the current diskette file pointers.

IOCLSN has been incremented by the number of sectors read into the buffer ((IOCSBI-IOCSBS+1)/128).

IOCSBP contains the starting address of the sector buffer (the same as IOCSBS).

IOCSBI contains the address of the last valid data byte in the sector buffer. If only a partial segment was read into the buffer, IOCSBI will not be the same as IOCSBE (maximum end of buffer). The following relationship should be used to calculate the number of sectors read:

```
IOCSBI-IOCSBS+1
--------------- = # SECTORS READ
```

The remainder of the IOCB is unchanged.

The sector buffer contains the data from the sectors read beginning with the logical sector whose number was in IOCLSN.

25.3.9.2 Output logical sectors -- .PUTLS

The .PUTLS function writes one or more logical sectors from a sector buffer to an opened file. Additional space may be allocated to the file to accommodate the increased space requirements. The space allocation is performed automatically. The amount of secondary allocation will depend on the available space; however, an attempt will be made to allocate the default number of clusters. If less space is available than the default, then the largest available block will be allocated.

ENTRY PARAMETERS: X = The address of an IOCB which has been properly reserved and opened (i.e., no errors occurred) via the .RESRV and .OPEN functions, respectively.

IOCDTT must have the "S" bit set to one (sector I/O). The mode flag (bit "M") must specify either the output or the update modes as configured prior to opening the file.

IOCLSN must contain the logical sector number that is to be written into. The actual number of sectors written depends on the size of the sector buffer (see below). The data sectors of the file begin with logical sector zero. If the RIB is to be accessed via the .PUTLS function, then IOCLSN must contain the value $FFFF.

IOCSBS must contain the starting address of a sector buffer containing the data to be written. The sector buffer must be an integral number of sectors in size (see section 25.3.1.20). This buffer does not necessarily have to be the same one used to open the file. The sector buffer can be in a different location for each .PUTLS call; however, if the sector buffer is to be moved after a file has been opened, then IOCSBS, IOCSBE, and IOCSBI must be changed by the calling program.

IOCSBE is not used during the .PUTLS function; however, it should not have been changed since the file was opened (with restrictions mentioned above for IOCSBS).

IOCSBI must contain the address of the last data byte to be written from the sector buffer. The sector buffer, as described by IOCSBS and IOCSBI, must be an integral number of sectors in size (see section 25.3.1.20).

EXIT CONDITIONS: A is indeterminate.

B = The contents of the IOCSTA entry. If no errors occurred, B will be zero. A non-zero value indicates that an error occurred.

X is unchanged.

C = 0 and Z = 1 if no errors occurred (B = 0). The remainder of CC is indeterminate.

C = 1 and Z = 0 if an error occurred (B not zero). The remainder of CC is indeterminate.

The IOCB is affected in the following manner if an error occurred:

IOCSTA contains the error status. The following error statuses can be returned: I$CLOS, I$DYTP, I$SECB, I$RANG, IRIB, IFSPC, I$SSPC.

IOCMLS, IOCSDW, IOCSLS, IOCLSN, IOCEOF, IOCSBP, and IOCSBI are indeterminate.

The remainder of the IOCB and the contents of the sector buffer are unchanged.

The IOCB is affected in the following manner if no errors occurred:

IOCSTA = 0.

IOCMLS, IOCSDW, and IOCSLS contain the system-maintained parameters as described in section 25.3.1. They reflect the current diskette file pointers.

IOCLSN has been incremented by the number of sectors written ((IOCSBI-IOCSBS+1)/128). If the sector specified by the entry value of IOCLSN or any of the sectors written from the buffer was outside of the range of the file's allocated space, additional file space will have been allocated (if available).

IOCEOF contains the logical sector number of the logical end-of-file. If additional file space was allocated, IOCEOF will contain the new end-of-file LSN. IOCEOF is unchanged otherwise.

IOCSBP contains the starting address of the sector buffer (the same as IOCSBS).

The remainder of IOCB and the contents of the sector buffer are unchanged.

25.3.9.3 Rewind file -- .REWND

The .REWND function resets the pointers of the IOCB so that subsequent I/O functions will access the diskette file as if it had just been opened, i.e., from the beginning. Only files that have been opened in the update or input mode can be rewound. Files opened in the output mode will cause the .REWND function to return an error condition.

ENTRY PARAMETERS: X = The address of an IOCB which has been properly reserved and opened (i.e., no errors occurred) via the .RESRV and .OPEN functions, respectively.

IOCDTT can have the "S" bit set to indicate either record or sector I/O. The mode flag (bit "M") must specify either input or update modes as configured prior to opening the file.

IOCSBS must contain the starting address of a sector buffer. The sector buffer must be an integral number of sectors in size (see section 25.3.1.20). This buffer does not necessarily have to be the same one used to open the file; however, if the sector buffer is to be moved after a file has been opened, then IOCSBS, IOCSBE, and IOCSBI must be changed by the calling program.

IOCSBE must contain the address of the last byte of a sector buffer. The sector buffer must be an integral number of sectors in size (see section 25.3.1.20).

EXIT CONDITIONS:

A is indeterminate.

B = The contents of the IOCSTA entry. If no errors occurred, B will be zero. A non-zero value indicates that an error occurred.

X is unchanged.

C = 0 and Z = 1 if no errors occurred (B = 0). The remainder of CC is indeterminate.

C = 1 and Z = 0 if an error occurred (B not zero). The remainder of CC is indeterminate.

The IOCB is affected in the following manner if an error occurred:

IOCSTA contains the error status. The same error statuses can be returned as those that can be returned by the .OPEN and .CLOSE functions.

IOCMLS, IOCSDW, IOCSLS, IOCLSN, IOCEOF, IOCSBP, and IOCSBI are indeterminate.

The remainder of the IOCB is unchanged.

The contents of the sector buffer are indeterminate.

The IOCB is affected in the following manner if no errors occurred:

IOCSTA = 0.

IOCDTT has the "T" bit set to zero. If

the bit was set to one before the .REWND call was issued, space may have been deallocated from the file and the end-of-file pointer in the RIB updated. The remainder of IOCDTT is unchanged.

IOCMLS contains the value $FFFF.

IOCSDW contains the first SDW from the file's RIB.

IOCSLS contains the value $FFFF.

IOCLSN contains the value zero.

IOCEOF contains the LSN of the logical end-of-file from the file's RIB.

IOCSBP contains the value zero.

IOCSBI contains the value in IOCSBE.

The remainder of the IOCB is unchanged.

The contents of the sector buffer are indeterminate.

The effect of rewinding a file is the same as if a .CLOSE and a .OPEN function were performed; however, the .REWND function reopens the file without having the calling program re-specify the file's name and suffix. Thus, when the file is rewound, the same space deallocation and end-of-file pointer considerations take effect as if the file were closed. Since the truncate flag is set to zero after the .REWND call (opening an existing file), the calling program may have to reset the flag if space is to be deallocated or the end-of-file pointer updated upon calling the subsequent .CLOSE function.

25.3.9.4 Example of logical sector I/O

The following example uses the logical sector I/O functions. The IOCB shown below is used in the example as the control block for reading from and writing to a diskette file. The initial values set up in this IOCB are similar to those in the example given in section 25.3.8; however, the sector I/O and update modes are specified in the IOCDTT entry. Only a single sector is used for a sector buffer to make the management of logical sectors easier (eliminates calculation of the number of sectors read or written). The logical unit number, file name, and suffix are going to be initialized by an operator-supplied parameter obtained from the command line. The system symbols from the MDOS equate file are used throughout this example.

```
TEXFIL EQU    *          . START OF TEXFIL IOCB
       FCB    0          . IOCSTA
       FCB    DT$OPU+DT$SIO+DT$CLS .  IOCDTT
       FDB    0          . IOCDBP
       FDB    0          . IOCDBS
       FDB    0          . IOCDBE
       FCC    2,DK       . IOCGDW
       FCB    '0+0       . IOCLUN -- DEFAULT = 0
       FCC    8,          . IOCNAM
       FCC    2,SA       . IOCSUF -- DEFAULT = SA
       FDB    0          . IOCRIB
       FDB    FD$FMA!<8 .  IOCFDF --  ASCII
       FDB    0          . RESERVED
       FDB    0          . IOCDEN
       FDB    0          . IOCSBP
       FDB    SECBUF     . IOCSBS
       FDB    SECBUF+SC$SIZ-1 .  IOCSBE
       FDB    0          . IOCSBI
*
SECBUF BSZ    SC$SIZ     . SECTOR BUFFER
```

The code that is shown below performs the following functions. First, a file name specification which must have been entered on the MDOS command line is extracted from the command line buffer and placed into the IOCB. This is accomplished with the .PFNAM system function described in Chapter 27. Then, the IOCB is reserved and opened. Next, one sector is read from the file and all upper case alphabetic characters are converted into lower case characters. A special check is made for punctuation marks (period, exclamation point, and question mark) so that the first alphabetic character following such punctuation is left upper case. After all bytes within the sector have been processed, they are rewritten into the same sector from which they were read. The process is repeated until an end-of-file condition is encountered. Finally, after the file is closed and released, control is returned to the MDOS command interpreter via the function .MDENT. Since the file does not expand, it was opened in the update mode so that sectors could be both read from and written to the file. It should be noted that the logical sector number should be decremented before a sector is written back from where it was read.

The error message function, .MDERR, is used to display standard error messages if an invalid file name specification is entered, if a file name is missing, of if one of the I/O functions returns an error condition. The system function .ALPHA is used to test for alphabetic characters. Both of these functions are discussed in detail in Chapter 27.

In this example, the assumption is made that the program is invoked from the MDOS command line. Thus, it must be origined to load above location $1FFF. The stack pointer is automatically initialized through the loading process to point to the last-loaded program location. The stack area has been set up so that the default value of the stack pointer can be used without having to execute a load stack pointer instruction.

```
*
* DEFINE SOME WORKING STORAGE
*
PFNPAK FDB    0,0        . PROCESS FILE NAME PACKET
UCFLG  FCB    0          . UPPER CASE CONVERSION FLAG
*
*
* EXTRACT NAME FROM COMMAND LINE
*
START  LDX    #TEXFIL+IOCLUN . DESTINATION OF NAME
       STX    PFNPAK+2 .
       LDX    CBUFP$     . SOURCE OF NAME
       STX    PFNPAK     .
       LDX    #PFNPAK    .
       SCALL  .PFNAM     . EXTRACT FILE NAME
       TSTB              . CHECK FOR VALID NAME
       BEQ    STARTA     . EQ => GOOD
       ASLB              .
       BCS    ERR1       . CS => NAME MISSING
       LDAB   #7         . ILLEGAL NAME MSG NUMBER
       BRA    ERR2       .
*
ERR1   LDAB   #5         . NAME REQUIRED MSG NUMBER
ERR2   SCALL  .MDERR     .
       BRA    EXIT       . DISPLAY ERROR, THEN EXIT PROGRAM
*
ERR3   CLRB              . I/O FUNCTION ERROR MSG NUMBER
       BRA    ERR2       .
*
* RESERVE AND OPEN THE IOCB
*
STARTA LDX    #TEXFIL    .
       SCALL  .RESRV     .
       BCS    ERR3       . CS => ERROR
       SCALL  .OPEN      .
       BCS    ERR3       . CS => ERROR
*
* READ A LOGICAL SECTOR INTO BUFFER
*
```

```
LOOP1  LDX    #TEXFIL   .
       SCALL  .GETLS    .
       BCS    EOF       . CS => ERROR, POSSIBLE END OF FILE
*
* CONVERT DATA WITHIN SECTOR BUFFER
*
LOOP2  LDX    TEXFIL+IOCSBP .
       LDAA   X         . GET CHAR FROM BUFFER
       BSR    CONVRT    .
       STAA   X         . PUT CHARACTER BACK
       INX              . INCREMENT BUFFER POINTER
       STX    TEXFIL+IOCSBP . SAVE POINTER
       CPX    TEXFIL+IOCSBE . CHECK FOR LAST CHARACTER
       BNE    LOOP2     . NE => MORE DATA TO CONVERT
       LDAA   X         . CONVERT LAST CHARACTER
       BSR    CONVRT    .
       STAA   X         .
*
* WRITE LOGICAL SECTOR BACK INTO FILE
*
       LDX    TEXFIL+IOCLSN . PICK UP LSN
       DEX              . POINT BACK TO LAST READ SECTOR
       STX    TEXFIL+IOCLSN .
       LDX    #TEXFIL   .
       SCALL  .PUTLS    . WRITE THE SECTOR BACK
       BCS    ERR3      . CS => ERROR
       BRA    LOOP1     . READ NEXT SECTOR AND CONTINUE
*
* END-OF-FILE DETECTED ON INPUT
*
EOF    CMPB   #I$EOF    .
       BNE    ERR3      . NE => I/O ERROR
       LDX    #TEXFIL   .
       SCALL  .CLOSE    .
       BCS    ERR3      . CS => ERROR
       SCALL  .RELES    .
       BCS    ERR3      . CS => ERROR
EXIT   SCALL  .MDENT    . RETURN TO MDOS COMMAND INTERPRETER
*
* CONVERT ALL UPPER CASE ALPHABETIC CHARACTERS TO LOWER
* CASE CHARACTERS.  FIRST ALPHABETIC
* CHARACTER FOLLOWING A PERIOD, EXCLAMATION POINT, OR
* QUESTION MARK IS NOT CHANGED.
*
CONVRT SCALL  .ALPHA    . CHECK FOR U/C ALPHABETIC
       BCS    CONTRM    .
       TST    UCFLG     .
       BNE    CONVEX    . NE => DON'T CONVERT
       ORAA   #SPACE    . CONVERT TO L/C
CONVEX CLR    UCFLG     . RESET FLAG TO CONVERT NEXT ALFA
CONEX2 RTS              .
*
CONTRM CMPA   #'.       . PERIOD
       BEQ    SETFLG    .
       CMPA   #'!      . EXCLAMATION
       BEQ    SETFLG   .
       CMPA   #'?      . QUESTION
       BNE    CONEX2   .
SETFLG INC    UCFLG    .
       BRA    CONEX2   .
*
* SAVE SOME ROOM FOR STACK
*
       BSZ    80       . STACK POINTER SET HERE BY LOAD
*
       END    START    .
```

25.3.10 Error handling

All of the I/O functions discussed in this section use the IOCB. The first entry of the IOCB will contain an error status upon returning from one of these functions. The calling program is responsible for processing these error conditions. If the error status is to be decoded and displayed as a message on the system console, the system error message function, .MDERR, can be used. This function is described in detail in Chapter 27; however, it should be noted here that a common mistake is made in calling the error message function with the value returned in the B accumulator by the I/O functions. It is true that this value is the same as IOCSTA's contents; however this is not the parameter that should be used to invoke the error message function. The error message function will decode the contents of IOCSTA only if it is called with the B accumulator equal to zero and with the X register pointing to the IOCB.

None of the I/O functions described here will return control to the calling program if a diskette controller error is detected (only applicable if the device type is DK). These errors are fatal errors and will cause the program to be aborted (i.e., the files will not be closed). An error message is displayed on the system console before giving control to MDOS.

In order to guarantee the integrity of data files (especially on the diskette), it cannot be stressed often enough that it is necessary for the calling program to check for an error condition after each I/O function call. A common mistake is to fail to check for errors after a file has been closed. Since output can still take place during the closing, data at the end of the file can be lost without being apparent. Another common mistake is to initialize the IOCB with the "O" flag of IOCDTT and the "R" flag of IOCLUN in the wrong sense. If the "R" flag is cleared before the IOCB is reserved, the "O" flag will be properly set by the functions themselves.

CHAPTER 26

26. INPUT/OUTPUT PROVISIONS FOR NON-SUPPORTED DEVICES

It is assumed that the reader is familiar with the device independent I/O functions described in section 25.3 before this chapter is read.

This chapter describes how the I/O functions interface with the hardware device and how a user can interface non-standard devices for use with the device independent I/O functions.

26.1 Device Dependent I/O

The device dependent I/O functions described in Chapter 25 for accessing the console and the line printer cannot be changed to access non-standard devices. These routines are a part of MDOS and its basic environment requirements; however, a user can construct his own device drivers that are accessed by his programs. If the standard MDOS commands are to utilize non-standard devices, the user should be using MODOS (OEM MDOS) which can be configured to work in that manner. The COPY command (Chapter 7) is an exception. It can load a user-defined device driver into memory to copy a file from that device to the diskette or from the diskette to that device.

26.2 Device Independent I/O

This section describes how the device independent I/O functions interface to the device drivers which, in turn, interface directly to the hardware device. This description applies to both standard and non-standard devices.

26.2.1 Controller Descriptor Block -- CDB

The Controller Descriptor Block, or CDB, is a table that describes a physical device and the types of input and output operations that can be performed by the device. Unlike the IOCB, the CDB is configured only once for each device. It is the memory location of the CDB that replaces the contents of the IOCGDW entry of an IOCB after the .RESRV function has been called. The format of the CDB is shown in the following diagram.

```
Byte
 !    7   6   5   4   3   2   1   0     <-- Bit position
 V   --------------------------------
 00  !                              !
     --         IOCB address       --   CDBIOC
 01  !                              !
     --------------------------------
 02  !         Device driver        !
     --           address          --   CDBSDA
 03  !                              !
     --------------------------------
 04  !                              !
     --       Hardware address     --   CDBHAD
 05  !                              !
     --------------------------------
 06  ! R ! O ! I ! F ! W ! S ! L ! D !  CDBDDF - Device descrip-
     --------------------------------            tor flags
 07  ! N !               ! B !      !   CDBVDT - Valid data
     --------------------------------            types
 08  !       Device dependent       !
     --          area              --   CDBDDA
 09  !                              !
     --------------------------------
 0A  !                              !
     --       Working storage      --   CDBWST
 0B  !                              !
     --------------------------------
```

CDB FLAG DESCRIPTION SUMMARY

Field	Name	Bit	Content
CDBDDF	R	7	Reservable device flag 0 => Not reservable 1 => Reservable
	O	6	Output device flag 0 => Cannot perform output 1 => Can perform output
	I	5	Input device flag 0 => Cannot perform input 1 => Can perform input
	F	4	File-type device flag 0 => Cannot open/close files 1 => Can open/close files
	W	3	Rewindable device flag 0 => Cannot rewind files 1 => Can rewind files
	S	2	System console flag 0 => Not system console device 1 => System console device
	L	1	Logical sector I/O flag 0 => Cannot perform logical sector I/O 1 => Can perform logical sector I/O
	D	0	Default binary record format flag 0 => Binary record is default binary format 1 => ASCII-converted-binary record is default binary format
CDBVDT	N	7	Non-file format flag 0 => Non-file format mode is invalid 1 => Non-file format mode is valid
	-	3-6	Not used (=0)
	B	2	Binary I/O flag 0 => Eight-bit data is invalid 1 => Eight-bit data is valid
	-	0-1	Not used (=0)

26.2.1.1 CDBIOC -- Current IOCB address

These two-bytes of the CDB are reserved for expansion. They are currently not being used by the device drivers. These two bytes should be initialized to zero.

26.2.1.2 CDBSDA -- Software driver address

This two-byte field of the CDB must contain the starting address of the device driver program that controls the device. It is this address that is used to access the individual device driver entry points. Therefore, this entry must be provided in every CDB. The format of the device driver is explained in section 26.2.2.

26.2.1.3 CDBHAD -- Hardware address

These two bytes of the CDB are intended to contain the lowest address of the hardware device (PIA or ACIA) used to interface with the external device. The actual usage of this CDB entry depends exclusively on the device driver program. The device independent I/O functions do not access this entry.

26.2.1.4 CDBDDF -- Device descriptor flags

The CDBDDF byte contains the basic description about the types of I/O accesses that the device can perform. The format of the CDBDDF byte is shown below:

```
  7   6   5   4   3   2   1   0
---------------------------------
! R ! O ! I ! F ! W ! S ! L ! D !
---------------------------------
  :   :   :   :   :   :   :   :.... Default binary format
  :   :   :   :   :   :   :........ Logical sector I/O flag
  :   :   :   :   :   :............ System console flag
  :   :   :   :   :................ Rewindable device flag
  :   :   :   :.................... File-type device flag
  :   :   :........................ Input device flag
  :   :............................ Output device flag
  :................................ Reservable device flag
```

These flags are constant once defined. The flags are interrogated by the various device independent I/O functions in order to verify that the requested function can be performed on the specified device. The properties controlled by the various bits of the CDBDDF are explained below.

R (Bit 7) -- Reservable device flag

This bit determines whether a device can be reserved by multiple IOCBs at the same time. Certain devices, like diskette devices, by nature of their operation, can allow input/output accesses to be performed from different callers (IOCBs). Other devices, like a line printer, cannot logically allow multiple output accesses from different IOCBs to be processed. If the "R" bit is set to one, it means that the device is reservable. In other words, only one IOCB can communicate with the device at a time. If the "R" bit is set to zero, it means that the device is non-reservable (i.e., the device can communicate with multiple IOCBs).

O (Bit 6) -- Output device flag

This bit indicates whether a device can be used by output functions. If the "O" bit is set to one, then the device can be used for output. If the "O" flag is set to zero, then the device cannot be used for output.

I (Bit 5) -- Input device flag

This bit indicates whether a device can be used by input functions. If the "I" bit is set to one, then the device can be used for input. If the "I" flag is set to zero, then the device cannot be used for input.

F (Bit 4) -- File-type device flag

This bit determines whether or not a device can open and close files. A file-type device (e.g., diskette drive, and cassette or paper tape reader/punch) will be handled differently by the .OPEN and .CLOSE functions than a non-file-type device (e.g., console printer, line printer, keyboard). In addition to having FDR processing performed on them, file-type devices are also sensitive to end-of-file records. Non-file-type devices are not subject to FDR processing, nor are end-of-file records read from them or written to them. A file-type device is indicated by the "F" bit being set to one. Non-file-type devices have the "F" bit set to zero.

W (Bit 3) -- Rewindable device flag

This bit indicates whether the .REWND function is valid for the device. In the current version of MDOS, it may appear as if the "W" flag and the "L" flag are redundant, because only the diskette device can be used for logical sector I/O and only the diskette device can be "rewound"; however, in order to allow for expansion, the .REWND function's processing depends on the "W" flag. If the "W" flag is set to one, the device can be rewound. If the "W" flag is set to zero, the device cannot be rewound.

S (Bit 2) -- System console flag

This flag distinguishes the system console from all other devices. This is needed since the record input function does special processing for the certain control characters which are treated differently when being input from another device. These special characters are described in section 25.3.4. If the "S" bit is set to one, the device is the system console. If the "S" bit is set to zero, the device is not the system console.

L (Bit 1) -- Logical sector I/O flag

This flag is used to distinguish the diskette drives from all other devices. Since the two specialized I/O calls, .GETLS and .PUTLS, are only valid for the diskette drives, a flag is necessary that identifies that device. If the "L" flag is set to one, logical sector I/O is valid (i.e., the device is the diskette drive). If the "L" flag is set to zero, logical sector I/O is invalid (i.e., the device is not the diskette drive).

D (Bit 0) -- Default binary record format flag

Some devices cannot receive or transmit eight-bit data bytes. For those types of devices a special record format has been designed so that binary records can be processed. Devices that can process eight-bit data can process either type of record format. The "D" bit controls the default record format to be used when dealing with "binary" records. The FMT field of the IOCFDF entry in the IOCB has a special value that will cause the default binary record format to be used for the indicated device. If the "D" bit is set to one, the default record format will be the ASCII-converted-binary format (only if binary records are being processed). If the "D" bit is set to zero, then the default record format will be the binary format (only if binary records are being processed). If the device can process eight-bit data, then the setting of the "D" bit is independent of the device type; however, for devices which can only process seven-bit data, the "D" bit must be set to one. Otherwise, the device may respond unpredictably when binary data are being transmitted to it.

26.2.1.5 CDBVDT -- Valid data types

This byte of the CDB is an extension of the CDBDDF entry. It contains some additional flags that govern the types of I/O accesses that can be made on the device. The format of the CDBVDT entry is shown below.

```
  7   6   5   4   3   2   1   0
 ---------------------------------
 ! N !               ! B !       !
 ---------------------------------
   :         :         :     :...... Not used (=0)
   :         :         :............ Binary device flag
   :         :...................... Not used (=0)
   :................................ Non-file format flag
```

The properties controlled by the various bits of the CDBVDT entry are explained below.

N (Bit 7) -- Non-file format flag

This bit indicates whether or not the device can be used to perform FDR processing. Certain devices (i.e., those with the file-type bit set to zero in CDBDDF) can never perform FDR processing; however, devices which are file-type devices can, in some cases, be used in either the file format or the non-file format mode (see IOCDTT description, section 25.3.1.2). If the "N" bit is set to one, then the device can be used in the non-file format mode. If the "N" bit is set to zero, then the device cannot be used in the non-file format mode. The diskette drive is an example of a device that can only be used in the file format mode. The console reader is an example of a device that can be used in either mode. The line printer is an example of a device that can only be used in the non-file format mode.

NOT USED (Bits 3-6, 0-1) -- Reserved area

These bits of the CDBVDT byte are reserved for future expansion. They must be zero.

B (Bit 2) -- Binary device flag

This bit indicates whether a device can process eight-bit data or not. If the "B" flag is set to one, then eight-bit data are valid. If the "B" flag is set to zero, then eight-bit data are invalid.

26.2.1.6 CDBDDA -- Device dependent area

These two-bytes of the CDB are available to the device drivers as working storage. For the MDOS-supported devices, this field has been provided for future expansion. For other devices, this field can be used for whatever purposes are deemed appropriate.

26.2.1.7 CDBWST -- Working storage

These two-bytes of the CDB are available to the device drivers as working storage.

26.2.2 Device drivers

Each device type that is to be accessed via the device independent I/O functions (section 25.3) must have its own driver program. All device drivers must be accessible for the following five functions:

1. Turn the device on,
2. Turn the device off,
3. Perform device initialization,
4. Perform device termination,
5. Input and/or output a single character.

Not necessarily all of the five functions apply to each device; however, an entry point must be provided in each device driver for each of the five functions, regardless of whether or not the function is performed.

Since the only address that is available to the device independent I/O functions is the starting address of the device driver (CDBSDA of CDB), the following convention must be used by each device driver. The starting address contained in the CDBSDA entry must be the address of the beginning of a jump table, one jump for each of the five functions listed above. An example of such jump table is given below:

```
DVDRV$    EQU      *        . ADDRESS KEPT IN CDBSDA
          JMP      DEVON    . DEVICE ON ROUTINE
          JMP      DEVOFF   . DEVICE OFF ROUTINE
          JMP      DEVINT   . INITIALIZATION ROUTINE
          JMP      DEVTRM   . TERMINATION ROUTINE
DEVIO     EQU      *        . CHARACTER I/O ROUTINE
```

Each entry point to the device driver is accessed from the device independent I/O functions by executing an indexed subroutine call. The offset (index value) is defined by the displacement of the entry point from the beginning of the device driver. Since these offsets must be the same for all device drivers, a set of system symbols is defined in the MDOS equate file for the device driver entry point offsets.

The device on and off entry points are accessed at the beginning and at the end of every record I/O function call (.GETRC and .PUTRC). These entry points allow the device driver to turn the device on and off, respectively. If such actions are not defined for the device, then the entry points should jump to a routine which simply exits the driver with a "no error" status condition.

The device initialization and termination entry points are called once by the .OPEN and .CLOSE functions, respectively. These entry points are intended to allow leader to be punched on a paper tape device, for example. If such actions are not defined for the device, then the entry points should jump to a routine which simply exits the driver with a "no error" status condition.

The character I/O entry point to the driver is used to receive or transmit one byte of data. The transmitted or received byte is passed between the I/O functions and the device driver in the "B" accumulator. For devices that can process both input and output, the IOCB must be interrogated ("IO" of IOCDTT) by the device driver to determine which function is to be performed. Since the index register is required to execute the jump to subroutine instruction, the address of the IOCB is passed to the device driver using the following convention:

```
                 JSR      DV$IO,X . CALL TO DRIVER
                 FDB      IOCPTR  . POINTER TO IOCB'S POINTER
                 BCS      ERROR   . RETURN HERE FROM DRIVER
                  .
                  .
        IOCPTR   FDB      IOCB    . ADDRESS OF IOCB
```

With this convention, the address pushed on the stack as a result of executing the jump to subroutine instruction will point to the double byte which contains a pointer. It is the data at the address identified by the pointer that is the actual address of the IOCB itself. As a result, the device driver cannot just execute a return from subroutine instruction to get back to the I/O function. This calling sequence applies to all entry points into all device drivers.

Before returning to the I/O function, the device driver must set an error status condition indicating the state of the performed action. Two things must be configured by the driver to indicate an error. First, the IOCSTA byte of the IOCB must be initialized with one of the standard I/O error statuses (section 25.3.1.1). Second, the carry condition code must be set to one. If no error occurred, only the carry condition code must be set to zero. The IOCSTA entry of the IOCB need not be changed to zero since the I/O function will set a normal return status before exiting. The "A" and "X" registers need not be preserved by the device driver in any case. The "B" register returns the character received if the device driver was called upon for an input request.

26.2.3 Example of device driver

The following example illustrates a CDB and its associated device driver for a high-speed papertape reader (specifically, the EXORtape reader). The system symbols from the MDOS equate file are used throughout this example. First, the CDB is shown:

```
*
* CONTROLLER DESCRIPTOR BLOCK (CDB)
*
HR$CDB EQU    *            .
       FDB    0            . CDBIOC
       FDB    HRDRV$       . CDBSDA
       FDB    $E404        . CDBHAD
       FCB    DD$RES+DD$INP+DD$OCF . CDBDDF
       FCB    VD$NFF+VD$BIN . CDBVDT
       FDB    0            . CDBDDA
       FDB    0            . CDBWST
```

Logically, the paper tape reader should not be accessed by multiple IOCBs at the same time. Thus, the device is considered to be reservable (Bit "R" of CDBDDF set to 1). The paper tape reader is an input device only. Therefore, bit "O" of CDBDDF is zero and bit "I" is one. The paper tape reader is sensitive to end-of-file records. Thus, it must be a file-type device (Bit "F" of CDBDDF set to 1). Bits "W", "S", and "L" are all zero since the paper tape reader is not rewindable (according to the definition in section 26.2.1.4), is not the system console, and is not able to perform logical sector I/O. The default binary format has been arbitrarily identified as binary record.

The paper tape reader is capable of being used in the non-file format mode and is capable of transmitting eight-bit data to the device. Thus, both bits "N" and "B" of CDBVDT are set to one.

The only other required field of the CDB is the address of the device driver in CDBSDA. The remainder of the CDB is reserved for expansion or is used for working storage by the device driver.

Next, the device driver itself is shown. Of the five entry points that are required by each device driver, only two are used for the paper tape reader driver. The other three (device on, device off, and device termination) are dummy vectors that set a "no error" return status and then return to the I/O function.

```
*
*DEVICE DRIVER ENTRY POINTS
*
HRDRV$ EQU    *
       CLC              . TURN DEVICE ON
       BRA    RETURN    .
*
       CLC              . TURN DEVICE OFF
       BRA    RETURN    .
*
       JSR    INITR     . DEVICE INITIALIZATION
*
       CLC              . DEVICE TERMINATION
       BRA    RETURN    .
*
       BSR    GETCP     . CHARACTER INPUT
       TAB              . RETURN WITH CHAR IN "B"
       BCC    RETURN    . CC => NO ERROR
       TSX              . CS => END OF MEDIA (TIMEOUT)
       LDX    0,X       . GET ADR OF FDB FOLLOWING JSR
       LDX    0,X       . GET CONTENTS OF FDB
       LDX    0,X       . GET ADR OF IOCB
       LDAA   #I$EOM    . SET END OF MEDIA STATUS
       STAA   IOCSTA,X .
RETURN TSX              . RETURN TO CALLER
       LDX    X         . GET ADR OF FDB FOLLOWING JSR
       INS              . ADJUST STACK FOR RETURN
       INS              .
       JMP    2,X       . JUMP TO ADR FOLLOWING FDB
*
* READER INITIALIZATION ROUTINE
*
INITR  STX    HR$CDB+CDBDDA   . SAVE INDEX REGISTER
       LDX    HR$CDB+CDBHAD   . GET THE PIA ADDRESS
       CLR    PTCTL,X   .
       CLR    PTDTA,X   .
       LDAA   #$3C      .
       STAA   PTCTL,X   .
       LDX    HR$CDB+CDBDDA   . RESTORE INDEX REGISTER
       RTS              .
*
* INPUT ONE CHARACTER
*
GETCP  STX    HR$CDB+CDBDDA   . SAVE THE INDEX REGISTER
       LDX    HR$CDB+CDBHAD   . GET THE PIA ADDRESS
       LDAA   PTDTA,X   . CLR INTERRUPT
       LDAA   #$34      . STROBE READER
       STAA   PTCTL,X   .
       LDAA   #$3C      .
       STAA   PTCTL,X   .
       CLR    HR$CDB+CDBWST   . INIT THE TIMEOUT COUNTER
       CLR    HR$CDB+CDBWST+1 . AND CLEAR CARRY
GETC1  LDAA   PTCTL,X   . READY TO READ?
```

```
         BMI    GETC2     . MI => YES
         DEC    HR$CDB+CDBWST+1 . PL => CHECK TIMEOUT
         BNE    GETC1     . NE => KEEP LOOPING
         DEC    HR$CDB+CDBWST   .
         BNE    GETC1     . NE => KEEP LOOPING
         SEC              . SET CARRY FOR TIMEOUT
GETC2    LDAA   PTDTA,X   . GET CHAR
         BCS    GETC4     . CS => TIMEOUT
*
* IF ASCII FILE, STRIP PARITY
*
         TSX              . GET ADR OF IOCB
         LDX    2,X       . GET BACK TO 1ST LEVEL SUBRTN
         LDX    0,X       . GET CONTENTS OF 2ND FDB
         LDX    0,X       . GET ADR OF IOCB
         LDAB   IOCFDF,X  . PICK UP FILE ATTRIBUTES
         ANDB   #7        . ISOLATE FMT BITS
         CMPB   #FM$FMA   . ASCII FILE ?
         BNE    GETC3     . NE => NO, LEAVE 8 BITS
         ANDA   #$7F      . STRIP PARITY IF ASCII
GETC3    CLC              . SET STATUS TO OK
GETC4    LDX    HR$CDB+CDBDDA   . RESTORE X
         RTS              .
```

26.2.4 Adding a non-standard device

If the device driver defined in the above example is to be used by a user's program with the device independent I/O functions, then the only function that is treated differently is the .RESRV function. Since .RESRV must be used to link the IOCB with a known CDB, the .RESRV call is bypassed altogether by the user program; however, before the .OPEN function is invoked, the IOCB must be parameterized as if it had been properly reserved.

Thus, the IOCGDW entry of the IOCB must be configured to contain the address of the CDB with which communication is to take place. In addition, bit "R" of IOCLUN must indicate that the IOCB has been reserved. This information is also found in the EXIT CONDITIONS description of the .RESRV function (section 25.3.2).

Once the IOCB has been configured in this manner, the other I/O functions can be used in the normal fashion.

CHAPTER 27

27. OTHER SYSTEM FUNCTIONS

In the following description of the system functions these symbols will be used:

Symbol	Meaning
A	A accumulator
B	B accumulator
X	Index register
S	Stack pointer register
CC	Condition code register
Z	Zero flag of condition code register (bit 2)
C	Carry flag of condition code register (bit 0)
XH	Most significant byte of X
XL	Least significant byte of X
B,A	The register pair B and A treated as a sixteen bit register

For MDOS09, the registers Y, U and DP are unchanged by the system function calls.

It is assumed that the reader is familiar with what system functions are, how they are invoked, what precautions must be taken when testing programs using system functions, and how errors are handled by system functions (see section 24.8).

The remainder of this chapter is devoted to the description of all system functions not described thus far. The description is divided into the following sections: register functions, double-byte arithmetic functions, character string functions, diskette file functions, and miscellaneous functions.

27.1 Register Functions

The register functions are used by some of the other system functions as an extension of the M6800 instruction set. Many operations that involve the transfer and exhange of information between the register pair "B,A" and the X register are made feasible by the fact that the SWI instruction (used for accessing system function handler) automatically saves all registers on the stack. Since the sixteen bit registers are pushed on the stack least significant byte first, most significant byte last, the register pair "B,A" was chosen instead of "A,B". The relationship of the B and A registers on the stack is then identical to the other sixteen bit registers saved in this fashion (for the M6800 only). For the M6809, it is not anticipated that the user will use the register system functions as there are instructions to perform most of these functions. However, the system function calls remain identical between MDOS and MDOS09 to allow portability of program written initially for the 6800.

27.1.1 Transfer X to B,A -- .TXBA

The .TXBA function transfers the contents of the X register into the register pair B,A.

ENTRY PARAMETERS:	None.
EXIT CONDITIONS:	A contains XL. B contains XH. X is unchanged. CC is indeterminate.

27.1.2 Transfer B,A to X -- .TBAX

The .TBAX function transfers the contents of the register pair B,A into the X register.

ENTRY PARAMETERS:	None.
EXIT CONDITIONS:	A is unchanged. B is unchanged. XH contains B. XL contains A. CC is indeterminate.

27.1.3 Exchange B,A with X -- .XBAX

The .XBAX function exchanges the contents of the register pair B,A with the contents of the X register.

ENTRY PARAMETERS:	None.
EXIT CONDITIONS:	A contains entry value of XL. B contains entry value of XH. XH contains entry value of B. XL contains entry value of A. CC is unchanged.

27.1.4 Add B to X -- .ADBX

The .ADBX function adds the contents of the B register to the contents of the X register. The addition is performed as if B were an unsigned binary number.

ENTRY PARAMETERS:	None.
EXIT CONDITIONS:	A is unchanged.
	B is unchanged.
	X has been incremented by the contents of B.
	CC has been set as in a normal unsigned addition.

27.1.5 Add A to X -- .ADAX

The .ADAX function adds the contents of the A register to the contents of the X register. The addition is performed as if A were an unsigned binary number.

ENTRY PARAMETERS:	None.
EXIT CONDITIONS:	A is unchanged.
	B is unchanged.
	X has been incremented by the contents of A.
	CC has been set as in a normal unsigned addition.

27.1.6 Add B,A to X -- .ADBAX

The .ADBAX function adds the contents of the register pair B,A to the contents of the X register.

ENTRY PARAMETERS:	None.
EXIT CONDITIONS:	A is unchanged.
	B is unchanged.
	X has been incremented by the contents of B,A.
	CC has been set as in a normal unsigned addition.

27.1.7 Add X to B,A -- .ADXBA

The .ADXBA function adds the contents of the X register to the contents of the register pair B,A.

ENTRY PARAMETERS:	None.
EXIT CONDITIONS:	A has been incremented by XL.
	B has been incremented by XH and C.

X is unchanged.
CC has been set as in a normal unsigned addition.

27.1.8 Subtract B from X -- .SUBX

The .SUBX function subtracts the contents of the B register from the contents of the X register. The subtraction is performed as if B were an unsigned binary number.

ENTRY PARAMETERS:	None.
EXIT CONDITIONS:	A is unchanged. B is unchanged. X has been decremented by the contents of B. CC has been set as in a normal, unsigned subtraction.

27.1.9 Subtract A from X -- .SUAX

The .SUAX function subtracts the contents of the A register from the contents of the X register. The subtraction is performed as if A were an unsigned binary number.

ENTRY PARAMETERS:	None.
EXIT CONDITIONS:	A is unchanged. B is unchanged. X has been decremented by the contents of A. CC has been set as in a normal unsigned subtraction.

27.1.10 Subtract B,A from X -- .SUBAX

The .SUBAX function subtracts the contents of the register pair B,A from the contents of the X register.

ENTRY PARAMETERS:	None.
EXIT CONDITIONS:	A is unchanged. B is unchanged. X has been decremented by the contents of B,A. CC has been set as in a normal unsigned subtraction.

27.1.11 Subtract X from B,A -- .SUXBA

The .SUXBA function subtracts the contents of the X register from the contents of the register pair B,A.

ENTRY PARAMETERS:	None.
EXIT CONDITIONS:	A has been decremented by XL. B has been decremented by XH and C. X is unchanged. CC has been set as in a normal unsigned subtraction.

27.1.12 Compare B,A with X -- .CPBAX

The .CPBAX function compares the contents of the register pair B,A to the contents of the X register.

ENTRY PARAMETERS:	None.
EXIT CONDITIONS:	A is unchanged. B is unchanged. X is unchanged. CC has been set as in a normal unsigned subtraction.

27.1.13 Shift X right -- .ASRX

The .ASRX function shifts the contents of the X register to the right by one bit position. Bit 15 is held constant and bit 0 is moved into C.

ENTRY PARAMETERS:	None.
EXIT CONDITIONS:	A is unchanged. B is unchanged. X is shifted right one bit position. The sign bit is propagated into the lower bits upon subsequent shifts. C contains bit zero of the entry value of X. The remainder of CC is indeterminate.

27.1.14 Shift X left -- .ASLX

The .ASLX function shifts the contents of the X register to the left by one bit position. Bit 0 is filled with zero.

ENTRY PARAMETERS: None.

EXIT CONDITIONS: A is unchanged.
B is unchanged.
X is shifted left one bit position. Bit zero is filled with zero.
C contains bit 15 of the entry value of X. The remainder of CC is indeterminate.

27.1.15 Push X on stack -- .PSHX

The .PSHX function pushes the contents of the X register on the current stack.

ENTRY PARAMETERS: None.

EXIT CONDITIONS: A is unchanged.
B is unchanged.
X is unchanged.
CC is unchanged.
S has been decremented by 2. The contents of XL have been pushed on the stack followed by the contents of XH.

27.1.16 Pull X from stack -- .PULX

The .PULX function pulls the contents from the stack into the X register.

ENTRY PARAMETERS: None.

EXIT CONDITIONS: A is unchanged.
B is unchanged.
XH contains the contents located at the entry value of S + 1.
XL contains the contents located at the entry value of S + 2.
CC is unchanged.
S has been incremented by 2.

27.2 Double-byte Arithmetic Functions

The double-byte arithmetic functions are used by some of the other system functions and the MDOS commands as an extension of the M6800 instruction set. These functions are not as general purpose as the register functions, but they are useful in special cases.

27.2.1 Add A to memory -- .ADDAM

The .ADDAM function increments a double byte in memory by the contents of the A register. The addition is performed as if A is an unsigned binary number.

ENTRY PARAMETERS:	X = The address of most significant byte of a double byte in memory.
EXIT CONDITIONS:	A is indeterminate. B is unchanged. X is unchanged. CC is indeterminate. The double byte in memory has been incremented by the contents of A.

27.2.2 Subtract A from memory -- .SUBAM

The .SUBAM function decrements a double byte in memory by the contents of the A register. The subtraction is performed as if A is an unsigned binary number.

ENTRY PARAMETERS:	X = The address of the most significant byte of a double byte in memory.
EXIT CONDITIONS:	A is indeterminate. B is unchanged. X is unchanged. CC is indeterminate. The double byte in memory has been decremented by the contents of A.

27.2.3 Shift memory right -- .DMA

The .DMA function shifts the contents of a double byte in memory to the right by the number of bit positions represented by the contents of the A register. The effect is to divide the double byte by a power of 2. The exponent is given by the value of the A register.

ENTRY PARAMETERS:	X = The address of the most significant byte of a double byte in memory.
EXIT CONDITIONS:	A is unchanged. B is unchanged. X is unchanged. CC is indeterminate. The double byte in memory has been shifted to the right by the number of bits represented by the contents of A. Zero bits are brought in from the

left as the shift takes place.

27.2.4 Shift memory left -- .MMA

The .MMA function shifts the contents of a double byte in memory to the left by the number of bit positions represented by the contents of the A register. The effect is to multiply the double byte by a power of 2. The exponent is given by the value of the A register.

ENTRY PARAMETERS: X = The address of the most significant byte of a double byte in memory.

EXIT CONDITIONS: A is unchanged.
B is unchanged.
X is unchanged.
CC is indeterminate.
The double byte in memory has been shifted to the left by the number of bits represented by the contents of A. Zero bits are brought in from the right as the shift takes place.

27.3 Character String Functions

The character string functions are used by some of the more complex system functions and the MDOS commands as macro instructions or subroutines.

27.3.1 String move -- .MOVE

The .MOVE function transfers a series of contiguous bytes in memory from one location into another location. The move is made starting with the lowest addressed byte of the source string.

ENTRY PARAMETERS: B = The number of bytes to be moved. If B is intially zero, 256 (decimal) bytes will be moved.
X = The address of the first byte of a four-byte parameter packet. The parameter packet has the following format:

```
        ------------------------------
  0  |        Address of           |
     --          the               --
  1  |      source string          |
        ------------------------------
  2  |        Address of           |
     --          the               --
  3  |    destination string       |
        ------------------------------
```

EXIT CONDITIONS: A is indeterminate.
B = 0.
X is unchanged.
CC is indeterminate.
The addresses of the source and destination strings in the parameter packet have both been incremented by the entry value of B.
The source string has been moved into the destination string.

27.3.2 String comparison -- .CMPAR

The .CMPAR function compares a series of contiguous bytes in memory from one location with a series of bytes at another location. The comparison is made starting with the lowest addressed byte of the source string.

ENTRY PARAMETERS: B = The number of bytes to be compared. If B is intially zero, 256 (decimal) bytes will be compared.
X = The address of the first byte of a four-byte parameter packet. The parameter packet has the following format:

```
        ------------------------------
  0  |        Address of           |
     --          the               --
  1  |      source string          |
        ------------------------------
  2  |        Address of           |
     --          the               --
  3  |    destination string       |
        ------------------------------
```

EXIT CONDITIONS: A is indeterminate.
B = The number of bytes remaining in the string which did not compare. If B is zero, the strings were identical. If the strings mis-compared on the first byte, B is unchanged.

X is unchanged.
Z = 1 if the strings compared (B = 0). The remainder of CC is indeterminate.
Z = 0 if the strings mis-compared. The remainder of CC is indeterminate.
The addresses of the source and destination strings in the parameter packet have both been incremented by the entry value of B if the two strings compared. Otherwise, the source string pointer will contain the address of the character following the mis-comparison, and the destination string pointer will contain the address of the character of the mis-comparison.
The source and destination strings are unchanged.

27.3.3 Character-fill a string -- .STCHR

The .STCHR function stores a specific character into a series of contiguous bytes in memory.

ENTRY PARAMETERS:

A = The character to be stored into the string.
B = The number of bytes to be filled with the character. If B is initially zero, 256 (decimal) bytes will be filled.
X = The address of the first byte of the string to be filled.

EXIT CONDITIONS:

A is unchanged.
B = 0.
X is unchanged.
CC is indeterminate.
The string is filled with the character in A.

27.3.4 Blank-fill a string -- .STCHB

The .STCHB function stores blanks ($20) into a series of contiguous bytes in memory.

ENTRY PARAMETERS:

B = The number of bytes to be filled with blanks. If B is initially zero, 256 (decimal) bytes will be filled.
X = The address of the first byte of the string to be filled.

EXIT CONDITIONS:

A = $20 (space).

	B = 0.
	X is unchanged.
	CC is indeterminate.
	The string is filled with blanks.

27.3.5 Test for alphabetic character -- .ALPHA

The .ALPHA function examines the character in the A register for being an upper case alphabetic character (A-Z).

ENTRY PARAMETERS:	A = The character to be tested.
EXIT CONDITIONS:	A is unchanged.
	B is unchanged.
	X is unchanged.
	C = 0 if A contains a valid alphabetic character. The remainder of CC is indeterminate.
	C = 1 if A contains an invalid alphabetic character. The remainder of CC is indeterminate.

27.3.6 Test for decimal digit -- .NUMD

The .NUMD function examines the character in the A register for being a valid ASCII decimal digit (0-9).

ENTRY PARAMETERS:	A = The character to be tested.
EXIT CONDITIONS:	A is unchanged if it contained an invalid digit. Otherwise, A contains the binary equivalent of the decimal digit (bits 4-7 will be zero).
	B is unchanged.
	X is unchanged.
	C = 0 if A contained a valid digit. The remainder of CC is indeterminate.
	C = 1 if A contained an invalid digit. The remainder of CC is indeterminate.

27.4 Diskette File Functions

The diskette file functions can be used in conjunction with the device dependent I/O functions (section 25.2) for diskette accessing. These functions are used by the device independent I/O functions to perform directory searches and diskette space allocation and deallocation. The MDOS commands use these functions for changing file names and attributes and for loading programs from memory-image files from the diskette into memory.

All of the functions described in this section require a twenty-five byte parameter table called the diskette file table, or DFT. The format of the table is shown here so that it will not have to be repeated for each function. It will be seen that the first sixteen bytes of the DFT are identical in format with an MDOS directory entry. Also, the entire DFT is of the same format as part of an IOCB (starting with IOCLUN and ending with IOCSBE). The contents of the individual fields are not described in this section since they have been adequately discussed in sections 24.1.4 and 25.3.1. All of the diskette file functions will change the diskette controller variables below location $0020.

```
        --------------------------------
   00   !     Logical unit number      !   LUN
        --------------------------------
   01   !                              !
        --                            --
   02   !                              !
        --                            --
   03   !                              !
        --                            --
   04   !          File Name           !
        --                            --   NAM
   05   !                              !
        --                            --
   06   !                              !
        --                            --
   07   !                              !
        --                            --
   08   !                              !
        --------------------------------
   09   !                              !
        --           Suffix           --   SUF
   0A   !                              !
        --------------------------------
   0B   !    Physical sector number    !
        --        of file's RIB       --   RIB
   0C   !                              !
        --------------------------------
   0D   ! W ! D ! S ! C ! N !    FMT   !   FDF - File descrip-
        --                            --         tor flags
   0E   !        (reserved; =0)        !
        --------------------------------
   0F   !                              !
        --       (reserved; =0)       --   RES
   10   !                              !
        --------------------------------
   11   !        PSN        !    EN    !   DEN - Directory
        --                            --         entry number
   12   !        (reserved; =0)        !
        --------------------------------
   13   !                              !
        --   Initial new file size    --   SIZ
   14   !                              !
        --------------------------------
   15   !        Sector buffer         !
        --       start address        --   SBS
   16   !                              !
        --------------------------------
   17   !        Sector buffer         !
        --        end address         --   SBE
   18   !                              !
        --------------------------------
```

27.4.1 Directory search -- .DIRSM

The .DIRSM function performs directory searches based on various criteria. This function can be used for finding, creating, or deleting directory entries on an MDOS diskette.

ENTRY PARAMETERS:

B contains a function code that specifies the action to be performed by .DIRSM.

X = The address of the DFT. All calls to .DIRSM require that LUN contains the logical unit number to be accessed (ASCII number 0-3, $30-$33), that SBS contains the starting address of a 128 (decimal) byte sector buffer, and that SBE contains the ending address of the sector buffer. If the sector buffer is larger than a single sector, only the first 128 bytes will be used.

The following function codes for the B register are defined:

B = 1 indicates to search for and retrieve the next, non-deleted directory entry. The DFT must have DEN = 0 for the initial call. The DEN must then remain unchanged for subsequent calls since it is used to determine where to resume the search. The contents of the sector buffer must also remain unchanged between successive calls for this function code.

B = 2 indicates to search for and retrieve a directory entry with a specific file name and suffix. The DFT entries NAM and SUF are used to specify the file name.

B = 4 indicates to create a new unique directory entry of a given name and suffix. Initial diskette space allocation is performed if the directory entry is created. The DFT entries NAM and SUF are used to specify the directory entry to be created. A search of the directory is performed for this entry to ensure that it does not already exist. The DFT entries FDF and SIZ must also be specified. FDF must specify both the inherent and the changeable attributes to be initially assigned to the file. SIZ is used to describe the initial diskette space that is to be allocated. If SIZ is zero, the default space allocation will be performed. If SIZ is non-zero, the allocation will be performed using the contents of SIZ as the minimum number of sectors to be allocated.

B = 8 indicates a similar function to be performed as for the B=4 case; however, in the event that a directory entry already exists with the NAM and SUF found in the DFT, that file's directory entry information will be returned in the DFT. Otherwise, the DFT is parameterized identically to the B=4 case.

B = 16 ($10) indicates that a specific directory entry is to be deleted from the directory. The DFT entries NAM and SUF are used to specify the entry to be deleted.

B = 32 ($20) indicates to search for the next, non-deleted directory entry with a specific set of file attributes. Entries encountered with different attributes will not be returned by the search. The DFT must have DEN = 0 for the initial call. The DEN must then remain unchanged for subsequent calls since it is used to determine where to resume the search. The contents of the sector buffer must also remain unchanged between successive calls for this function code. The FDF entry must contain the specific attributes to be searched for.

EXIT CONDITIONS: A is indeterminate.

B contains the return status. The following return statuses are defined:

B = 0 indicates that no errors occurred (normal return).

B = 1 indicates that the directory entry specified by LUN, NAM, and SUF was not found in the directory.

B = 2 indicates that B contained an invalid function code upon entry to .DIRSM.

B = 3 indicates the physical end of the directory was encountered during a "search for next directory entry" request (Entry value of B = 1 or 32).

B = 4 indicates that the directory is full and cannot accomodate a new entry.

B = 5 indicates that insufficient diskette space exists to satisfy the initial space requirements of SIZ when attempting to create a new directory entry. The .ALLOC function (section 27.4.4) should be consulted for a full description of the allocation scheme and the reasons for arriving at this error.

B = 7 indicates that an attempt was made to create a duplicate entry in the directory. The file name identified by LUN, NAM, and SUF already exists in the directory.

B = 8 indicates that a new directory entry was created as specified by LUN, NAM, and SUF.

B = 9 indicates that an attempt was made to delete a protected file.

X is unchanged.

C = 0 if no errors occurred (B = 0). The remainder of CC is indeterminate.

C = 1 if an error occurred (B not zero). The remainder of CC is indeterminate.

The DFT entries were changed in the following manner depending on the various entry values of B:

B = 1. If a non-deleted directory entry was found, then NAM, SUF, RIB, FDF, and RES contain the full image of the directory entry. DEN will contain the computed directory entry number. The remainder of the DFT is unchanged. The sector buffer contains the current directory sector. If no directory entry was found, the directory entry fields NAM through RES, inclusive, will be unchanged. DEN and the contents of the sector buffer are indeterminate.

B = 2. The DFT is affected the same as for B=1.

B = 4. If a new directory entry was created, RIB and DEN will reflect the appropriate values for the new entry. The sector buffer will contain the current directory sector. If a new entry was not created (duplicate file name), then the DFT will be affected in the same way as for B=1.

B = 8. The exit conditions for this case are the same as for B=4. In addition, if a duplicate entry already existed in the directory, the directory entry fields NAM through RES, inclusive, will contain the full image of the duplicate entry. DEN will also contain the duplicate entry's directory entry number.

B = 16. If the entry is deleted, the complete directory entry will be returned in fields NAM through RES, inclusive. In addition, RIB will be zero. The contents of the sector buffer are indeterminate. If the entry is not deleted, all parameters except RES and DEN will be unchanged. RES, DEN and the contents of the sector buffer will be indeterminate.

B = 32. The DFT is affected in the same way as for B=1.

27.4.2 Change file name/attributes -- .CHANG

The .CHANG function allows a directory entry to have its name, suffix, and/or attribute fields changed.

ENTRY PARAMETERS:

B = A function code that specifies the action to be taken by .CHANG. If bit 0 is set to one, .CHANG will change the file name and suffix fields of a directory entry. If bit 1 is set to one, the function will change the attribute field of a directory entry. Bits 2-7 are not used and should be zero. Bits 0 and 1 are independent of each other. Thus, .CHANG can be used to change file name, suffix, and attributes at the same time.

X = The address of a file table packet. The packet has the following format:

```
    ---------------------------
0   !       Address of        !
    --       old DFT         --
1   !                         !
    ---------------------------
2   !       Address of        !
    --       new DFT         --
3   !                         !
    ---------------------------
```

The "old DFT" contains the LUN, NAM, and SUF fields of an existing directory entry that is to be changed. The SBS contains the starting address of a 128 (decimal) byte sector buffer. SBE contains the ending address of the sector buffer. If the sector buffer is larger than one sector, only the first 128 bytes will be used. The "new DFT" contains the information that is to be placed into the directory entry. LUN in both DFTs must be the same (ASCII number 0-3, $30-$33). The new DFT must contain NAM, SUF, and/or FDF fields as indicated by the function code in the B register. A sector buffer is not required by the new DFT.

EXIT CONDITIONS:

A is indeterminate.

B contains the return status. The following return statuses are defined:

B = 0 indicates that no errors occurred (normal return).

B = 1 indicates that B contained an invalid function code upon entry to .CHANG.

B = 3 indicates that the directory entry specified by LUN, NAM, and SUF of the old DFT could not be found in the directory. The old DFT directory entry must exist in order for the change to be possible.

B = 4 indicates that the directory entry specified by LUN, NAM, and SUF of the new DFT already existed in the directory. The new DFT directory entry must have a unique file name and suffix (only if changing the old entry's file name).

B = 5 indicates that an invalid attribute change was attempted. Only the changeable attributes (system file, write protection, delete protection) can be changed. The inherent attributes of a file remain constant for the duration of the file's existence.

X is unchanged.

C = 0 if no errors occurred (B = 0). The remainder of CC is indeterminate.

C = 1 if an error occurred (B not zero). The remainder of CC is indeterminate.

The four-byte file packet is unchanged.

The old DFT and its sector buffer have been changed as a result of performing a directory search (.DIRSM with B = 2). The new DFT has been changed as a result of performing a directory search (.DIRSM with B = 4); however, no diskette space allocation was performed. A file name change is affected by deleting the old directory entry and by creating a new directory entry. Thus, the directory entry's DEN (and its position within the directory) may have changed; however, no space is deleted or reallocated.

27.4.3 Load program into memory -- .LOAD

The .LOAD function reads a program from a memory-image file from the diskette into memory. Control can be passed to the resident debug monitor, to the calling program, or to the loaded program. In addition, the program can be loaded into the User Memory Map of EXORciser II systems with the dual memory map configuration.

The .LOAD function does not verify that memory exists for the areas into which a program gets loaded. Programs which load above location $1F and below the end of contiguous memory known to MDOS are guaranteed that memory exists since the memory was sized during MDOS initialization; however, programs loading beyond the end of contiguous memory known to MDOS or programs loading into the User Memory Map of an EXORciser II system with the dual memory map configured are not guaranteed that memory exists. The operator is responsible for knowing where memory is configured in his system and where his programs are loaded. Also, due to the nature of the diskette controller, it is not possible for the .LOAD function to compare what is read from the file with what is stored into memory. Only diskette controller read errors can be detected during the load process.

Programs brought into memory from the diskette will be loaded in multiples of eight bytes. This fact must be considered when programs are loaded into adjacent blocks of memory close to other programs, or if programs are loaded into the upper end of a block of memory.

ENTRY PARAMETERS: B = A function code that specifies the action to be performed by .LOAD. This action includes selecting the memory map; checking the limits of the loaded program against the memory map; and the passing of control to the debug monitor, loaded program, or calling program. The following function codes are defined:

Bit 0 = 1 indicates that control is to be given to the loaded program at its starting execution address as obtained from the file's RIB. Bit 0 is mutually exclusive with bits 1 and 2.

Bit 1 = 1 indicates that control is to be given to the resident debug monitor after the program is loaded. Bit 1 is mutually exclusive with bits 0 and 2.

Bit 2 = 1 indicates that control is to be given to the loaded program at a starting execution address specified in the DFT, not at the address contained in the file's RIB. The starting execution address must be specified in DEN of the DFT. Bit 2 is mutually exclusive with bits 0 and 1.

Bit 4 = 1 indicates that the program can only be loaded above the resident MDOS (location $1FFF) and below the last location of contiguous memory established during MDOS initialization. Programs loaded in this manner require an additional eight bytes of memory beyond the last address loaded into by the program. The MDOS variable ENDUS$ will be changed to reflect the last address loaded into by the program. The MDOS SWI vector will be unchanged to allow access to MDOS system functions. Bit 4 is mutually exclusive with bits 5 and 7.

Bit 5 = 1 indicates that the program can only be loaded into the User Memory Map of an EXORciser II system with the dual memory map configuration. The MDOS SWI vector will be restored to point back to the debug monitor if control is passed to the loaded program or to the monitor. If control is returned to the calling program, the MDOS SWI vector will be unchanged. The only requirement placed on programs loading into the User Memory Map of a dual memory map configuration is that the ending load address not be greater than $FFFF. Otherwise, any memory locations ($0000-FFFF) can be loaded into. Bit 5 is mutually exclusive with bits 4 and 7.

Bit 6 = 1 indicates that no directory search is to be performed. The RIB entry of the DFT contains the physical sector number of the RIB of the file from which the program is to be loaded.

Bit 7 = 1 indicates that the program can be loaded anywhere in memory above location $1F. The only other requirement is that the ending load address not exceed $FFFF. No checks are made for overlaying the resident MDOS or for loading into discontiguous memory. As a result, the MDOS SWI vector is restored to point back into the debug monitor, making MDOS system functions unaccessible. This function requires one of the control passage bits (0, 1, or 2) to be set to one. Control must be passed to either the loaded program or to the debug monitor. Control cannot be returned to the calling program. Bit 7 is mutually exclusive with bits 4 and 5.

If none of bits 0-2 are set, then control will be returned to the calling program after the program is loaded.

X = The address of the DFT. All calls to the .LOAD function require that LUN contains the logical unit number to be accessed (ASCII number 0-3, $30-$33), that SBS contains the starting address of a 128 (decimal) byte sector buffer, and that SBE contains the ending address of the sector buffer. If the sector buffer is larger than one sector, only the first 128 bytes will be used. For all cases but one (Bit 6 set to 1), the DFT must also contain the file name and suffix in NAM and SUF. For the Bit 6 case, NAM and SUF are not required. Instead, the physical sector number of the file's RIB must be placed into RIB.

EXIT CONDITIONS: A is indeterminate.

B contains the return status. The following return statuses are defined (only if control is returned to the calling program):

B = 0 indicates that no errors occurred (normal return).

B = 1 indicates that B contained an

invalid function code upon entry to .LOAD. An invalid function may be one that is not defined, or use of more than one of the mutually exclusive bits. This error will also occur when attempting to load into the User Memory Map in a system which does not have the dual memory map configured.

B = 3 indicates that the directory entry specified by LUN, NAM, and SUF was not found in the directory.

B = 4 indicates that the directory entry specified by LUN, NAM, and SUF does not have the memory-image format. Only programs from memory-image files can be loaded from the diskette.

B = 5 indicates that an attempt was made to load a program into an invalid range of memory. If bit 4 was set, the program must load above $1FFF and eight bytes below the end of contiguous memory. If bit 5 was set, the program must load within the range $0000-$FFFF, inclusive, in the User Memroy Map of an EXORciser II system with the dual memory map configured. If bit 7 was set, the program must load within the range $20-$FFFF, inclusive.

B = 6 indicates that the starting execution address is invalid. The starting execution address must be within the range of memory loaded by the program.

B = a diskette controller error status ($31-$39) if a diskette controller error occurred during the load attempt. This status can only be returned if control was to be passed back to the calling program (Bits 0-2 all zero and Bit 5 zero in entry value of B) or if the program was to be loaded into the User Memory Map of a dual memory map configuration and executed (Bit 5 set to one and bits 0 or 2 set to 1). Otherwise, any diskette controller errors that are detected while the program is being loaded will cause the two-character diskette controller error message to be displayed and control passed to the debug monitor. These two-character error messages are discussed in detail in section 28.1.

X is unchanged if control is returned to the calling program (Bits 0-2 all zero in entry value of B). Otherwise, X will contain the starting load address of the program (lowest address loaded into).

C = 0 if no errors occurred (B = 0). The remainder of CC is indeterminate.

C = 1 if an error occurred (B not zero). The remainder of CC is indeterminate.

S is configured depending on which range of memory is loaded into. If loading above the resident MDOS (Bit 4 set to one in entry value of B), the stack pointer will contain the highest address loaded into (eight bytes greater than the highest program location; twenty bytes for MDOS09). If loading over the resident MDOS or into discontiguous memory (Bit 7 set to one in entry value of B), the stack pointer will contain the address of the EXbug stack area. If loading into the User Memory Map of an EXORciser II system with the dual memory map configured, the stack pointer will contain the highest address loaded into.

The DFT has been changed as if a directory search has been performed (.DIRSM with B = 2). In addition, RES contains the starting load address and DEN contains the starting execution address as found in the file's RIB. The DFT contents can only be accessed if control is returned to the calling program.

If the resident debug monitor is given control (Bit 1 set to one in entry value of B), the pseudo registers are initialized as follows:

Pseudo register	Contents
P	Starting execution address
S	See description of S above. Contents vary depending on load mode.
X	Starting load address.
A, B, C	Indeterminate.
Y	Indeterminate (MDOS09)
U=S	MDOS09 only
DP=0	MDOS09 only

This feature facilitates starting the execution of a program from the debug monitor since the starting execution address need not be remembered by the operator; however, caution must be exercised if programs are loaded into the User Memory Map of an EXORciser II with the dual memory map configured. Since the stack pointer contains the address of the last loaded program location, use of the debug commands ";P" or ";N" will cause seven locations of the program to be destroyed. This may alter program data or instructions. It is recommended that the stack pointer first be changed via the ";S" command; that the "nnnn;G" command be used to initiate execution; or that stack area be provided at the end of the program area. For programs not loaded into the User Memory Map of an EXORciser II system with the dual memory map configured, this precaution does not apply.

Particular attention should be placed on programs that load into the highest memory address $FFFF. Since the diskette controller can only load programs in a multiple of eight bytes, such programs should have a starting load address that is a multiple of eight. Otherwise, the calculated ending load address will be greater than $FFFF, causing an error.

Caution must also be exercised if MDOS is to be reinitialized from the debug monitor after having loaded a program. The ABORT or RESTART pushbuttons must first be depressed before the debug command "E800;G" or "MDOS" is executed.

27.4.4 Allocate diskette space -- .ALLOC

The .ALLOC function allocates contiguous segments of diskette space for a file. The file's Retrieval Information Block and the system's Cluster Allocation Table are updated to account for the allocated space. Since space allocation is performed automatically by the device independent I/O functions, the .ALLOC function should only be used by programs that are doing physical sector I/O on MDOS compatible diskettes.

ENTRY PARAMETERS: X = The address of the DFT.

The DFT must contain the following parameters:

LUN must contain the logical unit number on which the file resides (ASCII number 0-3, $30-$33).

RIB must contain the physical sector number of the file's RIB if the directory entry has already been created (additional space allocation). Otherwise, RIB must contain the value zero to indicate that no Retrieval Information Block exists for the file (initial space allocation).

FDF should have the "C" bit set to indicate whether space is to be allocated contiguously to the already existing space (RIB not zero). If the "C" bit is set to zero, additional space can be allocated anywhere on the diskette. If RIB is zero, the FDF entry is not required.

SIZ must contain the number of sectors that are to be allocated. If SIZ is zero, the default allocation size (32 clusters) will be used.

SBS must contain the starting address of a 128 (decimal) byte sector buffer.

SBE must contain the ending address of the sector buffer. If the sector buffer is larger than one sector, only the first 128 bytes will be used.

EXIT CONDITIONS: A is indeterminate.

B contains the return status. The return statuses are taken from the set of codes defined for the device independent I/O functions. Only the system symbols are given here for those return statuses. The exact values can be found from the MDOS equate file, section 25.3.1.1, or section 28.3. The following return statuses are defined:

B = 0 indicates that no errors occurred (normal return).

B = I$RIB indicates that the file had an existing Retrieval Information Block that was invalid (see section 24.2).

B = I$FSPC indicates that insufficient space is available to accommodate the allocation requirements. If SIZ contained a non-zero value at the entry to .ALLOC, this error indicates that the specific amount of space requested could not be allocated. This can occur for two reasons. First, if the file is segmented ("C" of FDF set to zero), the number of sectors specified in SIZ could not be allocated in a single, contiguous block anywhere. Second, if the file is contiguous ("C" of FDF set to one), the number of sectors specified in SIZ could not be allocated contiguously with the existing space. If SIZ contained a zero value, this error indicates that no space is available at all on the diskette, or that no space is available that is contiguous to the existing space, depending on "C" being zero or one in FDF. If the default of 32 clusters (SIZ = 0) cannot be allocated, .ALLOC will allocate whatever space it can without generating an error. If SIZ is non-zero, an error will be generated if the exact number of sectors cannot be allocated.

B = I$SSPC indicates that the file's Retrieval Information Block could not accommodate the required number of SDWs for the requested allocation. This error occurs if a file is very fragmented.

X is unchanged.

C = 0 if no errors occurred (B = 0). The remainder of CC is indeterminate.

C = 1 if an error occurred (B not zero). The remainder of CC is indeterminate.

The DFT is unchanged if an error

occurred. If no errors occurred, the DFT has been changed in the following manner. Bytes 3 and 4 contain the SDW of the last allocated segment. Bytes 5 and 6 contain the starting, logical sector number of the last allocated segment. SUF contains the logical sector number of the logical end-of-file, and RIB, if originally zero, contains the physical sector number of the file's Retrieval Information Block. The contents of the sector buffer are indeterminate.

27.4.5 Deallocate diskette space -- .DEALC

The .DEALC function deallocates segments of diskette space from a file. The file's Retrieval Information Block and the system's Cluster Allocation Table are updated to account for the deallocated space. Since space deallocation is performed automatically by the device independent I/O functions, the .DEALC function should only be used by programs that are doing physical sector I/O on MDOS compatible diskettes.

ENTRY PARAMETERS: X = The address of the DFT.

The DFT must contain the following parameters:

LUN must contain the logical unit number on which the file resides (ASCII number 0-3, $30-$33).

Bytes 1 and 2 must contain the file's logical sector number beyond which space is to be deallocated. If these two bytes contain the value $FFFF, then the entire space belonging to the file will be deallocated; however, in this special case, the file's directory entry must already have been flagged as deleted.

RIB must contain the physical sector number of the file's Retrieval Information Block.

DEN must contain the file's directory entry number.

SBS must contain the starting address of a 128 (decimal) byte sector buffer.

SBE must contain the ending address of the sector buffer. If the sector buffer is larger than one sector, only the first 128 bytes will be used.

EXIT CONDITIONS:

A is indeterminate.

B contains the return status. The return statuses are taken from the set of codes defined for the device independent I/O functions. Only the system symbols are given here for those return statuses. The exact values can be found from the MDOS equate file, section 25.3.1.1, or section 28.3. The following return statuses are defined:

B = 0 indicates that no errors occurred (normal return).

B = I$RIB indicates that the file had an existing Retrieval Information Block that was invalid (see section 24.2).

B = I$RANG indicates that the maximum referenced logical sector number specified in bytes 1 and 2 does not belong to the file. That is, the LSN specified is greater than the number of sectors belonging (allocated) to the file.

B = I$IDEN indicates that an invalid DEN was specified.

B = I$DEAL indicates that an attempt was made to deallocate all of a file's space (bytes 1 and 2 set to $FFFF), but the directory entry for the file was not flagged as deleted.

X is unchanged.

C = 0 if no errors occurred (B = 0). The remainder of CC is indeterminate.

C = 1 if an error occurred (B not zero). The remainder of CC is indeterminate.

The DFT is only changed if the all of a file's space was to be deallocated. In that case, RIB will contain the value zero. Otherwise, the DFT is unchanged. The contents of the sector buffer are indeterminate.

27.4.6 Display system error message -- .MDERR

The .MDERR function displays on the system console one of the standard system error messages contained in the MDOS error message file. The error message to be displayed is indicated by an index number which is passed in one of the registers. This index number will also be used to modify the system error status word (see section 28.4).

Certain error messages contain references to external parameters that must be supplied by the calling program (e.g., a file name specification or an address). These parameters are shown in the list of error messages below as a backslash character (\) followed by a numeric digit which indentifies the format of the parameter. When an external parameter reference is encountered in the message, the corresponding parameter from the calling program will be inserted into the message before it is displayed on the system console. The following external parameters are defined:

Parameter reference	Calling program specification
\0	The X register contains the address of a standard MDOS file name. Eleven bytes comprise an MDOS file name: logical unit number (1 byte), file name (eight bytes), suffix (two bytes).
\1	The X register's contents are to be converted into four displayable hexadecimal digits.
\3	The X register contains an address of a byte in memory whose contents are to be converted into two displayable hexadecimal digits.
\8	The return address on the stack is decremented by two (pointing to the system call of the error message function) and converted into four displayable hexadecimal digits. This parameter allows the location of the call to .MDERR to be incorporated into the error message for system diagnostic purposes.

The following table lists the standard error messages from the MDOS error message file in order of their error message index numbers (number required as entry parameter to display the message). This number is not to be confused with the two-digit decimal reference number that is displayed with each message on the system console. The displayed reference number only serves as a quick way of locating the error messages' descriptions in Chapter 28.

INDEX NUMBER	ERROR MESSAGE
02	** 40 DIRECTORY SPACE FULL
03	** 41 INSUFFICIENT DISK SPACE
04	** 29 INVALID LOGICAL UNIT NUMBER
05	** 02 NAME REQUIRED
06	** 03 \0 DOES NOT EXIST
07	** 25 INVALID FILENAME
08	** 05 \0 DUPLICATE FILE NAME
09	** 28 DEVICE NAME NOT FOUND
0A	** 31 INVALID DEVICE
0B	** 01 COMMAND SYNTAX ERROR
0C	** 46 INTERNAL SYSTEM ERROR AT \8
0D	** 07 OPTION CONFLICT
0E	** 12 INVALID TYPE OF OBJECT FILE
0F	** 13 INVALID LOAD ADDRESS
10	** 42 SEGMENT DESCRIPTOR SPACE FULL
11	** 32 INVALID RIB
12	** 30 INVALID EXECUTION ADDRESS
13	** 14 INVALID FILE TYPE
14	** 36 FILE EXHAUSTED BEFORE LINE FOUND
15	** 24 LOGICAL SECTOR NUMBER OUT OF RANGE
16	** 34 INVALID START/END SPECIFICATIONS
17	** 35 INVALID PAGE FORMAT
18	** 38 INVALID LINE NUMBER OR RANGE
19	** 39 LINE NUMBER ENTERED BEFORE SOURCE FILE
1A	** 06 DUPLICATE FILE NAME
1B	** 04 FILE NAME NOT FOUND
1C	** 10 FILE IS DELETE PROTECTED
1D	** 33 TOO MANY SOURCE FILES
1E	** 16 CONFLICTING FILE TYPES
1F	** 15 \0 HAS INVALID FILE TYPE
20	** 27 \0 IS WRITE PROTECTED
21	** 47 INVALID SCALL
22	** 18 DEVICE ALREADY RESERVED
23	** 19 DEVICE NOT RESERVED
24	** 11 DEVICE NOT READY
25	** 20 INVALID OPEN/CLOSED FLAG
26	** 21 END OF FILE
27	** 17 INVALID DATA TRANSFER TYPE
28	** 37 END OF MEDIA

INDEX NUMBER	ERROR MESSAGE
29	** 22 BUFFER OVERFLOW
2A	** 23 CHECKSUM ERROR
2B	** 26 FILE IS WRITE PROTECTED
2C	** 43 INVALID DIRECTORY ENTRY NO. AT \8
2D	** 44 CANNOT DEALLOCATE ALL SPACE, DIRECTORY ENTRY EXISTS AT \8
2E	** 45 RECORD LENGTH TOO LARGE
2F	** 48 CHAIN OVERLAY DOES NOT EXIST
30	** 08 CHAIN ABORTED BY BREAK KEY
31	** 09 CHAIN ABORTED BY SYSTEM ERROR STATUS WORD
32	** 49 CHAIN ABORTED BY ILLEGAL OPERATOR
33	** 50 CHAIN ABORTED BY UNDEFINED LABEL
34	** 51 CHAIN ABORTED BY PREMATURE END OF FILE
35	** 52 SECTOR BUFFER SIZE ERROR
36	** 53 INSUFFICIENT MEMORY

In addition, two error messages have specific calling sequences. These two messages have the following format when displayed:

INDEX NUMBER	ERROR MESSAGE
00	**UNIF. I/O ERROR -- STATUS = \3 AT \8
01	**PROM I/O ERROR -- STATUS = \3 AT h DRIVE i - PSN j

The first case (index number 00) should be used for displaying standard error messages as a result of the device independent I/O functions. The .MDERR function expects the X register to contain the address of an IOCB. The status byte of the IOCB will be decoded into one of the standard system error messages shown above. In the event that an illegal status code is contained in the IOCB, the error message will take on the form as shown above. The "\3" parameter will contain the value of the status byte, and the "\8" parameter will contain the address of the call to the error message function.

The second case (index number 01) should be used for displaying standard diskette controller error messages (as returned by .EREAD, .EWRIT, .MERED, .MEWRT). The .MDERR function expects the X register to contain the address of a three-byte packet. The format of the packet is shown below:

```
        ------------------------------
   0    | Controller error status |
        ------------------------------
   1    |        Address of           |
        --     function call         --
   2    | to sector I/O function  |
        ------------------------------
```

In addition, the .MDERR function will pick up the logical unit number and the physical sector number from the diskette controller variables in locations $0000-$0002, inclusive. When the error message is displayed, the parameter "h" will have been replaced with the address of the call to the error message function, the parameter "i" will have been replaced with the logical unit number, and the parameter "j" will have been replaced with the physical sector number at which the error occurred.

ENTRY PARAMETERS: B = The index number of the error message as shown in the above tables.

X may not have to be parameterized. If the error message calls for an external parameter, X will have to contain the parameter or the address of the parameter that is to be placed into the error message. The contents of X depend on the type of message displayed as shown in the above tables.

EXIT CONDITIONS: A is indeterminate.

B is indeterminate.

X is indeterminate.

C = 0. The remainder of CC is indeterminate.

The Error Type of the system error status word has been changed to contain the index number of the displayed error message. In addition, the Error Status Flag of the system error status word has been set to one. Section 28.4 contains a complete description of the system error status word.

If the .MDERR function is called with an index number for which no valid error message exists, or if the MDOS error message file cannot be accessed on the diskette without an error, a special message will be displayed. This message has the format:

** INVALID MESSAGE \3 AT \8

The "\3" parameter will have been replaced with the index number of the error message that the .MDERR function was trying to display. This may or may not be a valid index number, depending on whether or not the MDOS error message file could be properly accessed. The "\8" parameter will have been replaced with the address of the call to the .MDERR system function. In the event that this message is displayed, the Error Type portion of the system error status word will contain the value $FF (the Error Status Flag will also be set to one).

27.5 Other Functions

The remaining system functions are so diverse that they fail to fall into one of the previous categories. These functions are used by the MDOS commands and are available for user programs in order to extract file name or device specifications from the MDOS command line, allocate program memory in the remaining block of contiguous memory, set the system error status word when non-standard error messages are displayed so that CHAIN processing will work properly, and to return control to the MDOS command interpreter.

27.5.1 Process file name -- .PFNAM

The .PFNAM function scans a specified input buffer for a file name or device specification. The information is returned in a format which is called the standard MDOS file name format. This format fits into the other parameter tables required by the device independent I/O functions (IOCB) and the diskette file functions (DFT). The .PFNAM function will also recognize family indicators in either the file name or the suffix.

Due to the nature of the free format of the MDOS command line, any character that will not be confused with a device name indicator, a family indicator, a suffix delimiter, a logical unit delimiter, an option field delimiter, or an end of line delimiter will be used to terminate the scan for a valid file name or device specification.

The scan will never continue beyond an option delimiter (;) or an end of line delimiter (carriage return), regardless of the number of times .PFNAM is called with the scan pointer pointing to such a character.

ENTRY PARAMETERS: X = The address of a file name packet.

This packet has the following format:

```
       -----------------------
  0  !      Address of       !
     --    input buffer     --
  1  !                       !
       -----------------------
  2  !      Address of       !
     --      standard       --
  3  !    file name area     !
       -----------------------
```

Since .PFNAM is designed to be called more than once to extract multiple file name or device specifications from a single input buffer, the first pointer of the file name packet, or scan pointer, must be pointing to a character which previously terminated the scan. When .PFNAM is called the first time, special care must be taken to ensure that the first byte of the input buffer is a valid terminator (this is automatically handled by the MDOS command interpreter in using the MDOS command line buffer). This character is normally a space or a comma; however, any other valid terminator will suffice.

The second pointer of the file name packet defines where the standard file name is to be placed. This area must be eleven bytes long. The first byte will contain the logical unit number. The next eight bytes will contain the device name or the file name, and the last two bytes will contain the suffix.

EXIT CONDITIONS:

A = The character that terminated the scan.

B contains the return status. The following return statuses are defined:

B = 0 indicates that a standard MDOS file name specification was found.

Bit 0 = 1 indicates that a family indicator was found in the file name.

Bit 1 = 1 indicates that a family indicator was found in the suffix.

Bit 2 = 1 indicates that a device specification was found.

Bits 3-6 are unused and will be zero.

Bit 7 = 1 indicates a null file name was found. This does not necessarily mean that a null suffix or a null logical unit number was found.

X is unchanged.

CC is indeterminate.

The scan pointer (first two bytes of file packet) will contain the address of the character that terminated the scan.

The standard file name pointer (second two bytes of file packet) will have been incremented by eleven (points to location following the suffix).

The standard file name area is only changed if a corresponding element is found in the input buffer. Thus, if no logical unit number is found in the input buffer, the logical unit part of the standard file name area will not be changed. The same is true for the file name and for the suffix fields. This feature allows appropriate default values for the logical unit number, file name, and suffix to be placed into the standard file name area before .PFNAM is invoked. Then, after the input buffer is scanned, those parts of the file name specification which were not explicitly found will assume the default values which were unchanged.

No delimiters of any sort are placed into the standard file name area. The presence of device name indicators and family indicators is indicated by the return status in the B register only. The file name (or device name) and suffix will be left justified within the file name area. Unused parts of the file name or suffix will be space-filled automatically.

When the scan is initiated, leading spaces in front of the file name or device specification will be treated as a single space (ignored). Any space, however, encountered after the first character of a specification is found will be treated as a terminator.

If the file name, suffix, or logical unit number contains more valid characters than required, they will be automatically flushed from the input stream. Thus, even if a ten character file name is specified, only the first eight characters will be returned in the file name area.

The following examples illustrate how .PFNAM extracts the file name or device specification from the input buffer. The left column shows a string as it is encountered in the input buffer. The double quotation marks delimit the start and end of the string. It should be noted that an initial terminator begins each string. The right column shows the extracted information as it would appear in the standard file name area. The dashes indicate unchanged parts of the standard file name area (those areas where the default values would be found).

Input string	Extracted file name
" FILE,"	-FILE --
" FILE1:0,"	0FILE1 --
" F.SA,"	-F SA
" FILE.RO:1,"	1FILE RO
" :0,"	0----------
" .LX:1,"	1--------LX
" FILENAMETOOLONG.AB:1,"	1FILENAMEAB
" FILE$AB:1,"	-FILE --
" #LP,"	-LP --
" #UD:1,"	1UD --
" FILE*.*:1,"	1FILE --

27.5.2 Re-enter resident MDOS -- .MDENT

The .MDENT function passes control from a calling program to the MDOS command interpreter. It is one of the few functions which does not return control to the calling program. .MDENT can only be used if the resident operating system area has not be changed by the calling program (or any programs that may have executed prior to it).

ENTRY PARAMETERS:	The diskette in drive zero must not have been replaced with another diskette since the last time MDOS was initialized via the resident debug monitor.
EXIT CONDITIONS:	There is no return from this function; however, the following actions are performed:
	The SWI and IRQ vectors are configured for the MDOS function handler.

The user SWI and IRQ vectors maintained by MDOS (SWI$UV and IRQ$UV) are reset to point to an RTI instruction. The user program is no longer resident, thus user-defined SWI and IRQ interrupts cannot be processed after MDOS regains control.

The end of user memory pointer, END$US, is reset.

The command line buffer is initialized.

The version/revision numbers of MDOS in memory are compared with the version/revision numbers in the ID sector. The addresses of the system overlays are also compared in this fashion. If a discrepancy exists between memory and the diskette, EXbug is given control.

The system IOCBs for the console, printer, and the MDOS error message file are configured.

The input prompt (=) is displayed and a new command line accepted from the system console.

The system error status word is cleared (Error Type and Error Status Flag) if a valid command is interpreted.

27.5.3 Reload MDOS from diskette -- .BOOT

The .BOOT function reloads the resident operating system from the diskette in drive zero via the diskette controller firmware. This function should be used if the resident operating system has been changed by the current program (SWI handler must still be intact). This function should also be used if the diskette in drive zero has been replaced with another MDOS diskette since the last time MDOS was initialized via the debug monitor. .BOOT is one of the few functions that does not return control to the calling program.

This function has the same effect as if the ABORT or RESTART pushbuttons were depressed on the EXORciser and the debug command "E800;G" or "MDOS" executed.

ENTRY PARAMETERS: A valid MDOS diskette must be ready in drive zero.

EXIT CONDITIONS: This function does not return to the calling program. A new copy of MDOS is brought from the diskette into memory. All of the functions performed during this type of initialization are described in section 2.1 and section 24.6. Control is given to the MDOS command interpreter after MDOS has been initialized.

27.5.4 Set system error status word -- .EWORD

The .EWORD function configures the system error status word with a specific error type. This allows a calling program to indicate that an error occurred during its execution. The system error status word can then be tested from within a CHAIN procedure (Chapter 6).

ENTRY PARAMETERS: B = The value that is to be placed into the Error Type field of the system error status word. Any value is valid. Section 28.4 describes the format of the error status word.

EXIT CONDITIONS: A is unchanged.

B is unchanged.

X is unchanged.

CC is indeterminate.

The lower byte of the system error status word contains the value passed in B. The Error Status Flag has also been set to one. The remainder of the error status word is unchanged.

27.5.5 Allocate user program memory -- .ALUSM

The .ALUSM function adjusts the MDOS pointer ENDUS$ to reflect the end of the user program area. This function facilitates the dynamic allocation of variable buffer space adjacent to the highest loaded program location so that programs can take advantage of the variable amount of contiguous memory that may be configured for a given installation.

The user program area consists of all contiguous memory between the end of the resident operating system and the end of contiguous memory. The pointer ENDUS$ is automatically adjusted to reflect the end of a loaded program (only if the program is loaded directly from the command line or via the LOAD command without the "U" or "V" option). Thus, the program can obtain information about the remaining amounts of memory without having to size memory itself.

ENTRY PARAMETERS:

B contains a function code that specifies the action to be taken by .ALUSM. The following function codes (and their impact the the X register) are defined:

B = 0 indicates that the X register contains the address of the last address that is to be made a part of the current user program area.

B = 1 indicates that the X register contains the number of bytes of memory that are to be allocated to the end of the current user program.

B = 2 indicates that all of the remaining contiguous memory is to be allocated to the current user program area.

X contains the parameters as described above.

EXIT CONDITIONS:

A is unchanged.

B contains the return status. The following return statuses are defined:

B = 0 indicates that no errors occurred (normal return).

B = 1 indicates that the allocation request would have caused ENDUS$ to be greater than ENDSY$. The user program area cannot extend beyond the end of contiguous memory in the system.

B = 2 indicates that the allocation request would have caused ENDUS$ to be less than or equal to ENDOS$. The allocated memory block must reside completely above the address contained in ENDOS$.

X contains an indeterminate value if an error occurred (exit value of B not zero) or if the entry value of B was zero.

X contains the old value plus one (value before the call to .ALUSM) of ENDUS$ if the entry value of B was one. Thus, X points to the starting address of the newly allocated block.

X contains the number of bytes allocated if the entry value of B was two.

Z = 1 and C = 0 if no errors occurred (B = 0). The remainder of CC is indeterminate.

Z = 0 and C = 1 if an error occurred (B not zero). The remainder of CC is indeterminate.

The MDOS variable ENDUS$ is unchanged if an error occurred. Otherwise, ENDUS$ will contain the following: if the entry value of B was zero, ENDUS$ will contain the entry value of the X register; if the entry value of B was one, ENDUS$ will have been incremented by the entry value of the X register; and if the entry value of B was two, ENDUS$ will contain the value of ENDSY$.

CHAPTER 28

28. ERROR MESSAGES

This chapter contains a summary and an explanation of all of the standard error messages that can be displayed during the operation of MDOS. Standard error messages include those displayed by the diskette controller firmware during initialization, the PROM I/O messages that can be displayed when any fatal diskette error is detected by an MDOS command or overlay, and the standard error messages displayed by the commands themselves. The standard command error messages are recognizable by the fact that a pair of asterisks followed by two-digit reference number is displayed before the actual message. Explanations of messages without the two-digit number should be looked for in the detailed command descriptions in chapters 3-23.

28.1 Diskette Controller Errors

The diskette controller errors can be displayed in two forms depending on the phase MDOS is in. During the initialization phase, the error messages from the controller take on the form of the letter "E" followed by a decimal digit 0-9. Control is given to the debug monitor after the message is displayed. After MDOS has been properly initialized, the diskette controller errors are identified by the text "PROM I/O ERROR". Control is returned to the MDOS command interpreter.

28.1.1 Errors during initialization

If for some reason the drive electronics are not properly initialized, or if the diskette in drive zero cannot be read properly to load the Bootblock or the resident operating system, then a two-character error message will be displayed and control returned to the debug monitor. The function resulting in the error has been tried five times. After the fifth failure, the error message is displayed.

Message	Probable Cause
E1	A cyclical redundancy check (CRC) error was detected while reading the resident operating system into memory.

E2	The diskette has the write protection tab punched out. During the initialization process, certain information is written onto the diskette. The diskette is not damaged and can still be used for a system diskette; however, the write protection tab must first be covered with a piece of opaque tape to allow writing on the diskette.
E3	The drive is not ready. The door is open or the diskette is not yet turning at the proper speed. If the diskette has been inserted into the drive with the wrong orientation, the "not ready" error will be also generated. This error will also occur if a double-sided diskette is placed into a single-sided diskette drive. Closing the door, waiting a little bit longer before entering the "E800;G" or "MDOS" command, or turning the diskette around so it is properly oriented should eliminate this error.
E4	A deleted data mark was detected while reading the resident operating system into memory.
E5	A timeout interrupt occurred. This indicates that a diskette controller command was not completed within the allotted time. This error is also produced if a non-maskable interrupt (such as depressing the ABORT pushbutton on the EXORciser's front panel) is generated during a diskette operation.

E6	The diskette controller has been presented with a cylinder-sector address that is invalid. This error occurs when the sum of STRSCT and NUMSCT (see Appendix D) is larger than the total number of sectors on the diskette. This error indicates some type of a hardware problem. For example, the error can be caused by missing or overlapping memory, bad memory, or pending IRQs that cannot be serviced.
E7	A seek error occurred while trying to read the resident operating system into memory. Like E6 errors, this one indicates some type of a hardware problem.
E8	A data mark error was detected while trying to read the resident operating system into memory.
E9	A CRC error was found while reading the address mark that identifies sector locations on the diskette.

The diskette controller errors E1, E4, E8, and E9 indicate that the diskette cannot be used to load the operating system; however, a new operating system can be generated on that diskette, making it useful again. The DOSGEN (Chapter 10) and/or FORMAT (Chapter 15) commands should be consulted for generating a new diskette. Depending on the extent of the errors, the diskette may be used in drive one to recover any files that may be on it (see section 2.8.9).

The diskette controller error E5 can occur for a variety of reasons. The most common reason, and the most fatal, is the destruction of the addressing information on the diskette. If the addressing information has been destroyed (verified by using the DUMP command to examine areas of the diskette), the FORMAT command may be used to rewrite the addressing; however, information on the damaged diskette cannot be recovered. Occasionally, after a system has just been unpacked, the read/write head may have been positioned past its normal restore point on cylinder zero. In this case, trying the event which caused the error three or more times may position the head to the proper place. If this fails, the head will have to be manually repositioned past cylinder zero; however, this problem rarely occurs. The E5 errors can also occur if a user-written program accesses drives 1-3 without using one of the system functions and without first restoring the read/write head on that drive.

Even after the resident operating system has been successfully read into memory, certain errors can occur in the subsequent initialization procedure. During initialization the resident operating system cannot access the error message processor since it has not been initialized. Messages similar in format to those generated by the diskette controller are displayed to indicate such errors. They differ from the diskette controller errors in that the second character of the two-character message is a non-numeric character. The following errors can occur during initialization, but only after the resident operating system has been read into memory.

Message	Probable cause
E?	This error indicates that the RIB of the resident operating system file MDOS.SY is in error. The operating system cannot be loaded. The diskette probably is not an MDOS system diskette, or the system files have been moved from their original places. The REPAIR command (Chapter 22) can be used to identify which files are missing or if their places have been changed.
EM	This error indicates that there was insufficient memory to accommodate the resident portion of the operating system. The memory requirements described in section 1.1 should be reviewed. If the minimum requirements are satisfied, then the existing memory should be carefully examined for bad locations.

EI	The version and revision of MDOS already loaded into memory is not the same as that on diskette. This error usually occurs as the result of switching diskettes in drive zero without following the initialization procedure outlined in section 2.1. This error can also occur is the ID sector has been damaged. The error can be avoided if the initialization procedure is followed correctly every time a new system diskette is inserted into drive zero.
ER	The addresses of the RIBs of the MDOS overlays are not the same as those at the time of the last initialization. This error may occur for the same reasons as the "EI" error.
EU	An input/output system function returned an error during the initialization. Errors of this sort indicate a possible memory problem or the opening of the door to drive zero while the initialization is taking place.
EV	One of the system files is missing or cannot be loaded into memory. If a system file is missing, the diskette has been improperly generated or the file was intentionally deleted. If a file cannot be loaded, then the diskette should be regenerated. The diskette may be used in drive one to save any files that may be on it (section 2.8.9). This error may also occur if the door to drive zero is opened while initialization is in progress.

28.1.2 Errors after initialization

If a diskette controller error is detected after MDOS has been initialized, then an error message of the following format will be displayed.

```
**PROM I/O ERROR--STATUS=nn AT h DRIVE i-PSN j
```

This message indicates that an unrecoverable error occurred while trying to access the diskette. The error status "nn" is a value returned by the diskette controller. The errors are of the same type that cause the initialization process to give control to EXbug; however, instead of beginning with the letter "E", the status (nn) begins with the digit "3". The second digit of the status corresponds directly to the diskette controller error number discussed in the previous section. The "E" has been replaced by the "3". Thus, status

```
31 is the same as E1
32 is the same as E2
          .
          .
          .
39 is the same as E9.
```

A memory address (only meaningful for system diagnostics) is substituted for the letter "h"; the logical unit number is substituted for the letter "i"; and the physical sector number (PSN) at which the error occurred is substituted for the letter "j".

For errors that are retryable (status 31, 34, 38, and 39), the following actions have been taken in an attempt to bypass the error. First, the ROM firmware tried to re-access the sector five times. The head was then positioned a maximum of five cylinders outward from the sector in error, repositioned back over the sector, and another five accesses attempted. Lastly, the head was positioned a maximum of five cylinders inward from the sector in error, repositioned back over the sector, and another five accesses attempted.

Occassionally, if the diskette in drive zero was changed without properly reinitializing the system, or if an MDOS system file is moved, renamed, or deleted from the directory, the error messages EI, ER, EU, or EV can be displayed and control given to the debug monitor. These error messages are explained in the previous section.

28.2 Standard Command Errors

The following list contains all of the standard error messages than can be displayed by the MDOS commands. They are listed in order of their two-digit reference number for easy location. This number is not to be confused with the error message index number that is loaded into the B accumulator when the system error message function (.MDERR, section 27.4) is accessed.

In some cases, the error message applies also to user-written programs using the device independent I/O functions. Then, the error condition returned in the IOCB entry IOCSTA (section 25.3.1.20) will contain a value, which when decoded by the .MDERR function, would result in the standard error message being displayed.

The first error message is standard, but is only displayed by the MDOS command interpreter, not by a command. It has no number identifying it. The second error message is only displayed if the MDOS error message function is called with an invalid error message index number, or if the system error message file cannot be accessed without error.

WHAT?

This message indicates that the first file name specification entered on the command line was not the name of a file in the diskette's directory. Most often this error occurs as the result of a mistyped command name.

Some commands, such as DUMP and PATCH, display this message to indicate an unrecognizable command.

** INVALID MESSAGE mm AT nnnn

This message is displayed by the .MDERR system function if it is called with an index number for which no valid error message exists, or if the MDOS error message file cannot be accessed on the diskette without an error. The number "mm" shows the index number of the error message that the .MDERR function was trying to display. The number "nnnn" shows the address of the call to the .MDERR function.

** 01 COMMAND SYNTAX ERROR

The syntax of the command line parameters as seen by the command could not be interpreted. Most often this message refers to undefined characters appearing in the <options> field of the command line.

If this message is displayed during the execution phase of the CHAIN command, it may mean that an execution operator was encountered that had an illegal operand field.

** 02 NAME REQUIRED

One or more of the file names required by the command as parameters was omitted from the command line.

** 03 <name> DOES NOT EXIST

The displayed file name was not found in the diskette's directory. The file must exist prior to using the command. The <name> is displayed to show which file name of the multiple names specified as parameters caused the error.

** 04 FILE NAME NOT FOUND

The file name entered on the command line as a parameter does not exist in the diskette's directory. The file must exist prior to using the command. No file name is displayed since only one parameter is required by the command.

This error can also occur during the FDR processing of the .OPEN function when a file is being opened in the input or update modes.

** 05 <name> DUPLICATE FILE NAME

The displayed file name already exists in the diskette's directory. The file must not exist prior to using the command. The <name> is displayed to show which file name of the multiple names specified as parameters caused the error.

** 06 DUPLICATE FILE NAME

The file name entered on the command line as a parameter already exists in the diskette's directory. The file must not exist prior to using the command. No file name is displayed since only one parameter is required by the command.

This error can also occur during the FDR processing of the .OPEN function when a diskette file is being opened in the output mode.

** 07 OPTION CONFLICT

The specified options were not valid for the type of function that was to be performed by the command. Several of the options are mutually exclusive and cannot be specified at the same time. The specific command descriptions should be consulted for the restrictions concerning the various options.

** 08 CHAIN ABORTED BY BREAK KEY

This message is displayed by the CHAIN command to indicate that the operator depressed the break key during the execution phase, causing it to be aborted.

** 09 CHAIN ABORTED BY SYSTEM ERROR STATUS WORD

The last program invoked from the CHAIN process set an error status into the system error status word which was not masked by a SET operator. If no SET operators are used in a CHAIN file, any error status word change will cause the CHAIN process to be aborted.

** 10 FILE IS DELETE PROTECTED

An attempt was made to delete a file which had the delete protection bit set in its directory entry. The file is not deleted.

** 11 DEVICE NOT READY

Most frequently this error indicates that a command is trying to output to the printer while the printer is not ready or out of paper; however, the message can apply to any of the supported devices whether being used for input or output.

** 12 INVALID TYPE OF OBJECT FILE

Most frequently this message indicates that an attempt was made to load a program into memory from a file which does not have the memory-image attribute.

This message can also indicate that the RIB of a memory-image file has been damaged (LOAD command, Chapter 18).

** 13 INVALID LOAD ADDRESS

This message indicates that an attempt was made to load a program into memory which, depending on the method of loading: 1) loads outside of the range of contiguous memory established at initialization; 2) loads over the resident operating system; 3) loads below hexadecimal location $20; or 4) loads beyond location $FFFF. The latter case implies that the file's RIB may be damaged. If this is the suspected cause, the REPAIR command (Chapter 22) should be used to correct the error. Programs which load into the highest memory address ($FFFF) which do not have a starting load address that is a multiple of eight, can also cause this error.

** 14 INVALID FILE TYPE

The file name entered on the command line as a parameter has the wrong file format (the numeric portion of a displayed directory entry's attribute field) for the intended operation. No file name is displayed since only one parameter is required by the command.

This error can also occur if a binary record transfer is being requested to a device that does not support binary transfers; if a non-record format (e.g., memory-image format) is specified when opening a non-diskette device; or if a non-ASCII record format is specified when using the non-file format mode.

** 15 <name> HAS INVALID FILE TYPE

The displayed file name has the wrong file format (the numeric portion of a displayed directory entry's attribute field) for the intended operation. The <name> is displayed to show which file name of the multiple names specified as parameters caused the error.

The MERGE command (Chapter 19) can display this message if a memory-image file has an invalid RIB. The REPAIR command (Chapter 22) should be used to correct the error.

** 16 CONFLICTING FILE TYPES

A command was expecting files of the same format. The files specified have different file formats and/or attributes.

** 17 INVALID DATA TRANSFER TYPE

An attempt was made to read from an output device or file, to write to an input device or file, to perform record I/O with the logical sector mode set, to perform logical sector I/O with the record mode set, to open a non-input/output device in the update mode, or to open a non-diskette device in the update mode.

** 18 DEVICE ALREADY RESERVED

Bit "R" of the IOCLUN byte in an IOCB was set to one when the .RESRV system function was called.

** 19 DEVICE NOT RESERVED

Bit "R" of the IOCLUN byte in an IOCB was set to zero when the .OPEN or .RELES system functions were called.

** 20 INVALID OPEN/CLOSED FLAG

Bit "O" of the IOCDTT byte in an IOCB was set to one when the .CLOSE, .GETRC, .GETLS, .PUTRC, .PUTLS, .REWND, or .RELES system function was called, or bit "O" of the IOCDTT byte was set to zero when the .OPEN system function was called.

** 21 END OF FILE

An end-of-file record was read from a non-diskette device or an attempt was made to read beyond the logical end-of-file in a diskette file. Attempting to read from a diskette file after the end-of-file error has occurred will result in the same error. Reading from a device after the end-of-file error occurred may or may not result in the same error, depending on what caused the initial end-of-file condition. Reading a record from a diskette file which contains no carriage returns will result in this error.

** 22 BUFFER OVERFLOW

An attempt was made to read a record which was larger than the data buffer provided for the record. The overflow of the record is truncated.

During the CHAIN command's execution phase, a supplied input response exceeded the maximum number of characters acceptable for the input request.

** 23 CHECKSUM ERROR

A binary record or an ASCII-converted-binary record was read whose calculated checksum did not agree with the checksum byte contained in the record.

This error can also occur during the FDR processing of the .OPEN function. If the file format mode is specified, and the device is read in search of an FDR, any record that begins with the FDR header character but which is not an FDR (e.g., created in non-file format mode) will cause this error.

** 24 LOGICAL SECTOR NUMBER OUT OF RANGE

An attempt was made to read a logical sector beyond the physical end of the file. The physical end of the file is the highest numbered logical sector allocated to the file. This error can also be caused if the IOCSDW and IOCSLS entries of the IOCB are changed by the calling program after the file has been opened.

** 25 INVALID FILE NAME

A file name was specified that contained the family indicator (*), began with a device name indicator (#), or began with a non-alphabetic character.

The NAME command (Chapter 20) limits the use of the family indicator. Failure to do so may result in this error.

** 26 FILE IS WRITE PROTECTED

An attempt was made to write into a file which has the write protection attribute set in its directory entry.

This error can also be caused by attempting to open a diskette file in the update mode which already has the write protection bit set.

** 27 <name> IS WRITE PROTECTED

The file <name> had the write protection attribute set in its directory entry when an attempt was made to write to the file.

** 28 DEVICE NAME NOT FOUND

A device name was specified which is not defined as an MDOS-supported device. This usually occurs if the device name is mistyped. The valid device names for the I/O functions are CN, CR, CP, DK, and LP. If a logical unit number is specified for a proper device that is greater than the number of units present for that device, then this error may also occurr (e.g., specifying units greater than 3 for for diskette drives or units greater than 0 for other devices).

The COPY command (Chapter 7) will also accept the device names HR and UD.

** 29 INVALID LOGICAL UNIT NUMBER

A logical unit number was specified that is invalid. If the device is a diskette, the valid logical unit numbers are zero through three. For non-diskette supported devices only logical unit numbers of zero are allowed.

** 30 INVALID EXECUTION ADDRESS

The starting execution address of a program in a memory-image file is less than the lowest address or greater than the highest address loaded into by the program. This indicates a RIB error. The REPAIR command (Chapter 22) should be used to correct the error.

The EXBIN command (Chapter 14) uses this message to refer to an illegal specification of an execution address in the options field (i.e., a non-hexadecimal digit).

** 31 INVALID DEVICE

A valid device name was used in an illegal context. For example, the device LP cannot be used in the context of an input device. The name DK cannot be used on the command line of any of the MDOS commands. The COPY command does not allow the CN device to be used as an input specification.

This message can also indicate an attempt to perform logical sector I/O on a non-diskette device, or an attempt to perform non-file format I/O on a device that does not support the non-file format mode.

If a non-standard device is being interfaced to the system using the device independent I/O functions, this error can indicate that the IOCGDW entry of an IOCB (address of CDB) is zero, or that the address of the software driver (CDBSDA of CDB) is zero.

** 32 INVALID RIB

An attempt was made to open a file (usually a memory-image file) that has an invalid RIB. The criteria for a valid RIB are explained in detail section 24.2. The REPAIR command (Chapter 22) should be used to correct the error.

** 33 TOO MANY SOURCE FILES

More file names were specified on the command line than could be accommodated by a command which can accept multiple file names as parameters.

** 34 INVALID START/END SPECIFICATIONS

The start and end specifications entered on the command line for the LIST command did not start with the letters "S" or "L". This error can occur if the starting specification starts with "S" and the ending specification starts with "L", or vice versa. If the end specification has a value less than the value of the start specification, then this error will also occur.

** 35 INVALID PAGE FORMAT

A non-standard page format was specified which had an invalid number of columns/line or lines/page. The specific command description should be consulted for the limits of these specifications.

** 36 FILE EXHAUSTED BEFORE LINE FOUND

A start specification entered on the command line of the LIST command (Chapter 17) specified a physical line number whose value was larger than the total number of lines in the file. The same type of error can be caused by a line number specification in a BLOKEDIT command file (Chapter 5).

** 37 END OF MEDIA

A File Descriptor Record was being searched for on a non-diskette device or a record output transfer was taking place on a non-diskette device when the device ran out of medium (e.g., end of cassette or paper tape).

** 38 INVALID LINE NUMBER OR RANGE

A line number was encountered in the BLOKEDIT command file (Chapter 5) which did not begin with an asterisk, a double quote, a decimal digit (0-9), or an alphabetic character (A-Z), and the line was not a quoted line. If the command line started with a digit, then the physical line number had a value outside of the range 1-65535, or the starting number of a line number range was greater than the ending line number of the range.

** 39 LINE NUMBER ENTERED BEFORE SOURCE FILE

A line number was encountered in the BLOKEDIT command file (Chapter 5) before an input file was opened.

** 40 DIRECTORY SPACE FULL

An attempt was made to add a new entry to the directory when no empty directory entry could be found (first byte equal to zero or to $FF). The directory can accommodate 160 (decimal) entries.

** 41 INSUFFICIENT DISK SPACE

While trying to write to a file or close a file, an allocation request for more space returned with insufficient room to accommodate the space requirements. This can occur when trying to extend a file whose attributes demand contiguous space allocation. In this case, even though more space may be available on the diskette than is actually required, the space is not adjacent to the already allocated space. This error can also occur when trying to create a file with contiguous allocation on a diskette where the largest available contiguous block is smaller than the requested size. This error can also occur if the diskette is 100% full when a new file is being created or when an existing file is attempting to expand by even a single sector. File reorganization (section 3.3) will consolidate fragmented space, possibly increasing the size of the available contiguous space.

** 42 SEGMENT DESCRIPTOR SPACE FULL

During an allocation request for additional space, the file's Retrieval Information Block was found to have the maximum number of Segment Descriptors already in use. File reorganization (section 3.3) will consolidate segment descriptors.

** 43 INVALID DIRECTORY ENTRY NO. AT nnnn

An IOCB (or DFT) contained a value in its IOCDEN (or DEN) entry which was outside of the allowable limits of valid directory entry numbers. The address "nnnn" gives the location of the call to the error message function.

** 44 CANNOT DEALLOCATE ALL SPACE, DIRECTORY ENTRY EXISTS AT nnnn

This message indicates a hardware or system software malfunction if generated by one of the MDOS commands. A directory entry must be flagged as deleted prior to having the file's space deallocated. The address "nnnn" gives the location of the call to the error message function.

** 45 RECORD LENGTH TOO LARGE

An attempt was made to write a binary record or an ASCII-converted-binary record which had more than 254 (decimal) data bytes.

** 46 INTERNAL SYSTEM ERROR AT nnnn

This message indicates a hardware or system software malfunction. Careful notes should be made regarding the events leading up to this error. Motorola Microsystems should be notified. The address "nnnn" gives the location of the call to the error message function.

** 47 INVALID SCALL

This message indicates that a program attempted to access the MDOS SWI (system function) handler with a function byte following the SWI instruction that is not defined. If breakpoints are patched into memory without using the EXbug command "nnnn;V", this error may occur if the SWI vector is still configured for MDOS functions.

** 48 CHAIN OVERLAY DOES NOT EXIST

The CHAIN overlay's file name does not exist in the directory. The REPAIR command (Chapter 22) should be used to check the diskette for other errors.

** 49 CHAIN ABORTED BY ILLEGAL OPERATOR

An illegal execution operator was encountered in the intermediate file during the CHAIN command's execution phase.

** 50 CHAIN ABORTED BY UNDEFINED LABEL

A JMP execution operator was encountered which referenced a label that did not exist in the intermediate file (forward direction only) during the CHAIN command's execution phase.

** 51 CHAIN ABORTED BY PREMATURE END OF FILE

An access to the intermediate file returned an end-of-file condition when an input request was made by a program that was invoked by the CHAIN process. All input that is expected by the program must be supplied by the intermediate file.

** 52 SECTOR BUFFER SIZE ERROR

The sector buffer pointers of an IOCB do not describe a sector buffer that is an integral number of sectors in size. When a file is opened, the IOCSBS and the IOCSBE entries of the IOCB must point to the first and last bytes of a sector buffer. The following relationship must be true:

$$\frac{\text{IOCSBE-IOCSBS+1}}{128} = \text{INTEGRAL NUMBER OF SECTORS}$$

When using the logical sector I/O functions (.GETLS, .PUTLS), the above relationship must be true also. In addition, the .PUTLS function requires that the sector buffer to be output be described by the pointers IOCSBS and IOCSBI (instead of IOCSBE). Then, the buffer described by IOCSBS and IOCSBI must also be an integral number of sectors in size.

** 53 INSUFFICIENT MEMORY

This message indicates that a command could not allocate sufficient memory in the user program area to complete its task. The minimum memory requirements described in section 1.1 is sufficient for all MDOS commands. Thus, this message indicates a problem with the existing memory, or tampering with the memory map. The same is true for the MDOS-Supported software products that display this message; however, the memory requirements for the particular product that displayed the error message should be reviewed (Appendix H), rather than those for the standard MDOS commands in section 1.1.

The ROLLOUT command (Chapter 23) may display this message to indicate that the address given as the destination of the position-independent routine is outside of a valid addressing range (missing memory).

28.3 Input/Ouput Function Errors

The MDOS system functions that perform I/O through an IOCB parameter table will return an error status in the IOCSTA entry of the IOCB. These error conditions can be decoded and displayed as messages by the MDOS error message function by loading the B accumulator with a zero and leaving the IOCB's address in the X register. The errors are part of the standard error messages explained above. This section contains the system symbols from the MDOS equate file that are used to reference the I/O errors. The following table shows the value of the IOCSTA byte, the system symbol equated to that value from the MDOS equate file, and the error message.

IOCSTA Value	System Symbol	Standard Error Message Displayed by .MDERR (B=0, X=IOCB address)
00	I$NOER	Normal return, no error
01	I$NODV	** 28 DEVICE NAME NOT FOUND
02	I$RESV	** 18 DEVICE ALREADY RESERVED
03	I$NORV	** 19 DEVICE NOT RESERVED
04	I$NRDY	** 11 DEVICE NOT READY
05	I$IVDV	** 31 INVALID DEVICE
06	I$DUPE	** 06 DUPLICATE FILE NAME
07	I$NONM	** 04 FILE NAME NOT FOUND
08	I$CLOS	** 20 INVALID OPEN/CLOSED FLAG
09	I$EOF	** 21 END OF FILE
0A	I$FTYP	** 14 INVALID FILE TYPE
0B	I$DTYP	** 17 INVALID DATA TRANSFER TYPE
0C	I$EOM	** 37 END OF MEDIA
0D	I$BUFO	** 22 BUFFER OVERFLOW
0E	I$CKSM	** 23 CHECKSUM ERROR
0F	I$WRIT	** 26 FILE IS WRITE PROTECTED
10	I$DELT	** 10 FILE IS DELETE PROTECTED
11	I$RANG	** 24 LOGICAL SECTOR NUMBER OUT OF RANGE
12	I$FSPC	** 41 INSUFFICIENT DISK SPACE
13	I$DSPC	** 40 DIRECTORY SPACE FULL
14	I$SSPC	** 42 SEGMENT DESCRIPTOR SPACE FULL
15	I$IDEN	** 43 INVALID DIRECTORY ENTRY NO. AT nnnn
16	I$RIB	** 32 INVALID RIB
17	I$DEAL	** 44 CANNOT DEALLOCATE ALL SPACE, DIRECTORY ENTRY EXISTS AT nnnn
18	I$RECL	** 45 RECORD LENGTH TOO LARGE
19	I$SECB	** 52 SECTOR BUFFER SIZE ERROR

28.4 System Error Status Word

Within the operating system's resident variables is a two-byte error status word. Each MDOS command will set or clear a bit within this status word to indicate the status of the command's completion. The error status word has the following format:

```
  F  E  D  C  B  A  9  8  7  6  5  4  3  2  1  0
 ------------------------------------------------
 !           !          !                       !
 ! Error     ! Error    !      Error Type       !
 ! Status    ! Mask     !                       !
 !           !          !                       !
 ------------------------------------------------
  :           :           :
  :           :           :... Bits 0-7 describe
  :           :                error
  :           :
  :           :............... Error Mask Flag
  :                            Bit B (8-A unused)
  :
  :........................... Error Status Flag
                               Bit F (C-E unused)
```

Normally, after the completion of each command all bits of the Error Status and the Error Type are cleared (= 0). If an error occurred during the command, the Error Status Flag (bit F) will be set by the command. In addition, an Error Type will be set into the lower half of the status word (bits 0-7). The Error Type is used to indicate which error was detected by the command.

Usually, the CHAIN process will abort anytime the Error Status Flag is set by one of the commands invoked from the intermediate file; however, the Error Mask can be used to inhibit CHAIN process aborting due to command errors. The Error Mask Flag (bit B) will inhibit CHAIN process aborting if it is set to one. The process of setting the Error Mask is described in section 6.4.

28.5 Commands Affecting Error Status Word

All MDOS commands that are intended to be invoked by the CHAIN process have been programmed to configure error types into the system error status word. These error types are summarized here to facilitate the user who is taking advantage of the TST execution operator during the CHAIN process.

All MDOS commands use the system function .MDERR for displaying the common error messages. Thus, the error types that correspond to these messages will always be the same; namely, the error message's index number used to call the MDERR function (not the same as the displayed, two-digit, error message reference number); however, commands have other error messages that are displayed independently of the .MDERR function. These errors will cause a value to be set into the Error Type field of the error status word that is greater than or equal to 128 ($80). It is these values, which are unique to each command, that are summarized here. The following table contains the name of the MDOS command or system function that sets the Error Type, the value of the Error Type in hexadecimal, and the error message or condition that caused the error. If the text in the table is in capital letters, it is an actual error message. If the text is in upper/lower case letters, then it is an error condition.

MDOS Function	Error Type	Error Message or Condition
MDOS Command Interpreter	$80	WHAT?
.MDERR	$FF	**INVALID MESSAGE mm AT nnnn
BACKUP	$80	SOURCE FILE COPY ERROR
	$81	OBJECT FILE CREATION COPY ERROR
	$82	CANNOT DELETE DUPLICATE NAME
	$83	INVALID TO COPY/VERIFY FROM DOUBLE TO SINGLE SIDED
	$84	DIRECTORY READ/WRITE ERROR
	$85	SYSTEM SECTOR COPY ERROR
	$86	SYNTAX ERROR
	$87	Sector verify error
BINEX	-	-
BLOKEDIT	-	-
CHAIN	-	-
COPY	$80	Response other than "Y" to overwrite question
	$81	Verify error
DEL	$80	<name> DOES NOT EXIST
	$81	<name> IS PROTECTED
DIR	$80	NO DIRECTORY ENTRY FOUND
	$81	NO TERMINATOR FOUND IN FILE'S R.I.B.
	$82	*NO SDWS*
DOSGEN	$80	INVALID SECTOR NUMBER
	$81	SECTOR xxxx LOCKED OUT
DUMP	$80	SYNTAX ERROR
	$81	MODE ERROR
	$82	BOUNDARY ERROR
	$83	INVALID SECTOR ADDRESS
	$84	WHAT?

ECHO	-	-
EMCOPY	-	-
EXBIN	$80	SOURCE FILE NOT ASCII
	$81	RECORD FORMAT ERROR
	$82	START ADDRESS OUT OF RANGE
	$83	CHECKSUM ERROR
FORMAT	-	-
FREE	-	-
LIST	-	-
LOAD	-	-
NAME	-	-
MERGE	$80	Response other than "Y" to overwrite question
PATCH	$80	INITIALIZATION ERROR
	$81	WHAT?
	$82	SYNTAX ERROR
	$83	ILLEGAL OP CODE
	$84	ILLEGAL OPERAND
	$85	ILLEGAL ADDRESS
REPAIR	-	-
ROLLOUT	-	-

The following MDOS-supported commands (available at time of publication) change the Error Type in the error status word:

Command	Error Type	Error Message or Condition
ASM	-	The error message number of the last encountered error will appear in the Error Type.
ASM1000	-	The error message number of the last encountered error will appear in the Error Type.
ASM3870	-	The error message number of the last encountered error will appear in the Error Type.
BASIC	-	-

FORM1000	-	-
FORT	$80	Any compiler-detected error
MASM	-	-
MPL	$80	Any compiler-detected error
RASM	-	The error message number of the last encountered error will appear in the Error Type.
RASM09	-	The error message number of the last encountered error will appear in the Error Type.
RLOAD	$80	Illegal Commmand
	$81	Illegal command syntax
	$83	User assignment error
	$84	Undefined intermediate file
	$85	Phasing error
	$86	Section overflow
	$87	Undefined object file
	$88	Illegal object record
	$89	Local symbol table overflow
	$8D	Undefined symbol
	$8E	Multiply defined symbol
	$8F	Illegal addressing mode
	$90	Global symbol table overflow

APPENDIX

A. Cylinder-Sector/Physical Sector Conversion Table

The following tables give the physical sector numbers for the first sector of every cylinder. The first table is for single-sided diskettes. All sectors are recorded on surface zero, or the top surface, of a single-sided diskette.

The second table is for double-sided diskettes. The physical sector numbers are given for the first sector of a cylinder on each surface. Surface zero is the top surface and surface one is the bottom surface.

The following notation is used in the table headings:

NOTATION	MEANING
CYLINDER	The numbers in these columns are the cylinder numbers on the diskette. They are given in both decimal and hexadecimal.
PSN	The numbers in these columns are the hexadecimal physical sector numbers of the first sector on a cylinder surface.
DEC	Numbers in these columns are decimal.
HEX	Numbers in these columns are hexadecimal.
SFC 0	The top surface, surface zero.
SFC 1	The bottom surface, surface one.

SINGLE-SIDED DISKETTES

CYLINDER		PSN
DEC	HEX	HEX
00	00	000
01	01	01A
02	02	034
03	03	04E
04	04	068
05	05	082
06	06	09C
07	07	0B6
08	08	0D0
09	09	0EA
10	0A	104
11	0B	11E
12	0C	138
13	0D	152
14	0E	16C
15	0F	186
16	10	1A0
17	11	1BA
18	12	1D4
19	13	1EE
20	14	208
21	15	222
22	16	23C
23	17	256
24	18	270
25	19	28A
26	1A	2A4
27	1B	2BE
28	1C	2D8
29	1D	2F2
30	1E	30C
31	1F	326
32	20	340
33	21	35A
34	22	374
35	23	38E
36	24	3A8
37	25	3C2
38	26	3DC
39	27	3F6
40	28	410
41	29	42A
42	2A	444
43	2B	45E
44	2C	478
45	2D	492
46	2E	4AC
47	2F	4C6
48	30	4E0
49	31	4FA
50	32	514
51	33	52E
52	34	548
53	35	562
54	36	57C
55	37	596
56	38	5B0
57	39	5CA
58	3A	5E4
59	3B	5FE
60	3C	618
61	3D	632
62	3E	64C
63	3F	666
64	40	680
65	41	69A
66	42	6B4
67	43	6CE
68	44	6E8
69	45	702
70	46	71C
71	47	736
72	48	750
73	49	76A
74	4A	784
75	4B	79E
76	4C	7B8

DOUBLE-SIDED DISKETTES

CYLINDER		PSN		CYLINDER		PSN	
DEC	HEX	SFC 0	SFC 1	DEC	HEX	SFC 0	SFC 1
0	000	000	01A	39	027	7EC	806
1	001	034	04E	40	028	820	83A
2	002	068	082	41	029	854	86E
3	003	09C	0B6	42	02A	888	8A2
4	004	0D0	0EA	43	02B	8BC	8D6
5	005	104	11E	44	02C	8F0	90A
6	006	138	152	45	02D	924	93E
7	007	16C	186	46	02E	958	972
8	008	1A0	1BA	47	02F	98C	9A6
9	009	1D4	1EE	48	030	9C0	9DA
10	00A	208	222	49	031	9F4	A0E
11	00B	23C	256	50	032	A28	A42
12	00C	270	28A	51	033	A5C	A76
13	00D	2A4	2BE	52	034	A90	AAA
14	00E	2D8	2F2	53	035	AC4	ADE
15	00F	30C	326	54	036	AF8	B12
16	010	340	35A	55	037	B2C	B46
17	011	374	38E	56	038	B60	B7A
18	012	3A8	3C2	57	039	B94	BAE
19	013	3DC	3F6	58	03A	BC8	BE2
20	014	410	42A	59	03B	BFC	C16
21	015	444	45E	60	03C	C30	C4A
22	016	478	492	61	03D	C64	C7E
23	017	4AC	4C6	62	03E	C98	CB2
24	018	4E0	4FA	63	03F	CCC	CE6
25	019	514	52E	64	040	D00	D1A
26	01A	548	562	65	041	D34	D4E
27	01B	57C	596	66	042	D68	D82
28	01C	5B0	5CA	67	043	D9C	DB6
29	01D	5E4	5FE	68	044	DD0	DEA
30	01E	618	632	69	045	E04	E1E
31	01F	64C	666	70	046	E38	E52
32	020	680	69A	71	047	E6C	E86
33	021	6B4	6CE	72	048	EA0	EBA
34	022	6E8	702	73	049	ED4	EEE
35	023	71C	736	74	04A	F08	F22
36	024	750	76A	75	04B	F3C	F56
37	025	784	79E	76	04C	F70	F8A
38	026	7B8	7D2				

APPENDIX

B. ASCII Character Set

BITS 4 TO 6 --		0	1	2	3	4	5	6	7
	0	NUL	DLE	SP	0	@	P	`	p
B	1	SOH	DC1	!	1	A	Q	a	q
I	2	STX	DC2	"	2	B	R	b	r
T	3	ETX	DC3	#	3	C	S	c	s
S	4	EOT	DC4	$	4	D	T	d	t
	5	ENQ	NAK	%	5	E	U	e	u
O	6	ACK	SYN	&	6	F	V	f	v
	7	BEL	ETB	'	7	G	W	g	w
T	8	BS	CAN	(	8	H	X	h	x
O	9	HT	EM	)	9	I	Y	i	y
	A	LF	SUB	*	:	J	Z	j	z
3	B	VT	ESC	+	;	K	[	k	{
	C	FF	FS	,	<	L	\	l	¦
	D	CR	GS	-	=	M	]	m	}
	E	SO	RS	.	>	N	^	n	~
	F	SI	US	/	?	O	_	o	DEL

APPENDIX

C. MDOS Command Syntax Summary

```
Chapter  Command Line              Options
-------  ------------              -------

3*    BACKUP [[:<source unit>,]:<destination unit>] [;<options>]
                                   null - Normal copy
                                   A - Append
                                   R - Reorganize
                                   V - Verify

                                   C - Disk error continue
                                   D - Deleted data mark continue
                                   I - ID sector
                                   L - Line printer
                                   N - No printing
                                   S - Sector number only
                                   U - Unallocated space
                                   Y - Delete duplicate
                                   Z - Skip duplicate

4     BINEX <memory-image file>[,<EXbug-loadable file>]

5     BLOKEDIT <command file>,<new file>

6     CHAIN <command file> [;<tag i>[%<value i>%] ...]
      CHAIN N*
      CHAIN *

7     COPY <source name>[,<destination name>] [;<options>]
                                   B - Automatic verify after copy
                                   C - Convert binary records
                                   D=<file>[,] - Driver file
                                   L - Line printer
                                   M - Test driver via debug monitor
                                   N - Non-file format
                                   V - Verify
                                   W - Overwrite

8*    DEL [<file>] [;<options>]
                                   S - System files
                                   Y - Yes, delete

9*    DIR [<file>] [;<options>]
                                   A - Allocation information
                                   E - Entire entry
                                   L - Line printer
                                   S - System files
```

```
Chapter  Command Line            Options
-------  ------------            -------

10    DOSGEN [:<unit>] [;<options>]
                                 T - Write/read surface test
                                 U - User diskette (minimum system files)

11    DUMP [<file>]

12    ECHO [;<options>]
                                 N - Turn echo off

13    EMCOPY [<EDOS file>][,<MDOS file>] [;<options>]
                                 A - ASCII record format
                                 C - Contiguous allocation
                                 D - Delete protection
                                 E - Entire disk copy
                                 R - Binary record format
                                 S - Selected file copy

14    EXBIN <EXbug lodadable file>[,<memory-image file>] [;<start address>

15    FORMAT [:<unit>]

16    FREE [:<unit>] [;<options>]
                                 L - Line printer

17    LIST <ASCII file>[,[<start>][,<end>]] [;<options>]
                                 F[mmm].[nn] - Page format
                                 H - Input heading
                                 L - Line printer
                                 N - Line numbers

18    LOAD [<memory-image file>] [;<options>]
                                 null - Go to EXbug
                                 null - Load above MDOS
                                 G - Load and go
                                 U - EXORciser II User Memory Map
                                 V - Overlay MDOS; discontiguous memory
                                 (<string>) - Initialize command buffer

19    MERGE <file 1>[,<file 2>,...,<file n>],<destination file> [;<options
                                 W - Overwrite
                                 <start address>

20*   NAME <old name>[,<new name>] [;<options>]
                                 D - Delete protection
                                 N - Non-system file
                                 S - System file
                                 W - Write protection
                                 X - No protection

21    PATCH <memory-image file>
```

```
Chapter  Command Line            Options
-------  ------------            -------

22     REPAIR [:<unit>]

23     ROLLOUT [<memory-image file>] [;<options>]
                                 null - Memory above MDOS
                                 D - Build file from scratch diskette
                                 U - EXORciser II User Memory Map
                                 V - Any memory to scratch diskette

     *   These  commands  allow  the  family indicator in the file
         name specification.
```

APPENDIX

D. Diskette Controller Entry Points

The floppy diskette controller module firmware is used to control all of the EXORdisk II/III hardware functions. The entry points to the various functions are described in this section. Parameters required by the firmware functions are stored in RAM in the locations described by the following table:

Name	Address	Definition
CURDRV	$0000	This byte contains the binary logical unit number of the drive to be selected (zero through three).
STRSCT	$0001	These two bytes contain the physical sector number of the first sector to be used (starting sector).
NUMSCT	$0003	These two bytes contain the number of sectors to be used. This number includes a partial sector, if a partial sector read is being requested. The sum of STRSCT and NUMSCT cannot be greater than $7D2 (single-sided diskettes) or $FA4 (double-sided diskettes).
LSCTLN	$0005	This byte contains the number of bytes to be read from the last sector during a read operation. This number should be a multiple of eight and cannot be greater than 128 ($80). If a number is specified that is not a multiple of eight, the next larger multiple of eight bytes will be read.
CURADR	$0006	These two bytes contain the first address in memory that is to be used during a read or write operation. This location is updated after each sector is read or written. During write test operations, these two bytes contain the address of a two-byte data buffer.
FDSTAT	$0008	This byte contains a status indication of the performed function. If an error occurred during a diskette operation, the carry bit in the condition code register

		will be set to one upon returning to the calling program. In addition, FDSTAT will contain a number indicating the error type ($31 - $39). The error types are explained in Chapter 28. If no error occurs, then the carry bit of the condition code register will be set to zero and FDSTAT will contain the value $30.
SIDES	$000D	This byte contains an indication of the type of diskette that is in a drive. If the sign bit (bit 7) of this location is set to one after a diskette has been accessed, then the diskette is single-sided. If the sign bit of this location is set to zero after a diskette has been accessed, then the diskette is double-sided. In earlier versions of the diskette controller firmware (EXORdisk II), this location will always have the sign bit set to one.

For all of the firmware entry points described below, the content of the registers is unspecified both upon entry and exit from the routine. Each entry point is accessed by executing a "jump to subroutine" instruction (JSR). The parameters must have been set up in RAM as indicated for each specific function. It should be noted that the ROM routines for the diskette functions run with the interrupt mask bit set to one in the condition code register. The routines also use the NMI vector. Both the NMI vector and the interrupt mask are restored before returning to the calling program.

Name	Address	Function
OSLOAD	$E800	This entry point initializes the drive electronics and loads the Bootblock and MDOS retrieval information block from the diskette in drive zero. The Bootblock is given control after it has been loaded from the diskette. It, in turn, causes the rest of the operating system to be loaded into memory. No parameters are required for this entry point. This function does not return control to the calling program. If an error occurs during the Bootblock load process, the error number will be displayed on the system console and control passed to the resident debug monitor. At least $120 bytes of memory are required starting at location zero. If less memory exists,

the Bootblock program may not be able to display an error message indicating that there is insufficient memory in the system. The SWI vector must be configured for the debug monitor before this entry point can be used (e.g., the ABORT or RESTART pushbutton on the front panel of the EXORciser must have been depressed).

FDINIT	$E822	This entry point initializes the PIA and SSDA. No parameters are required by this routine and none are modified by it.
CHKERR	$E853	This entry point is used to check for a diskette controller error if called immediately after returning from another ROM entry point. The routine will check the state of the carry flag in the condition code register. If the carry flag is set to zero, the CHKERR routine will simply return to the calling program. If the carry flag is set to one (an error occurred), then the routine will print an "E" followed by the contents of FDSTAT and two spaces on the system console. Control is given to the resident debug monitor after printing the error message. CHKERR does not change any of the parameters.
PRNTER	$E85A	This entry point will print an "E" followed by the contents of FDSTAT followed by two spaces on the system console. PRNTER does not change any of the parameters.
READSC	$E869	This entry point causes the number of sectors contained in NUMSCT beginning with STRSCT from CURDRV to be read into memory starting at the address contained in CURADR. CURADR is updated to the next address that is to be written into after each sector is read. The parameter LSCTLN is automatically set to 128 ($80) so that a complete sector is read into memory when the last sector is processed. The parameters CURDRV, STRSCT, and NUMSCT are not changed. FDSTAT will contain the status of the read operation.
READPS	$E86D	This entry point is similar to READSC with the exception that the last sector is only partially read according to the

		contents of LSCTLN. If LSCTLN contains 128 ($80), then this entry point is identical to READSC. The restrictions placed on LSCTLN are described in the preceding table of the parameters.
RDCRC	$E86F	This entry point causes the number of sectors contained in NUMSCT beginning with STRSCT from CURDRV to be read to check their CRCs. The contents of the sectors are not read into memory. The only parameter changed is FDSTAT.
RWTEST	$E872	This entry point causes the two bytes located at the address (and at address + 1) contained in CURADR to be written into alternating bytes of NUMSCT sectors beginning with STRSCT of CURDRV. After NUMSCT sectors are written in this fashion, they are read back to verify their CRCs. The only parameter changed is FDSTAT.
RESTOR	$E875	This entry point causes the read/write head on CURDRV to be positioned to cylinder zero. The only parameter required is CURDRV. The only parameter changed is FDSTAT.
SEEK	$E878	This entry point causes the read/write head of CURDRV to be positioned to the cylinder containing STRSCT (see Appendix A). The only parameter changed is FDSTAT.
WRTEST	$E87B	This entry point causes the two bytes of data located at the address (and at address + 1) contained in CURADR to be written into alternating bytes of NUMSCT sectors beginning with STRSCT of CURDRV. The only parameter changed is FDSTAT.
WRDDAM	$E87E	This entry point causes a deleted data mark to be written to NUMSCT sectors beginning with STRSCT of CURDRV. The only parameter changed is FDSTAT.
WRVERF	$E881	This entry point causes NUMSCT sectors beginning at STRSCT of CURDRV to be written from memory starting at the address contained in CURADR. CURADR is updated to the address of the next byte to be read from memory after each sector is written. After all sectors have been

		written to the diskette, they are read back to verify their CRCs as checked by the routine RDCRC. The only parameters changed are CURADR and FDSTAT.
WRITSC	$E884	This entry point is identical to WRVERF with the exception that the written sectors are not read back to verify their CRCs. The only parameters changed are CURADR and FDSTAT.

When an error occurs, the physical sector number at which the error occurred can be computed from the following relationship:

PSN = STRSCT + NUMSCT - SCTCNT -1

where PSN is the physical sector number at which the error occurred, and SCTCNT is a two-byte value contained in locations $000B-000C.

The following entry points are also in the firmware but have nothing to do with the diskette functions. These entry points can be used to access a line printer.

Name	Address	Function
LPINIT	$EBC0	This entry point intializes the PIA from a reset condition.
LIST	$EBCC	This entry point sends the contents of the A accumulator to the line printer. If the "paper empty" or "printer not selected" status condition is detected, the LIST entry point will return with the carry flag of the condition code register set to one. If these conditions are not detected, the carry flag will be set to zero.
LDATA	$EBE4	This entry point sends a character string to the line printer. The string is pointed to by the X register and must be terminated with an EOT ($04). Prior to printing the string, a carriage return and a line feed are sent to the printer. If a printer error is detected by LDATA, it will loop until aborted or until the error is corrected.
LDATA1	$EBF2	This entry point performs the same function as LDATA with the exception that the initial carriage return and line feed

are not printed.

For a complete description of the diskette controller module the "Floppy Disk Controller Module User's Guide" should be consulted.

APPENDIX

E. Mini-Diagnostic Facility

A mini-diagnostic routine is available in the EXORdisk II diskette controller firmware (version numbers less than 1.2). This routine permits the user to execute any diskette controller function a single time or continuously. The parameters required by the mini-diagnostic routines are similar to those used by the other diskette controller functions (Appendix D). The reader should be familiar with those parameters before attempting to use the mini-diagnostics.

The following parameters and entry points are required by the mini-diagnostic routine:

Name	Address	Definition
CURADR	$0006	This parameter is automatically set up by the mini-diagnostic routine from LDADDR (see below) before each execution of the specified function.
LDADDR	$0020	These two bytes contain the data that would normally be placed into CURADR. The diagnostic routine will update CURADR from LDADDR before each function is executed.
EXADDR	$0022	These two bytes must contain the address of the entry point of the function (READSC, WRTEST, etc.) that is to be executed by the diagnostic routine.
ONECON	$0024	This byte should contain a zero if the function is to be executed continuously. A non-zero value in this location will cause the function to only be executed once.
$0060-$0073		This area contains a two-byte counter for each of the possible states returned by a function in FDSTAT. Locations $60-61 contain a counter for the status of "0"; locations $62-63 contain a counter for the status of "1"; and so on.
CLRTOP	$EB90	This location is the entry point to the mini-diagnostic routine that initially

		zeroes the counters in locations $60-73 before executing the function.
TOP	$EB98	This location is the entry point to the mini-diagnostic routine that will leave the counters at locations $60-73 unchanged before executing the function.

Single Execution

In order to execute a diskette function a single time, the parameters CURDRV, STRSCT, NUMSCT, LSCTLN, and LDADDR should be configured as required for the specific function. The address of the specific function should then be placed into EXADDR. The location ONECON should be initialized with a non-zero value. The stack register should be pointing to a valid area in memory (the EXbug stack is acceptable). Then, the debug monitor command

```
EB98;G
```

will give control to the mini-diagnostic routine causing the PIA and SSDA to be initialized, CURDRV to be restored, and the function in EXADDR to be executed a single time. Upon completion of the function, the letter "E" followed by a digit "0" through "9" will be printed and control returned to the debug monitor. The displayed message will indicate the completion status of the function as returned in FDSTAT.

Continuous Execution

In order to execute a diskette function continuously, the parameters CURDRV, STRSCT, NUMSCT LSCTLN, and LDADDR should be configured as required for the specific function. The address of the specific function should then be placed into EXADDR. The location ONECON should be initialized to the value of zero. The the debug monitor command

```
EB98;G (to start at TOP)

or

EB90;G (to start at CLRTOP and zero counters)
```

will give control to the mini-diagnostic routine. This will cause the PIA and SSDA to be initialized, CURDRV to be restored, and the function in EXADDR to be executed continuously until one of the two-byte counters is incremented to zero. When one of the two-byte counters reaches zero, an "E" followed by an error indication will be printed at the console and control returned to the debug monitor. The error indication following the letter "E" will not be the normal value in the range 0-9. Rather, it will be the ASCII character that corresonds to twice the value of the normal error code $30-$39. Thus, the following correlation exists between the normal error and the printed character following the "E":

Normal Error	Printed character
0	`
1	b
2	d
3	f
4	h
5	j
6	l
7	n
8	p
9	r

If the user initializes a counter to the value $FFFF, for example, the mini-diagnostic will run continuously until the first error of the type monitored by the counter occurs.

APPENDIX

F. Diskette Description, Handling, and Format

The flexible disk, or diskette, is permanently enclosed by a durable, plastic covering. This outside jacket allows the diskette to be handled and at the same time gives a certain degree of protection for the oxide surface within. The covering also provides rigidity to the diskette, allowing it to be easily inserted into and removed from the diskette drives.

To extend the usable life of a diskette and to maximize trouble-free operation, the diskette should be handled with reasonable care. The following points of diskette care should be followed. Most manufacturers usually list these points on the protective envelope of the diskette as a reminder.

1. The diskette should be returned to its protective envelope when not in a drive unit.

2. The diskette in its envelope should be stored vertically. It should not be stacked or placed under heavy pressure as this can cause warping of the oxide surface.

3. Too many diskettes should not be forced into one box.

4. The diskette should not be exposed to any magnetizing force in excess of 50 oersted. The 50 oersted level can be reached about three inches away from a typical source such as electric motors, transformers, etc.

5. Diskettes should not be subjected to extremes of heat. They should not be kept in direct sunlight. Warping can result.

6. The label on the diskette should only be written on with a felt-tipped pen. Pencils, ballpoint pens, or extreme pressure from felt-tipped pens can emboss the oxide surface within.

7. The physical oxide surface should never be touched. Skin oils transferred to the surface in this manner can attract and retain dust and other contaminants.

8. The surface of the diskette should never be wiped or cleaned. Any physical contact with the surface should be avoided.

9. The diskette should never be forced into the drive. Neither should the diskette be folded or bent.

10. The door on the diskette drive should not be closed before the diskette has been inserted all the way. Damage to the drive hub hole can result. Likewise, the door on the drive should be fully opened before the diskette is removed.

The diskette may or may not have a write-protect hole along the edge that is inserted first into the drive. This hole is located 6.25 inches from the right edge as seen from above the diskette. When the hole is not covered, the diskette is write protected. The hole must be covered in order to write on the diskette. An opaque adhesive-backed label or tape can be used to cover the hole.

The single-sided diskette is recorded in a format that is similar to the single-sided single-density format of an IBM-3740 diskette. The detailed format description is contained in the IBM document number GA21-9190-3, "IBM One-sided Diskette OEM Information", Appendix B. The format described in that appendix is in reference to IBM part number 2305830.

The single-sided format is similar to the IBM 3740 format insofar as the addressing information is concerned. The usage and content of the actual sectors and cylinders is not necessarily similar.

The double-sided diskette is recorded in the Motorola single-density double-sided format. This format is an extension of the single-sided single-density format onto the other side of the diskette. Appendix A gives the location of the phsyical sectors with respect to surface and cylinder for both single- and double-sided diskettes.

APPENDIX

G. Directory Hashing Function

In order to speed up a directory search for a specific file name, a hashing function is used to map a file's name into one of the directory's sectors. As a result, the number of sectors that have to be read before a match is found or not found is minimized.

All ten bytes of the file name and suffix are used by the hashing function. The function computes a number which, when added to the physical sector number of the start of the directory, is the sector number of the first sector used in a linear search of the directory.

An entry in the directory will have in its first byte a value of zero, indicating that this entry has never been used; a value of $FF, indicating that the entry is deleted; or an ASCII character, indicating the presence of a file name.

Initially, all directory sectors are filled with zeroes. New names are added sequentially to the sector identified by the hashing function. New entries can be made into those entries which have a zero or an $FF in their first byte. Thus, a search for a name can stop whenever an entry is found which has the first byte equal to zero.

A directory search begins in the sector identified by the hashing function. If no entries within this sector contain zero in their first byte, and if no match is found, the next sector in the directory is searched. The sectors will continue to be searched in this round-robin fashion until a match or an entry with first byte of zero is found, or until all sectors have been examined. The only time all sectors of the directory are searched is if every entry contains a valid file name or a deleted file name. Thus, directory searches are faster if the directory has been reorganized with the BACKUP command (section 3.3).

The following routine is similar to the one used in MDOS to perform the directory hashing function. It is documented here to allow users who wish to write disk-oriented programs to access the directory without using MDOS.

```
*
* MDOS DIRECTORY HASHING FUNCTION
*
* ENTRY:  X = ADDRESS OF 10 BYTE FILE NAME
*             AND SUFFIX
*
* EXIT:   A ACCUMULATOR CONTAINS THE
*             HASH CODE -- A NUMBER IN THE
*             RANGE 0-19, DECIMAL.
*
TMP1    RMB     1         .
TMP2    RMB     1         .
TMP3    RMB     1         .
*
HASH    LDAB    #10       .
        STAB    TMP3      .
        CLC               .
        CLRB              .
HASH2   STAB    TMP1      .
        TPA               .
        STAA    TMP2      .
        LDAB    0,X       . GET FILE CHAR
        SUBB    #$25      . MAKE IT UNIQUE
        BPL     HASH25    .
        CLRB              .
HASH25  LDAA    TMP2      .
        TAP               .
        ADCB    TMP1      .
        ROLB              .
        INX               .
        DEC     TMP3      .
        BNE     HASH2     .
        RORB              .
        TBA               .
        RORA              .
        RORA              .
        RORA              .
        RORA              .
        ABA               .
        TAB               .
        ANDB    #%00011111
        CMPB    #19       .
        BLS     HASH3     .
        SUBB    #20       .
        CMPB    #9        .
        BHI     HASH3     .
        ASRA              .
        ROLB              .
HASH3   STAB    TMP3      .
        RTS               .
```

APPENDIX

H. MDOS-Supported Software Products

This Appendix contains a list of the MDOS-Supported software products available at the time of publication. These products are capable of running in an MDOS environment even though some of them have been developed independently. All MDOS-Supported products are purchased and shipped separately from MDOS. At the time of publication, only the following supported products are available for MDOS09: RASM09, RLOAD, EDIT, and E.

These descriptions contain a brief discussion of how the product is invoked from the MDOS command line. Any additional hardware requirements are also noted. The product's manual that is shipped along with its diskette should be consulted for details about its operation.

H.1 ASM -- M6800 Assembler

The ASM command processes source program statements written in the M6800 Assembly Language. The M6800 Assembler, ASM, translates these source statements into object programs.

The M6800 Assembler is invoked from the MDOS command line as are other MDOS commands. No additional hardware requirements are needed to run the assembler other than the minimum configuration used for MDOS. The format of the command line is:

```
ASM <name> [;<options>]
```

where <name> is the name of source file. The source file <name> is in the standard MDOS file name format

```
<file name> [.<suffix>] [:<logical unit number>]
```

The default values of "SA" and "0" are used if <suffix> and <logical unit number> are not explicitly entered.

The <options> may be one or more of the options listed in the following table. All options except those that control the destination of the source listing and the destination of the object file can be specified from within the source program with the OPT directive. Certain options are automatically used as a default condition. These conditions can be reversed or overridden by preceding the option letter with a minus sign (-). The following options are recognized by the assembler:

OPTION	DEFAULT	ATTRIBUTE CONTROLLED BY OPTION
G	-G	Printing of generated code from FCB, FDB, and FCC directives
L	-L	Print source listing on line printer
L=#CN,	-L	Print source listing on console
O	O	Create object file with name of source file and suffix "LX" on same logical unit as source file on command line
O=<name>,	O	Create object file with name <name>
S	-S	Print symbol table

Certain options (L=, O=) require a terminating comma only if other options follow. Options are specified without any intervening blanks or separators.

Each symbol in the symbol table requires eight bytes. Thus, if the minimum of 16K bytes of memory is used, the M6800 Assembler can accommodate about 300 (decimal) symbols.

For more details about the M6800 Assembler, the "M6800 Co-Resident Assembler Reference Manual" should be consulted. The following enhancements have been made in the MDOS version of the M6800 Assembler over the specifications in its reference manual.

The symbols may contain the special characters period (.) and dollar sign ($); however, the dollar sign may not be used as the first character of a symbol.

The END directive has been changed so that it now has the following format:

```
END [<expression>]
```

where the value of the optional <expression> will be placed into the S9 record of the object file. This record is used to specify the starting execution address of the object file. If no expression is specified, the value of zero will be used.

Like other MDOS commands, the ASM command is sensitive to the BREAK and CTL-W keys of the system console.

The object file produced is in the EXbug-loadable format. The file must be converted into a memory-image file before it can be loaded from the diskette into memory.

H.2 ASM1000 -- M141000 Cross Assembler

The ASM1000 command processes source program statements written in the M141000 Assembly Language. The M141000 Cross Assembler, ASM1000, translates these source statements into object programs that can be executed by the M141000 Simulator, SIM1000.

The M141000 Cross Assembler is invoked from the MDOS command line as are other MDOS commands; however, the Cross Assembler requires that the system has a minimum of 24K bytes of memory. The format of the command line is:

```
    ASM1000 <name 1>[,<name 2>,...,<name n>] [;<options>]
```

where <name i> are the names of source files. Each file name in the list is in the standard MDOS file name format

```
      <file name> [.<suffix>] [:<logical unit number>]
```

The default values of "SA" and "0" are used if <suffix> and <logical unit number> are not explicitly entered. Up to twenty file names can be accommodated by the assembler.

The <options> may be one or more of the options listed in the following table. All options except those that control the destination of the source listing, the destination of the object file, and the printing of error messages on the printer if no listing is desired, can be specified from within the source program with the OPT directive. Certain options are automatically used as a default condition. These conditions can be reversed or overridden by preceding the option letter with a minus sign (-). The following options are recognized by the assembler:

OPTION	DEFAULT	ATTRIBUTE CONTROLLED BY OPTION
C	C	Printing of macro calls
D	D	Printing of macro definitions
E	-E	Printing of macro expansions
F	F	Printing of conditional directives
G	-G	Printing of generated code from OPLA directive
H	-H	Input initial heading from the console
L	-L	Print source listing on line printer
L=#CN,	-L	Print source listing on console
L=<name>,	-L	Print source listing into diskette file <name> (default suffix is "AL", default logical unit number is zero). Such files should be printed with the COPY command.
M	-M	Print error messages only on line printer
N=ddd,	N=72	Set printed line length to "ddd" (decimal)
O	O	Create object file with name <name 1> and suffix "AO" on same logical unit as <name 1> of command line
O=<name>,	O	Create object file with name <name>
P=dd,	P=58	Set number of printed lines per page to "dd" (decimal). A -P suppresses paging.
S	-S	Print symbol table
T	-T	Print opcode usage statistics table
U	-U	Print unassembled code between conditional directives
X	-X	Print cross reference table

Certain options (L=, N=, O=, P=) require a terminating comma only if other options follow. Options are specified without any intervening blanks or separators.

Each symbol in the symbol table requires ten bytes. Thus, if the minimum of 24K bytes of memory is used, the M141000 Cross Assembler can accommodate about 490 (decimal) symbols; however, if the cross reference option is specified, the symbol table requirements differ. In this case, an additional ten bytes are required by each symbol for every four references to that symbol. If any macro definitions are used (either MACR or INST directives), the available symbol table space will be smaller.

For more details about the M141000 Cross Assembler, the "M141000 Cross Assembler Reference Manual" should be consulted.

Like other MDOS commands, the ASM1000 command is sensitive to the BREAK and CTL-W keys of the system console.

H.3 ASM3870 -- M3870 Cross Assembler

The ASM3870 command processes source program statements written in the M3870 Assembly Language. The M3870 Cross Assembler, ASM3870, translates these source statements into object programs that can be executed by the M3870 Emulator, EM3870.

The M3870 Cross Assembler is invoked from the MDOS command line as are other MDOS commands; however, the Cross Assembler requires that the system has a minimum of 20K bytes of memory. The format of the command line is:

```
ASM3870 <name 1>[,<name 2>,...,<name n>] [;<options>]
```

where <name i> are the names of source files. Each file name in the list is in the standard MDOS file name format

```
<file name> [.<suffix>] [:<logical unit number>]
```

The default values of "SA" and "0" are used if <suffix> and <logical unit number> are not explicitly entered. Up to twenty file names can be accommodated by the assembler.

The <options> may be one or more of the options listed in the following table. All options except those that control the destination of the source listing, the destination of the object file, and the printing of error messages on the printer if no listing is desired, can be specified from within the source program with the OPT directive. Certain options are automatically used as a default condition. These conditions can be reversed or overridden by preceding the option letter with a minus sign (-). The following options are recognized by the assembler:

OPTION	DEFAULT	ATTRIBUTE CONTROLLED BY OPTION
C	C	Printing of macro calls
D	D	Printing of macro definitions
E	-E	Printing of macro expansions
F	F	Printing of conditional directives
G	-G	Printing of generated code from DA and DC directives
H	-H	Input initial heading from the console
L	-L	Print source listing on line printer
L=#CN,	-L	Print source listing on console
L=<name>,	-L	Print source listing into diskette file <name> (default suffix is "AL", default logical unit number is zero). Such files should be printed with the COPY command.
M	-M	Print error messages only on line printer
N=ddd,	N=72	Set printed line length to "ddd" (decimal)
O	O	Create object file with name <name 1> and suffix "LX" on same logical unit as <name 1> of command line
O=<name>,	O	Create object file with name <name>
P=dd,	P=58	Set number of printed lines per page to "dd" (decimal). A -P suppresses paging.
S	-S	Print symbol table
U	-U	Print unassembled code between conditional directives
X	-X	Print cross reference table

Certain options (L=, N=, O=, P=) require a terminating comma only if other options follow. Options are specified without any intervening blanks or separators.

Each symbol in the symbol table requires ten bytes. Thus, if the minimum of 20K bytes of memory is used, the M3870 Cross Assembler can accommodate about 230 (decimal) symbols; however, if the cross reference option is specified, the symbol table requirements differ. In this case, an additional ten bytes are required by each symbol for every four references to that symbol. If any macro definitions are used (MACR directive), the available symbol table space will be smaller.

For more details about the M3870 Cross Assembler, the "M3870 Cross Assembler Reference Manual" should be consulted.

Like other MDOS commands, the ASM3870 command is sensitive to the BREAK and CTL-W keys of the system console.

H.4 BASIC -- BASIC Interpreter

The BASIC command processes source program statements written in the BASIC language. The BASIC interpreter, BASIC, can be used to create, modify, and interpret these source statements.

The BASIC interpreter is invoked from the MDOS command line as are other MDOS commands; however, the interpreter requires that the system has a minimum of 20K bytes of memory. The format of the command line is:

```
BASIC <name 1>[,<name 2>]
```

where <name 1> is the name of a source program file to be loaded or created, and <name 2> can be the name of a file into which the source program file is to be saved. Both file specifications are of the standard MDOS file name format

```
<file name> [.<suffix>] [:<logical unit number>]
```

The default suffix "SA" and the default logical unit number zero will be automatically supplied if none are explicitly entered.

If <name 1> is the name of file which already exists in the directory, then it must contain a valid BASIC program. The contents of the file <name 1> will then be automatically loaded into the work space. If <name 1> does not exist, it will be used to save the contents of the work space when the BASIC interpreter is terminated.

The file <name 2> can optionally be used to save the contents of the work space if <name 1> is to be left unchanged. If <name 2> is specified, it must be the name of a file that does not already exist.

For a detailed description of the BASIC interpreter, the "M6800 BASIC Interpreter Reference Manual" should be consulted.

H.5 E -- CRT Text Editor

The E command can be used to create or to modify ASCII record files on the diskette. Use of the Editor in conjunction with the EXORterm 200/220 or EXORterm 150/EXORciser system allows the user to perform editing, employing specifically designed features of the EXORterm.

The E command is invoked from the MDOS command line as are other MDOS commands; however, the Editor requires that the system has a minimum of 32K bytes of memory.

For a complete description of the E command's usage, the "M6800EDITORM Resident Editor Reference Manual" should be consulted.

H.6 EDIT -- Text Editor

The EDIT command can be used to create or to modify ASCII record files on the diskette. The EDIT command is invoked from the MDOS command line as are other MDOS commands. No additional hardware requirements are needed to run the EDIT command other than the minimum configuration used for MDOS.

The EDIT command is invoked with the following command line:

```
EDIT <name 1>[,<name 2>]
```

where <name 1> is the name of the file to be edited and <name 2> can be the name of an output or scratch file. Both file specifications are in the standard MDOS format:

```
<file name> [.<suffix>] [:<logical unit number>]
```

The default values "SA" and zero are used for the suffix and the logical unit number, respectively, if they are not explicitly entered.

If only <name 1> is specified on the command line, then it will be the name of the file to be edited. If <name 1> already exists, the input will be taken from it. If <name 1> does not already exist, then it will be automatically created, and all output written to it.

The second file name specification, <name 2>, can only be used if the file to be edited already exists on the diskette. Normally, <name 2> is not specified. In this case, the EDIT program will automatically create a temporary output file called SCRATCH.SA. The output file will be created on the same logical unit number as <name 1>, unless a specific logical unit number is entered for <name 2>. The output file is used to receive the data from <name 1> after it has been edited by the operator. When the edit process is ended, any unedited portion of the input file <name 1> will be copied into the output file. The output file will then contain a complete copy of the input file plus any changes that were made to it.

If the default output file is used, the file <name 1> will be automatically deleted and the output file renamed so it has the same name as the original input file. Thus, as far as the operator is concerned, the file <name 1> now contains the results of the edit. <name 1> will, therefore, always be the name of the input file and need not be changed as a result of editing it.

If, however, <name 2> was explicitly entered on the command line, then <name 1> will not be deleted when the EDIT command is terminated. In this way, a set of changes can be applied to the input file without affecting the original copy of the file. The result of the edit will be in <name 2> after the edit is ended. If only a logical unit number is entered for the <name 2> file name specification, then the result of the edit will be on the specified logical unit.

One of the standard MDOS error messages will be displayed if the input file <name 1> is delete or write protected and <name 2> is not specified. Since a protected file cannot be deleted, the edited output file SCRATCH.SA will contain the results of the edit; however, the input file must be manually deleted and the file SCRATCH.SA must be manually renamed by the operator.

If the file SCRATCH.SA already exists on the diskette when the EDIT command is invoked without a <name 2> specification, the error message

```
** 06 DUPLICATE FILE NAME
```

will be displayed. The file to receive the output, whether explicitly entered on the command line or implicitly used as SCRATCH.SA, cannot exist prior to the edit.

One of the standard error messages will also be displayed if during a cross-drive edit, <name 2> cannot be renamed after the original file <name 1> has been deleted. This can occur if <name 1> exists on both drives. In this case, the edited output will again be intact in the file SCRATCH.SA; however, it will have to be renamed manually.

For a complete description of the EDIT command's usage, the "M6800 Co-Resident Editor Reference Manual" should be consulted.

The EDIT command has been changed slightly for MDOS from the way it is described in the EDIT command's Manual. In an attempt to conform to the MDOS keyboard controls, the RUBOUT (DEL) key can be used to backspace a character out of the input buffer; however, the CTL-D key cannot be used to re-display the current line. In addition, the BREAK key can be used to prematurely terminate printing of lines (T command) and file searching (N command). Control will be returned to the EDIT command processor. The CTL-W can also be used to "hold" the lines for consoles that are CRTs. The "F" command (punch nulls for leader) is invalid. The "A" command appends 255 lines into the edit buffer.

H.7 EM3870 -- M3870 Emulator

The EM3870 command is the controlling software for the M3870 Emulator Module. It permits the user to load 3870 object programs from the diskette; to perform examine and change operations on the various programmable registers and memory; and to insert, to display and to remove breakpoints in the user program.

The EM3870 Emulator is invoked from the MDOS command line as are other MDOS commands; however, the Emulator requires that the system has a minimum of 20K bytes of memory as well as an M3870 Emulator Module. In addition, the user's development system must not contain memory between locations $D000 through $DFFF, inclusive.

The EM3870 Emulator is invoked from the following command line:

```
EM3870
```

For a complete description of the Emulator and its command structure, consult the "MC3870 Development System User's Guide".

H.8 FORM1000 -- M141000 Object File Conversion

The FORM1000 command takes the output file from the M141000 Cross Assembler and converts the data to an ASCII record file. The resultant file can then be copied to cassette or paper tape via the MDOS COPY command. No additional hardware requirements are needed to run the object file conversion program other than the minimum configuration needed to run the M141000 Cross Assembler.

The FORM1000 command is invoked with the following command line:

```
FORM1000 <name 1>[,<name 2>]
```

where <name 1> is the name of the object output file produced by the M141000 Cross Assembler, and <name 2> is the name of the file that is to be produced. Both file specifications take on the form:

```
<file name> [.<suffix>] [:<logical unit number>]
```

If <name 2> is not specified on the command line, then <name 1>'s file name and logical unit number will be used as default values for <name 2>. If either suffix is omitted from the command line, then the default values "AO" and "AF" will be used for <name 1> and <name 2>, respectively. If the logical unit number is not specified for <name 1>, then the default value zero will be used.

Once the command has been invoked, the specified directories will be searched to ensure that:

1. <name 1> exists, and
2. <name 2> does not exist.

If these conditions are met, <name 2> will be created. <name 1> will be read and its content converted into ASCII records that are written into <name 2>. Each record will be eighty bytes of data terminated by a carriage return. A total of sixty-six records will be written into <name 2> (64 data records and 2 OPLA records). The eighty-character records have the following format:

```
COLUMN
0                       4      5     6
1                       7      4     0

XX XX. . . . .XX XX XX XX      YYY   PAD 000 THRU 015
.
.
.
XX XX. . . . .XX XX XX XX      YYY   PAD F48 THRU F63
ZZ ZZ. . . . .ZZ ZZ ZZ ZZ      YYY   OPLA TERMS 00 THRU 15
ZZ ZZ. . . . .ZZ ZZ ZZ ZZ      YYY   OPLA TERMS 16 THRU 31
```

where "XX" are the instruction operation codes, "YYY" are the arithmetic sums of all "XX" or "ZZ" for that record, and "ZZ" are output PLA initialization values.

During the processing of the command, the BREAK key can be depressed at any time to cause a controlled termination of the program; however, the partially-generated output file will have to be deleted manually.

The output file, <name 2>, does not get created with space compression as do other MDOS ASCII files. Therefore, <name 2> must not be edited with the MDOS EDIT command since the editor automatically creates space-compressed files.

H.9 FORT -- Relocatable FORTRAN Compiler

The FORT command processes source program statements written in the M6800 FORTRAN Language. The FORTRAN compiler, FORT, compiles these source statements into relocatable object programs. The output from the FORTRAN compiler must be processed by the M6800 Linking Loader in order to obtain an executable object file.

The FORTRAN compiler is invoked from the MDOS command line as are other MDOS commands; however, the compiler requires that a system has a minimum of 24K bytes of memory. The format of the command line is

```
FORT <name 1>[,<name 2>,...,<name n>] [;<options>]
```

where <name i> are the names of source files. Each file name in the list is in the standard MDOS file name format:

```
<file name> [.<suffix>] [:<logical unit number>]
```

The default values "SA" and zero are used if <suffix> and <logical unit number> are not explicitly entered. Up to twenty file names can be accommodated by the compiler.

The <options> may be one or more of the options listed in the following table. Certain options are automatically used as a default condition. These conditions can be reversed or overridden by preceding the option letter with a minus sign (-). The following options are recogized by the compiler:

OPTION	DEFAULT	ATTRIBUTE CONTROLLED BY OPTION
H	-H	Input initial heading from the console
L	-L	Print source listing on line printer
L=#CN,	-L	Print source listing on console
L=<name>	-L	Print source listing into diskette file <name> (default suffix "AL", logical unit number zero). Such files should be printed with the COPY command.
N=ddd,	N=80	Set printed line length to "ddd" (decimal)
O	O	Create object file with name <name 1> and suffix "RO" on same logical unit as <name 1> of command line
O=<name>,	O	Create object file with name <name>
P=dd,	P=58	Set number of printed lines per page to "dd" (decimal). A -P suppresses paging.
S	-S	Print symbol table
X	-X	Conditional compilation of statements beginning with letter "X"

Certain options (L=, N=, O=, P=) require a terminating comma only if other options follow. Options are specified without any intervening blanks or separators.

For a complete description of the FORTRAN compiler consult the "M6800 Resident FORTRAN Compiler Reference Manual".

H.10 MASM -- MACE Cross Assembler

The MASM command processes source program statements written in a user-defined assembly language. The MACE Cross Assembler, MASM, allows the user to define the microword size and instruction field formats for a particular hardware configuration as well as to process source statements written in this format. The object files created by the MACE Cross Assembler can be loaded via the MACE Loader and Debug Module (MBUG).

The MACE Cross Assembler is invoked from the MDOS command line as are other MDOS commands; however, the Cross Assembler requires that the system has a minimum of 32K bytes of memory. The format of the command line is:

```
MASM <name 1>[,<name 2>,...,<name n>] [;<options>]
```

where <name i> are the names of source files. Each file name in the list is in the standard MDOS file name format

```
<file name> [.<suffix>] [:<logical unit number>]
```

The default values of "SA" and "0" are used if <suffix> and <logical unit number> are not explicitly entered.

The <options> may be one or more of the options listed in the following table. Certain options are automatically used as a default condition. These conditions can be reversed or overridden by preceding the option letter with a minus sign (-). The following options are recognized by the assembler:

OPTION	DEFAULT	ATTRIBUTE CONTROLLED BY OPTION
D	D	Build definition table in file <name 1> from command line; default suffix is "DT"; default logical unit number taken from <name 1>
D=<name>,	D	Build definition table in file <name>; default suffix is "DT" and logical unit number is zero
L	-L	Print source listing on line printer
L=#CN,	-L	Print source listing on console
L=<name>,	-L	Print source listing into diskette file <name> (default suffix is "AL", default logical unit number is zero). Such files should be printed with the COPY command.
M	-M	Print error messages only on line printer
N=ddd,	N=72	Set printed line length to "ddd" (decimal)
O	O	Create object file with name <name 1> and suffix "AO" on same logical unit as <name 1> of command line
O=<name>,	O	Create object file with name <name>
P=dd,	P=58	Set number of printed lines per page to "dd" (decimal). A -P suppresses paging.
T=<name>,	-T	Specifies name of file containing definition tables to be referenced during the assembly phase; -T implies tables are in memory
X	-X	Print cross reference table

Certain options (D=, L=, N=, O=, P=, T=) require a terminating comma only if other options follow. Options are specified without any intervening blanks or separators.

Each symbol in the symbol table requires a variable number of bytes depending on the complexity of the microword definition. If the minimum of 32K bytes of memory is used, the MACE Cross Assembler can accommodate about 8K of symbol table.

For more details about the MACE Cross Assembler, the "MACE 29/800 Development System User's Guide" should be consulted.

Like other MDOS commands, the MASM command is sensitive to the BREAK and CTL-W keys of the system console.

H.11 MBUG -- MACE Loader and Debug Module

The MBUG command allows a user to load a program from a diskette file created by the MACE Cross Assembler into the microprogram control storage. MBUG also allows the control storage to be examined, changed, and written back into the diskette file.

The MBUG command is invoked from the MDOS command line as are other MDOS commands; however, MBUG requires that the system has a minimum of 32K bytes of memory, the Memory Emulator, and the System Analyzer. The format of the command line is

```
MBUG [<name 1>][,<name 2>] [;<options>]
```

where <name 1> is the name of a file from which a program is to be loaded, and <name 2> is the name of an output file. Both file names are in the standard MDOS file name format:

```
<file name> [.<suffix>] [:<logical unit number>]
```

The default value "AO" will be used for the suffixes of <name 1> and <name 2> if none are explicitly entered. The default logical unit number for <name 1> is zero. The default logical unit number for <name 2> is taken from the logical unit number of <name 1>.

Only two letters can appear in the <options> field: "V" and "Q". The "V" option indicates that <name 1> is to be verified against the current contents of memory. If "V" is specified, <name 1> must exist.

The "Q" option indicates that all addresses entered will be interpreted as octal. All displayed addresses will also be in octal. If "Q" is not specified, the hexadecimal base will be used.

For a complete description of MBUG, consult the "MACE 29/800 Development System User's Guide".

H.12 MOTEST -- Component Tester Executive

The MDOS version of the MOTEST Component Tester has the same functional capabilities as described in the "MOTEST Component Tester Module Supplement". The operating procedure of the MOTEST executive is described in that supplement.

The MOTEST executive program is invoked by the following command line:

```
LOAD MOTST;VG
```

This MDOS command will both load and execute the executive program.

Since all versions of the MOTEST Component Tester are identical, regardless of the media on which they were supplied, the conversion to diskette will greatly speed up the amount of time it requires to initially load the program.

If the program is on either paper tape or cassette, it can be copied to the diskette by using the following MDOS command:

```
COPY #CR,MOTST.LX;N
```

If the program is on an EDOS diskette, it can be copied to the MDOS diskette by using the following command:

```
EMCOPY MOTST,.LX
```

Once the program is on an MDOS diskette, it must be converted into a memory-image file for loading by using the following MDOS command:

```
EXBIN MOTST;200
```

Thereafer, the LOAD command can be used as described above.

H.13 MPL -- MPL Compiler

The MPL command processes source program statements written in the M6800 MPL Language. The MPL compiler, MPL, compiles these source statements into assembly language source programs. The output from the MPL compiler must be assembled with the M6800 Macro Assembler. The output from the Macro Assembler must be processed by the M6800 Linking Loader in order to obtain an executable object file.

The MPL compiler is invoked from the MDOS command line as are other MDOS commands; however, the compiler requires that a system has a minimum of 56K bytes of memory. The format of the command line is

```
MPL <name 1>[,<name 2>,...,<name n>] [;<options>]
```

where <name i> are the names of source files. Each file name in the list is in the standard MDOS file name format:

```
<file name> [.<suffix>] [:<logical unit number>]
```

The default values "SA" and zero are used if <suffix> and <logical unit number> are not explicitly entered.

The <options> may be one or more of the options listed in the following table. Certain options are automatically used as a default condition. The sense of an option can be reversed by preceding the option letter with a minus sign (-). The following options are recognized by the compiler:

OPTION	DEFAULT	ATTRIBUTE CONTROLLED BY OPTION
L	-L	Produce source listing on the line printer
M	-M	Print error messages only on the line printer
N	-N	Sequence numbers are present on each source statement
O=<name>	-O	Generate compiler output (used for subsequent assembler input) in the file <name>. The file is given the default suffix "SA" and default logical unit number zero. The "O" option, if used, must be the last option specified on the command line.
S	S	Include MPL statements as comments in the output file

Options are specified without any intervening blanks or separators.

For a complete description of the MPL compiler consult the "M6800 Resident MPL Compiler Reference Manual".

The symbol table requirements for the MPL compiler are fairly complex; however, 6000 (decimal) bytes of symbol table space are available. This is sufficient to accommodate approximately 200 (decimal) symbols.

H.14 PPLO/PPHI -- PROM Programmer I

The MDOS version of the PROM Programmer I has the same functional capabilities as described in the "PROM Programmer Module Supplement". Both versions of the PROM programmer (PROMP HI and PROMP LO) are provided on the MDOS diskette in the files PPHI.LO and PPLO.LO, respectively. These files are in the memory-image format to allow them to be loaded into memory directly from the diskette.

The operating procedure for each version of the PROM Programmer I is described in the above-mentioned Supplement; however, the process of loading the PROM Programmer I from the diskette is explained here.

Either version of the PROM programmer I can be loaded and executed from the MDOS diskette by entering the MDOS command line

```
LOAD PPHI;VG or LOAD PPLO;VG
```

depending on which version is to be used. If a user program on the diskette is to be placed into a PROM, the following procedure can be used if the user program loads above the resident operating system and MDOS command interpreter. The file can be loaded into memory using the MDOS command

```
LOAD <name>;V
```

where <name> is the file name of the user's program. Since MDOS does not destroy memory during initialization, the system can be reinitialized and the PROM programmer loaded as explained above.

If the user program overlays the resident MDOS, then it must be "relocated" by changing the file's Retrieval Information Block before loading it into memory. The following sequence of commands should be used to alter a user programs's starting load address:

```
DUMP <name>
R FFFF
78/mm,nn/
W
Q
```

The values "mm" and "nn" represent the hexadecimal numbers of the most significant and least significant bytes of the new starting load address (above the resident MDOS). After the offset load address has been configured in this manner, the above procedure should be followed to load the user program and then load and execute the PROM Programmer I.

A user program whose file has been modified in this fashion cannot be executed after being loaded into memory. The file should be deleted after it has been placed into the PROM.

If the user has the PROM Programmer I on a non-MDOS diskette media, it can be copied to the MDOS diskette using the following procedure.

If the PROM Programmer I is on cassette or paper tape the commands

```
COPY #CR,PPHI.LX;N
COPY #CR,PPLO.LX;N
```

should be used. If the PROM Programmer I is on an EDOS diskette the commands

```
EMCOPY PPHI,.LX
EMCOPY PPLO,.LX
```

should be used. After the files are on the MDOS diskette, they must be converted into loadable memory-image files using the commands:

```
EXBIN PPLO;20
EXBIN PPHI;1000
```

H.15 PROMPROG -- PROM Programmer II/III

The PROM Programmer II/III is the controlling software for the Universal EROM/PROM Programmer Module. It provides the user with a means of programming a variety of 4-bit and 8-bit PROMs and EROMs.

The PROM Programmer II/III is invoked from the MDOS command line as are other MDOS commands; however, the PROM Programmer requires that the system contains the PROM Programmer II/III Module. The format of the command line is:

```
PROMPROG
```

For a complete description of the PROM Programmer II/III and its command structure, the "PROM Programmer II/III Reference Manual" should be consulted.

H.16 RASM -- Relocatable M6800 Macro Assembler

The RASM command processes source program statements written in the M6800/M6801 Assembly Language. The Macro Assembler, RASM, translates these source statements into object programs. If programs are assembled using the relocatable option, the M6800 Linking Loader is required to create a file that can be loaded from diskette into memory.

The Macro Assembler is invoked from the MDOS command line as are other MDOS commands; however, the Macro Assembler requires that the system has a minimum of 24K bytes of memory. The format of the command line is:

```
     RASM <name 1>[,<name 2>,...,<name n>] [;<options>]
```

where <name i> are the names of source files. Each file name in the list is in the standard MDOS file name format

```
      <file name> [.<suffix>] [:<logical unit number>]
```

The default values of "SA" and "O" are used if <suffix> and <logical unit number> are not explicitly entered. Up to twenty file names can be accommodated by the assembler.

The <options> may be one or more of the options listed in the following table. All options except those that control the destination of the source listing, the destination of the object file, and the printing of error messages on the printer if no listing is desired, can be specified from within the source program with the OPT directive. Certain options are automatically used as a default condition. These conditions can be reversed or overridden by preceding the option letter with a minus sign (-). The following options are recognized by the assembler:

OPTION	DEFAULT	ATTRIBUTE CONTROLLED BY OPTION
A	-A	Memory-image object file output
C	C	Printing of macro calls
D	D	Printing of macro definitions
E	-E	Printing of macro expansions
F	F	Printing of conditional directives
G	-G	Printing of generated code from FCB, FDB, and FCC directives
H	-H	Input initial heading from the console
L	-L	Print source listing on line printer
L=#CN,	-L	Print source listing on console
L=<name>,	-L	Print source listing into diskette file <name> (default suffix is "AL", default logical unit number is zero). Such files should be printed with COPY command.
M	-M	Print error messages only on line printer
N=ddd,	N=72	Set printed line length to "ddd" (decimal)
O	O	Create object file with name <name 1> and suffix "LX" (non-relocatable), suffix "RO" (relocatable), or suffix "LO" (memory-image) on same logical unit as <name 1> of command line
O=<name>,	O	Create object file with name <name>
P=dd,	P=58	Set number of printed lines per page to "dd" (decimal). A -P suppresses paging.
R	-R	Relocatable object file output
S	-S	Print symbol table
U	-U	Print unassembled code between conditional directives
X	-X	Print cross reference table
Z	-Z	Use M6801 instruction mnemonics instead of M6800 and create M6801 object output

Certain options (L=, N=, O=, P=) require a terminating comma only if other options follow. Options are specified without any intervening blanks or separators.

Each symbol in the symbol table requires ten bytes. Thus, if the minimum of 24K bytes of memory is used, the Macro Assembler can accommodate about 195 (decimal) symbols; however, if the cross reference option is specified, the symbol table requirements differ. In this case, an additional ten bytes are required by each symbol for every four references to that symbol. If macro definitions are used (MACR directive), the available symbol table space will be smaller. For more details about the Macro Assembler, the

"M6800/M6801/M6809 Macro Assembler Reference Manual" should be consulted.

Like other MDOS commands, the RASM command is sensitive to the BREAK and CTL-W keys of the system console.

H.17 RASM09 -- Relocatable M6809 Cross Assembler

The RASM09 command processes source program statements written in the M6809 Assembly Language. The M6809 Cross Assembler, RASM09, translates these source statements into object programs. RASM09 is the resident macro assembler for MDOS09. If programs are assembled using the relocatable option, the Linking Loader is required to create a file that can be loaded from diskette by the M6809 Simulator.

The M6809 Cross Assembler is invoked from the MDOS command line as are other MDOS commands; however, the Macro Assembler requires that the system has a minimum of 32K bytes of memory. The format of the command line is:

```
RASM09 <name 1>[,<name 2>,...,<name n>] [;<options>]
```

where <name i> are the names of source files. Each file name in the list is in the standard MDOS file name format

```
<file name> [.<suffix>] [:<logical unit number>]
```

The default values of "SA" and "0" are used if <suffix> and <logical unit number> are not explicitly entered. Up to twenty file names can be accommodated by the assembler.

The <options> may be one or more of the options listed in the following table. All options except those that control the destination of the source listing, the destination of the object file, and the printing of error messages on the printer if no listing is desired, can be specified from within the source program with the OPT directive. Certain options are automatically used as a default condition. These conditions can be reversed or overridden by preceding the option letter with a minus sign (-). The following options are recognized by the assembler:

OPTION	DEFAULT	ATTRIBUTE CONTROLLED BY OPTION
A	-A	Memory-image object file output
C	C	Printing of macro calls
D	D	Printing of macro definitions
E	-E	Printing of macro expansions
F	F	Printing of conditional directives
G	-G	Printing of generated code from FCB, FDB, and FCC directives
H	-H	Input initial heading from the console
L	-L	Print source listing on line printer
L=#CN,	-L	Print source listing on console
L=<name>,	-L	Print source listing into diskette file <name> (default suffix is "AL", default logical unit number is zero). Such files should be printed with COPY command.
M	-M	Print error messages only on line printer
N=ddd,	N=72	Set printed line length to "ddd" (decimal)
O	O	Create object file with name <name 1> and suffix "LX" (non-relocatable), suffix "RO" (relocatable), or suffix "LO" (memory-image) on same logical unit as <name 1> of command line
O=<name>,	O	Create object file with name <name>
P=dd,	P=58	Set number of printed lines per page to "dd" (decimal. A -P suppresses paging.
R	-R	Relocatable object file output
S	-S	Print symbol table
U	-U	Print unassembled code between conditional directives
X	-X	Print cross reference table

Certain options (L=, N=, O=, P=) require a terminating comma only if other options follow. Options are specified without any intervening blanks or separators.

Each symbol in the symbol table requires ten bytes. Thus, if the minimum of 32K bytes of memory is used, the M6809 Cross Assembler can accommodate about 700 (decimal) symbols; however, if the cross reference option is specified, the symbol table requirements differ. In this case, an additional ten bytes are required by each symbol for every four references to that symbol. If macro definitions are used (MACR directive), the available symbol table space will be smaller. For more details about the M6809 Cross Assembler, the "M6800/M6801/M6809 Macro Assembler Reference Manual" should be consulted.

Like other MDOS commands, the RASM09 command is sensitive to the BREAK and CTL-W keys of the system console.

H.18 RLOAD -- Linking Loader

The RLOAD command combines relocatable object files created by the M6800/M6801/M6805/M6809 Macro Assemblers or the M6800 FORTRAN Compiler and produces an absolute object file in either memory-image or EXbug-loadable format.

The Linking Loader is invoked from the MDOS command line as are other MDOS commands; however, the Linking Loader requires that the system has a minimum of 24K bytes of memory. The format of the command line is

RLOAD

RLOAD works basically the same as described in the "M6800 Linking Loader Reference Manual"; however, the following changes have been made in the MDOS version of RLOAD over the specifications in the manual.

Some commands have been removed from RLOAD since they were originally intended for a cassette version of the Linking Loader which is no longer supported. These commands are: EXBUG, OI, SRCH, SKIP, FILE, and MODU.

The STR, CUR (without backslash option), and END commands allow the use of either a defined ASCT symbol or a numeric constant to the right of the equal sign.

The default BSCT address that RLOAD will assign is $0020 if assembly language programs are being linked; however, the default address of BSCT will become $0040 if FORTRAN programs are linked. In addition, FORTRAN programs will be automatically assigned memory locations so that DSCT and PSCT fall on even addresses. Therefore, the CUR commands with the backslash option (\) need not be used when linking FORTRAN programs; however, if the CUR command with the backslash option is used when linking FORTRAN programs, the user must ensure that the supplied number is an even number.

Programs with uninitialized BSCT and/or DSCT will not be allocated space on the diskette when an absolute, memory-image file is created; however, all of the BSCT and DSCT must be uninitialized for this feature to be of use.

The format of the load map is slightly improved over the examples shown in the Linking Loader manual. Each program's symbols are printed separately, in alphabetical order, so that an individual symbol can be more easily located in the printed maps.

The following two cautions should be observed when RLOAD is invoked from within a CHAIN file. Since CHAIN uses a forcing character of a backslash (\), two backslash characters have to be entered for the RLOAD commands that use that character. Systems which have a CRT as a console may lose the error messages displayed by RLOAD if errors are inhibited within the CHAIN process. Since such errors are not reflected in any printed MAPs, it is possible to lose sight of the fact that an error occurred, resulting in an invalid output file.

Each symbol in RLOAD requires twelve bytes. If the minimum memory configuration of 24K is used, about 85 entries can be made into the local symbol table and about 265 entries can be made into the global symbol table; however, other items besides symbols occupy this area. The exact symbol table requirements can be calculated from the following:

$$SIZE = GST + \text{largest } LST$$

where SIZE is the total size of the symbol table in bytes and GST and LST are computed from the formulas given below:

$$GST = 12 * (5 + ASCT + NC + XDEF + UXREF + NMOD)$$

$$LST = 12 * (5 + ASCT + NC + XDEF + XREF)$$

The symbols have the following meanings:

Symbol	Meaning
GST	Size of Global Symbol Table.
LST	Size of Local Symbol Table. An LST is created for each file loaded; however, only one LST is kept in memory at any one time.
ASCT	Number of absolute sections.
NC	Number of named common sections.
XDEF	Number of external definitions.
XREF	Number of external references.
UXREF	Number of external references not satisfied (defined) by an external definition.
NMOD	Number of files loaded.

RLOAD divides the available memory so that about three fourths of it is available for the global symbol table and one fourth is available for the local symbol table. The global symbol table contains all of the external definitions and all undefined external references from all loaded files. The local symbol table contains the external definitions and references that pertain to an individual program. Thus, if a global symbol table overflows (GOV error), more memory should be added to the system, or fewer external definitions should be made. If a local symbol table overflow occurs (LOV error), then more memory should be added or the program causing the error should be split into smaller programs.

The following error messages are defined in the RLOAD manual; however, some expansions and new causes for the errors are listed here. All error messages that are generated by RLOAD take on the following format:

ERR-<cause>

where <cause> can be any of the following messages:

<cause>	Explanation
BAE	BSCT Assignment Error. The size of the base section is greater than $100 bytes. This message can be displayed only after a MAP or OBJ command.
COV	Common Section Overflow Error. The size of a common section is greater than $FFFF bytes.
GAE	General Assignment Error. The Linking Loader cannot assign absolute memory addresses for one or more of the following reasons: -- The combined length of all sections is greater than $FFFF bytes. -- Due to the location of ASCTs or user assigned sections, the remaining unassigned sections cannot be placed into unassigned areas of memory. -- The automatic sequence in which sections are assigned memory locations (BSCT, CSCT, DSCT, PSCT) results in the Linking loader being unable to assign memory. User specified starting and/or ending addresses can possibly be used to override the automatic sequence of assigning memory to force a successful link/load.
GOV	Global Symbol Table Overflow Error. The amount of memory available for the global symbol table was too small to accommodate all section information and external definitions.
IAM	Illegal Address Mode Error. A four-digit hexadecimal number will be displayed following this error message. This number is the address of a reference to a global symbol which is used in the program as a one-byte operand; however, the most significant byte of this symbol's value

is not zero. One byte relocation will be performed on the byte located at the specified address, using only the least significant byte of the symbol's value. The object file should be examined to ensure it can be executed.

ICM	Illegal Command Error. An entered command was not recognized by the Linking Loader.
IOR	Illegal Object Record Error. A record in the input file is not a valid relocatable object record.
ISY	Illegal Command Syntax Error. An error occurred in the option or specification field of a command. The following causes are examples of syntax errors:

-- A command separator other than a space, semicolon, or carriage return was used.
-- A command (e.g., OBJA, DEF) was entered without the required equal sign.
-- A <name> was used when a <number> was required by the command (e.g., CURP=\LABEL).
-- An invalid section specification was used with the DEF command.
-- A non-ASCT symbol was used to the right of the equal sign of a STR, CUR, or END command.
-- A backslash was used with the STR or END commands.
-- An undefined global symbol was used to the right of the equal sign of the EXIT command.
-- The file/module qualifier was invalid with the LOAD or LIB command.
-- A logical unit number greater than 3 was specified with a file name.
-- A non-numeric logical unit number was specified with a file name.
-- A numeric constant was used after the device delimiter of the MO command.

LOV	Local Symbol Table Overflow Error. The amount of memory available for the local symbol table was too small to accommodate the section and symbol information for a single program.
MDS	Multiply Defined Symbol Error. The symbol in error is shown following the MDS message. Only one external definition (from files loaded or via DEF command) can be encountered by the Linking Loader. Only the first definition is valid and will be used.
PHS	Phasing Error. The value of a symbol's absolute

address assigned at the end of Pass I (prior to OBJ command) does not agree with the value obtained during Pass II (after the OBJ command). This error can also occur if a program is being searched for during Pass II and it is not found.

SOV	Section Overflow Error. The length of a section is greater than $FFFF bytes (non-BSCT section).
UAE	User Assignment Error. This error can occur for anyone or more of the following reasons:

-- If the OBJA command is being used, the starting load address is less than $0020.
-- If the OBJA command is being used, the calculated ending load address is greater than $FFFF.
-- A user assigned start address for a section is less than the user assigned end address for that section.
-- The user assigned space (end-start) for a section is too small to contain the actual section.
-- The user assigned addresses for sections overlap.
-- The execution address specified with the EXIT command is less than the starting load address or greater than the ending load address of the program.
-- The user assigned starting/ending address for BSCT is greater than $0100.

UDS	Undefined Symbol Error. The symbol in question is displayed following the UDS message. The symbol was not defined during Pass I via a loaded program's external defintiions or via a DEF command. This error can occur after a LOAD, LIB, DEF, STR, CUR, or END command. A value of zero will be used for the undefined symbol.
UIF	Undefined Intermediate File Error. The IFON command was issued but no intermediate file has been defined via the IF command.

In addition, some of the standard MDOS error messages can be displayed by RLOAD. The following are the most frequently seen messages:

** 03 <name> DOES NOT EXIST

The file <name> was used with the LOAD or LIB command but does not exist on the specified logical unit.

** 05 <name> DUPLICATE FILE NAME

The file <name> was used with OBJA, OBJX, MO, or IF commands. These commands require the named file to not exist prior to execution.

** 11 DEVICE NOT READY

A MAP command is trying to write to the printer which is not ready.

** 14 INVALID FILE TYPE

The file specified with the LOAD or LIB command was not a binary record file.

** 24 LOGICAL SECTOR NUMBER OUT OF RANGE

During Pass II (after OBJA command), the programs loaded required the accessing of allocated diskette space outside of the range that was calculated as sufficient during Pass I. This can occur if different files are loaded during the two passes. This message will again occur when the EXIT command is issued, resulting in the output file being deleted.

** 41 INSUFFICIENT DISK SPACE

Any memory-image file for which an appropriate contiguous block of space does not exist will cause this error. Usually, this occurs when creating a file with initialized BSCT or DSCT at low memory addresses and PSCT at high memory addresses. If an intermediate file is being used (which also requires disk space), it is suggested that the link/load process be run without the intermediate file (using CHAIN for example). Map output files also require disk space and can cause this error.

H.19 SIM1000 -- 141000 Simulator

The SIM1000 command is the controlling software for the M141000 Simulator Module. It permits the user to load 141000 object programs from the diskette, to examine and change the various registers and memory, to debug the program, and to rewrite the program with changes back to the diskette.

The SIM1000 Simulator is invoked from the MDOS command line as are other MDOS commands; however, the Simulator requires that the system has a minimum of 24K bytes of memory as well as an M141000 Simulator Module.

The SIM1000 Simulator is invoked from the following command line:

```
SIM1000
```

For a complete description of the Simulator and its command structure, consult the "MC141000/1200 Simulator User's Guide".

H.20 USE with MDOS

Several versions of the Floppy Diskette Controller Module are available for use with MDOS. If a crystal on the controller board is used to generate timing for the diskette interface, this section is not applicable; however, if the memory clock from the EXORciser bus is being used to generate the timing, the following precautions must be taken when using MDOS and USE together.

The user clock must run at 1 MHz, plus or minus a few percent (variable clock rate acceptable in Series II versions), to permit loading user memory with a program from a file from the diskette. If the user clock is not near 1 MHz, the object file should first be converted to an EXbug-loadable file and copied to cassette or paper tape in the regular MDOS environment. Then, the user can load the tape via EXbug in his own environment running with the user clock.

The other precaution is the possibility of having a PIA or ACIA in the user memory generate an IRQ when MDOS is initializing. When memory resides at the same addresses in both EXORciser and user system, the EXORciser memory responds when such a redundant location is read; however, both locations respond (one in each system) when the EXORciser memory is written to. Thus, if an I/O device resides in the user's system at an address that is within the range of contiguous memory in the EXORciser system, the device will be written to when MDOS sizes memory at initialization. It is possible, therefore, to configure the I/O device to generate an IRQ. MDOS does not run with IRQs pending. Thus, a switch should be installed to allow the IRQ line to be opened. This has been done in the buffer box of USE2B.

For a more detailed discussion of USE and the Floppy Diskette Controller Module one or more of the following three manuals should be consulted: "MEXUSE2B User's Guide", "Floppy Disk Controller Module User's Guide", or the appendix of the "USE User's Guide".

APPENDIX

I. MDOS Equate File Listing

This appendix contains a modified listing of the MDOS and MDOS09 equate files. Only the pertinent parts the assembler output are shown. The leftmost column contains the value of the location counter which represents the value equated to the system symbol. The MDOS equate file can be assembled on a user's system if the M6800 Macro Assembler is available. The MDOS equate file is shown first, followed by the MDOS09 equate file.

```
          *
          * MDOS VERSION 03.00 -- SYSTEM EQUATE FILE -- JULY 25,1978
          *
          *
          *DEFINE TYPE OF MDOS--RESIDENT MDOS ONLY
          *
 0000   A MDOSF$ EQU     0         . 0 => MDOS, 1 => OEM MDOS
 0000   A MDOS9$ EQU     0         . 0 => MDOS, 1 => MDOS09
          *
          * S K I P 2     M A C R O
          *
          * THE GENERATED BYTE IS A "COMPARE INDEX IMMEDIATE".
          * THE EXECUTION OF THE BYTE WILL CHANGE THE CONDITION CODE
          * NO REGISTERS ARE AFFECTED.  THUS, A ONE BYTE INSTRUCTION
          * IS FORMED THAT SKIPS FORWARD TWO BYTES.
          *
          SKIP2  MACR
           FCB $8C .
           ENDM
          *
          * S K I P 1     M A C R O
          *
          * THE SAME CONCEPT AS THE "SKIP2" MACRO IS USED, EXCEPT TH
          * A "BIT TEST ACCUMULATOR A IMMEDIATE" OP CODE IS GENERATE
          *
          SKIP1  MACR
           FCB $85
           ENDM
          *
          * S C A L L     M A C R O    (SYSTEM FUNCTION CALL)
          *
          SCALL  MACR
           IFEQ NARG-1
           SWI
           FCB \0!.%01111111
           ENDC
          *
           IFNE NARG-1
```

```
                  FAIL * UNDEFINED SWI CALL ARGUMENT *
                  ENDC
                  ENDM
                 *
                 * U C A L L    M A C R O    (USER FUNCTION CALL)
                 *
                 UCALL  MACR
                  IFEQ NARG-1
                  SWI
                  FCB \O!+%10000000
                  ENDC
                 *
                  IFNE NARG-1
                  SCALL
                  ENDC
                  ENDM
                 *
                 * S E Q    M A C R O     (NUMBERING SEQUENTIAL EQUATES)
                 *
                 SEQ    MACR
                  IFNE NARG
                 \O EQU * .
                  ENDC
                  ORG *+1 .
                  ENDM
                 *
                 * S Y S T E M    F U N C T I O N    D E F I N I T I O N
                 *
                 *
                 * SET LOCATION COUNT TO O FOR THE EQUATE DEFINITIONS
                 *
        0000  A .$SAV   SET    *        .  SAVE OLD LOCATION COUNT
0000                    ORG    $0       .
                 *
                 *
                 *
0000                    SEQ    .RESRV   .  RESERVE A DEVICE
0001                    SEQ    .RELES   .  RELEASE A DEVICE
0002                    SEQ    .OPEN    .  OPEN A FILE
0003                    SEQ    .CLOSE   .  CLOSE A FILE
0004                    SEQ    .GETRC   .  READ A RECORD
0005                    SEQ    .PUTRC   .  WRITE A RECORD
0006                    SEQ    .REWND   .  POSITION TO BEGINNING OF FILE
0007                    SEQ    .GETLS   .  READ LOGICAL SECTOR
0008                    SEQ    .PUTLS   .  WRITE LOGICAL SECTOR
0009                    SEQ    .KEYIN   .  CONSOLE INPUT
000A                    SEQ    .DSPLY   .  CONSOLE OUTPUT (TERM W/ CR)
000B                    SEQ    .DSPLX   .  CONSOLE OUTPUT (TERM W/ EOT)
000C                    SEQ    .DSPLZ   .  CONSOLE OUTPUT (TERM W/ EOT, NO C
000D                    SEQ    .CKBRK   .  CHECK CONSOLE FOR BREAK KEY
000E                    SEQ    .DREAD   .  EROM DISK READ
000F                    SEQ    .DWRIT   .  EROM DISK WRITE
0010                    SEQ    .MOVE    .  MOVE A STRING
0011                    SEQ    .CMPAR   .  COMPARE STRINGS
```

```
0012                  SEQ    .STCHB    .  STORE BLANKS
0013                  SEQ    .STCHR    .  STORE CHARACTERS
0014                  SEQ    .ALPHA    .  CHECK ALPHABETIC CHARACTER
0015                  SEQ    .NUMD     .  CHECK DECIMAL DIGIT
0016                  SEQ    .ADDAM    .  INCREMENT MEMORY (DOUBLE BYTE) BY
0017                  SEQ    .SUBAM    .  DECREMENT MEMORY (DOUBLE BYTE) BY
0018                  SEQ    .MMA      .  MULTIPLY (SHIFT LEFT) MEMORY BY A
0019                  SEQ    .DMA      .  DIVIDE (SHIFT RIGHT) MEMORY BY A
001A                  SEQ    .MDENT    .  ENTER MDOS WITHOUT RELOADING
001B                  SEQ    .LOAD     .  LOAD A FILE FROM DISK
001C                  SEQ    .DIRSM    .  DIRECTORY SEARCH AND MODIFY
001D                  SEQ    .PFNAM    .  PROCESS FILE NAME
001E                  SEQ    .ALUSM    .  ALLOCATE USER MEMORY
001F                  SEQ    .CHANG    .  CHANGE NAME/ATTRIBUTES
0020                  SEQ    .MDERR    .  MDOS ERROR MESSAGE HANDLER
0021                  SEQ    .ALLOC    .  ALLOCATE DISK SPACE
0022                  SEQ    .DEALC    .  RETURN DISK SPACE
0023                  SEQ    .EWORD    .  SET ERROR STATUS WORD FOR CHAIN
0024                  SEQ    .TXBA     .  TRANSFER X TO B,A
0025                  SEQ    .TBAX     .  TRANSFER B,A TO X
0026                  SEQ    .XBAX     .  EXCHANGE B,A AND X
0027                  SEQ    .ADBX     .  ADD B TO X
0028                  SEQ    .ADAX     .  ADD A TO X
0029                  SEQ    .ADBAX    .  ADD B,A TO X
002A                  SEQ    .ADXBA    .  ADD X TO B,A
002B                  SEQ    .SUBX     .  SUBTRACT B FROM X
002C                  SEQ    .SUAX     .  SUBTRACT A FROM X
002D                  SEQ    .SUBAX    .  SUBTRACT B,A FROM X
002E                  SEQ    .SUXBA    .  SUBTRACT X FROM B,A
002F                  SEQ    .CPBAX    .  COMPARE B,A TO X
0030                  SEQ    .ASRX     .  SHIFT X RIGHT (ARITHMETIC)
0031                  SEQ    .ASLX     .  SHIFT X LEFT (ARITHMETIC/LOGICAL)
0032                  SEQ    .PSHX     .  PUSH X ON STACK
0033                  SEQ    .PULX     .  PULL X FROM STACK
0034                  SEQ    .PRINT    .  PRINT-TERMINATE WITH CR
0035                  SEQ    .PRINX    .  PRINT-TERMINATE WITH EOT
0036                  SEQ    .GETFD    .  READ FDR (RESIDENT MDOS ONLY)
0037                  SEQ    .PUTFD    .  WRITE FDR (RESIDENT MDOS ONLY)
0038                  SEQ    .PUTEF    .  WRITE EOF (RESIDENT MDOS ONLY)
0039                  SEQ    .EREAD    .  DISK READ W/ ERR RETN
003A                  SEQ    .EWRIT    .  DISK WRITE W/ ERR RETN
003B                  SEQ    .MREAD    .  MULTIPLE SECTOR READ
003C                  SEQ    .MWRIT    .  MULTIPLE SECTOR WRITE
003D                  SEQ    .MERED    .  MULTIPLE SECTOR READ W/ ERR RETUR
003E                  SEQ    .MEWRT    .  MULTIPLE SECTOR WRITE W/ ERR RETU
003F                  SEQ    .BOOT     .  RELOAD MDOS
                *
0000                  ORG    .$SAV     .  RESTORE LOCATION COUNTER
                *
                * A S C I I    C O N T R O L    C H A R A C A T E R S
                *
        0000   A NULL  EQU    0         .  NULL
        0001   A SOH   EQU    1         .  START OF HEADING
        0002   A STX   EQU    2         .  START OF TEXT
```

```
0003  A ETX    EQU   3          . END OF TEXT
0004  A EOT    EQU   4          . END OF TRANSMISSION
0005  A ENQ    EQU   5          . ENQUIRY (WRU - WHO ARE YOU)
0006  A ACK    EQU   6          . ACKNOWLEDGE
0007  A BEL    EQU   7          . BELL
0008  A BS     EQU   8          . BACKSPACE
0009  A HT     EQU   9          . HORIZONTAL TAB
000A  A LF     EQU   $A         . LINE FEED
000B  A VT     EQU   $B         . VERTICAL TAB
000C  A FF     EQU   $C         . FORM FEED
000D  A CR     EQU   $D         . CARRIAGE RETURN
000E  A SO     EQU   $E         . SHIFT OUT
000F  A SI     EQU   $F         . SHIFT IN
0010  A DLE    EQU   $10        . DATA LINK ESCAPE
0011  A DC1    EQU   $11        . DEVICE CONTROL 1
0012  A DC2    EQU   $12        . DEVICE CONTROL 2
0013  A DC3    EQU   $13        . DEVICE CONTROL 4
0014  A DC4    EQU   $14        . DEVICE CONTROL 4
0015  A NAK    EQU   $15        . NEGATIVE ACKNOWLEDGE
0016  A SYN    EQU   $16        . SYNCHRONOUS IDLE
0017  A ETB    EQU   $17        . END OF TRANSMISSION BLOCK
0018  A CAN    EQU   $18        . CANCEL
0019  A EM     EQU   $19        . END OF MEDIUM
001A  A SUB    EQU   $1A        . SUBSTITUTE
001B  A ESC    EQU   $1B        . ESCAPE
001C  A FS     EQU   $1C        . FILE SEPARATOR
001D  A GS     EQU   $1D        . GROUP SEPARATOR
001E  A RS     EQU   $1E        . RECORD SEPARATOR
001F  A US     EQU   $1F        . UNIT SEPARATOR
0020  A SPACE  EQU   $20        . SPACE (WORD SEPARATOR)
007F  A RUBOUT EQU   $7F        . DELETE (RUB OUT)
        *
        * S P E C I A L    C H A R A C T E R    E Q U A T E S
        *
002E  A SUFDLM EQU   '.         . SUFFIX DELIMETER
003B  A OPTDLM EQU   ';         . OPTIONS DELIMETER
003A  A DRVDLM EQU   ':         . LOGICAL DRIVER DELIMETER
0023  A DEVDLM EQU   '#         . GENERIC DEVICE NAME DELIMETER
002A  A FAMDLM EQU   '*         . FAMILY NAME/SUFFIX DELIMETER
0080  A E$FATL EQU   1!<7       . FATAL ERROR BIT
        *
        * M D O S    S E C T O R    E Q U A T E S
        *
0000  A SC$DID EQU   0          . DISK ID PHYSICAL SECTOR NUMBER
0001  A SC$CAT EQU   1          . CLUSTER ALLOCATION TABLE PHSYICAL
0002  A SC$LOK EQU   2          . LOCKOUT CLUSTER TABLE PHYSICAL SE
0003  A SC$DIR EQU   3          . DIRECTORY START PHYSICAL SECTOR N
0016  A SC$DRE EQU   $16        . DIRECTORY END PHYSICAL SECTOR NUM
0017  A SC$BB  EQU   $17        . BOOT BLOCK PHYSICAL SECTOR NUMBER
0018  A SC$DOS EQU   $18        . OPERATING SYSTEM PHSYICAL SECTOR
0080  A SC$SIZ EQU   128        . SECTOR SIZE IN BYTES
001A  A SC$TRK EQU   26         . NUMBER OF SECTORS/TRACK (SINGLE S
0034  A SC$TKD EQU   52         . NUMBER OF SECTORS/CYLINDER (DOUBL
0004  A SC$CLS EQU   4          . NUMBER OF SECTORS / CLUSTER
```

```
07D0   A SC$MAX EQU    2000       . MAXIMUM NO. OF USABLE SECTORS (SI
0FA4   A SC$MXD EQU    4004       . MAXIMUM NO. OF USABLE SECTORS (DO
0020   A DFCLS$ EQU    32         . DEFAULT NO. OF CLUSTERS
         *
         * D I S K   I D    S E C T O R    O F F S E T S
         *
0000   A DID$ID EQU    0          . OFFSET TO DISK ID (8 BYTES)
0008   A DID$VN EQU    8          . OFFSET TO VERSION NUMBER (2 BYTES
000A   A DID$RN EQU    10         . OFFSET TO REVISION NUMBER (2 BYTE
000C   A DID$DT EQU    12         . OFFSET TO DATE (6 BYTES)
0012   A DID$NM EQU    18         . OFFSET TO USER NAME (20 BYTES)
0026   A DID$RB EQU    38         . OFFSET TO RIB ADDRS. (20 BYTES)
         *
         * D I R E C T O R Y    E N T R Y    O F F S E T S
         *
0000   A DIR$NM EQU    0          . OFFSET TO NAME (8 BYTES)
0008   A DIR$SX EQU    8          . OFFSET TO SUFFIX (2 BYTES)
000A   A DIR$RB EQU    10         . OFFSET TO RIB ADDRESS (2 BYTES)
000C   A DIR$AT EQU    12         . OFFSET OF ATTRIBUTES (2 BYTES)
000E   A DIR$NU EQU    14         . OFFSET TO NOT USED AREA (2 BYTES)
         *
         * R . I . B .   B I N A R Y    F I L E    O F F S E T
         *
0075   A RIB$LB EQU    117        . NUMBER OF BYTES IN LAST SECTOR
0076   A RIB$SL EQU    118        . NUMBER OF SECTORS TO LOAD
0078   A RIB$LA EQU    120        . MEMORY LOAD ADDRESS
007A   A RIB$SA EQU    122        . START EXECUTION ADDRESS
         *
         * U N I F I E D   I / O    C O N T R O L    B L O C K
         *
         *                    O F F S E T S
         *
         *
0000   A IOCSTA EQU    0          . ERROR STATUS
0001   A IOCDTT EQU    1          . DATA TRANSFER TYPE
0002   A IOCDBP EQU    2          . DATA BUFFER POINTER
0004   A IOCDBS EQU    4          . DATA BUFFER START ADDRESS
0006   A IOCDBE EQU    6          . DATA BUFFER END ADDRESS
0008   A IOCGDW EQU    8          . GENERIC DEVICE TYPE/CDB ADDRESS
000A   A IOCLUN EQU    10         . LOGICAL UNIT NUMBER
000B   A IOCNAM EQU    11         . FILE NAME
000B   A IOCMLS EQU    11         . MAXIMUM REFERENCED LSN
000D   A IOCSDW EQU    13         . CURRENT SEGMENT DESCRIPTOR WORD
000F   A IOCSLS EQU    15         . 1ST LOGICAL SECTOR OF CURRENT SEG
0011   A IOCLSN EQU    17         . CURRENT LOGICAL SECTOR NUMBER
0013   A IOCSUF EQU    19         . FILE NAME SUFFIX
0013   A IOCEOF EQU    19         . LOGICAL END OF FILE
0015   A IOCRIB EQU    21         . PHYSICAL DISK ADDRESS OF R.I.B.
0017   A IOCFDF EQU    23         . FILE DESCRIPTOR FLAGS
001B   A IOCDEN EQU    27         . DIRECTORY ENTRY NUMBER
001D   A IOCSBP EQU    29         . SECTOR BUFFER POINTER/INITIAL SIZ
001F   A IOCSBS EQU    31         . SECTOR BUFFER START ADDRESS
0021   A IOCSBE EQU    33         . SECTOR BUFFER END ADDRESS
0023   A IOCSBI EQU    35         . SECTOR BUFFER INTERNAL PTR
```

```
        0025  A IOCBLN EQU    IOCSBI+2-IOCSTA . IOCB LENGTH
                *
                * U N I F I E D    I / O    E R R O R    S T A T U S E
                *
        0000  A .$SAV  SET    *         . REMEMBER THE CURRENT LOCATION COU
0000                   ORG    $0        . RESET IT TO ZERO TO USE THE SEQ M
                *
0000                   SEQ    I$NOER    . NO ERRORS, NORMAL RETURN
0001                   SEQ    I$NODV    . NO SUCH DEVICE
0002                   SEQ    I$RESV    . DEVICE RESERVED ALREADY
0003                   SEQ    I$NORV    . DEVICE NOT RESERVED
0004                   SEQ    I$NRDY    . DEVICE NOT READY
0005                   SEQ    I$IVDV    . INVALID DEVICE
0006                   SEQ    I$DUPE    . DUPLICATE FILE NAME
0007                   SEQ    I$NONM    . FILE NAME NOT FOUND
0008                   SEQ    I$CLOS    . INVALID OPEN/CLOSED FLAG
0009                   SEQ    I$EOF     . END OF FILE
000A                   SEQ    I$FTYP    . INVALID FILE TYPE
000B                   SEQ    I$DTYP    . INVALID DATA TRANSFER TYPE
000C                   SEQ    I$EOM     . END OF MEDIA
000D                   SEQ    I$BUFO    . BUFFER OVERFLOW
000E                   SEQ    I$CKSM    . CHECKSUM ERROR
000F                   SEQ    I$WRIT    . FILE IS WRITE PROTECTED
0010                   SEQ    I$DELT    . FILE IS DELETE PROTECTED
0011                   SEQ    I$RANG    . LOGICAL SECTOR NUMBER OUT OF RANG
0012                   SEQ    I$FSPC    . NO DISK FILE SPACE AVAILABLE
0013                   SEQ    I$DSPC    . NO DIRECTORY SPACE AVAILABLE
0014                   SEQ    I$SSPC    . NO SEGMENT DESCRIPTOR SPACE AVAIL
0015                   SEQ    I$IDEN    . INVALID DIR.  ENTRY NO.
0016                   SEQ    I$RIB     . INVALID RIB
0017                   SEQ    I$DEAL    . CAN'T DEALLOCATE ALL SPACE
0018                   SEQ    I$RECL    . BINARY RECORD LENGTH TOO LRGE
0019                   SEQ    I$SECB    . SECTOR BUFFER SIZE ERROR
                *
0000                   ORG    .$SAV     . RESTORE THE LOCATION COUNTER
                *
                *
                * M D O S    I N T E R N A L    V A R I A B L E
                *
                *       A N D    L O C A T I O N    E Q U A T E S
                *
        0100  A MDOS$  EQU    $100      . START OF MDOS ASECT
        0050  A CBUFL$ EQU    80        . COMMAND BUFFER LENGTH
        00AE  A CBUFF$ EQU    MDOS$-CBUFL$-2 . COMMAND BUFFER LOCATION
        00FE  A CBUFP$ EQU    CBUFF$+CBUFL$ . COMMAND BUFFER SCAN POINTER
        0100  A VERS$$ EQU    MDOS$     . VERSION #
        0102  A REVS$$ EQU    VERS$$+2 . REVISION #
        0104  A KYI$SV EQU    REVS$$+2 . SAVE AREA FOR KEYIN$ VECTOR
        0106  A ENDOS$ EQU    KYI$SV+2 . END OF MDOS
        0108  A ENDUS$ EQU    ENDOS$+2 . END OF USER PROGRAM AREA
        010A  A ENDSY$ EQU    ENDUS$+2 . END OF SYSTEM (MDOS) RAM
        010E  A RIBBA$ EQU    ENDSY$+4 . RIB BUFFER ADDRESS
        0110  A ENDRV$ EQU    RIBBA$+2 . END OF MDOS ROM VARIABLES
        0112  A GDBA$  EQU    ENDRV$+2 . GENERIC DEVICE TABLE ADDRESS
```

```
0114  A SYERR$ EQU    GDBA$+2  .  SYSTEM ERROR STATUS WORD
0116  A SWI$SV EQU    SYERR$+2 .  SWI VECTOR SAVE AREA
0118  A SWI$UV EQU    SWI$SV+2 .  SWI USER VECTOR
011A  A IRQ$UV EQU    SWI$UV+2 .  IRQ USER VECTOR
011C  A IRQ$SV EQU    IRQ$UV+2 .  IRQ VECTOR SAVE AREA
011E  A CHFLG$ EQU    IRQ$SV+2 .  CHAIN FUNCTION FLAG WORD
0120  A SYIOCB EQU    CHFLG$+2 .  SYSTEM CONSOLE  IOCB
0145  A SYPOCB EQU    SYIOCB+IOCBLN .  SYSTEM PRINTER IOCB
016A  A SYEOCB EQU    SYPOCB+IOCBLN .  ERR MSG FILE
        *
        *L O G I C A L   U N I T   N U M B E R - - B I T   D E F.
        *
0040  A LU$RES EQU    %01000000 .  IOCB RESERVED FLAG
        *
        * I O C D T T    --    B I T    D E F I N I T I O N S
        *
0000  A DT$OPP EQU    %00000000 .  OPEN UPDATE/INPUT
0001  A DT$OPI EQU    %00000001 .  OPEN INPUT MODE
0002  A DT$OPO EQU    %00000010 .  OPEN OUTPUT MODE
0003  A DT$OPU EQU    %00000011 .  OPEN UPDATE MODE
0004  A DT$NFF EQU    %00000100 .  NON-FILE FORMAT I/O FLAG
0008  A DT$TRU EQU    %00001000 .  TRUNCATE FLAG
0010  A DT$CLS EQU    %00010000 .  FILE OPEN/CLOSE FLAG
0020  A DT$SIO EQU    %00100000 .  SECTOR I/O FLAG
0040  A DT$OUT EQU    %01000000 .  OUTPUT TRANSFER TYPE
0080  A DT$INP EQU    %10000000 .  INPUT TRANSFER TYPE
        *
        * I O C F D F    --    B I T    D E F I N I T I O N S
        *
0000  A FD$FMU EQU    %00000000 .  USER DEFINED FORMAT (SECTOR I/O
0001  A FD$FMD EQU    %00000001 .  DEFAULT OBJECT REC'D FORMAT
0002  A FD$FML EQU    %00000010 .  BINARY LOAD FORMAT
0003  A FD$FMB EQU    %00000011 .  BINARY RECORD FORMAT
0005  A FD$FMA EQU    %00000101 .  ASCII RECORD FORMAT
0007  A FD$FMC EQU    %00000111 .  ASCI-CONVERTED-BINARY REC'D FORM
0008  A FD$CMP EQU    %00001000 .  SPACE COMPRESSION FLAG
0010  A FD$CON EQU    %00010000 .  CONTIGUOUS ALLOCATION FLAG
0020  A FD$SYS EQU    %00100000 .  SYSTEM FILE ATTRIBUTE
0040  A FD$DEL EQU    %01000000 .  DELETE PROTECTION ATTRIBUTE
0080  A FD$WRT EQU    %10000000 .  WRITE PROTECTION ATTRIBUTE
        *
        * U N I F I E D   I / O   C O N T R O L   D E S C R I
        *
        *                 B L O C K    O F F S E T S
        *
0000  A CDBIOC EQU    0        .  ADDRESS OF IOCB
0002  A CDBSDA EQU    2        .  SOFTWARE DRIVER ADDRESS
0004  A CDBHAD EQU    4        .  HARDWARE ADDRESS
0006  A CDBDDF EQU    6        .  DEVICE DESCRIPTOR FLAGS
0007  A CDBVDT EQU    7        .  VALID DATA TYPE
0008  A CDBDDA EQU    8        .  DEVICE DEPENDENT AREA
000A  A CDBWST EQU    10       .  WORKING STORAGE
000C  A CDBLEN EQU    CDBWST+2 .  CDB LENGTH
        *
```

```
               * C D B D D F    --    B I T   D E F I N I T I O N S
               *
0001   A DD$FMC EQU    %00000001 . ASCII-CONVERTED-BINARY IS DEFAUL
0002   A DD$LOG EQU    %00000010 . LOGICAL SECTOR I/O FLAG
0004   A DD$CNS EQU    %00000100 . CONSOLE FLAG
0008   A DD$RWD EQU    %00001000 . REWIND FLAG
0010   A DD$OCF EQU    %00010000 . OPEN/CLOSE FLAG
0020   A DD$INP EQU    %00100000 . INPUT DEVICE FLAG
0040   A DD$OUT EQU    %01000000 . OUTPUT DEVICE FLAG
0080   A DD$RES EQU    %10000000 . RESERVABLE DEVICE FLAG
               *
               * C D B V D T    --     B I T   D E F I N I T I O N S
               *
0004   A VD$BIN EQU    %00000100 . BINARY OBJECT FLAG
0008   A VD$GDB EQU    %00001000 . TEMP GDB POINTER FLAG
0010   A VD$SDA EQU    %00010000 . TEMP SDA POINTER FLAG
0080   A VD$NFF EQU    %10000000 . NON-FILE FORMAT FLAG
               *
               * D E V I C E   D R I V E R   E N T R Y   O F F S E T
               *
0000   A DV$ON  EQU    0         . DEVICE ON OFFSET
0003   A DV$OFF EQU    3         . DEVICE OFF OFFSET
0006   A DV$INT EQU    6         . DEVICE INTIALIZATION OFFSET
0009   A DV$TRM EQU    9         . DEVICE TERMINATION OFFSET
000C   A DV$IO  EQU    12        . DEVICE CHARACTER INPUT/OUTPUT OFF
               *
               * D I S K    E R O M    E Q U A T E S
               *
0000   A CURDRV EQU    0         . CURRENT DRIVE NUMBER
0001   A STRSCT EQU    1         . STARTING PHYSICAL SECTOR NUMBER
0003   A NUMSCT EQU    3         . NUMBER OF SECTORS TO OPERATE UPON
0005   A LSCTLN EQU    5         . # OF BYTES TO READ FROM LAST SECT
0006   A CURADR EQU    6         . MEMORY ADDRESS FOR DISK TRANSFER
0008   A FDSTAT EQU    8         . DISK TRANSFER STATUS
000B   A SCTCNT EQU    11        . SECTOR COUNT USED IN DETERMINING
000D   A SIDES  EQU    $D        . - ->SINGLE; + -> DOUBLE SIDED
               *
               * E R O M   E N T R Y   P O I N T S
               *
E800   A OSLOAD EQU    $E800     . BOOTSTRAP THE OPERATING SYSTEM
E822   A FDINIT EQU    $E822     . INITIALIZE THE DISK'S PIA AND SSD
E853   A CHKERR EQU    $E853     . CHECK AND PRINT ERROR FROM FDSTAT
E85A   A PRNTER EQU    $E85A     . PRINT ERROR FROM FDSTAT
E869   A READSC EQU    $E869     . READ SECTOR(S)
E86D   A READPS EQU    $E86D     . READ PARTIAL SECTOR
E86F   A RDCRC  EQU    $E86F     . READ AND CHECK FOR CRC
E872   A RWTEST EQU    $E872     . WRITE/READ TEST
E875   A RESTOR EQU    $E875     . MOVE HEAD TO TRACK 0
E878   A SEEK   EQU    $E878     . POSITION HEAD TO TRACK OF "STRSCT
E87B   A WRTEST EQU    $E87B     . WRITE TEST
E87E   A WRDDAM EQU    $E87E     . WRITE DELETED DATA MARK
E881   A WRVERF EQU    $E881     . WRITE AND VERIFY CRC
E884   A WRITSC EQU    $E884     . WRITE SECTOR(S)
               *
```

```
          * E R O M    E R R O R    E Q U A T E S
          *
0031   A ER$CRC EQU    '1        . DATA CRC ERROR
0032   A ER$WRT EQU    '2        . WRITE PROTECTED DISK
0033   A ER$RDY EQU    '3        . DISK NOT READY
0034   A ER$MRK EQU    '4        . DELETED DATA MARK ENCOUNTERED
0035   A ER$TIM EQU    '5        . TIMEOUT
0036   A ER$DAD EQU    '6        . INVALID DISK ADDRESS
0037   A ER$SEK EQU    '7        . SEEK ERROR
0038   A ER$DMA EQU    '8        . DATA ADDRESS MARK ERROR
0039   A ER$ACR EQU    '9        . ADDRESS MARK CRC ERROR
          *
          * M I S C E L L A N E O U S    E R O M    E Q U A T E S
          *
0005   A RETRY$ EQU    5         . RETRY COUNT FOR DISK READ/WRITE E
          *
          * L I N E   P R I N T E R    E R O M    E Q U A T E S
          *
EBC0   A LPINIT EQU    $EBC0     . INIT PRINTER PIA
EBCC   A LIST   EQU    $EBCC     . PRINT CONTENTS OF 'A'
EBE4   A LDATA  EQU    $EBE4     . PRINT STRING, CR/LF
EBF2   A LDATA1 EQU    $EBF2     . PRINT STRING, NO CR/LF
          *
          * E X B U G    E Q U A T E S    F O R    M D O S
          *      (PARTIAL LIST ONLY)
          *
F015   A INCHNP EQU    $F015     . INPUT CHARACTER (NO PARITY)
F018   A OUTCH  EQU    $F018     . OUTPUT ONE CHARACTER
F021   A PCRLF  EQU    $F021     . PRINT LF/CR
F024   A PDATA  EQU    $F024     . PRINT STRING
FCFD   A SBIT$  EQU    $FCFD     . BIT 7 INDICATES IRQ OCCURRED (IF
FF1F   A BRKPT$ EQU    $FF1F     . MAID'S BREAKPOINT TABLE (8 FDB'S)
FF4F   A BKPIN$ EQU    $FF4F     . EXBUG BREAKPOINTS IN MEMORY
FF53   A AECHO  EQU    $FF53     . INPUT CHARACTER ECHO FLAG (0=>ECH
FFF8   A IRQ$VC EQU    $FFF8     . IRQ VECTOR
FFFA   A SWI$VC EQU    $FFFA     . SWI VECTOR
FFFC   A NMI$VC EQU    $FFFC     . NMI VECTOR
FF8A   A XSTAK$ EQU    $FF8A     . EXBUG STACK
F0F3   A MAID$  EQU    $F0F3     . MAID ENTRY POINT
FF16   A XREG$P EQU    $FF16     . MAID P-REG.
FF18   A XREG$X EQU    $FF18     . MAID X-REG.
FF1A   A XREG$A EQU    $FF1A     . MAID A-REG.
FF1B   A XREG$B EQU    $FF1B     . MAID B-REG.
FF1C   A XREG$C EQU    $FF1C     . MAID C-REG.
FF1D   A XREG$S EQU    $FF1D     . MAID S-REG.
FF63   A BRKPE$ EQU    $FF63     . END OF MAID BREAKPOINT TABLE
FCF4   A CNACI$ EQU    $FCF4     . CONSOLE ACIA
          *
          * SPECIAL MACRO FOR THE CENTRONIX PRINTERS TO PRINT TITLES
          *       (NO LONGER USED)
          TITLE  MACR
           TTL \0
           ENDM
```

```
TOTAL ERRORS 00000--00000

.$SAV   0000  .ADAX   0028  .ADBAX  0029  .ADBX   0027  .ADDAM  0016
.ADXBA  002A  .ALLOC  0021  .ALPHA  0014  .ALUSM  001E  .ASLX   0031
.ASRX   0030  .BOOT   003F  .CHANG  001F  .CKBRK  000D  .CLOSE  0003
.CMPAR  0011  .CPBAX  002F  .DEALC  0022  .DIRSM  001C  .DMA    0019
.DREAD  000E  .DSPLX  000B  .DSPLY  000A  .DSPLZ  000C  .DWRIT  000F
.EREAD  0039  .EWORD  0023  .EWRIT  003A  .GETFD  0036  .GETLS  0007
.GETRC  0004  .KEYIN  0009  .LOAD   001B  .MDENT  001A  .MDERR  0020
.MERED  003D  .MEWRT  003E  .MMA    0018  .MOVE   0010  .MREAD  003B
.MWRIT  003C  .NUMD   0015  .OPEN   0002  .PFNAM  001D  .PRINT  0034
.PRINX  0035  .PSHX   0032  .PULX   0033  .PUTEF  0038  .PUTFD  0037
.PUTLS  0008  .PUTRC  0005  .RELES  0001  .RESRV  0000  .REWND  0006
.STCHB  0012  .STCHR  0013  .SUAX   002C  .SUBAM  0017  .SUBAX  002D
.SUBX   002B  .SUXBA  002E  .TBAX   0025  .TXBA   0024  .XBAX   0026
ACK     0006  AECHO   FF53  BEL     0007  BKPIN$  FF4F  BRKPE$  FF63
BRKPT$  FF1F  BS      0008  CAN     0018  CBUFF$  00AE  CBUFL$  0050
CBUFP$  00FE  CDBDDA  0008  CDBDDF  0006  CDBHAD  0004  CDBIOC  0000
CDBLEN  000C  CDBSDA  0002  CDBVDT  0007  CDBWST  000A  CHFLG$  011E
CHKERR  E853  CNACI$  FCF4  CR      000D  CURADR  0006  CURDRV  0000
DC1     0011  DC2     0012  DC3     0013  DC4     0014  DD$CNS  0004
DD$FMC  0001  DD$INP  0020  DD$LOG  0002  DD$OCF  0010  DD$OUT  0040
DD$RES  0080  DD$RWD  0008  DEVDLM  0023  DFCLS$  0020  DID$DT  000C
DID$ID  0000  DID$NM  0012  DID$RB  0026  DID$RN  000A  DID$VN  0008
DIR$AT  000C  DIR$NM  0000  DIR$NU  000E  DIR$RB  000A  DIR$SX  0008
DLE     0010  DRVDLM  003A  DT$CLS  0010  DT$INP  0080  DT$NFF  0004
DT$OPI  0001  DT$OPO  0002  DT$OPP  0000  DT$OPU  0003  DT$OUT  0040
DT$SIO  0020  DT$TRU  0008  DV$INT  0006  DV$IO   000C  DV$OFF  0003
DV$ON   0000  DV$TRM  0009  E$FATL  0080  EM      0019  ENDOS$  0106
ENDRV$  0110  ENDSY$  010A  ENDUS$  0108  ENQ     0005  EOT     0004
ER$ACR  0039  ER$CRC  0031  ER$DAD  0036  ER$DMA  0038  ER$MRK  0034
ER$RDY  0033  ER$SEK  0037  ER$TIM  0035  ER$WRT  0032  ESC     001B
ETB     0017  ETX     0003  FAMDLM  002A  FD$CMP  0008  FD$CON  0010
FD$DEL  0040  FD$FMA  0005  FD$FMB  0003  FD$FMC  0007  FD$FMD  0001
FD$FML  0002  FD$FMU  0000  FD$SYS  0020  FD$WRT  0080  FDINIT  E822
FDSTAT  0008  FF      000C  FS      001C  GDBA$   0112  GS      001D
HT      0009  I$BUFO  000D  I$CKSM  000E  I$CLOS  0008  I$DEAL  0017
I$DELT  0010  I$DSPC  0013  I$DTYP  000B  I$DUPE  0006  I$EOF   0009
I$EOM   000C  I$FSPC  0012  I$FTYP  000A  I$IDEN  0015  I$IVDV  0005
I$NODV  0001  I$NOER  0000  I$NONM  0007  I$NORV  0003  I$NRDY  0004
I$RANG  0011  I$RECL  0018  I$RESV  0002  I$RIB   0016  I$SECB  0019
I$SSPC  0014  I$WRIT  000F  INCHNP  F015  IOCBLN  0025  IOCDBE  0006
IOCDBP  0002  IOCDBS  0004  IOCDEN  001B  IOCDTT  0001  IOCEOF  0013
IOCFDF  0017  IOCGDW  0008  IOCLSN  0011  IOCLUN  000A  IOCMLS  000B
IOCNAM  000B  IOCRIB  0015  IOCSBE  0021  IOCSBI  0023  IOCSBP  001D
IOCSBS  001F  IOCSDW  000D  IOCSLS  000F  IOCSTA  0000  IOCSUF  0013
IRQ$SV  011C  IRQ$UV  011A  IRQ$VC  FFF8  KYI$SV  0104  LDATA   EBE4
LDATA1  EBF2  LF      000A  LIST    EBCC  LPINIT  EBC0  LSCTLN  0005
LU$RES  0040  MAID$   F0F3  MDOS$   0100  MDOS9$  0000  MDOSF$  0000
NAK     0015  NMI$VC  FFFC  NULL    0000  NUMSCT  0003  OPTDLM  003B
OSLOAD  E800  OUTCH   F018  PCRLF   F021  PDATA   F024  PRNTER  E85A
RDCRC   E86F  READPS  E86D  READSC  E869  RESTOR  E875  RETRY$  0005
REVS$$  0102  RIB$LA  0078  RIB$LB  0075  RIB$SA  007A  RIB$SL  0076
RIBBA$  010E  RS      001E  RUBOUT  007F  RWTEST  E872  SBIT$   FCFD
```

```
SC$BB  0017  SC$CAT 0001  SC$CLS 0004  SC$DID 0000  SC$DIR 0003
SC$DOS 0018  SC$DRE 0016  SC$LOK 0002  SC$MAX 07D0  SC$MXD 0FA4
SC$SIZ 0080  SC$TKD 0034  SC$TRK 001A  SCTCNT 000B  SEEK   E878
SI     000F  SIDES  000D  SO     000E  SOH    0001  SPACE  0020
STRSCT 0001  STX    0002  SUB    001A  SUFDLM 002E  SWI$SV 0116
SWI$UV 0118  SWI$VC FFFA  SYEOCB 016A  SYERR$ 0114  SYIOCB 0120
SYN    0016  SYPOCB 0145  US     001F  VD$BIN 0004  VD$GDB 0008
VD$NFF 0080  VD$SDA 0010  VERS$$ 0100  VT     000B  WRDDAM E87E
WRITSC E884  WRTEST E87B  WRVERF E881  XREG$A FF1A  XREG$B FF1B
XREG$C FF1C  XREG$P FF16  XREG$S FF1D  XREG$X FF18  XSTAK$ FF8A
```

```
                *
                * 6809 MDOS VERSION 03.00 -- SYSTEM EQUATE FILE -- 01/26/79
                *


                *
                *DEFINE TYPE OF MDOS--RESIDENT MDOS ONLY
                *
        0000  A MDOSF$ EQU    0          . 0 => MDOS, 1 => OEM MDOS
        0001  A MDOS9$ EQU    1          . 0 => MDOS, 1 => MDOS09


                *
                * S K I P 2     M A C R O
                *
                * THE GENERATED BYTE IS A "COMPARE INDEX IMMEDIATE".
                * THE EXECUTION OF THE BYTE WILL CHANGE THE CONDITION CODES
ONLY.
                * NO REGISTERS ARE AFFECTED.  THUS, A ONE BYTE INSTRUCTION
                * IS FORMED THAT SKIPS FORWARD TWO BYTES.
                *
                SKIP2  MACR
                 FCB $8C .
                 ENDM
                *
                * S K I P 1     M A C R O
                *
                * THE SAME CONCEPT AS THE "SKIP2" MACRO IS USED, EXCEPT THAT

                * A "BIT TEST ACCUMULATOR A IMMEDIATE" OP CODE IS GENERATED.

                *
                SKIP1  MACR
                 FCB $85
                 ENDM
                *
                * S C A L L     M A C R O    (SYSTEM FUNCTION CALL)
                *
                SCALL  MACR
                 IFEQ NARG-1
                 SWI
                 FCB \0!.%01111111
                 ENDC
                *
                 IFNE NARG-1
                 FAIL * UNDEFINED SWI CALL ARGUMENT *
                 ENDC
                 ENDM
                *
                * U C A L L     M A C R O    (USER FUNCTION CALL)
                *
```

```
                   UCALL  MACR
                    IFEQ NARG-1
                    SWI
                    FCB \0!+%10000000
                    ENDC
                   *
                    IFNE NARG-1
                    SCALL
                    ENDC
                    ENDM
                   *
                   * S E Q    M A C R O      (NUMBERING SEQUENTIAL EQUATES)
                   *
                   SEQ    MACR
                    IFNE NARG
                   \0 EQU * .
                    ENDC
                    ORG *+1 .
                    ENDM
                   *
                   * S Y S T E M    F U N C T I O N    D E F I N I T I O N S
                   *
                   *
                   * SET LOCATION COUNT TO 0 FOR THE EQUATE DEFINITIONS
                   *
           0000  A .$SAV  SET    *         .  SAVE OLD LOCATION COUNT
                          ORG    $0        .
                   *
                   *
                   *
0000                      SEQ    .RESRV    .  RESERVE A DEVICE
0001                      SEQ    .RELES    .  RELEASE A DEVICE
0002                      SEQ    .OPEN     .  OPEN A FILE
0003                      SEQ    .CLOSE    .  CLOSE A FILE
0004                      SEQ    .GETRC    .  READ A RECORD
0005                      SEQ    .PUTRC    .  WRITE A RECORD
0006                      SEQ    .REWND    .  POSITION TO BEGINNING OF FILE
0007                      SEQ    .GETLS    .  READ LOGICAL SECTOR
0008                      SEQ    .PUTLS    .  WRITE LOGICAL SECTOR
0009                      SEQ    .KEYIN    .  CONSOLE INPUT
000A                      SEQ    .DSPLY    .  CONSOLE OUTPUT (TERM W/ CR)
000B                      SEQ    .DSPLX    .  CONSOLE OUTPUT (TERM W/ EOT)
000C                      SEQ    .DSPLZ    .  CONSOLE OUTPUT (TERM W/ EOT, NO CR
000D                      SEQ    .CKBRK    .  CHECK CONSOLE FOR BREAK KEY
000E                      SEQ    .DREAD    .  EROM DISK READ
000F                      SEQ    .DWRIT    .  EROM DISK WRITE
0010                      SEQ    .MOVE     .  MOVE A STRING
0011                      SEQ    .CMPAR    .  COMPARE STRINGS
0012                      SEQ    .STCHB    .  STORE BLANKS
0013                      SEQ    .STCHR    .  STORE CHARACTERS
0014                      SEQ    .ALPHA    .  CHECK ALPHABETIC CHARACTER
0015                      SEQ    .NUMD     .  CHECK DECIMAL DIGIT
0016                      SEQ    .ADDAM    .  INCREMENT MEMORY (DOUBLE BYTE) BY
0017                      SEQ    .SUBAM    .  DECREMENT MEMORY (DOUBLE BYTE) BY
```

```
0018                       SEQ     .MMA      .  MULTIPLY (SHIFT LEFT) MEMORY BY A
0019                       SEQ     .DMA      .  DIVIDE (SHIFT RIGHT) MEMORY BY A
001A                       SEQ     .MDENT    .  ENTER MDOS WITHOUT RELOADING
001B                       SEQ     .LOAD     .  LOAD A FILE FROM DISK
001C                       SEQ     .DIRSM    .  DIRECTORY SEARCH AND MODIFY
001D                       SEQ     .PFNAM    .  PROCESS FILE NAME
001E                       SEQ     .ALUSM    .  ALLOCATE USER MEMORY
001F                       SEQ     .CHANG    .  CHANGE NAME/ATTRIBUTES
0020                       SEQ     .MDERR    .  MDOS ERROR MESSAGE HANDLER
0021                       SEQ     .ALLOC    .  ALLOCATE DISK SPACE
0022                       SEQ     .DEALC    .  RETURN DISK SPACE
0023                       SEQ     .EWORD    .  SET ERROR STATUS WORD FOR CHAIN
0024                       SEQ     .TXBA     .  TRANSFER X TO B,A
0025                       SEQ     .TBAX     .  TRANSFER B,A TO X
0026                       SEQ     .XBAX     .  EXCHANGE B,A AND X
0027                       SEQ     .ADBX     .  ADD B TO X
0028                       SEQ     .ADAX     .  ADD A TO X
0029                       SEQ     .ADBAX    .  ADD B,A TO X
002A                       SEQ     .ADXBA    .  ADD X TO B,A
002B                       SEQ     .SUBX     .  SUBTRACT B FROM X
002C                       SEQ     .SUAX     .  SUBTRACT A FROM X
002D                       SEQ     .SUBAX    .  SUBTRACT B,A FROM X
002E                       SEQ     .SUXBA    .  SUBTRACT X FROM B,A
002F                       SEQ     .CPBAX    .  COMPARE B,A TO X
0030                       SEQ     .ASRX     .  SHIFT X RIGHT (ARITHMETIC)
0031                       SEQ     .ASLX     .  SHIFT X LEFT (ARITHMETIC/LOGICAL)
0032                       SEQ     .PSHX     .  PUSH X ON STACK
0033                       SEQ     .PULX     .  PULL X FROM STACK
0034                       SEQ     .PRINT    .  PRINT-TERMINATE WITH CR
0035                       SEQ     .PRINX    .  PRINT-TERMINATE WITH EOT
0036                       SEQ     .GETFD    .  READ FDR (RESIDENT MDOS ONLY)
0037                       SEQ     .PUTFD    .  WRITE FDR (RESIDENT MDOS ONLY)
0038                       SEQ     .PUTEF    .  WRITE EOF (RESIDENT MDOS ONLY)
0039                       SEQ     .EREAD    .  DISK READ W/ ERR RETN
003A                       SEQ     .EWRIT    .  DISK WRITE W/ ERR RETN
003B                       SEQ     .MREAD    .  MULTIPLE SECTOR READ
003C                       SEQ     .MWRIT    .  MULTIPLE SECTOR WRITE
003D                       SEQ     .MERED    .  MULTIPLE SECTOR READ W/ ERR RETURN
003E                       SEQ     .MEWRT    .  MULTIPLE SECTOR WRITE W/ ERR RETURN

003F                       SEQ     .BOOT     .  RELOAD MDOS
                    *
0000                       ORG     .$SAV     .  RESTORE LOCATION COUNTER
                    *
                    * A S C I I    C O N T R O L    C H A R A C A T E R S
                    *
        0000   A NULL      EQU     0         .  NULL
        0001   A SOH       EQU     1         .  START OF HEADING
        0002   A STX       EQU     2         .  START OF TEXT
        0003   A ETX       EQU     3         .  END OF TEXT
        0004   A EOT       EQU     4         .  END OF TRANSMISSION
        0005   A ENQ       EQU     5         .  ENQUIRY (WRU - WHO ARE YOU)
        0006   A ACK       EQU     6         .  ACKNOWLEDGE
        0007   A BEL       EQU     7         .  BELL
```

```
0008   A BS     EQU    8         . BACKSPACE
0009   A HT     EQU    9         . HORIZONTAL TAB
000A   A LF     EQU    $A        . LINE FEED
000B   A VT     EQU    $B        . VERTICAL TAB
000C   A FF     EQU    $C        . FORM FEED
000D   A CR     EQU    $D        . CARRIAGE RETURN
000E   A SO     EQU    $E        . SHIFT OUT
000F   A SI     EQU    $F        . SHIFT IN
0010   A DLE    EQU    $10       . DATA LINK ESCAPE
0011   A DC1    EQU    $11       . DEVICE CONTROL 1
0012   A DC2    EQU    $12       . DEVICE CONTROL 2
0013   A DC3    EQU    $13       . DEVICE CONTROL 4
0014   A DC4    EQU    $14       . DEVICE CONTROL 4
0015   A NAK    EQU    $15       . NEGATIVE ACKNOWLEDGE
0016   A SYN    EQU    $16       . SYNCHRONOUS IDLE
0017   A ETB    EQU    $17       . END OF TRANSMISSION BLOCK
0018   A CAN    EQU    $18       . CANCEL
0019   A EM     EQU    $19       . END OF MEDIUM
001A   A SUB    EQU    $1A       . SUBSTITUTE
001B   A ESC    EQU    $1B       . ESCAPE
001C   A FS     EQU    $1C       . FILE SEPARATOR
001D   A GS     EQU    $1D       . GROUP SEPARATOR
001E   A RS     EQU    $1E       . RECORD SEPARATOR
001F   A US     EQU    $1F       . UNIT SEPARATOR
0020   A SPACE  EQU    $20       . SPACE (WORD SEPARATOR)
007F   A RUBOUT EQU    $7F       . DELETE (RUB OUT)
         *
         * S P E C I A L    C H A R A C T E R    E Q U A T E S
         *
002E   A SUFDLM EQU    '.        . SUFFIX DELIMETER
003B   A OPTDLM EQU    ';        . OPTIONS DELIMETER
003A   A DRVDLM EQU    ':        . LOGICAL DRIVER DELIMETER
0023   A DEVDLM EQU    '#        . GENERIC DEVICE NAME DELIMETER
002A   A FAMDLM EQU    '*        . FAMILY NAME/SUFFIX DELIMETER
0080   A E$FATL EQU    1!<7      . FATAL ERROR BIT
         *
         * M D O S    S E C T O R    E Q U A T E S
         *
0000   A SC$DID EQU    0         . DISK ID PHYSICAL SECTOR NUMBER
0001   A SC$CAT EQU    1         . CLUSTER ALLOCATION TABLE PHSYICAL
0002   A SC$LOK EQU    2         . LOCKOUT CLUSTER TABLE PHYSICAL SEC
0003   A SC$DIR EQU    3         . DIRECTORY START PHYSICAL SECTOR NUM

0016   A SC$DRE EQU    $16       . DIRECTORY END PHYSICAL SECTOR NUM
0017   A SC$BB  EQU    $17       . BOOT BLOCK PHYSICAL SECTOR NUMBER
0018   A SC$DOS EQU    $18       . OPERATING SYSTEM PHSYICAL SECTOR N
0080   A SC$SIZ EQU    128       . SECTOR SIZE IN BYTES
001A   A SC$TRK EQU    26        . NUMBER OF SECTORS/TRACK (SINGLE SID

0034   A SC$TKD EQU    52        . NUMBER OF SECTORS/CYLINDER (DOUBLE
0004   A SC$CLS EQU    4         . NUMBER OF SECTORS / CLUSTER
07D0   A SC$MAX EQU    2000      . MAXIMUM NO. OF USABLE SECTORS (SI
0FA4   A SC$MXD EQU    4004      . MAXIMUM NO. OF USABLE SECTORS (DOU
0020   A DFCLS$ EQU    32        . DEFAULT NO. OF CLUSTERS
```

```
                *
                * D I S K   I D   S E C T O R   O F F S E T S
                *
0000   A DID$ID EQU    0          . OFFSET TO DISK ID (8 BYTES)
0008   A DID$VN EQU    8          . OFFSET TO VERSION NUMBER (2 BYTES)
000A   A DID$RN EQU    10         . OFFSET TO REVISION NUMBER (2 BYTES)

000C   A DID$DT EQU    12         . OFFSET TO DATE (6 BYTES)
0012   A DID$NM EQU    18         . OFFSET TO USER NAME (20 BYTES)
0026   A DID$RB EQU    38         . OFFSET TO RIB ADDRS.  (20 BYTES)
                *
                * D I R E C T O R Y   E N T R Y   O F F S E T S
                *
0000   A DIR$NM EQU    0          . OFFSET TO NAME (8 BYTES)
0008   A DIR$SX EQU    8          . OFFSET TO SUFFIX (2 BYTES)
000A   A DIR$RB EQU    10         . OFFSET TO RIB ADDRESS (2 BYTES)
000C   A DIR$AT EQU    12         . OFFSET OF ATTRIBUTES (2 BYTES)
000E   A DIR$NU EQU    14         . OFFSET TO NOT USED AREA (2 BYTES)
                *
                * R . I . B .   B I N A R Y   F I L E   O F F S E T S
                *
0075   A RIB$LB EQU    117        . NUMBER OF BYTES IN LAST SECTOR
0076   A RIB$SL EQU    118        . NUMBER OF SECTORS TO LOAD
0078   A RIB$LA EQU    120        . MEMORY LOAD ADDRESS
007A   A RIB$SA EQU    122        . START EXECUTION ADDRESS
                *
                * U N I F I E D   I / O   C O N T R O L   B L O C K
                *
                *                 O F F S E T S
                *
                *
0000   A IOCSTA EQU    0          . ERROR STATUS
0001   A IOCDTT EQU    1          . DATA TRANSFER TYPE
0002   A IOCDBP EQU    2          . DATA BUFFER POINTER
0004   A IOCDBS EQU    4          . DATA BUFFER START ADDRESS
0006   A IOCDBE EQU    6          . DATA BUFFER END ADDRESS
0008   A IOCGDW EQU    8          . GENERIC DEVICE TYPE/CDB ADDRESS
000A   A IOCLUN EQU    10         . LOGICAL UNIT NUMBER
000B   A IOCNAM EQU    11         . FILE NAME
000B   A IOCMLS EQU    11         . MAXIMUM REFERENCED LSN
000D   A IOCSDW EQU    13         . CURRENT SEGMENT DESCRIPTOR WORD
000F   A IOCSLS EQU    15         . 1ST LOGICAL SECTOR OF CURRENT SEGM
0011   A IOCLSN EQU    17         . CURRENT LOGICAL SECTOR NUMBER
0013   A IOCSUF EQU    19         . FILE NAME SUFFIX
0013   A IOCEOF EQU    19         . LOGICAL END OF FILE
0015   A IOCRIB EQU    21         . PHYSICAL DISK ADDRESS OF R.I.B.
0017   A IOCFDF EQU    23         . FILE DESCRIPTOR FLAGS
001B   A IOCDEN EQU    27         . DIRECTORY ENTRY NUMBER
001D   A IOCSBP EQU    29         . SECTOR BUFFER POINTER/INITIAL SIZE
001F   A IOCSBS EQU    31         . SECTOR BUFFER START ADDRESS
0021   A IOCSBE EQU    33         . SECTOR BUFFER END ADDRESS
0023   A IOCSBI EQU    35         . SECTOR BUFFER INTERNAL PTR
0025   A IOCBLN EQU    IOCSBI+2-IOCSTA . IOCB LENGTH
                *
```

```
          *U N I F I E D    I / O    E R R O R    S T A T U S E S
          *
0000   A .$SAV   SET    *         . REMEMBER THE CURRENT LOCATION COUN
                 ORG    $0        . RESET IT TO ZERO TO USE THE SEQ MA
          *
                 SEQ    I$NOER    . NO ERRORS, NORMAL RETURN
                 SEQ    I$NODV    . NO SUCH DEVICE
                 SEQ    I$RESV    . DEVICE RESERVED ALREADY
                 SEQ    I$NORV    . DEVICE NOT RESERVED
                 SEQ    I$NRDY    . DEVICE NOT READY
                 SEQ    I$IVDV    . INVALID DEVICE
                 SEQ    I$DUPE    . DUPLICATE FILE NAME
                 SEQ    I$NONM    . FILE NAME NOT FOUND
                 SEQ    I$CLOS    . INVALID OPEN/CLOSED FLAG
                 SEQ    I$EOF     . END OF FILE
                 SEQ    I$FTYP    . INVALID FILE TYPE
                 SEQ    I$DTYP    . INVALID DATA TRANSFER TYPE
                 SEQ    I$EOM     . END OF MEDIA
                 SEQ    I$BUFO    . BUFFER OVERFLOW
                 SEQ    I$CKSM    . CHECKSUM ERROR
                 SEQ    I$WRIT    . FILE IS WRITE PROTECTED
                 SEQ    I$DELT    . FILE IS DELETE PROTECTED
                 SEQ    I$RANG    . LOGICAL SECTOR NUMBER OUT OF RANGE
                 SEQ    I$FSPC    . NO DISK FILE SPACE AVAILABLE
                 SEQ    I$DSPC    . NO DIRECTORY SPACE AVAILABLE
                 SEQ    I$SSPC    . NO SEGMENT DESCRIPTOR SPACE AVAIL
                 SEQ    I$IDEN    . INVALID DIR. ENTRY NO.
                 SEQ    I$RIB     . INVALID RIB
                 SEQ    I$DEAL    . CAN'T DEALLOCATE ALL SPACE
                 SEQ    I$RECL    . BINARY RECORD LENGTH TOO LRGE
                 SEQ    I$SECB    . SECTOR BUFFER SIZE ERROR
          *
                 ORG    .$SAV     . RESTORE THE LOCATION COUNTER
          *
          *
          *M D O S    I N T E R N A L    V A R I A B L E
          *
          *     A N D     L O C A T I O N     E Q U A T E S
          *
0100   A MDOS$   EQU    $100      . START OF MDOS ASECT
0050   A CBUFL$  EQU    80        . COMMAND BUFFER LENGTH
00AE   A CBUFF$  EQU    MDOS$-CBUFL$-2 . COMMAND BUFFER LOCATION
00FE   A CBUFP$  EQU    CBUFF$+CBUFL$ . COMMAND BUFFER SCAN POINTER
0100   A VERS$$  EQU    MDOS$     . VERSION #
0102   A REVS$$  EQU    VERS$$+2  . REVISION #
0104   A KYI$SV  EQU    REVS$$+2  . SAVE AREA FOR KEYIN$ VECTOR
0106   A ENDOS$  EQU    KYI$SV+2  . END OF MDOS
0108   A ENDUS$  EQU    ENDOS$+2  . END OF USER PROGRAM AREA
010A   A ENDSY$  EQU    ENDUS$+2  . END OF SYSTEM (MDOS) RAM
010E   A RIBBA$  EQU    ENDSY$+4  . RIB BUFFER ADDRESS
0110   A ENDRV$  EQU    RIBBA$+2  . END OF MDOS ROM VARIABLES
0112   A GDBA$   EQU    ENDRV$+2  . GENERIC DEVICE TABLE ADDRESS
0114   A SYERR$  EQU    GDBA$+2   . SYSTEM ERROR STATUS WORD
0116   A SWI$SV  EQU    SYERR$+2  . SWI VECTOR SAVE AREA
```

```
0118  A SWI$UV EQU    SWI$SV+2 .  SWI USER VECTOR
011A  A CHFLG$ EQU    SWI$UV+2 .  CHAIN FUNCTION FLAG WORD
011C  A SYIOCB EQU    CHFLG$+2 .  SYSTEM CONSOLE  IOCB
0141  A SYPOCB EQU    SYIOCB+IOCBLN .  SYSTEM PRINTER IOCB
0166  A SYEOCB EQU    SYPOCB+IOCBLN .  ERR MSG FILE
        *
        *L O G I C A L   U N I T   N U M B E R - - B I T   D E F.
        *
0040  A LU$RES EQU    %01000000 .  IOCB RESERVED FLAG
        *
        * I O C D T T    --     B I T    D E F I N I T I O N S
        *
0000  A DT$OPP EQU    %00000000 .  OPEN UPDATE/INPUT
0001  A DT$OPI EQU    %00000001 .  OPEN INPUT MODE
0002  A DT$OPO EQU    %00000010 .  OPEN OUTPUT MODE
0003  A DT$OPU EQU    %00000011 .  OPEN UPDATE MODE
0004  A DT$NFF EQU    %00000100 .  NON-FILE FORMAT I/O FLAG
0008  A DT$TRU EQU    %00001000 .  TRUNCATE FLAG
0010  A DT$CLS EQU    %00010000 .  FILE OPEN/CLOSE FLAG
0020  A DT$SIO EQU    %00100000 .  SECTOR I/O FLAG
0040  A DT$OUT EQU    %01000000 .  OUTPUT TRANSFER TYPE
0080  A DT$INP EQU    %10000000 .  INPUT TRANSFER TYPE
        *
        * I O C F D F    --    B I T    D E F I N I T I O N S
        *
0000  A FD$FMU EQU    %00000000 .  USER DEFINED FORMAT (SECTOR I/O
0001  A FD$FMD EQU    %00000001 .  DEFAULT OBJECT REC'D FORMAT
0002  A FD$FML EQU    %00000010 .  BINARY LOAD FORMAT
0003  A FD$FMB EQU    %00000011 .  BINARY RECORD FORMAT
0005  A FD$FMA EQU    %00000101 .  ASCII RECORD FORMAT
0007  A FD$FMC EQU    %00000111 .  ASCI-CONVERTED-BINARY REC'D FORMA

0008  A FD$CMP EQU    %00001000 .  SPACE COMPRESSION FLAG
0010  A FD$CON EQU    %00010000 .  CONTIGUOUS ALLOCATION FLAG
0020  A FD$SYS EQU    %00100000 .  SYSTEM FILE ATTRIBUTE
0040  A FD$DEL EQU    %01000000 .  DELETE PROTECTION ATTRIBUTE
0080  A FD$WRT EQU    %10000000 .  WRITE PROTECTION ATTRIBUTE
        *
        * U N I F I E D   I / O   C O N T R O L   D E S C R I
        *
        *                B L O C K    O F F S E T S
        *
0000  A CDBIOC EQU    0          .  ADDRESS OF IOCB
0002  A CDBSDA EQU    2          .  SOFTWARE DRIVER ADDRESS
0004  A CDBHAD EQU    4          .  HARDWARE ADDRESS
0006  A CDBDDF EQU    6          .  DEVICE DESCRIPTOR FLAGS
0007  A CDBVDT EQU    7          .  VALID DATA TYPE
0008  A CDBDDA EQU    8          .  DEVICE DEPENDENT AREA
000A  A CDBWST EQU    10         .  WORKING STORAGE
000C  A CDBLEN EQU    CDBWST+2 .  CDB LENGTH
        *
        * C D B D D F    --    B I T    D E F I N I T I O N S
        *
0001  A DD$FMC EQU    %00000001 .  ASCII-CONVERTED-BINARY IS DEFAULT
```

```
0002  A DD$LOG EQU    %00000010 . LOGICAL SECTOR I/O FLAG
0004  A DD$CNS EQU    %00000100 . CONSOLE FLAG
0008  A DD$RWD EQU    %00001000 . REWIND FLAG
0010  A DD$OCF EQU    %00010000 . OPEN/CLOSE FLAG
0020  A DD$INP EQU    %00100000 . INPUT DEVICE FLAG
0040  A DD$OUT EQU    %01000000 . OUTPUT DEVICE FLAG
0080  A DD$RES EQU    %10000000 . RESERVABLE DEVICE FLAG
        *
        * C D B V D T   --    B I T   D E F I N I T I O N S
        *
0004  A VD$BIN EQU    %00000100 . BINARY OBJECT FLAG
0008  A VD$GDB EQU    %00001000 . TEMP GDB POINTER FLAG
0010  A VD$SDA EQU    %00010000 . TEMP SDA POINTER FLAG
0080  A VD$NFF EQU    %10000000 . NON-FILE FORMAT FLAG
        *
        * D E V I C E   D R I V E R   E N T R Y   O F F S E T S
        *
0000  A DV$ON  EQU    0         . DEVICE ON OFFSET
0003  A DV$OFF EQU    3         . DEVICE OFF OFFSET
0006  A DV$INT EQU    6         . DEVICE INTIALIZATION OFFSET
0009  A DV$TRM EQU    9         . DEVICE TERMINATION OFFSET
000C  A DV$IO  EQU    12        . DEVICE CHARACTER INPUT/OUTPUT OFF
        *
        * D I S K   E R O M    E Q U A T E S
        *
0000  A CURDRV EQU    0         . CURRENT DRIVE NUMBER
0001  A STRSCT EQU    1         . STARTING PHYSICAL SECTOR NUMBER
0003  A NUMSCT EQU    3         . NUMBER OF SECTORS TO OPERATE UPON
0005  A LSCTLN EQU    5         . # OF BYTES TO READ FROM LAST SECTOR

0006  A CURADR EQU    6         . MEMORY ADDRESS FOR DISK TRANSFER
0008  A FDSTAT EQU    8         . DISK TRANSFER STATUS
000B  A SCTCNT EQU    11        . SECTOR COUNT USED IN DETERMINING E
000D  A SIDES  EQU    $D        . - ->SINGLE; + -> DOUBLE SIDED
        *
        * E R O M   E N T R Y   P O I N T S
        *
E800  A OSLOAD EQU    $E800     . BOOTSTRAP THE OPERATING SYSTEM
E822  A FDINIT EQU    $E822     . INITIALIZE THE DISK'S PIA AND SSDA
E853  A CHKERR EQU    $E853     . CHECK AND PRINT ERROR FROM FDSTAT
E85A  A PRNTER EQU    $E85A     . PRINT ERROR FROM FDSTAT
E869  A READSC EQU    $E869     . READ SECTOR(S)
E86D  A READPS EQU    $E86D     . READ PARTIAL SECTOR
E86F  A RDCRC  EQU    $E86F     . READ AND CHECK FOR CRC
E872  A RWTEST EQU    $E872     . WRITE/READ TEST
E875  A RESTOR EQU    $E875     . MOVE HEAD TO TRACK 0
E878  A SEEK   EQU    $E878     . POSITION HEAD TO TRACK OF "STRSCT"
E87B  A WRTEST EQU    $E87B     . WRITE TEST
E87E  A WRDDAM EQU    $E87E     . WRITE DELETED DATA MARK
E881  A WRVERF EQU    $E881     . WRITE AND VERIFY CRC
E884  A WRITSC EQU    $E884     . WRITE SECTOR(S)
        *
        * E R O M   E R R O R   E Q U A T E S
        *
```

```
0031   A ER$CRC EQU    '1        . DATA CRC ERROR
0032   A ER$WRT EQU    '2        . WRITE PROTECTED DISK
0033   A ER$RDY EQU    '3        . DISK NOT READY
0034   A ER$MRK EQU    '4        . DELETED DATA MARK ENCOUNTERED
0035   A ER$TIM EQU    '5        . TIMEOUT
0036   A ER$DAD EQU    '6        . INVALID DISK ADDRESS
0037   A ER$SEK EQU    '7        . SEEK ERROR
0038   A ER$DMA EQU    '8        . DATA ADDRESS MARK ERROR
0039   A ER$ACR EQU    '9        . ADDRESS MARK CRC ERROR
         *
         * M I S C E L L A N E O U S    E R O M    E Q U A T E S
         *
0005   A RETRY$ EQU    5         . RETRY COUNT FOR DISK READ/WRITE ER
         *
         * L I N E    P R I N T E R    E R O M    E Q U A T E S
         *
EBC0   A LPINIT EQU    $EBC0     . INIT PRINTER PIA
EBCC   A LIST   EQU    $EBCC     . PRINT CONTENTS OF 'A'
EBE4   A LDATA  EQU    $EBE4     . PRINT STRING, CR/LF
EBF2   A LDATA1 EQU    $EBF2     . PRINT STRING, NO CR/LF
         *
         * E X B U G    E Q U A T E S    F O R    M D O S
         *      (INCLUDES ALL REFERENCES BUT ROLLOUT)
         *
F015   A INCHNP EQU    $F015     . INPUT CHARACTER (NO PARITY)
F018   A OUTCH  EQU    $F018     . OUTPUT ONE CHARACTER
F021   A PCRLF  EQU    $F021     . PRINT LF/CR
F024   A PDATA  EQU    $F024     . PRINT STRING
FCFD   A SBIT$  EQU    $FCFD     . BIT 7 INDICATES IRQ OCCURRED (IF 0)

FF24   A BRKPT$ EQU    $FF24     . MAID'S BREAKPOINT TABLE (8 FDB'S)
FF54   A BKPIN$ EQU    $FF54     . EXBUG BREAKPOINTS IN MEMORY
FF58   A AECHO  EQU    $FF58     . INPUT CHARACTER ECHO FLAG (0=>ECHO)

FFF2   A SW3$VC EQU    $FFF2     . SWI 3 VECTOR
FFF4   A SW2$VC EQU    $FFF4     . SWI 2 VECTOR
FFF6   A FIR$VC EQU    $FFF6     . FAST IRQ VECTOR
FFF8   A IRQ$VC EQU    $FFF8     . IRQ VECTOR
FFFA   A SWI$VC EQU    $FFFA     . SWI VECTOR
FFFC   A NMI$VC EQU    $FFFC     . NMI VECTOR
FF8F   A XSTAK$ EQU    $FF8F     . EXBUG STACK
F0F3   A MAID$  EQU    $F0F3     . MAID ENTRY POINT
FF16   A XREG$P EQU    $FF16     . MAID P-REG.
FF18   A XREG$U EQU    $FF18     . MAID U-REG.
FF1A   A XREG$Y EQU    $FF1A     . MAID Y-REG.
FF1C   A XREG$X EQU    $FF1C     . MAID X-REG.
FF1E   A XREG$D EQU    $FF1E     . MAID DP-REG.
FF1F   A XREG$B EQU    $FF1F     . MAID B-REG.
FF20   A XREG$A EQU    $FF20     . MAID A-REG.
FF21   A XREG$C EQU    $FF21     . MAID C-REG.
FF22   A XREG$S EQU    $FF22     . MAID S-REG.
FF68   A BRKPE$ EQU    $FF68     . END OF MAID BREAKPOINT TABLE
FCF4   A CNACI$ EQU    $FCF4     . CONSOLE ACIA
FF92   A LINES$ EQU    $FF92     SEARCH/LOAD/VERIFY BUFFER
```

```
        F9CF   A OCHAR$ EQU     $F9CF     OUTPUT CHAR ROUTINE WITHOUT NULL PAD
        FF02   A XPEED$ EQU     $FF02     TERMINAL SPEED FLAG
        FF67   A CAS$ET EQU     $FF67     PUNCH ON FLAG
                 *
                 * SPECIAL MACRO FOR THE CENTRONIX PRINTERS TO PRINT TITLES
                 *      (NO LONGER USED)
                 TITLE  MACR
                  TTL \0
                  ENDM
                        OPT    LIST
                 *
                 * SPECIAL OPTION -- TURN ON THE LISTING
                 *
                        TITLE  (SYMBOL  TABLE)
                        END
TOTAL ERRORS 00000--00000
TOTAL WARNINGS 00000--00000



.$SAV  0000  .ADAX  0028  .ADBAX 0029  .ADBX  0027  .ADDAM 0016
.ADXBA 002A  .ALLOC 0021  .ALPHA 0014  .ALUSM 001E  .ASLX  0031
.ASRX  0030  .BOOT  003F  .CHANG 001F  .CKBRK 000D  .CLOSE 0003
.CMPAR 0011  .CPBAX 002F  .DEALC 0022  .DIRSM 001C  .DMA   0019
.DREAD 000E  .DSPLX 000B  .DSPLY 000A  .DSPLZ 000C  .DWRIT 000F
.EREAD 0039  .EWORD 0023  .EWRIT 003A  .GETFD 0036  .GETLS 0007
.GETRC 0004  .KEYIN 0009  .LOAD  001B  .MDENT 001A  .MDERR 0020
.MERED 003D  .MEWRT 003E  .MMA   0018  .MOVE  0010  .MREAD 003B
.MWRIT 003C  .NUMD  0015  .OPEN  0002  .PFNAM 001D  .PRINT 0034
.PRINX 0035  .PSHX  0032  .PULX  0033  .PUTEF 0038  .PUTFD 0037
.PUTLS 0008  .PUTRC 0005  .RELES 0001  .RESRV 0000  .REWND 0006
.STCHB 0012  .STCHR 0013  .SUAX  002C  .SUBAM 0017  .SUBAX 002D
.SUBX  002B  .SUXBA 002E  .TBAX  0025  .TXBA  0024  .XBAX  0026
ACK    0006  AECHO  FF58  BEL    0007  BKPIN$ FF54  BRKPE$ FF68
BRKPT$ FF24  BS     0008  CAN    0018  CAS$ET FF67  CBUFF$ 00AE
CBUFL$ 0050  CBUFP$ 00FE  CDBDDA 0008  CDBDDF 0006  CDBHAD 0004
CDBIOC 0000  CDBLEN 000C  CDBSDA 0002  CDBVDT 0007  CDBWST 000A
CHFLG$ 011A  CHKERR E853  CNACI$ FCF4  CR     000D  CURADR 0006
CURDRV 0000  DC1    0011  DC2    0012  DC3    0013  DC4    0014
DD$CNS 0004  DD$FMC 0001  DD$INP 0020  DD$LOG 0002  DD$OCF 0010
DD$OUT 0040  DD$RES 0080  DD$RWD 0008  DEVDLM 0023  DFCLS$ 0020
DID$DT 000C  DID$ID 0000  DID$NM 0012  DID$RB 0026  DID$RN 000A
DID$VN 0008  DIR$AT 000C  DIR$NM 0000  DIR$NU 000E  DIR$RB 000A
DIR$SX 0008  DLE    0010  DRVDLM 003A  DT$CLS 0010  DT$INP 0080
DT$NFF 0004  DT$OPI 0001  DT$OPO 0002  DT$OPP 0000  DT$OPU 0003
DT$OUT 0040  DT$SIO 0020  DT$TRU 0008  DV$INT 0006  DV$IO  000C
DV$OFF 0003  DV$ON  0000  DV$TRM 0009  E$FATL 0080  EM     0019
ENDOS$ 0106  ENDRV$ 0110  ENDSY$ 010A  ENDUS$ 0108  ENQ    0005
EOT    0004  ER$ACR 0039  ER$CRC 0031  ER$DAD 0036  ER$DMA 0038
ER$MRK 0034  ER$RDY 0033  ER$SEK 0037  ER$TIM 0035  ER$WRT 0032
ESC    001B  ETB    0017  ETX    0003  FAMDLM 002A  FD$CMP 0008
FD$CON 0010  FD$DEL 0040  FD$FMA 0005  FD$FMB 0003  FD$FMC 0007
FD$FMD 0001  FD$FML 0002  FD$FMU 0000  FD$SYS 0020  FD$WRT 0080
FDINIT E822  FDSTAT 0008  FF     000C  FIR$VC FFF6  FS     001C
```

```
GDBA$  0112  GS     001D  HT     0009  I$BUFO 000D  I$CKSM 000E
I$CLOS 0008  I$DEAL 0017  I$DELT 0010  I$DSPC 0013  I$DTYP 000B
I$DUPE 0006  I$EOF  0009  I$EOM  000C  I$FSPC 0012  I$FTYP 000A
I$IDEN 0015  I$IVDV 0005  I$NODV 0001  I$NOER 0000  I$NONM 0007
I$NORV 0003  I$NRDY 0004  I$RANG 0011  I$RECL 0018  I$RESV 0002
I$RIB  0016  I$SECB 0019  I$SSPC 0014  I$WRIT 000F  INCHNP F015
IOCBLN 0025  IOCDBE 0006  IOCDBP 0002  IOCDBS 0004  IOCDEN 001B
IOCDTT 0001  IOCEOF 0013  IOCFDF 0017  IOCGDW 0008  IOCLSN 0011
IOCLUN 000A  IOCMLS 000B  IOCNAM 000B  IOCRIB 0015  IOCSBE 0021
IOCSBI 0023  IOCSBP 001D  IOCSBS 001F  IOCSDW 000D  IOCSLS 000F
IOCSTA 0000  IOCSUF 0013  IRQ$VC FFF8  KYI$SV 0104  LDATA  EBE4
LDATA1 EBF2  LF     000A  LINES$ FF92  LIST   EBCC  LPINIT EBC0
LSCTLN 0005  LU$RES 0040  MAID$  F0F3  MDOS$  0100  MDOS9$ 0001
MDOSF$ 0000  NAK    0015  NMI$VC FFFC  NULL   0000  NUMSCT 0003
OCHAR$ F9CF  OPTDLM 003B  OSLOAD E800  OUTCH  F018  PCRLF  F021
PDATA  F024  PRNTER E85A  RDCRC  E86F  READPS E86D  READSC E869
RESTOR E875  RETRY$ 0005  REVS$$ 0102  RIB$LA 0078  RIB$LB 0075
RIB$SA 007A  RIB$SL 0076  RIBBA$ 010E  RS     001E  RUBOUT 007F
RWTEST E872  SBIT$  FCFD  SC$BB  0017  SC$CAT 0001  SC$CLS 0004
SC$DID 0000  SC$DIR 0003  SC$DOS 0018  SC$DRE 0016  SC$LOK 0002
SC$MAX 07D0  SC$MXD 0FA4  SC$SIZ 0080  SC$TKD 0034  SC$TRK 001A
SCTCNT 000B  SEEK   E878  SI     000F  SIDES  000D  SO     000E
SOH    0001  SPACE  0020  STRSCT 0001  STX    0002  SUB    001A
SUFDLM 002E  SW2$VC FFF4  SW3$VC FFF2  SWI$SV 0116  SWI$UV 0118
SWI$VC FFFA  SYEOCB 0166  SYERR$ 0114  SYIOCB 011C  SYN    0016
SYPOCB 0141  US     001F  VD$BIN 0004  VD$GDB 0008  VD$NFF 0080
VD$SDA 0010  VERS$$ 0100  VT     000B  WRDDAM E87E  WRITSC E884
WRTEST E87B  WRVERF E881  XPEED$ FF02  XREG$A FF20  XREG$B FF1F
XREG$C FF21  XREG$D FF1E  XREG$P FF16  XREG$S FF22  XREG$U FF18
XREG$X FF1C  XREG$Y FF1A  XSTAK$ FF8F
```

APPENDIX

J. MDOS 3.00 Differences

The following appendix contains a description of the differences between MDOS 3.00 and prior versions of MDOS. The first part of the appendix contains those differences that may have an impact on user-written programs which were based on prior versions of MDOS. The second part of the appendix contains the enhancements that are apparent to the operator at the MDOS command level. These enhancements have been separated by the version number of MDOS in which they first appeared. All of the listed enhancements are incorporated into MDOS 3.00.

J.1 Impact of MDOS 3.00 on Previous MDOS Programs

MDOS version 3.00 accommodates both the single-sided and the double-sided diskettes, a four-drive system, and multiple sector I/O. There are several items which as a result of these new features must be checked in all programs that have been developed to use prior versions of MDOS. These items are listed below.

1. A program making explicit checks for logical unit numbers 0 and 1 must be changed to accommodate the new numbers 2 and 3.

2. A program referring to the maximum number of sectors on a diskette as 2000 (decimal) or 2002, or the symbol from the MDOS equate file (SC$MAX), must be changed to accommodate the possible larger diskette sizes that can be encountered with the double-sided systems. Once a diskette has been accessed, the diskette controller variable SIDES (location $000D) will have bit seven set or cleared to indicate the number of sides on the diskette. If bit seven is one, a single-sided diskette has been accessed. If bit seven is zero, a double-sided diskette has been accessed. This variable is set up properly in all versions of the MDOS diskette controller.

   A single-sided diskette can be accessed in a double-sided drive; however, a double-sided diskette cannot be accessed in a single-sided drive.

   A new symbol has been placed into the MDOS equate file that gives the maximum number of sectors on a double-sided diskette (SC$MXD).

The double-sided diskette has no unused sectors as do the single-sided diskettes, since an integral number of clusters exists.

3. A program using the IOCB for diskette I/O must have the full IOCB configured as described in all MDOS manuals. This includes the until-now-unused entry IOCSBI.

4. A program using the IOCB for diskette I/O will have to be changed if the sector buffer at the time of the .OPEN function call is not exactly an integral number of sectors. In previous versions of MDOS, the sector size did not get checked until a subsequent I/O transfer was made (even though the entry parameters for the .OPEN call specified that IOCSBS and IOCSBE must be set up).

5. A program using the IOCB for diskette I/O will have to be changed if the sector buffer pointers (IOCSBS and IOCSBE) are altered after the .OPEN function has been called. Since .OPEN sets IOCSBI to the same value as IOCSBE, moving the sector buffer requires that all sector buffer pointers (IOCSBS, IOCSBE, and IOCSBI) be changed accordingly.

6. A program accessing logical unit 1 without first using the system function .OPEN, .DIRSM, .CHANG, or .LOAD, will have to be changed so that the read head is restored before the unit is accessed. Previous versions of MDOS restored both logical units 0 and 1 each time the system initialized and each time the MDOS command interpreter received control; however, in MDOS 3.0 this is no longer true. Thus, the diskette controller firmware entry point RESTOR must be used to restore the head on the unit to be accessed if not using one of the above system functions (which do the restore themselves). The same is true if the program is to access units 2 or 3 without first using one of these functions.

7. A program that has been designating logical unit numbers in the diskette controller variable CURDRV (location $0000) as either a zero or a non-zero value (to access either unit 0 or 1), will have to be changed so that the actual binary number is used instead. A non-zero value no longer guarantees that unit 1 will be accessed (physical I/O).

8. If a program has been using the system function .ALUSM, the entry and exit conditions have been changed. Section 27.5.5 should be consulted for a detailed description of the current entry and exit conditions.

9. Four new system functions have been provided for multiple sector physical diskette I/O. These new functions are described in sections 25.2.7 and 25.2.8. The existing system functions have not had their function numbers changed.

10. The device independent I/O functions (Unified I/O functions in previous versions of MDOS manuals) for accessing the diskette have been enhanced with the multiple sector I/O capability. Now, a sector buffer can be larger than a single sector in order to minimize the number of diskette accesses that must be made (and therefore decrease the amount of time it takes a program to run). The following areas have been affected:

    IOCSBI, the IOCB sector buffer internal pointer, is now used. This pointer indicates the end of valid data within the user's sector buffer. It is initialized by the .OPEN function to point to the end of the sector buffer (IOCSBE). It is changed by the input functions to reflect the end of the valid data (if only using a single sector, IOCSBI will always be the same as IOCSBE).

    IOCSBE, the IOCB ending sector buffer pointer, still points to the last byte in the sector buffer; however, the sector buffer can be an integral number of sectors in length (one or more).

    No program modification will be required if a program is using record I/O and if the sector buffer stays in the same place; however, changing the size of the sector buffer should speed up the program.

    Programs using logical sector I/O will not require modification if only a single sector is accommodated by the buffer and if the sector buffer is always in the same place. Thus, existing programs should be minimally impacted. If the sector buffer changes locations (single sector size), then the IOCSBI entry must be adjusted along with the IOCSBE entry to reflect the end of the valid data within the sector buffer.

    If the user supplies a sector buffer larger than one sector, then he must realize that after a .GETLS function, he may have more sectors in the buffer than just the logical sector number requested. IOCLSN will be updated to point to the logical sector to be read next (incremented by the number of sectors that were read into the buffer). Upon return from the .GETLS call, IOCSBI will point to the last valid data byte within the sector buffer (less than or equal to

IOCSBE). Thus, the user must check IOCSBI to determine the end of the data in the buffer and to calculate the number of sectors read.

The .PUTLS function will write the logically contiguous sectors from IOCSBS through IOCSBI from the buffer to the file starting at IOCLSN. IOCMLS and IOCLSN are updated as expected, and additional space may have been allocated.

J.2 Enhancements to MDOS 2.20/2.21

MDOS 2.20 was released to support the dual memory map of the EXORciser II system. Other enhancements, however, were added to the MDOS commands at the same time. MDOS 2.21 is almost identical to MDOS 2.20. A change was implemented to aid in the proper sizing of contiguous memory during initialization when running with the USE module.

1. A new command, ROLLOUT, was added to the standard command package. ROLLOUT allows the user to write blocks of memory to a diskette.

2. A new command, ECHO, was added to the standard command package. ECHO allows users with an EXORciser II system to echo all console input/output to a line printer

3. The Bootblock program may generate a new error message: EM. This message indicates that insufficient contiguous RAM exists in the system to load the resident MDOS.

4. When MDOS 2.20 or 2.21 initializes via the E800;G or MDOS command to the debug monitor, it sizes memory using a technique that will not change the contents of memory. Thus, programs can be loaded above MDOS, the system reinitialized, and another program loaded with the first program image still intact in memory (first program must load above MDOS command interpreter and LOAD command).

5. The FREE command can be invoked with the "L" option, causing its output to be directed to the line printer.

6. The REPAIR command will default to drive zero if no logical unit number is specified. In addition, if a file with a RIB error is not deleted, the user will not be able to update the CAT on diskette. A message is displayed to that effect during the last phase of the REPAIR process.

7. The COPY command allows users with an EXORtape paper tape reader to use that device. In addition, a user can provide his own device driver to be used by COPY for an input or output device.

8. The LOAD command allows files to be loaded into the User Memory Map of an EXORciser II system that has the dual memory map configured. This is done with the "U" option. The "V" option now allows programs to be loaded anywhere in memory (not below $20 or beyond $FFFF, however). In addition, the stack pointer is set to the EXbug stack area when the "V" option is specified.

9. The DEL command will display the logical unit numbers along with the file names shown as being deleted or protected. In addition, the command line processing has been changed so that a null file name among a list of multiple file names is invalid.

10. The BINEX command generates an S0 record containing the memory image file name and suffix.

12. All standard error messages are displayed with a two-digit decimal reference number to allow them to be easily looked up in the error message chapter of the new MDOS manual. Most error messages are still the same. However, the wording was changed on several to make them more uniform. Also, the "AT nnnn" phrase that accompanied many error messages has been removed to make the messages less cryptic. A new error message was added (as was a new error code) for sector I/O functions that are called with a sector buffer not 128 bytes in size.

13. The EXBIN command ignores null records (carriage return only) if encountered in the EXbug-loadable file.

14. The EOT character ($04) which was output by the .PRINX, .DSPLX and .DSPLZ functions, is no longer written to the output device.

15. The sequence of line feed, carriage return, null written to console and/or printer, has been changed to carriage return, line feed, null to eliminate the overprinting problem encountered on certain printers with an 80 character buffer when lines of 80 characters were printed.

16. The FORMAT command has been upgraded to function

with either a 1 MHz or a 2 MHz system.

17. The recovery of accidentally deleted files has been made easier for those users who refuse to keep directory listings or backup copies. The directory entry is only changed so that the first two bytes are changed to an $FF when deleted. This retains 8 characters of name and suffix out of the original 10 to make the entry visible in the directory. In addition, the RIB is no longer zero-filled when the file is deleted. Thus, the user has only to use DUMP to rebuild the two FF-ed name bytes in the directory. Then, the REPAIR program must be run immediately afterwards to reconstruct the allocation table.

J.3 Enhancements to MDOS 3.00

MDOS 3.00 was released to support EXORdisk III (four-drives, double-sided). The other major enhancement was the addition of multiple sector I/O. The implementation of this enhancement has considerably reduced the amount of time it takes all MDOS commands to execute. Commands like LIST, MERGE, COPY, DOSGEN, and EDIT show the greatest increases in speed. The other enhancements are listed below.

1. The BACKUP command has been modified to allow single-sided diskettes to be copied onto double-sided diskettes. A logical unit number specification can be entered on the command line to allow copying diskettes from units other than zero to units other than one. The "R" option no longer copies the LCAT from the source diskette. The destination diskette's LCAT is initialized completely. The "V" option no longer terminates the verify process if the system sectors in cylinder zero miscompare since the BREAK key can be used to abort the process at any time.

2. The COPY command's "V" option will cause the miscomparisons to be displayed between sectors or records when verifying files. The "L" option can be used to direct this display to the line printer. The "B" option has been added to automatically verify files after the copy has completed (diskette-to-diskette copy only).

3. The DOSGEN command has been changed allow logical unit number specifications 1-3. Either single- or double-sided diskettes can be DOSGENed. The write/read test has been changed to verify that the sectors locked out indeed have the deleted data mark written in them. The BREAK key is

sensed at times other than the file copy phase. Only one sector range can be locked out by the user. All input from the operator is entered on the same line as the input prompts.

4. The FORMAT command has been changed to allow logical units other than number one to be formatted. Both single- and double-sided diskettes can be formatted.

5. The REPAIR command has been changed so that it will work with logical units 0-3 and with single- and double-sided diskettes. In addition, the version numbers between the resident MDOS and the ID sector are compared and made the same. The version and revision numbers can no longer be changed by the operator. Several of the messages have been changed.

APPENDIX

K. IOCB Input Parameter Summary

The following appendix contains a summary of the twelve different modes in which an IOCB can be used. The tables show the entries of an IOCB labelled on the left. Across the top of each table are the names of the valid device independent I/O functions. Immediately underneath each I/O function will be the letter "N" or "Y". The "N" indicates that the function cannot be used in the mode described by the title line under each table. A "Y" indicates that the function can be used.

An "X" appears in those places where a given IOCB entry is required as an input parameter to the function in whose column the "X" appears. At the bottom of each table, the values that must be placed into the IOCB entries are summarized. Periods in the table serve as place holders to show the columns.

	RESRV	OPEN	GETRC	PUTRC	CLOSE	RELES	GETLS	PUTLS	REWND
VALID CALL	Y	Y	Y	N	Y	Y	N	N	Y
IOCB ENTRY									
IOCSTA	.	.	.	.	.	.	.	.	.
IOCDTT	.	X	.	.	.	.	.	.	.
IOCDBP	.	.	.	.	.	.	.	.	.
IOCDBS	.	.	X	.	.	.	.	.	.
IOCDBE	.	.	X	.	.	.	.	.	.
IOCGDW	X	.	.	.	.	.	.	.	.
IOCLUN	X	.	.	.	.	.	.	.	.
IOCNAM/MLS	.	X	.	.	.	.	.	.	.
/SDW	.	X	.	.	.	.	.	.	.
/SLS	.	X	.	.	.	.	.	.	.
/LSN	.	X	.	.	.	.	.	.	.
IOCSUF/EOF	.	X	.	.	.	.	.	.	.
IOCRIB	.	.	.	.	.	.	.	.	.
IOCFDF	.	.	.	.	.	.	.	.	.
----	.	.	.	.	.	.	.	.	.
IOCDEN	.	.	.	.	.	.	.	.	.
IOCSBP/SIZ	.	.	.	.	.	.	.	.	.
IOCSBS	.	X	.	.	.	.	.	.	.
IOCSBE	.	X	.	.	.	.	.	.	.
IOCSBI	.	.	.	.	.	.	.	.	.

Diskette Device -- Record Processing, Input (Existing File)

```
IOCDTT = DT$CLS + DT$OPI
IOCGDW = DK
IOCLUN = 0-3
IOCNAM = File name of existing file
IOCSUF = Suffix
IOCSBS = Sector buffer start
IOCSBE = Sector buffer end
IOCDBS = Data buffer start
IOCDBE = Data buffer end
```

	RESRV	OPEN	GETRC	PUTRC	CLOSE	RELES	GETLS	PUTLS	REWND
VALID CALL	Y	Y	N	Y	Y	Y	N	N	N
IOCB ENTRY									
IOCSTA	.	.	.	.	.	.	.	.	.
IOCDTT	.	X	.	.	.	.	.	.	.
IOCDBP	.	.	.	.	.	.	.	.	.
IOCDBS	.	.	.	X	.	.	.	.	.
IOCDBE	.	.	.	X	.	.	.	.	.
IOCGDW	X	.	.	.	.	.	.	.	.
IOCLUN	X	.	.	.	.	.	.	.	.
IOCNAM/MLS	.	X	.	.	.	.	.	.	.
/SDW	.	X	.	.	.	.	.	.	.
/SLS	.	X	.	.	.	.	.	.	.
/LSN	.	X	.	.	.	.	.	.	.
IOCSUF/EOF	.	X	.	.	.	.	.	.	.
IOCRIB	.	.	.	.	.	.	.	.	.
IOCFDF	.	X	.	.	.	.	.	.	.
----	.	.	.	.	.	.	.	.	.
IOCDEN	.	.	.	.	.	.	.	.	.
IOCSBP/SIZ	.	X	.	.	.	.	.	.	.
IOCSBS	.	X	.	.	.	.	.	.	.
IOCSBE	.	X	.	.	.	.	.	.	.
IOCSBI	.	.	.	.	.	.	.	.	.

Diskette Device -- Record Procesing, Output (New file)

IOCDTT	= DT$CLS + DT$OPO
IOCGDW	= DK
IOCLUN	= 0-3
IOCNAM	= File name of new file
IOCSUF	= Suffix
IOCFDF	= FD$FMA or FD$FMB plus other optional attributes
IOCSIZ	= 0 (Default size) or specific size
IOCSBS	= Sector buffer start
IOCSBE	= Sector buffer end
IOCDBS	= Data buffer start
IOCDBE	= Data buffer end

	RESRV	OPEN	GETRC	PUTRC	CLOSE	RELES	GETLS	PUTLS	REWND
VALID CALL	Y	Y	Y	Y	Y	Y	N	N	Y
IOCB ENTRY									
IOCSTA	.	.	.	.	.	.	.	.	.
IOCDTT	.	X	.	.	.	.	.	.	.
IOCDBP	.	.	.	.	.	.	.	.	.
IOCDBS	.	.	X	X	.	.	.	.	.
IOCDBE	.	.	X	X	.	.	.	.	.
IOCGDW	X	.	.	.	.	.	.	.	.
IOCLUN	X	.	.	.	.	.	.	.	.
IOCNAM/MLS	.	X	.	.	.	.	.	.	.
/SDW	.	X	.	.	.	.	.	.	.
/SLS	.	X	.	.	.	.	.	.	.
/LSN	.	X	.	.	.	.	.	.	.
IOCSUF/EOF	.	X	.	.	.	.	.	.	.
IOCRIB	.	.	.	.	.	.	.	.	.
IOCFDF	.	X	.	.	.	.	.	.	.
----	.	.	.	.	.	.	.	.	.
IOCDEN	.	.	.	.	.	.	.	.	.
IOCSBP/SIZ	.	X	.	.	.	.	.	.	.
IOCSBS	.	X	.	.	.	.	.	.	.
IOCSBE	.	X	.	.	.	.	.	.	.
IOCSBI	.	.	.	.	.	.	.	.	.

Diskette Device -- Record Processing, Update (New File)

IOCDTT	= DT$CLS + DT$OPU
IOCGDW	= DK
IOCLUN	= 0-3
IOCNAM	= File name of new file
IOCSUF	= Suffix
IOCFDF	= FD$FMA or FD$FMB plus other optional attributes
IOCSIZ	= 0 (Default size) or specific size
IOCSBS	= Sector buffer start
IOCSBE	= Sector buffer end
IOCDBS	= Data buffer start
IOCDBE	= Data buffer end

	RESRV	OPEN	GETRC	PUTRC	CLOSE	RELES	GETLS	PUTLS	REWND
VALID CALL	Y	Y	Y	Y	Y	Y	N	N	Y
IOCB ENTRY									
IOCSTA	.	.	.	.	.	.	.	.	.
IOCDTT	.	X	.	.	X	.	.	.	.
IOCDBP	.	.	.	.	.	.	.	.	.
IOCDBS	.	.	X	X	.	.	.	.	.
IOCDBE	.	.	X	X	.	.	.	.	.
IOCGDW	X	.	.	.	.	.	.	.	.
IOCLUN	X	.	.	.	.	.	.	.	.
IOCNAM/MLS	.	X	.	.	X	.	.	.	.
/SDW	.	X	.	.	.	.	.	.	.
/SLS	.	X	.	.	.	.	.	.	.
/LSN	.	X	.	.	.	.	.	.	.
IOCSUF/EOF	.	X	.	.	.	.	.	.	.
IOCRIB	.	.	.	.	.	.	.	.	.
IOCFDF	.	.	.	.	.	.	.	.	.
----	.	.	.	.	.	.	.	.	.
IOCDEN	.	.	.	.	.	.	.	.	.
IOCSBP/SIZ	.	.	.	.	.	.	.	.	.
IOCSBS	.	X	.	.	.	.	.	.	.
IOCSBE	.	X	.	.	.	.	.	.	.
IOCSBI	.	.	.	.	.	.	.	.	.

Diskette Device -- Record Processing, Update (Existing file)

IOCDTT	= DT$CLS + DT$OPP
IOCGDW	= DK
IOCLUN	= 0-3
IOCNAM	= File name
IOCSUF	= Suffix
IOCSBS	= Sector buffer start
IOCSBE	= Sector buffer end
IOCDBS	= Data buffer start
IOCDBE	= Data buffer end

	RESRV	OPEN	GETRC	PUTRC	CLOSE	RELES	GETLS	PUTLS	REWND
VALID CALL	Y	Y	N	N	Y	Y	Y	N	Y
IOCB ENTRY									
IOCSTA	.	.	.	.	.	.	.	.	.
IOCDTT	.	X	.	.	.	.	.	.	.
IOCDBP	.	.	.	.	.	.	.	.	.
IOCDBS	.	.	.	.	.	.	.	.	.
IOCDBE	.	.	.	.	.	.	.	.	.
IOCGDW	X	.	.	.	.	.	.	.	.
IOCLUN	X	.	.	.	.	.	.	.	.
IOCNAM/MLS	.	X	.	.	.	.	.	.	.
/SDW	.	X	.	.	.	.	.	.	.
/SLS	.	X	.	.	.	.	.	.	.
/LSN	.	X	.	.	.	.	X	.	.
IOCSUF/EOF	.	X	.	.	.	.	.	.	.
IOCRIB	.	.	.	.	.	.	.	.	.
IOCFDF	.	.	.	.	.	.	.	.	.
----	.	.	.	.	.	.	.	.	.
IOCDEN	.	.	.	.	.	.	.	.	.
IOCSBP/SIZ	.	.	.	.	.	.	.	.	.
IOCSBS	.	X	.	.	.	.	.	.	.
IOCSBE	.	X	.	.	.	.	.	.	.
IOCSBI	.	.	.	.	.	.	.	.	.

Diskette Device -- Logical Sector Processing, Input
(Existing file)

IOCDTT	=	DT$CLS + DT$OPI + DT$SIO
IOCGDW	=	DK
IOCLUN	=	0-3
IOCNAM	=	File name of existing file
IOCSUF	=	Suffix
IOCLSN	=	Starting logical sector number to be read
IOCSBS	=	Sector buffer start
IOCSBE	=	Sector buffer end

	RESRV	OPEN	GETRC	PUTRC	CLOSE	RELES	GETLS	PUTLS	REWND
VALID CALL	Y	Y	N	N	Y	Y	N	Y	N
IOCB ENTRY									
IOCSTA	.	.	.	.	.	.	.	.	.
IOCDTT	.	X	.	.	.	.	.	.	.
IOCDBP	.	.	.	.	.	.	.	.	.
IOCDBS	.	.	.	.	.	.	.	.	.
IOCDBE	.	.	.	.	.	.	.	.	.
IOCGDW	X	.	.	.	.	.	.	.	.
IOCLUN	X	.	.	.	.	.	.	.	.
IOCNAM/MLS	.	X	.	.	.	.	.	.	.
/SDW	.	X	.	.	.	.	.	.	.
/SLS	.	X	.	.	.	.	.	.	.
/LSN	.	X	.	.	.	.	.	X	.
IOCSUF/EOF	.	X	.	.	.	.	.	.	.
IOCRIB	.	.	.	.	.	.	.	.	.
IOCFDF	.	X	.	.	.	.	.	.	.
----	.	.	.	.	.	.	.	.	.
IOCDEN	.	.	.	.	.	.	.	.	.
IOCSBP/SIZ	.	X	.	.	.	.	.	.	.
IOCSBS	.	X	.	.	.	.	.	X	.
IOCSBE	.	X	.	.	.	.	.	.	.
IOCSBI	.	.	.	.	.	.	.	X	.

Diskette Device -- Logical Sector Processing, Output
(New file)

IOCDTT	=	DT$CLS + DT$OPO + DT$SIO
IOCGDW	=	DK
IOCLUN	=	0-3
IOCNAM	=	File name of new file
IOCSUF	=	Suffix
IOCFDF	=	Optional attributes
IOCLSN	=	Starting logical sector number to be written
IOCSIZ	=	0 (Default size) or specific size
IOCSBS	=	Sector buffer start
IOCSBI	=	Sector buffer end

	RESRV	OPEN	GETRC	PUTRC	CLOSE	RELES	GETLS	PUTLS	REWND
VALID CALL	Y	Y	N	N	Y	Y	Y	Y	Y
IOCB ENTRY									
IOCSTA	.	.	.	.	.	.	.	.	.
IOCDTT	.	X	.	.	.	.	.	.	.
IOCDBP	.	.	.	.	.	.	.	.	.
IOCDBS	.	.	.	.	.	.	.	.	.
IOCDBE	.	.	.	.	.	.	.	.	.
IOCGDW	X	.	.	.	.	.	.	.	.
IOCLUN	X	.	.	.	.	.	.	.	.
IOCNAM/MLS	.	X	.	.	.	.	.	.	.
/SDW	.	X	.	.	.	.	.	.	.
/SLS	.	X	.	.	.	.	.	.	.
/LSN	.	X	.	.	.	.	X	X	.
IOCSUF/EOF	.	X	.	.	.	.	.	.	.
IOCRIB	.	.	.	.	.	.	.	.	.
IOCFDF	.	X	.	.	.	.	.	.	.
----	.	.	.	.	.	.	.	.	.
IOCDEN	.	.	.	.	.	.	.	.	.
IOCSBP/SIZ	.	X	.	.	.	.	.	.	.
IOCSBS	.	X	.	.	.	.	X	X	.
IOCSBE	.	X	.	.	.	.	X	.	.
IOCSBI	.	.	.	.	.	.	.	X	.

Diskette Device -- Logical Sector Processing, Update
(New file)

```
IOCDTT = DT$CLS + DT$OPU + DT$SIO
IOCGDW = DK
IOCLUN = 0-3
IOCNAM = File name of new file
IOCSUF = Suffix
IOCFDF = Optional attributes
IOCLSN = Starting logical sector number
IOCSIZ = 0 (Default size) or specific size
IOCSBS = Sector buffer start
IOCSBE = Sector buffer end
IOCSBI = Sector buffer end
```

	RESRV	OPEN	GETRC	PUTRC	CLOSE	RELES	GETLS	PUTLS	REWND
VALID CALL	Y	Y	N	N	Y	Y	Y	Y	Y
IOCB ENTRY									
IOCSTA	.	.	.	.	.	.	.	.	.
IOCDTT	.	X	.	.	X	.	.	.	.
IOCDBP	.	.	.	.	.	.	.	.	.
IOCDBS	.	.	.	.	.	.	.	.	.
IOCDBE	.	.	.	.	.	.	.	.	.
IOCGDW	X	.	.	.	.	.	.	.	.
IOCLUN	X	.	.	.	.	.	.	.	.
IOCNAM/MLS	.	X	.	.	X	.	.	.	.
/SDW	.	X	.	.	.	.	.	.	.
/SLS	.	X	.	.	.	.	.	.	.
/LSN	.	X	.	.	.	.	X	X	.
IOCSUF/EOF	.	X	.	.	.	.	.	.	.
IOCRIB	.	.	.	.	.	.	.	.	.
IOCFDF	.	.	.	.	.	.	.	.	.
----	.	.	.	.	.	.	.	.	.
IOCDEN	.	.	.	.	.	.	.	.	.
IOCSBP/SIZ	.	.	.	.	.	.	.	.	.
IOCSBS	.	X	.	.	.	.	X	X	.
IOCSBE	.	X	.	.	.	.	X	.	.
IOCSBI	.	.	.	.	.	.	.	X	.

Diskette Device -- Logical Sector Processing, Update
(Existing File)

IOCDTT	= DT$CLS + DT$OPP + DT$SIO
IOCGDW	= DK
IOCLUN	= 0-3
IOCNAM	= File name of existing file
IOCSUF	= Suffix
IOCLSN	= Starting logical sector number
IOCSBS	= Sector buffer start
IOCSBE	= Sector buffer end
IOCSBI	= Sector buffer end

	RESRV	OPEN	GETRC	PUTRC	CLOSE	RELES	GETLS	PUTLS	REWND
VALID CALL	Y	Y	Y	N	Y	Y	N	N	N
IOCB ENTRY									
IOCSTA	.	.	.	.	.	.	.	.	.
IOCDTT	.	X	.	.	.	.	.	.	.
IOCDBP	.	.	.	.	.	.	.	.	.
IOCDBS	.	.	X	.	.	.	.	.	.
IOCDBE	.	.	X	.	.	.	.	.	.
IOCGDW	X	.	.	.	.	.	.	.	.
IOCLUN	X	.	.	.	.	.	.	.	.
IOCNAM/MLS	.	.	.	.	.	.	.	.	.
/SDW	.	.	.	.	.	.	.	.	.
/SLS	.	.	.	.	.	.	.	.	.
/LSN	.	.	.	.	.	.	.	.	.
IOCSUF/EOF	.	.	X	.	.	.	.	.	.
IOCRIB	.	.	.	.	.	.	.	.	.
IOCFDF	X	.	.	.	.	.	.	.	.
----	.	.	.	.	.	.	.	.	.
IOCDEN	.	.	.	.	.	.	.	.	.
IOCSBP/SIZ	.	.	.	.	.	.	.	.	.
IOCSBS	.	.	.	.	.	.	.	.	.
IOCSBE	.	.	.	.	.	.	.	.	.
IOCSBI	.	.	.	.	.	.	.	.	.

Non-diskette Device -- Non-file Format, Input

```
IOCDTT = DT$CLS + DT$NFF + DT$OPI
IOCGDW = CN or CR
IOCLUN = 0
IOCFDF = FD$FMA
IOCSUF = Display prompt if device is CN
IOCDBS = Data buffer start
IOCDBE = Data buffer end
```

	RESRV	OPEN	GETRC	PUTRC	CLOSE	RELES	GETLS	PUTLS	REWND
VALID CALL	Y	Y	N	Y	Y	Y	N	N	N
IOCB ENTRY									
IOCSTA	.	.	.	.	.	.	.	.	.
IOCDTT	.	X	.	.	.	.	.	.	.
IOCDBP	.	.	.	.	.	.	.	.	.
IOCDBS	.	.	.	X	X	.	.	.	.
IOCDBE	.	.	.	X	X	.	.	.	.
IOCGDW	X	.	.	.	.	.	.	.	.
IOCLUN	X	.	.	.	.	.	.	.	.
IOCNAM/MLS	.	.	.	.	.	.	.	.	.
/SDW	.	.	.	.	.	.	.	.	.
/SLS	.	.	.	.	.	.	.	.	.
/LSN	.	.	.	.	.	.	.	.	.
IOCSUF/EOF	.	.	.	.	.	.	.	.	.
IOCRIB	.	.	.	.	.	.	.	.	.
IOCFDF	.	X	.	.	.	.	.	.	.
----	.	.	.	.	.	.	.	.	.
IOCDEN	.	.	.	.	.	.	.	.	.
IOCSBP/SIZ	.	.	.	.	.	.	.	.	.
IOCSBS	.	.	.	.	.	.	.	.	.
IOCSBE	.	.	.	.	.	.	.	.	.
IOCSBI	.	.	.	.	.	.	.	.	.

Non-diskette Device -- Non-file Format, Output

```
IOCDTT = DT$CLS + DT$NFF + DT$OPO
IOCGDW = LP, CN, or CP
IOCLUN = 0
IOCFDF = FD$FMA
IOCDBS = Data buffer start
IOCDBE = Data buffer end
```

	RESRV	OPEN	GETRC	PUTRC	CLOSE	RELES	GETLS	PUTLS	REWND
VALID CALL	Y	Y	Y	N	Y	Y	N	N	N
IOCB ENTRY									
IOCSTA	.	.	.	.	.	.	.	.	.
IOCDTT	.	X	.	.	.	.	.	.	.
IOCDBP	.	.	.	.	.	.	.	.	.
IOCDBS	.	X	X	.	.	.	.	.	.
IOCDBE	.	X	X	.	.	.	.	.	.
IOCGDW	X	.	.	.	.	.	.	.	.
IOCLUN	X	.	.	.	.	.	.	.	.
IOCNAM/MLS	.	X	.	.	.	.	.	.	.
/SDW	.	X	.	.	.	.	.	.	.
/SLS	.	X	.	.	.	.	.	.	.
/LSN	.	X	.	.	.	.	.	.	.
IOCSUF/EOF	.	X	.	.	.	.	.	.	.
IOCRIB	.	.	.	.	.	.	.	.	.
IOCFDF	.	.	.	.	.	.	.	.	.
----	.	.	.	.	.	.	.	.	.
IOCDEN	.	.	.	.	.	.	.	.	.
IOCSBP/SIZ	.	.	.	.	.	.	.	.	.
IOCSBS	.	.	.	.	.	.	.	.	.
IOCSBE	.	.	.	.	.	.	.	.	.
IOCSBI	.	.	.	.	.	.	.	.	.

Non-diskette Device -- File Format, Input

```
IOCDTT = DT$CLS + DT$OPI
IOCGDW = CR
IOCLUN = 0
IOCDBS = Data buffer start (used for FDR processing)
IOCDBE = Data buffer end
IOCNAM = File name of existing file
IOCSUF = Suffix
```

	RESRV	OPEN	GETRC	PUTRC	CLOSE	RELES	GETLS	PUTLS	REWND
VALID CALL	Y	Y	N	Y	Y	Y	N	N	N
IOCB ENTRY									
IOCSTA	.	.	.	.	.	.	.	.	.
IOCDTT	.	X	.	.	.	.	.	.	.
IOCDBP	.	.	.	.	.	.	.	.	.
IOCDBS	.	X	.	X	X	.	.	.	.
IOCDBE	.	X	.	X	X	.	.	.	.
IOCGDW	X	.	.	.	.	.	.	.	.
IOCLUN	X	.	.	.	.	.	.	.	.
IOCNAM/MLS	.	X	.	.	.	.	.	.	.
/SDW	.	X	.	.	.	.	.	.	.
/SLS	.	X	.	.	.	.	.	.	.
/LSN	.	X	.	.	.	.	.	.	.
IOCSUF/EOF	.	X	.	.	.	.	.	.	.
IOCRIB	.	.	.	.	.	.	.	.	.
IOCFDF	.	X	.	.	.	.	.	.	.
----	.	.	.	.	.	.	.	.	.
IOCDEN	.	.	.	.	.	.	.	.	.
IOCSBP/SIZ	.	.	.	.	.	.	.	.	.
IOCSBS	.	.	.	.	.	.	.	.	.
IOCSBE	.	.	.	.	.	.	.	.	.
IOCSBI	.	.	.	.	.	.	.	.	.

Non-diskette Device -- File Format, Output

```
IOCDTT = DT$CLS + DT$OPO
IOCGDW = CP
IOCLUN = 0
IOCDBS = Data buffer start (used for FDR processing)
IOCDBE = Data buffer end
IOCNAM = File name
IOCSUF = Suffix
IOCFDF = FD$FMA, FD$FMB, FD$FMC, or FD$FMD (only)
```

APPENDIX

L. EXORdisk II/III System Specifications

The following table lists the characteristics and specifications of the EXORdisk II/III system.

CHARACTERISTICS	SPECIFICATIONS
POWER REQUIREMENTS	
AC Power	
	110 Vac, 60 Hz, 3.4 Amps
	110 Vac, 50 Hz, 3.4 Amps
	220 Vac, 50 Hz, 1.8 Amps
DC Power supplied by EXORciser	+ 5 Vdc @ 2.75 Amps
	+12 Vdc @ 20 mAmps
	-12 Vdc @ 45 mAmps
BUS INTERFACE SIGNALS	
Address, Control busses	TTL compatible
Data bus	Bi-directional, three state TTL compatible
DISK-TO-CONTROLLER INTERFACE SIGNALS	Positive true TTL compatible
OPERATING TEMPERATURE	0-70 degrees Celsius
PHYSICAL CHARACTERISTICS	
Disk Drive Unit	
Width	17.75 inches
Depth	23.5 inches
Height	6.96 inches
Weight	48 pounds
Floppy Disk Controller	
Width	9.75 inches
Height	5.75 inches
Board thickness	0.06 inches
CONNECTOR TYPES	
Bus Connector (P1)	Stanford Applied Engineering SAC-43D/1-2 or equivalent
Disk Drive Unit Connector (P2)	AMP P/N 88393-7 or equivalent

SUGGESTION/PROBLEM REPORT

Motorola welcomes your comments on its products and publications. Please use this form.

To: Motorola Microsystems
P.O. Box 20912
Attention: Publications Manager
Mail Drop M374
Phoenix, Az. 85036

Comments
Product: ______________________ Manual: ______________________

Please Print

Name	Title	
Company	Division	
Street	Mail Drop	Phone Number
City	State	Zip

```
Hardware Support:  (800) 528-1908
Software Support:  (602) 962-3935
```

MOTOROLA ***Semiconductor Products Inc.***

P.O. BOX 20912 • PHOENIX, ARIZONA 85036 • A SUBSIDIARY OF MOTOROLA INC.

12405-2 PRINTED IN USA 1/26/81 IMPERIAL LITHO B92573 3000