

M68XDOS4 (D2)  
JUNE 1982

EXORset Disk Operating System User's Guide  
XDOS 4.0

The information in this document has been carefully checked and is believed to be entirely reliable. No responsibility, however, is assumed for inaccuracies. Furthermore, such information does not convey to the purchaser of the product described any license under the patent rights of Motorola, Inc. or others.

Motorola reserves the right to change specifications without notice.

EXORset, EXORbug, EXORciser, XDOS and MDOS are trademarks of Motorola, Inc.

Second Edition  
Copyright 1982 by Motorola, Inc.



## MANUAL ORGANIZATION

---

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

The User's Guide has been divided into two parts. PART I is intended for the new user of XDOS 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 simplified operation of XDOS. PART I contains descriptions and examples of the basic forms of the most frequently used XDOS 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.

PART II is intended as a detailed reference manual for those who need to know specific or extended information about the XDOS commands, the system structure, and the resident system functions.



## TABLE OF CONTENTS

	Page
MANUAL ORGANIZATION . . . . .	ii
1.1 General Description of Manual Organization . . . . .	ii
1.2 Table of Contents . . . . .	iii
1.3 Index . . . . .	iv
<b>PART I -- SIMPLIFIED XDOS USER'S GUIDE</b>	
<hr/>	
1. INTRODUCTION . . . . .	01-01
1.1 Hardware Support Required . . . . .	01-01
1.2 Additional Supported Hardware . . . . .	01-01
1.3 Software Support Required . . . . .	01-01
1.4 Program Compatibility . . . . .	01-02
1.5 Hardware Installation . . . . .	01-02
1.5.1 Operating Principles . . . . .	01-04
1.5.2 Logical Unit Numbers . . . . .	01-05
1.5.3 Configuration Requirements . . . . .	01-05
1.5.4 Configuration Parameters . . . . .	01-06
1.5.5 Operating on the Alternate Controller . . . . .	01-07
1.5.6 Disk Diagnostics . . . . .	01-07
1.6 Software Installation . . . . .	01-08
2. GENERAL SYSTEM OPERATION . . . . .	02-01
2.1 System Initialization . . . . .	02-01
2.2 Sign-on Message . . . . .	02-02
2.3 Initialization Error Messages . . . . .	02-02
2.4 Operator Command Format . . . . .	02-06
2.5 System Console . . . . .	02-07
2.5.1 Carriage return key . . . . .	02-07
2.5.2 Control-P . . . . .	02-07
2.5.3 Control-W . . . . .	02-08
2.5.4 Control-X . . . . .	02-08
2.5.5 DEL or RUBOUT . . . . .	02-08
2.5.6 Control-D . . . . .	02-08
2.6 Common Error Messages . . . . .	02-09
2.7 Diskette File Concepts . . . . .	02-11
2.7.1 File name specifications . . . . .	02-11
2.7.1.1 Family names . . . . .	02-12
2.7.1.2 Device specifications . . . . .	02-13
2.7.2 File creation . . . . .	02-13
2.7.3 File deletion . . . . .	02-13
2.7.4 File protection . . . . .	02-14
2.8 Typical Command Usage Examples . . . . .	02-15
2.8.1 DIR -- Directory display . . . . .	02-16
2.8.2 E.CM -- Program editing . . . . .	02-17
2.8.3 RASM09 -- Program assembling . . . . .	02-17
2.8.4 DEL -- File deletion . . . . .	02-18
2.8.5 LOAD -- Program loading/execution . . . . .	02-19
2.8.6 NAME -- File name changing . . . . .	02-20
2.8.7 NAME -- File protection changing . . . . .	02-20
2.8.8 COPY -- File copying . . . . .	02-21
2.8.9 BACKUP -- XDOS diskette creation . . . . .	02-22
2.9 Other Available Commands . . . . .	02-23
2.9.1 BACKUP -- Diskette copying . . . . .	02-23
2.9.2 MERGE -- File concatenation . . . . .	02-23
2.9.3 FREE -- Available file space display . . . . .	02-23
2.9.4 CHAIN -- XDOS command chaining . . . . .	02-23
2.9.5 DUMP -- Diskette sector display . . . . .	02-24
2.9.6 FORMAT -- Diskette reformatting . . . . .	02-24

## TABLE OF CONTENTS

## Page

2.9.7 DOSGEN -- XDOS diskette generation . . . . .	02-24
2.9.8 ROLLOUT -- Memory rollout to diskette . . . . .	02-24
2.10 XDOS-Supported Software Products . . . . .	02-24
2.11 Paper Alignment . . . . .	02-25

## PART II -- ADVANCED XDOS USER'S GUIDE

3. BACKUP COMMAND . . . . .	03-01
3.1 Use . . . . .	03-01
3.2 Diskette Copying . . . . .	03-02
3.3 File Reorganization . . . . .	03-03
3.4 File Appending . . . . .	03-07
3.5 Diskette Verification . . . . .	03-09
3.6 Other Options . . . . .	03-09
3.7 Messages . . . . .	03-11
3.8 Precautions with BACKUP . . . . .	03-15
3.8.1 BACKUP and the CHAIN process . . . . .	03-15
3.9 Examples . . . . .	03-16
4. CHAIN COMMAND . . . . .	04-01
4.1 Use . . . . .	04-01
4.2 Execution Operators . . . . .	04-02
4.2.1 Execution Comments . . . . .	04-03
4.2.2 Operator Breakpoints . . . . .	04-03
4.2.3 Error status word . . . . .	04-03
4.2.4 SET operator . . . . .	04-04
4.2.5 TST operator . . . . .	04-05
4.2.6 JMP operator . . . . .	04-06
4.2.7 LBL operator . . . . .	04-06
4.2.8 CMD operator . . . . .	04-06
4.3 Messages . . . . .	04-06
4.4 Resuming an Aborted CHAIN Process . . . . .	04-08
4.5 Examples . . . . .	04-09
5. COPY COMMAND . . . . .	05-01
5.1 Use . . . . .	05-01
5.1.1 Diskette-to-diskette copying . . . . .	05-02
5.1.2 Diskette-to-device copying . . . . .	05-02
5.1.3 Device-to-diskette copying . . . . .	05-04
5.1.4 Verification . . . . .	05-05
5.1.5 Automatic verification . . . . .	05-06
5.2 User-Defined Devices . . . . .	05-06
5.3 COPY Mode Summary . . . . .	05-07
5.4 Messages . . . . .	05-08
5.5 Examples . . . . .	05-09
5.5.1 Diskette-to-diskette example . . . . .	05-09
5.5.2 Diskette-to-device example . . . . .	05-10
5.5.3 Device-to-diskette example . . . . .	05-10
6. DEL COMMAND . . . . .	06-01
6.1 Use . . . . .	06-01
6.1.1 Single file name deletion . . . . .	06-01
6.1.2 Multiple file name deletion . . . . .	06-02

## TABLE OF CONTENTS

	Page
6.1.3 Family deletion . . . . .	06-02
6.2 Options . . . . .	06-03
6.3 Messages . . . . .	06-03
6.4 Examples . . . . .	06-04
<b>7. DIR COMMAND . . . . .</b>	<b>07-01</b>
7.1 Use . . . . .	07-01
7.1.1 Families . . . . .	07-02
7.1.2 System files . . . . .	07-02
7.1.3 Entire directory entry . . . . .	07-02
7.1.4 Segment descriptors . . . . .	07-04
7.1.5 Other options . . . . .	07-04
7.2 Messages . . . . .	07-04
7.3 Examples . . . . .	07-06
<b>8. DOSGEN COMMAND . . . . .</b>	<b>08-01</b>
8.1 Use . . . . .	08-01
8.2 Diskette Surface Test . . . . .	08-03
8.3 Minimum System Generation . . . . .	08-04
8.4 Messages . . . . .	08-04
8.5 Examples . . . . .	08-06
<b>9. DUMP COMMAND . . . . .</b>	<b>09-01</b>
9.1 Use . . . . .	09-01
9.1.1 Physical Mode of operation . . . . .	09-01
9.1.2 Logical Mode of operation . . . . .	09-02
9.1.3 Sector change buffer . . . . .	09-02
9.2 DUMP Command Set . . . . .	09-03
9.2.1 Quit -- Q . . . . .	09-03
9.2.2 Select logical unit -- U . . . . .	09-04
9.2.3 Open diskette file -- O . . . . .	09-04
9.2.4 Close diskette file -- C . . . . .	09-04
9.2.5 Show sector -- S . . . . .	09-05
9.2.6 Print sector -- L . . . . .	09-05
9.2.7 Read sector into change buffer -- R . . . . .	09-06
9.2.8 Write change buffer into sector -- W . . . . .	09-07
9.2.9 Fill change buffer -- F . . . . .	09-08
9.2.10 Examine/change sector buffer . . . . .	09-08
9.3 Messages . . . . .	09-09
9.4 Examples . . . . .	09-11
<b>10. FORMAT COMMAND . . . . .</b>	<b>10-01</b>
10.1 Use . . . . .	10-01
10.2 Messages . . . . .	10-02
10.3 Example . . . . .	10-02
<b>11. FREE COMMAND . . . . .</b>	<b>11-01</b>
11.1 Use . . . . .	11-01
11.2 Example . . . . .	11-01

## TABLE OF CONTENTS

## Page

12. LIST COMMAND . . . . .	12-01
12.1 Use . . . . .	12-01
12.1.1 Start/end specifications . . . . .	12-02
12.1.2 Physical line numbers . . . . .	12-02
12.1.3 User-supplied heading . . . . .	12-03
12.1.4 Non-standard page formats . . . . .	12-03
12.2 Messages . . . . .	12-03
12.3 Examples . . . . .	12-04
13. LOAD COMMAND . . . . .	13-01
13.1 Use . . . . .	13-01
13.1.1 Command-interpreter-loadable programs . .	13-02
13.1.2 Non-command-interpreter-loadable programs	13-04
13.1.3 Programs in the Alternate Memory Map . .	13-05
13.1.4 XDOS command line initialization . . . .	13-06
13.1.5 Entering the debug monitor . . . . .	13-07
13.2 Error Messages . . . . .	13-07
13.3 Examples . . . . .	13-08
14. MERGE COMMAND . . . . .	14-01
14.1 Use . . . . .	14-01
14.1.1 Merging non-memory-image files . . . . .	14-02
14.1.2 Merging memory-image files . . . . .	14-03
14.1.3 Other options . . . . .	14-04
14.2 Messages . . . . .	14-04
14.3 Examples . . . . .	14-05
15. NAME COMMAND . . . . .	15-01
15.1 Use . . . . .	15-01
15.1.1 Changing file names . . . . .	15-01
15.1.2 Changing file attributes . . . . .	15-02
15.2 Error Messages . . . . .	15-03
15.3 Examples . . . . .	15-03
16. ROLLOUT COMMAND . . . . .	16-01
16.1 Use . . . . .	16-01
16.1.1 Alternate Memory Map . . . . .	16-02
16.1.2 Non-overlaid memory . . . . .	16-03
16.1.3 Overlaid memory . . . . .	16-03
16.1.4 Scratch diskette conversion . . . . .	16-05
16.2 Messages . . . . .	16-05
16.3 Examples . . . . .	16-07

17.	SYSTEM DESCRIPTION . . . . .	17-01
17.1	Diskette Structure . . . . .	17-01
17.1.1	Diskette Identification Block . . . . .	17-02
17.1.2	Cluster Allocation Table . . . . .	17-02
17.1.3	Lockout Cluster Allocation Table . . . . .	17-03
17.1.4	Directory . . . . .	17-03
17.1.5	Bootblock . . . . .	17-05
17.2	File Structure . . . . .	17-06
17.2.1	Retrieval Information Block . . . . .	17-06
17.2.2	File formats . . . . .	17-08
17.3	Record Structure . . . . .	17-09
17.3.1	Binary records . . . . .	17-09
17.3.2	ASCII records . . . . .	17-10
17.3.3	ASCII-converted-binary records . . . . .	17-11
17.3.4	File descriptor records . . . . .	17-13
17.4	System Files . . . . .	17-14
17.4.1	System overlays . . . . .	17-14
17.4.2	System error message file . . . . .	17-15
17.5	Memory Map . . . . .	17-16
17.6	XDOS Command Interpreter . . . . .	17-19
17.7	Interrupt Handling . . . . .	17-20
17.8	System Function Calls . . . . .	17-21
17.9	XDOS Equate File . . . . .	17-23
18.	INPUT/OUTPUT FUNCTIONS FOR SUPPORTED DEVICES . . . . .	18-01
18.1	Supported Devices . . . . .	18-01
18.2	Device Dependent I/O Functions . . . . .	18-01
18.2.1	Console input -- .KEYIN . . . . .	18-02
18.2.2	Check for BREAK key -- .CKBRK . . . . .	18-04
18.2.3	Console output -- .DSPLY, .DSPLX, .DSPLZ . . . . .	18-04
18.2.3.1	Example of console I/O . . . . .	18-05
18.2.4	Printer output -- .PRINT, .PRINX . . . . .	18-06
18.2.4.1	Example of printer output . . . . .	18-07
18.2.5	Physical sector input -- .DREAD, .EREAD . . . . .	18-07
18.2.6	Physical sector output -- .DWRIT, .EWRIT . . . . .	18-10
18.2.7	Multiple sector input -- .MREAD, .MERED . . . . .	18-10
18.2.8	Multiple sector output -- .MWRIT, .MEWRT . . . . .	18-11
18.2.9	Diskette controller entry points . . . . .	18-12
18.3	Device Independent I/O Functions . . . . .	18-12
18.3.1	I/O Control Block -- IOCB . . . . .	18-13
18.3.1.1	IOCSTA -- Error status . . . . .	18-17
18.3.1.2	IOCDTT -- Data transfer type . . . . .	18-18
18.3.1.3	IOCDBP -- Data buffer pointer . . . . .	18-21
18.3.1.4	IOCDBS -- Data buffer start . . . . .	18-21
18.3.1.5	IOCDBE -- Data buffer end . . . . .	18-21
18.3.1.6	IOCGDW -- Generic device word . . . . .	18-22
18.3.1.7	IOCLUN -- Logical unit number . . . . .	18-22
18.3.1.8	IOCNAM -- File name . . . . .	18-22
18.3.1.9	IOCSUF -- Suffix . . . . .	18-22
18.3.1.10	IOCMLS -- Maximum LSN referenced . . . . .	18-23
18.3.1.11	IOCSDW -- Current SDW . . . . .	18-23
18.3.1.12	IOCSLS -- Starting LSN of SDW . . . . .	18-24
18.3.1.13	IOCLSN -- Next LSN . . . . .	18-24
18.3.1.14	IOCEOF -- LSN of end-of-file . . . . .	18-24
18.3.1.15	IOCRIB -- PSN of RIB . . . . .	18-24
18.3.1.16	IOCFDF -- File descriptor flags . . . . .	18-24
18.3.1.17	IOCDEN -- Directory entry number . . . . .	18-28
18.3.1.18	IOCSBP -- Sector buffer pointer . . . . .	18-28

## TABLE OF CONTENTS

## Page

18.3.1.19	IOCSBS -- Sector buffer start . . .	18-29
18.3.1.20	IOCSBE -- Sector buffer end . . .	18-29
18.3.1.21	IOCSBI -- Internal buffer pointer	18-29
18.3.2	Reserve a device -- .RESRV . . . . .	18-30
18.3.3	Open a file -- .OPEN . . . . .	18-31
18.3.4	Input a record -- .GETRC . . . . .	18-35
18.3.5	Output a record -- .PUTRC . . . . .	18-38
18.3.6	Close a file -- .CLOSE . . . . .	18-41
18.3.7	Release a device -- .RELES . . . . .	18-43
18.3.8	Example of device independent I/O . . .	18-44
18.3.9	Specialized diskette I/O functions . . .	18-47
18.3.9.1	Input logical sectors -- .GETLS .	18-47
18.3.9.2	Output logical sectors -- .PUTLS .	18-49
18.3.9.3	Rewind file -- .REWND . . . . .	18-51
18.3.9.4	Example of logical sector I/O . . .	18-53
18.3.10	Error handling . . . . .	18-56
19.	INPUT/OUTPUT PROVISIONS FOR NON-SUPPORTED DEVICES	19-01
19.1	Device Dependent I/O . . . . .	19-01
19.2	Device Independent I/O . . . . .	19-01
19.2.1	Controller Descriptor Block -- CDB . . .	19-01
19.2.1.1	CDBIOC -- Current IOCB address . .	19-04
19.2.1.2	CDBSDA -- Software driver address	19-04
19.2.1.3	CDBHAD -- Hardware address . . . .	19-04
19.2.1.4	CDBDDF -- Device descriptor flags	19-04
19.2.1.5	CDBVDT -- Valid data types . . . . .	19-06
19.2.1.6	CDBDDA -- Device dependent area .	19-07
19.2.1.7	CDBWST -- Working storage . . . . .	19-07
19.2.2	Device drivers . . . . .	19-08
19.2.3	Example of device driver . . . . .	19-09
19.2.4	Adding a non-standard device . . . . .	19-12

20.	OTHER SYSTEM FUNCTIONS . . . . .	20-01
20.1	Register Functions . . . . .	20-01
20.1.1	Transfer X to B,A -- .TXBA . . . . .	20-02
20.1.2	Transfer B,A to X -- .TBAX . . . . .	20-02
20.1.3	Exchange B,A with X -- .XBAX . . . . .	20-03
20.1.4	Add B to X -- .ADBX . . . . .	20-03
20.1.5	Add A to X -- .ADAX . . . . .	20-03
20.1.6	Add B,A to X -- .ADBA . . . . .	20-04
20.1.7	Add X to B,A -- .ADXBA . . . . .	20-04
20.1.8	Subtract B from X -- .SUBX . . . . .	20-04
20.1.9	Subtract A from X -- .SUAX . . . . .	20-05
20.1.10	Subtract B,A from X -- .SUBAX . . . . .	20-05
20.1.11	Subtract X from B,A -- .SUXBA . . . . .	20-05
20.1.12	Compare B,A with X -- .CPBAX . . . . .	20-05
20.1.13	Shift X right -- .ASRX . . . . .	20-05
20.1.14	Shift X left -- .ASLX . . . . .	20-06
20.1.15	Push X on stack -- .PSHX . . . . .	20-06
20.1.16	Pull X from stack -- .PULX . . . . .	20-06
20.2	Double-byte Arithmetic Functions . . . . .	20-07
20.2.1	Add A to memory -- .ADDAM . . . . .	20-07
20.2.2	Subtract A from memory -- .SUBAM . . . . .	20-07
20.2.3	Shift memory right -- .DMA . . . . .	20-08
20.2.4	Shift memory left -- .MMA . . . . .	20-08
20.3	Character String Functions . . . . .	20-08
20.3.1	String move -- .MOVE . . . . .	20-08
20.3.2	String comparison -- .CMPAR . . . . .	20-09
20.3.3	Character-fill a string -- .STCHR . . . . .	20-10
20.3.4	Blank-fill a string -- .STCHB . . . . .	20-10
20.3.5	Test for alphabetic character -- .ALPHA . . . . .	20-11
20.3.6	Test for decimal digit -- .NUMD . . . . .	20-11
20.4	Diskette File Functions . . . . .	20-11
20.4.1	Directory search -- .DIRSM . . . . .	20-12
20.4.2	Change file name/attributes -- .CHANG . . . . .	20-16
20.4.3	Load program into memory -- .LOAD . . . . .	20-18
20.4.4	Allocate diskette space -- .ALLOC . . . . .	20-23
20.4.5	Deallocate diskette space -- .DEALC . . . . .	20-25
20.4.6	Display system error message -- .MDERR . . . . .	20-27
20.5	Other Functions . . . . .	20-32
20.5.1	Process file name -- .PFNAM . . . . .	20-32
20.5.2	Re-enter resident XDOS -- .MDENT . . . . .	20-35
20.5.3	Reload XDOS from diskette -- .BOOT . . . . .	20-35
20.5.4	Set system error status word -- .EWORD . . . . .	20-36
20.5.5	Allocate user program memory -- .ALUSM . . . . .	20-36
20.5.6	Issue next command -- .COMND . . . . .	20-38
21.	ERROR MESSAGES . . . . .	21-01
21.1	Diskette Controller Errors . . . . .	21-01
21.1.1	Errors during initialization . . . . .	21-01
21.1.2	Errors after initialization . . . . .	21-05
21.2	Standard Command Errors . . . . .	21-06
21.3	Input/Ouput Function Errors . . . . .	21-16
21.4	System Error Status Word . . . . .	21-17
21.5	Commands Affecting Error Status Word . . . . .	21-18

## APPENDICES

---

A.	Track-Sector/Physical Sector Conversion Table . . .	A-01
----	---	------

## TABLE OF CONTENTS

## Page

B. ASCII Character Set . . . . .	B-01
C. XDOS Command Syntax Summary . . . . .	C-01
D. Diskette Controller Entry Points . . . . .	D-01
E. Mini-Diagnostic Facility . . . . .	E-01
F. Diskette Description, Handling, and Format . . . . .	F-01
G. Directory Hashing Function . . . . .	G-01
H. XDOS Equate File Listing . . . . .	H-01
I. XDOS 4.00 Differences . . . . .	I-01
J. IOCB Input Parameter Summary . . . . .	J-01

P A R T      I  
SIMPLIFIED XDOS USER'S GUIDE



## CHAPTER 1

### 1. INTRODUCTION

The EXORset is a single or double-sided, single-density mini-diskette drive system. Two diskette drives and their controller are included in the EXORset.

The EXORset Diskette Operating System (XDOS) is a subset of the EXORciser Disk Operating System (MDOS). Used in conjunction with the hardware environment of the EXORset (micromodule compatible), it provides a powerful and easy-to-use tool for software development. XDOS 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 XDOS 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 XDOS a good basis for a user's application system.

#### 1.1 Hardware Support Required

The minimum hardware configuration required to support XDOS consists of:

- a 68000-based computer system with a minimum of 16K RAM
- an EXORset with EXORbug firmware

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

#### 1.2 Additional Supported Hardware

XDOS also supports one or two external mass storage units (either 5.25" or 8" media, if an additional floppy disk controller card is installed (EXORdisk)). This allows storage expansion up to 2MB.

Output on a line printer is supported by XDOS. The line printer interfaces to the EXORset through the printer interface which consists of one PIA located on the EXORset main board.

### 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 XDOS commands and system files that are shipped on the system diskette must be used with that particular version of XDOS. XDOS commands and system files from other versions should never be intermixed.

Most user-written assembly language programs that were developed independently of XDOS can be executed on an XDOS 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 XDOS 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 software that he has developed using the EXORciser and MDOS, then Appendix J should be consulted for a list of differences between XDOS 4.00 and MDOS that may affect programs running with XDOS 4.00.

### 1.5 Hardware Installation

The EXORset should be inspected upon receipt for broken, damaged, or missing parts as well as for damage to the printed circuit boards. The packing materials should be saved in case reshipping is necessary.

8-inch disk drives connect to the EXORset via the MEX68SFDC (EXORDisk III) controller board, while 5-inch disk drives interface via the EXORset standard disk controller board. The MEX68SFDC module can control up to 4 disk drives, while the 5-inch controller can control a maximum of two disk units. Should the user need to use two controllers, they will have to be located in the two separate maps of the EXORset. This is the case when an EXORDisk III(E) mass storage peripheral is used in conjunction with 5-inch minidisk units. Two configurations are then possible which are described below.

**A. EXORdisk III Controller in Map 1.  
Minidisk controller in Map 2.**

**A1. EXORdisk III Controller**

- Install VXA jumper.
- Install disk driver firmware ROMC5009 in socket U19.

**A2. EXORset Minifloppy Controller**

- Configure jumpers SWD as follows:

SWD-1 OUT > Ready Signal From Drive  
 SWD-2 IN >

SWD-3 IN |  
 SWD-4 OUT > FDC and Disk Driver in Map 2  
 SWD-7 OUT > 16K Ram Block in Map 1

SWD-8 IN |  
 SWD-5 IN > R/W to FDC Chip

SWD-6 OUT > Read/Write Control for FDC

NOTE : IN = Jumper Installed

===== OUT = Jumper Left Open

- In socket U22, install PROM FDC 9.A for double-sided and/or single-sided minidisk drives, or PROM FDC 9.8 for single-sided minidisk drives only.

**B. Minidisk Controller in Map 1.  
EXORdisk III Controller in Map 2.**

**B1. EXORdisk III Controller**

- Install VMA jumper.
- Install disk driver firmware ROMC5009 in socket U19.

**B2. EXORset Minidisk Controller**

- Configure jumpers SWD as follows:

SWD-1	OUT	> Ready Signal From Drive
SWD-2	IN	>
SWD-3	OUT	
SWD-4	IN	> FDC, Disk Driver and
SWD-7	OUT	> 16K Ram Block in Map 1
SWD-8	IN	
SWD-5	IN	> R/W to FDC Chip
SWD-6	OUT	>
NOTE : IN = Jumper Installed		
===== OUT = Jumper Left Open		

- In socket U22, install PROM FDC 9.A for double-sided and/or single-sided minidisk drives, or PROM FDC 9.8 for single-sided minidisk drives only.

### 1.5.1 Operating Principles

---

As XDOS might be loaded from either controller, we will designate as the "current" map the one in which XDOS is booted, the other map being designated the "alternate" map. Conversely, the controller accessed in the current map will be designated the "current controller", and the controller in the alternate map, the "alternate controller". The terms "current disk" and "alternate disk" will also be used to refer to the disks that interface to the current and alternate controller, respectively.

Disk parameters switching technics are used when accessing an alternate disk, that is, for a given operation on an alternate disk, the following steps are taken:

- a/. the parameters pertaining to the current controller are saved,
- b/. the parameters associated with the alternate disk, that were saved on completion of the last operation performed on the alternate controller, are restored,
- c/. the desired operation is performed on the alternate controller,
- d/. the actual parameters of the alternate controller are saved,
- e/. the parameters of the current controller that were saved in step a/ are restored.

Obviously, the operations with an alternate disk will be slower than those performed on a current disk. Therefore the user will benefit from using the alternate disks as auxiliary devices rather than using them to hold scratch files. As the operations with alternate disks should be kept to a minimum, the user is recommended to install in the current map the controller that drives the disks having the largest speed and the largest storage capacity. This simply means that the EXORDisk III controller will best fit in map 1.

### 1.5.2 Logical Unit Numbers

As the EXORDisk III Controller can control up to 4 disk drives, and may be used as main or alternate controller, XDOS 4 supports up to 8 logical units which are defined as follows:

- .Logical unit numbers 0 thru 3 are assigned to the current controller,
- .Logical unit numbers 4 thru 7 are assigned to the alternate controller.

For instance, given an EXORset with an EXORDisk III as main controller, and a minidisk controller as alternate one, the following command will copy a file "MINI" from the minidisk responding to logical unit number 5 (drive 1 of the alternate controller) to a file "MAXI" in drive 0:

```
=COPY MINI:5,MAXI:0
```

If XDOS is booted in map2 following the EXORbug "TMAP" command that switches the maps, the minidisks will then respond to logical units 0 and 1, while the EXORDisk drives will assume the numbers 4 to 7. As a result, the same function as above will now be invoked in the following manner:

```
=COPY MINI:1,MAXI:4
```

Note that drive 0 is always the one from which XDOS is booted.

### 1.5.3 Configuration Requirements

Upon initiation, XDOS tests if an alternate controller can be accessed in the system. First, the diskette identification block is checked for valid configuration parameters. If the check does not fail, XDOS further checks if location E800-E801 in alternate map contain the value \$10CE which is the machine code for the LDS immediate instruction. If yes, XDOS will allow further accesses to alternate drives. If not, or if the first check failed, XDOS will limit the operations to the main controller only.

When using an alternate controller, the first 300 hexadecimal locations in the alternate map must consist of RAM. This is the case in the standard EXORset as the first 32K bytes of RAM are common to both maps.

#### **WARNING**

If XDOS is used in a system without an alternate controller, the user must insure that locations E800-E801 of the alternate map do no contain the value \$10CE (no EROM here starting with a LDS immediate instruction!). Failure to doing so will prevent XDOS from recognizing the absence of an alternate controller.

### 1.5.4 Configuration Parameters

---

In X DOS 3, disk configuration parameters were defined as constants in the equate file EQU.SA :

SC\$TRK	EQU	16	NUMBER OF SECTORS / TRACK (S.S DISK)
SC\$TKD	EQU	32	NUMBER OF SECTORS / CYLINDER (D.S DISK)
SC\$MAX	EQU	640	NUMBER OF USABLE SECTORS (S.S DISK)
SC\$MXD	EQU	1280	NUMBER OF USABLE SECTORS (D.S DISK)

Since X DOS 4 can support more than one type of drives, these definitions have been replaced by system configuration parameters which are kept in the system disk identification block (sector 0). When X DOS is booted, the configuration parameters are copied to RAM locations where they may be read later on by the user-program or by the X DOS commands and system calls. The symbolic definition of the configuration parameters can be found in the X DOS 4 equate file EQU.SA, along with their absolute addresses. Below are typical values for a system with an EXORDisk III controller as main controller, and a minidisk controller as alternate one.

	name	size	dec	hex	description
		byte	value	value	
SCTRKS\$	1	26	1A	1A	NUMBER OF SECTORS / TRACK (S.S. MAIN CONTROLLER)
SCMAX\$	2	2000	7D0	7D0	MAX NUMBER OF USABLE SECTORS (S.S. MAIN CONTROLLER)
SCTKD\$	1	52	34	34	NUMBER OF SECTORS / CYLINDER (D.S. MAIN CONTROLLER)
SCMXD\$	2	4004	FA4	FA4	MAX NUMBER OF USABLE SECTORS (D.S. MAIN CONTROLLER)
SATRK\$	1	16	10	10	NUMBER OF SECTORS / TRACK (S.S. ALTERNATE CONTROLLER)
SAMAX\$	2	640	280	280	MAX NUMBER OF USABLE SECTORS (S.S. ALTERNATE CONTROLLER)
SATKD\$	1	32	20	20	NUMBER OF SECTORS / TRACK (D.S. ALTERNATE CONTROLLER)
SAMXD\$	2	1280	500	500	MAX NUMBER OF USABLE SECTORS (D.S. ALTERNATE CONTROLLER)

In such a configuration, the configuration parameters of a minidisk will have these values reversed, i.e., SCTRKS\$=16, SATRK\$=26,...etc.

The configuration parameters are found, in that order, in the disk identification block, starting at offset DID\$CP (\$70). The first 6 bytes there contain the parameters related to the current controller while the last 6 bytes pertain to the alternate controller parameters.

The DOSGEN and BACKUP system commands take care of copying the appropriate configuration parameters to the identification block of the

destination diskette.

The configuration parameters of a system diskette are displayed in the XDOS sign-on message; they can also be visualized by invoking the FREE command with option "C".

### 1.5.5 Operating On The Alternate Controller

Since complete status of the alternate controller is memorized by XDOS, all operations where the alternate disks are involved must be issued through system calls SCALL's.

A .RESRV system call can be used to test if the access to the alternate controller is allowed (IOCGDW="DK", "IOCLUN='4").

If a physical sector I/O function is attempted on a non-accessible (not configured) alternate controller, a "NOT READY" indication will be returned (E3 or \*\*PROM I/O ERROR-STATUS=33 AT XXXX DRIVE Y-PSN ZZZZ).

If an unrecoverable disk error occurs while an operation is being performed with the alternate controller, or if during the same time the BREAK key (ABORT function) is hit, control will then be given to EXORbug with the alternate map being enabled (a colon is displayed as the EXORbug prompt). Consequently, the user will have to type in the EXORbug "TMAP" command prior to re-loading XDOS.

#### WARNING

The EXORbug stack is used when the alternate controller is accessed; consequently, the user calling program must not use it.

### 1.5.6 Disk Diagnostics

The firmware driver for the double-sided minidisk controller (FDC 9.A) does not make provision for the interactive diagnostics program which features automatic parameters initialization (see Appendix E). This program, which originates at \$EAD2, is only available with the single-sided minidisk controller (firmware FDC 9.8). Note also that the double-sided minidisk driver FDC 9.A can accomodate single-sided minidisk drives.

So, when using driver FDC 9.A, the user will have to configure the test disk parameters prior to initiating the disk diagnostics based at \$EB90.

The firmware driver for the EXORDisk III controller (ROMC5009) does not include diagnostics routine. Refer to the EXORDisk III User's Guide for fault isolation.

The table below summarizes the availability/unavailability of diagnostics routines depending on the controller firmware in use.

firmware	mini-diagnostics	Interactive test
ROMC5009	None	None
FDC 9.8	EB90;G	EAD2;G
FDC 9.A	EB90;G	None

Note that the SELFTEST program delivered with the EXORset provides another convenient means to exercise the disk drives, whatever their type.

#### 1.6 Software Installation

---

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

## CHAPTER 2 X DOS 4.0 User's Guide

### 2. GENERAL SYSTEM OPERATION

This chapter provides the user with the basic concepts that are necessary for the simplified and typical operation of XDOS. 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 XDOS 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 EXORset. Once the power is on, the following steps must be followed:

1. The prompt "EXORBUG V.R" will be displayed by EXORbug indicating it is waiting for operator input. "V" indicates the version and "R" the revision number of the EXORbug monitor in the system. If the power was already on and XDOS must be reloaded, press the restart button located on the main board through the back panel.

2. An XDOS diskette (one shipped from Motorola or one that has been properly prepared by the user (see section 2.8.9)) must be placed in drive zero. The door on the drive unit must then be closed in order for the diskette to begin rotating. When shipped from Motorola, the left side drive is configured as drive zero, the right side drive as drive one, as seen from the front.

The diskette must be oriented properly before being inserted into the drive. When the diskette is inserted properly, the label is facing right, 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. The EXORbug "XDOS" command must then be entered. It will give control to the diskette controller at address \$E800. 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, X DOS will display the message:

```
X DOS VV.RR

TOP MEM. ADDR. : $BFFF

DRV. 0-3
S.S. MODE : 26 SCT/CYL, 2000 USABLE SCT.
D.S. MODE : 32 SCT/CYL, 4004 USABLE SCT.

DRV. 4-7
S.S. MODE : 16 SCT/CYL, 640 USABLE SCT.
D.S. MODE : 32 SCT/CYL, 1280 USABLE SCT.
=
```

meaning that X DOS 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, the configuration parameters and the top address of the portion of contiguous RAM available in the current map are displayed. This last indication is derived from a memory sizing operation and provides a visual indication of possible memory system failures. The above is an example display when booting X DOS in a system configured with an EXORdisk III as main drive unit, and minidisks as alternate drive units. The current map includes 48K bytes of contiguous RAM starting at 0000, indicating that X DOS is ready to accept commands from the operator. The equal sign prompt is subsequently displayed each time the X DOS command interpreter gets control. The sign-on message showing the version and revision numbers is only displayed when X DOS 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 EXORbug 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. If the write protection tab is punched out, 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 fully inserted and 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. In either case, closing the door, waiting a little bit longer before entering the "XDOS" command, or reinserting 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. This error indicates some type of hardware problem.

**E5** This error status is returned when the track address has not been found after five attempts. This error indicates some type of hardware problem.

**E6** The diskette controller has been presented with a track-sector address that is invalid. This error indicates some type of hardware problem.

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 IROs 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 8, DOSGEN command, and chapter 10, 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 2.8.8).

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 track 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 track zero; however, this problem rarely occurs. The E7 errors can occur if a user-written program accesses drive 1 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 XDOS.SY is in error. The operating system cannot be loaded.  The diskette probably is not an XDOS system diskette, or the system files have been moved from their original places.
EM	This error indicates that there was insufficient memory to accommodate the resident portion of the operating system.

**MM** The memory requirements described in section 1.1.4 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 X DOS 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 X DOS 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 2.8.8). This error may also occur if the door to drive zero is opened while initialization is in progress.

**EN** A NMI has occurred and the X DOS NMI vector (NMI\$VC) was not initialized. This error may also occur after completion of the initialization.

**EQ** An IRQ has occurred and the X DOS IRQ vector (IRQ\$VC) was not initialized. This error may also occur after completion of the initialization.

- EF        A FIRQ has occurred and the XDOS FIRQ vector (FIR\$VC) was not initialized. This error may also occur after completion of the initialization.
- ES        A SWI2 has occurred and the XDOS SWI2 vector (SW2\$VC) was not initialized. This error may also occur after completion of the initialization.
- EW        A SWI3 has occurred and the XDOS SWI3 vector (SW3\$VC) was not initialized. This error may also occur after completion of the initialization.

#### 2.4 Operator Command Format

---

After the sign-on message is displayed, XDOS 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 XDOS 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 XDOS 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 XDOS commands. In addition, Appendix H contains a summary of the command line formats of all XDOS-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 XDOS 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. X DOS messages are displayed on the EXORset screen. X DOS 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 X DOS 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 Control-P

Control-P is actually a combination of two keys being depressed simultaneously: the CONTROL or CTL key and the P key. Depressing control-P is recognized as a special controlled abort function key. Most X DOS commands that take a long time to complete their function periodically check to see if the control-P has been depressed. If it has, the command will come to a premature, but controlled, termination point.

The control-P should be used, whenever possible, as an alternative to using the EXORset RESTART pushbutton or the keyboard BREAK key (The BREAK key is the equivalent of the EXORciser ABORT pushbutton). The controlled abort that is achieved with the control-P 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 BREAK key or RESTART pushbutton 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 Control-P abort function will also be "haltable" with the CTL-W key. Most XDOS 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 if the output rate is too high to read the messages. 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 inputs from the console support 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 XDOS 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 21. The messages are listed there in order of their two-digit reference numbers.

### WHAT?

**\*\* 01 NAME UNKNOWN** 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, or that contains a non-alphanumeric 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

The XDOS initialization process to give control to the user EXORbug; 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 and 32 is the same as E2.

The third digit of the status is substituted for the letter "h" in the error message. The fourth digit is substituted for the letter "i" and the fifth digit is substituted for the letter "j". Thus, 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 XDOS, 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 17. The following section describes how files are named, created, deleted, and protected.

### 2.7.1 File name specifications

An XDOS 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  
DATA115  
BACKUP  
S0  
X DOSNEWS  
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". XDOS 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:

```
XDOSNEWS.SA  
BACKUP.CM  
Z.SA  
PROCL.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 or 1). In most cases, XDOS 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.

```
BACKUP.CM:0  
TEST.X:1  
DIR:1  
Z456.D3:0  
ASM:1
```

#### 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 and \*PROGRAM.B and \*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

---

XDOS 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 XDOS 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 XDOS 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 XDOS 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 are also described. New XDOS 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 XDOS 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 and 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 E.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 7 contains a complete description of the DIR command's use.

### 2.8.2 E.CM -- Program editing

The E.CM command is used to create and/or to change a user-written source program and datafiles on a diskette. If the E.CM command resides on the system diskette, it is invoked with the following XDOS command line:

```
XDOS> E <filename1>[,<filename2>][;<options>]
```

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

```
XDOS> E:1 <filename1>[,<filename2>][;<options>]
```

```
XDOS> E:1 <filename1>[,<filename2>][;<options>]
```

A complete description of the E.CM command's format and usage is found in the relevant manual.

### 2.8.3 RASM09-- Program assembling

The RASM09 command (hereafter called the assembler) is used to assemble the source program files created with the E.CM command. The assembler translates these source programs into object programs. If the assembler resides on the system diskette, it is invoked with the following XDOS command line:

```
XDOS> RASM09 <name> [;<options>]
```

If the assembler is not copied to the system diskette in drive zero, it can be invoked from the diskette in drive one by using the following command line:

```
XDOS> RASM09:1 <name> [;<options>]
```

```
XDOS> 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 E.CM command. The assembler will automatically supply the default suffix for both the source file that is read (SA) and for the LX file that is created. If its name is not specified, the memory image 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.

```
XDOS> RASM09 <name> [;<options>]
```

Normally, a listing of the assembled program is desired. The assembler 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:

```
XDOS> RASM09 TESTPROG;L
```

```
XDOS> 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:

```
XDOS> RASM09 TESTPROG;L=#CN
```

```
XDOS> 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 assembler, however, requires 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

```
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 relevant manual should be consulted for a complete description of the assembler's 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., E.CM, RASM09) 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 6 contains a complete description of the DEL

command's other capabilities.

### 2.8.5 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 three processes of editing, assembling 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 originated at 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 XDOS system would be loaded and tested in this fashion. Programs that are developed with XDOS as a basis (i.e., programs that use the resident system functions) are loaded without the "V" option. Chapter 13 describes the details of the LOAD command and should be consulted if more information is required.

### 2.8.6 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,BLAJKACK
```

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.

### 2.8.7 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 15 describes in more detail the other features of the NAME command.

#### 2.8.8 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 a peripheral device and a diskette.

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

The command line COPY <name 1>,<name 2> copies all logical units zero and one of the specified files. The command line would be:

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.

```
COPY E.CM:1,:0
```

The command line COPY RASM09.CM:1,:0 will copy all logical units zero and one of the specified files. The command line 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 E.CM nor RASM09 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 XDOS, 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 wants to transfer external data to a disk file, he has to write his own driver, then invoke the COPY command as follows:

```
COPY #UD,<name 2>;D=<name 3>
```

where <name 2> is the name of the diskette file into which the data are to be written. The first parameter, #UD, specifies the user driver as source device, and the "D=<name 3>" option specifies the memory image file name in which the driver may be found (see 5.2 and 19).

The above process can be changed slightly so that a file on diskette can be written to a user defined peripheral. For example,

```
COPY <name 1>,#UD;D=<name 3>
```

will transfer the file named by <name 1> to the user device through the output driver found in <name 3>. Chapter 5 describes in more detail the other features of the COPY command.

#### 2.8.9 BACKUP -- XDOS diskette creation

---

New diskettes, or diskettes never before used on an XDOS system, must first be prepared for use with XDOS. The quickest way to generate a new XDOS diskette is to use the BACKUP command. Usually, a copy is retained of the original system diskette that was shipped with the EXORset. This diskette should be used to generate subsequent XDOS 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 formatted 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: `DISK 0,1,0;FORMAT 0,0,0;BACKUP 0,0,0;UV 0,0,0;FREE 0,0,0`.  
The first line sets up drive 0,0,0 for disk 0,0,0. The second line makes a copy of the entire diskette, naming it "BACKUP";UV generates a report, and FREE displays available space.

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 8 describes an alternative method of generating new system diskettes.

## 2.9 Other Available Commands

Several other powerful commands are included with each XDOS 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 XDOS 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 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. Chapter 14 contains the complete description of the MERGE command.

### 2.9.3 FREE -- Available file space display

The FREE command displays how many unallocated sectors and how many empty directory entries are on any diskette. Chapter 11 contains the complete description of the FREE command.

### 2.9.4 CHAIN -- XDOS command chaining

The CHAIN command allows predefined procedures to be automatically executed. A procedure consists of any sequence of XDOS 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 capability of conditional

directives to the CHAIN command at execution time permits an almost unlimited number of applications to be handled by a CHAIN file. Chapter 4 contains the complete description of the CHAIN command.

#### 2.9.5 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 9 contains the complete description of the DUMP command.

#### 2.9.6 FORMAT -- Diskette reformatting

---

The FORMAT command attempts to rewrite the sector addressing information on damaged diskettes. Upon receipt from supplier, diskettes may not be formatted. They then need be formatted before they can be used. The FORMAT command must be used to initialize them. Chapter 10 contains the complete description of the FORMAT command.

#### 2.9.7 DOSGEN -- XDOS diskette generation

---

The DOSGEN command allows specialized XDOS diskettes to be prepared. Diskettes that have bad sectors can have those sectors locked out so that the diskette can be used in an XDOS 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 8 contains the complete description of the DOSGEN command.

#### 2.9.8 ROLLOUT -- Memory rollout to diskette

---

The ROLLOUT command is used for writing the contents of memory to diskette. The ROLLOUT command does support the alternate map feature of the EXORset. Options exist for writing memory directly into a diskette file or for writing to a scratch diskette. Chapter 16 contains the complete description of the ROLLOUT command.

### 2.10 XDOS-Supported Software Products

---

Although the preceding list of commands provides the user with many powerful tools for software development, there are some other Motorola products which are capable of running in an XDOS environment, even though they were developed independently. These products are called XDOS-Supported software products. No attempt will be made in this User's

Guide to comprehensively describe any XDOS-Supported software product. Appendix H contains a list (complete at time of publication) of all products that can be invoked from an XDOS 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. XDOS-supported software products may be received on separate diskettes. Section 2.8.8 describes how such products can be copied onto the system diskette.

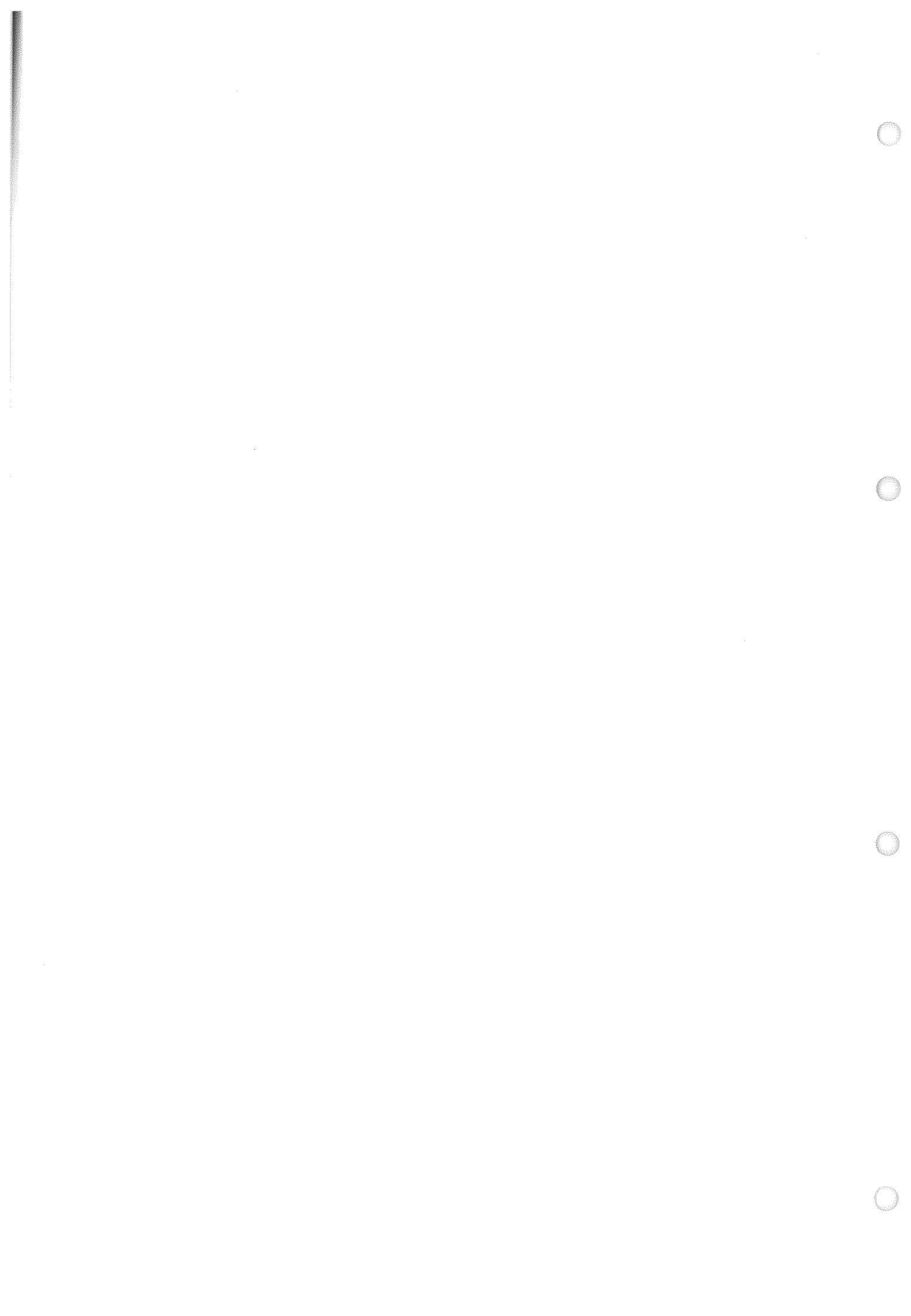
## 2.11 Paper Alignment

---

All XDOS 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 XDOS 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). XDOS commands use the standard format of 66 lines/page.

If an alternate paper size is desired the default line-per-page can be changed by patching the following bytes using the DUMP command.

Command	Logical Sector	Offset to first byte in Sector (Hex)	Current Value (Decimal)
DUMP.CM	0	4	66
DIR.CM	0	4	60
FREE.CM	0	4	66
LIST.CM	0	5	66
E.CM	0	6	66
BASICM.CM	0	9	66



P A R T      II  
ADVANCED XDOS USER'S GUIDE



## CHAPTER 3 XENIX COMMANDS

### 3. BACKUP COMMAND

The BACKUP command allows making copies of entire XDOS 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 XDOS-generated diskettes.

#### 3.1 Use

The BACKUP command is invoked with the following command line:

```
BACKUP [:<sn>,:<dn>] [<options>]
```

where <sn> is the logical drive number of the source drive and <dn> is the number of the destination drive. The default value for <sn> is 0. The default value for <dn> is 1. <options> can be one or more of the option letters described below.

If the command line is valid, the message:

```
BACKUP FROM DRIVE <sn> TO <dn>?
```

is displayed. If the response is "Y" or "y", the specified action is performed. If "N" or "n" is entered, the command line is repeated. If "APPEND FROM DRIVE <sn> TO <dn>?" is displayed, the user may type "Y" or "y" if continuing, or "N" or "n" if stopping. A question mark "?" is displayed after each option letter. The options will be displayed. A response of "Y" is required if BACKUP is to continue. Any other response will return control to XDOS. 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

Like the other system commands, BACKUP applies to different drive types; therefore, it is possible to copy, for instance, a 5-inch diskette to an 8-inch one. There are situations however, where the disk space available on the destination disk cannot accomodate all the data on the source diskette. This situation is checked when invoking the BACKUP command without a main option, or with main option "V". In all the following cases, attempting to copy/verify disks will cause the message

INSUFFICIENT DISK SPACE ON DESTINATION DRIVE

to be printed and BACKUP to abort.

Source disk	Destination disk
5-inch D.S.	5-inch S.S.
8-inch S.S	5-inch S.S.
8-inch S.S	5-inch D.S.
8-inch D.S	5-inch S.S.
8-inch D.S	5-inch D.S.
8-inch D.S	8-inch S.S.
<hr/>	
5-inch Single-Sided disks	= 80 Kbytes
5-inch Double-Sided disks	= 160 Kbytes
8-inch Single-Sided disks	= 256 Kbytes
8-inch Double-Sided disks	= 512 Kbytes

and it may copy over existing data without notice and can accidentally corrupt files and cause disk damage.

As entire diskette contents cannot be copied in the instances above, the user has then to specify the main options "A" or "R" for file appending and reorganization, respectively. 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 17). Thus, only XDOS-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:

```
XDOS> BACKUP FROM DRIVE 0 TO 1? Y
```

Information will be displayed concerning the current state of the diskettes and their available space. Assuming the user has selected the default options and no options are set, and would produce a copy on the destination diskette of the source diskette's allocated space.

**3.3 File Reorganization** (This section deals with how to keep your diskettes running smoothly and efficiently)

After an XDOS 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 XDOS diskettes, thereby improving the system's efficiency.

#### File reorganization and its benefits

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 ;R
```

would invoke the BACKUP command to reorganize the files on

the source diskette in drive zero during the copy to the destination diskette in drive one. The source diskette must be an XDOS diskette. It is unaffected by the reorganization. The message

BACKUP FROM DRIVE 0 TO 1?

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** (CR) Q Terminate the BACKUP command and return to XDOS. No copying will take place; however, if the destination diskette has been affected due to the reorganization option as explained above, a file selection will be required before proceeding.

**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:

S {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 XDOS. 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	COPY	.CM
02	DEL	.CM
03	DIR	.CM
04	DOSGEN	.CM
05	FREE	.CM
1D	LOAD	.CM
1E	FORLB	.RO
1F	EQU	.SA
20	IOCB	.SA

The range of numbers \$06-1C, 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,05

would cause the file names with numbers

After the sorted directory 01, 02, 03, 05, and 1E is displayed, the user may enter a number from 01 to 1E to indicate which files are 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 XDOS 01.SY is displayed and the file system directory structure continues to be copied when this option 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 XDOS 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 XDOS initialization process. After the XDOS system files have been copied, the message:

STARTING TO COPY FILES

is displayed, followed by messages of the form:

COPYING <name i>.SY

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 XDOS 01.SY
COPYING XDOSOV01.SY
COPYING XDOSOV1 .SY
COPYING XDOSOV2 .SY
COPYING XDOSOV3 .SY
COPYING XDOSOV4 .SY
COPYING XDOSOV5 .SY
COPYING XDOSOV6 .SY
COPYING XDOSER 00.SA
STARTING TO COPY FILES
COPYING BACKUP00.CM
COPYING DIR .CM
COPYING DOSGEN .CM
COPYING FREE00.CM
COPYING LOAD .CM
COPYING EQU00 .SA
COPYING IOCB 00.SA
```

After all eligible files from the sorted directory have been copied, BACKUP will return control to XDOS. 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 XDOS diskette. The file append process is invoked by using the Main Option "A" on the BACKUP command line:

```
BACKUP ;A
```

Instead of the "BACKUP FROM DRIVE 0 TO 1?" message normally displayed by BACKUP, the message:

```
APPEND FROM DRIVE 0 TO 1?
```

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 XDOS 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**

The message above 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 5). After the last file has been copied to the destination diskette, control will be returned to XDOS.

Contrary to XDOS 3, the XDOS 4 BACKUP command allows to append files to the diskette in drive 0. The following command would append selective files of drive 4 to drive 0.

```
=BACKUP :4,:0;A
APPEND FROM DRIVE 4 TO DRIVE 0?
Y
ENTER FILE COPY SELECTION COMMANDS:
SAVE (S), DELETE (D), PRINT (P), QUIT (Q), NO MORE (CR)
S, D, P, Q, (CR):
```

### 3.5 Configuration parameters

When copying an entire diskette (no main options) or

reorganizing it (main option "R"), BACKUP takes care of writing the appropriate configuration parameters to the destination diskette disk identification block. If the source and destination drives are in the same map, the parameters are merely duplicated to the destination disk. If they are not, the parameters associated with the current controller and those associated with the alternate controller are first read from the source disk, and then swapped prior to being written to the destination disk identification block.

### 3.6 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 ;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 DR0-jj DR1-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 XDOS.

### 3.7 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 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 and 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.

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 0 TO 1?

This indicates BACKUP will copy to the destination diskette in drive zero from the source diskette in drive one if a "Y" response is given. Any other response will cause control to be returned to XDOS.

APPEND FROM DRIVE 0 TO 1?

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 XDOS.

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 XDOS 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 XDOS 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 XDOS.

**SECTOR nnnn**

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

**OFFSET ii DR0-jj DR1-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.

### 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 XDOS.

#### 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 4). 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.

### 3.9 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):080679  
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 11 and 7, respectively). The date of the disk copy that is generated is August 6, 1979, 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 ;R
BACKUP FROM DRIVE 0 TO 1?
Y
ENTER FILE COPY SELECTION COMMANDS:
SAVE (S), DELETE (D), PRINT (P), QUIT (Q), NO MORE (CR)
S, D, P, Q, (CR):
COPYING XDOS .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 XDOS 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 FORTTEST1.SA
0D FORTTEST2.SA
0E FORTTEST3.SA
0F FORTTEST4.SA
10 FORTTEST5.SA
DELETE FILE NOS.:
B-E,10
DELETE FILE NOS.:

STARTING TO COPY FILES
COPYING FORT .CM
COPYING FORTLIB .RO
COPYING FORTTEST4.SA
FORTTEST4.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).

4. CHAIN COMMAND **TO** THE CHAIN COMMAND **CHAPTER 4**

The CHAIN command is used to automatically execute a sequence of command lines contained in a diskette file. This feature allows complicated operations to be defined once, and then invoked any number of times, requiring no operator intervention. The additional capabilities of conditional directives at the CHAIN command execution time permits an almost unlimited number of applications to be handled by a CHAIN file.

The CHAIN command allows predefined procedures to be automatically executed. A procedure consists of any sequence of XDOS 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 at the CHAIN command execution time permits an almost unlimited number of applications to be handled by a CHAIN file.

**4.1 Use of the CHAIN command** **TO** THE CHAIN COMMAND **CHAPTER 4**

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

**TO** THE CHAIN COMMAND **CHAPTER 4**

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.

**TO** THE CHAIN COMMAND **CHAPTER 4**

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 4.6.

**TO** THE CHAIN COMMAND **CHAPTER 4**

To restart an aborted CHAIN process, type either of the following command lines:

**TO** THE CHAIN COMMAND **CHAPTER 4**

CHAIN N\* or CHAIN \*  
The CHAIN executes a copy phase and an execution phase. In the copy phase, <name 1> is read from beginning to end and copied into an intermediate file named CHAIN.SY:0. The source records will then be read from this file during the execution phase of the CHAIN process. This file will be automatically deleted upon the subsequent successful completion of the CHAIN process.

**TO** THE CHAIN COMMAND **CHAPTER 4**

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.

**TO** THE CHAIN COMMAND **CHAPTER 4**

The CHAIN command only intercepts console input via the XDOS system function ".KEYIN" (see section 18.2). 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 18.3) 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 XDOS command interpreter, the CHAIN process will be properly terminated, XDOS 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 XDOS 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 drive one.

#### 4.2 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 XDOS, 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 XDOS 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

#### 4.2.1 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 displayed when they are encountered during the execution of the CHAIN process. 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 drive one, etc.

#### 4.4.2 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  
...  
ASM 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.

#### 4.2.3 Error status word

Among the operating system's resident variables is a two-byte error status word. Each XDOS 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:

The error status word is composed of two bytes, each consisting of eight bits. The first byte is the high-order byte and the second byte is the low-order byte. The bits are numbered from 7 down to 0. Bit 7 is the most significant bit and bit 0 is the least significant bit. The error status word is used to indicate the status of various operations. For example, if a command fails, the error status word will be set to indicate the type of error. The error status word is also used to indicate the status of the system. For example, if the system is in a power failure, the error status word will be set to indicate the power failure status.

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 XDOS 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. A mask feature is provided to isolate the Error Status Word bits which are to be preserved.

#### 4.2.4 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 XDOS commands will only set the Error Status and the Error Type. The SET operator has the following format:

```
@SET <mask> <value>
```

where <mask> is a hexadecimal number which is used to mask the bits to be preserved, and <value> is a hexadecimal number representing the bits which must be set in the error status word. The error status word is logically "anded" with <mask>, then logically "ored" with <value>. These two numbers may not exceed \$FFFF. 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 F0FF 800
```

This form will set bit B of the error status word; however, the other parts of the error status word are not changed. The following example sets the Error Mask Flag and clear the remainder of the error status word.

```
    @SET 0 800
```

#### 4.2.5 TST operator

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

```
    @TST <mask> <value> <condition>
```

where **<mask>** is a hexadecimal number used to mask the bits to be tested, **<condition>** is a hexadecimal value representing 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.

The following test conditions can be used in the **<condition>** field of the TST operator:

**<condition>** Test performed on word part

22	Higher than <value> (unsigned)
23	Less than or equal to <value> (unsigned)
24	Higher than or equal to <value> (unsigned)
25	Less than <value> (unsigned)
26	Not equal to <value>
27	Equal to <value>
2C	Greater than or equal to <value> (signed)
2D	Less than <value> (signed)
2E	Greater than <value> (signed)
2F	Less than or equal to <value> (signed)

The **<condition>** part of the TST operator must be one of the hexadecimal values listed.

The **<value>** and **<mask>** parts of the TST operator are hexadecimal numbers in the range 0-FFFF.

All tests are performed on 16-bit values. The error status word content is first logically "anded" with the 16-bit **<mask>** supplied, then compared to the given 16-bit **<value>**. Finally, the appropriate test is performed.

#### 4.2.6 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> is a hexadecimal value associated to a forward LBL operator. 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.

#### 4.2.7 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> is a hexadecimal value used to identify the LBL operator among the others. It must be unique within the LBL operators, this means that no other LBL operator in the intermediate file may define the same number. The <label> part of the LBL operator must be comprised between 0 and FFFF included.

#### 4.2.8 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 <value>
```

where value is an even hexadecimal number to enable the print, or an odd hexadecimal number to disable it. Initially during the execution phase, the print is enabled.

### 4.3 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 copy phase, and those that can be displayed during the execution phase.

The following error messages can be displayed during the copy 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.

#### \*\* 48 CHAIN OVERLAY DOES NOT EXIST

The XDOS system CHAIN overlay does not have an entry in the directory. The diskette in drive zero is unable to execute a CHAIN process. The BACKUP command (Chapter 3) must be used to copy the system overlays to that diskette.

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 CONTROL-P KEY

The operator depressed the CTL-P 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.

#### 4.4 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 XDOS 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.

#### 4.5 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 XDOS system diskette is on drive zero and that the source programs will be on drive one (although not all at the same time).

```
@CMD 1 position. Depress any key when ready to start with
@SET,M 8
@. INSERT DISK 1 INTO DRIVE 1 -- DEPRESS ANY KEY WHEN READY
@* ASSEMBLING MODULE ONE
DEL PROG1.LO:1
ASM NOL,EQU,LIS,PROG1:1;O=PROG1:1
@TST FF 27 0
@JMP 1
@* ASSEMBLING MODULE ONE
MERGE FINAL.LO,PROG1.LO:1,TEMP.LO
DEL FINAL.LO
NAME TEMP.LO,FINAL
@JMP 2
@LBL 1
@* ASSEMBLING MODULE ONE
MERGE FINAL.LO,PROG1.LO:1,TEMP.LO
@* SUCCESSFUL MODULE ONE ASSEMBLY. PROGRAM UPDATE WILL BE PERFORMED
@LBL 2
@*
@* ASSEMBLING MODULE TWO
DEL PROG2.LO:1
ASM NOL,EQU,LIS,PROG2:1;O=PROG2:1
@TST FF 27 0
@JMP 3
@* SUCCESSFUL MODULE TWO ASSEMBLY. PROGRAM UPDATE WILL BE PERFORMED
MERGE FINAL.LO,PROG2.LO:1,TEMP.LO
DEL FINAL.LO
NAME TEMP.LO,FINAL
@JMP 4
@LBL 3
@* ERROR(S) IN MODULE TWO ASSEMBLY. PROGRAM UPDATE WONT BE PERFORMED
@LBL 4
@. INSERT DISK 2 INTO DRIVE 1 -- DEPRESS ANY KEY WHEN READY
@*
@* ASSEMBLING MODULE THREE
DEL PROG3.LO:1
ASM PROG3:1;O=PROG3:1
@TST FF 26 0
@JMP 5
@* ERROR(S) IN MODULE THREE ASSEMBLY. PROGRAM UPDATE WONT BE PERFORMED
@JMP 6
@LBL 5
@* SUCCESSFUL MODULE THREE ASSEMBLY. PROGRAM UPDATE WILL BE PERFORMED
MERGE FINAL.LO,PROG3.LO:1,TEMP.LO
DEL FINAL.LO
```

```
NAME TEMP.LO,FINAL
@LBL 6
@*
@*      ASSEMBLING MODULE FOUR
DEL PROG4.LO:1
ASM PROG4:1;O=PROG4:1
@TST  FF 26 0
@JMP 7
@* ERROR(S) IN MODULE FOUR ASSEMBLY. PROGRAM UPDATE WONT BE PERFORMED
@JMP 8
@LBL 7
@* SUCCESSFUL MODULE FOUR ASSEMBLY. PROGRAM UPDATE WILL BE PERFORMED
MERGE FINAL.LO,PROG4.LO:1,TEMP.LO
DEL FINAL.LO
NAME TEMP.LO,FINAL
@LBL 8
```

The listing of the commands is inhibited during the CHAIN process (@CMD 1). A comment tells the user when the assembly of a module begins. After each module assembly, a test is performed on the error status word: if there have been any errors in the module, it is told to the user, if the assembly was successful, the object file produced is used to update a memory image file. Each time a diskette has to be mounted in drive one, an execution breakpoint is used to request it and pause the system.

It should be noted that the JMP operator may skip over LBL operators, providing that the searched label is not defined by these LBL operators. The first time that this 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.

## CHAPTER 5

### 5. 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.

#### 5.1 Use of the COPY command

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 Both copy and verify

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 X DOS-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.

### 5.1.1 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 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.

### 5.1.2 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      Device being copied from or to
Name       Associated Physical Device
-----+-----+-----+-----+-----+-----+-----+-----+
#CN      Console printer
#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 17.3). Thus, not every diskette file can be copied to a non-diskette device. Memory-image files may not be copied to a non-diskette device.

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 17.3). 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 17.3) 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 5.2).

### 5.1.3 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 15) 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 specification that can be used for <name 1> is:

Device	
Name	Associated Physical Device
#UD	User-defined device

Only ASCII record files can be copied using the "N" option. If media have been generated in a non-XDOS environment, they must conform to the XDOS format for ASCII record files (section 17.3). 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 media should contain the ASCII end-of-file record (section 17.3) or generate a timeout condition.

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.

#### 5.1.4 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 mmmmmm  
OFFSET kkk SRC-yy DST-zz

where "mmmmmm" 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.

#### 5.1.5 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 5.1.1 describes the copy process between two diskette files. Section 5.1.4 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.

#### 5.2 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 19.2 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 19).
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 EXORbug monitor is entered and the user is prompted for a monitor command. That indicates 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 EXORbug command

*monitor 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.

### 5.3 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
-----	Required
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.

#### 5.4 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 12) 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".

## 5.5 Examples

The following examples show how files are copied between drives and units.

### 5.5.1 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
=

```

### 5.5.2 Diskette-to-device example

---

The next example illustrates how source listings that have been directed to diskette by the assembler (ASM) 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.

### 5.5.3 Device-to-diskette example

---

The following example

```
COPY #UD;D=LINK
```

will read a FDR using the user driver located in file LINK.LO on drive 0. Then it will create a file according to the specifications found in this FDR, and write to this file the data following the FDR.

Note that the second file name has not been specified. In the following example:

```
COPY #UD,TESTPROG.LX;D=LINK
```

the user media is scanned until a FDR describing a file name TESTPROG.LX is found. Then the diskette file TESTPROG.LX will be created, containing the data following the FDR.

If no FDR is specified, the required command line must look like

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

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 19.2. The example shows the EXORbug command issued:

```
=COPY #UD,TEST;NMD=DRIVER  
.3056;V  
.3064;V  
.3082;V  
.;P
```

The EXORbug 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.



## CHAPTER 6

---

### 6. DEL COMMAND

---

The DEL command is used to remove X DOS 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.

#### 6.1 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.

It is recommended that files be configured with delete protection or that adequate backup copies be kept since it is not possible to recover an accidentally DELETED file.

#### 6.1.1 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.

#### 6.1.2 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.

#### 6.1.3 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.

## 6.2 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.

## 6.3 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 the current 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.

#### 6.4 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 one, and PROG.LO on drive zero, the following command line would be used. The system's responses are also shown:

```
=DEL SCRATCH.SA:1,TEST.LX:1,PROG.LO
SCRATCH .SA:1 DELETED
TEST     .LX:1 DELETED
PROG     .LO: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 one without having to respond "Y" to each prompt, the following command line could be used:

```
DEL :1;YS or DEL *.*:1;YS
```

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

## CHAPTER 7

---

### 7. DIR COMMAND

---

The DIR command displays XDOS 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.

#### 7.1 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 or one, and "xxxxxxxx" will be the eight-character ID that was assigned to the diskette via the DOSGEN command (Chapter 8) 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.

### 7.1.1 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 XDOS 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.

### 7.1.2 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 6). 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.

### 7.1.3 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:

FFFFFF.FS WDSCN# RRRR ZZZZ DD

where the symbols take on the following meanings:

Symbol	Meaning
FFFFFF	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 #	Meaning
: : : : :	File format (0=user defined, 1=memory-image, 2=binary record, 3=ASCII record, 5=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 17. 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 XDOS, this field will always be a multiple of the basic unit of allocation (see Chapter 17). 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)		
:							
:							

: ..... Position within sector  
(0-7)  
:  
: ..... Physical sector number  
(\$3-\$16)

#### 7.1.4 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:

```
EXFILE .SA .DS..3 00D0 0088 75 00 00D0 080
                                01 0140 008
```

The file EXFILE.SA consists of two segments. The first segment starts in physical sector \$D0 and is \$80 sectors long. The second segment starts in physical sector \$140 and is 8 sectors long. The file's physical size is \$88 sectors.

#### 7.1.5 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 XDOS 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.

### 7.2 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. "ddd" 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 system tables are probably corrupted: The best way to recover the good files is a "BACKUP ;R" command (chapter 3). This will re-build the system tables on a scratch diskette.

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. Again, the "BACKUP ;R" command (chapter 3) must be issued to recover the good files and the system tables.

### 7.3 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. : XDOS0300
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 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. : XDOS0300
XDOsov6 .SY
LIST .CM
XDOS .SY
MERGE .CM
DIR .CM
XDOser .SY
XDOsov1 .SY
XDOsov3 .SY
ROLLOUT .CM
FREE .CM
EQU .SA
XDOsov5 .SY
DUMP .CM
NAME .CM
XDOsov2 .SY
EDIT .CM
LOAD .CM
DEL .CM
XDOsov0 .SY
CHAIN .CM
BACKUP .CM
XDOsov4 .SY
DOSGEN .CM
FORMAT .CM
COPY .CM
TOTAL NUMBER OF ENTRIES SHOWN : 025/$19
=
```

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 XDOS-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 contiguously 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 8

---

### 8. DOSGEN COMMAND

---

The DOSGEN command allows specialized XDOS diskettes to be prepared. Diskettes that have bad sectors can have those sectors locked out so that the diskette can be used in an XDOS environment. DOSGEN will also create all system tables and files on the generated diskette.

#### 8.1 Use

---

New diskettes never before used on an XDOS system, must first be prepared for use with XDOS. One way to generate a new XDOS 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 XDOS diskette is by invoking the DOSGEN command from an already up-and-running XDOS system.

DOSGEN does not create the sector addressing information: The user is then responsible to create it via the FORMAT command (Chapter 10). Then, the DOSGEN command may be used to write other information on the new diskette in order to make it recognizable by XDOS. DOSGEN creates the system tables required by XDOS (see Chapter 17). 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 XDOS version number. The DOSGEN command also copies across the required XDOS family of system files which must be present on any diskette used in the XDOS environment. These files and tables must not be moved or changed in any way other than through the DOSGEN command or the BACKUP (Chapter 3) command. Optionally, the XDOS commands may be copied to the diskette.

The DOSGEN command is invoked with the following command line:

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

where <dn> is the logical drive number of the destination drive. The default value for <dn> is 1.  
<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).

The diskette to be DOSGENed may reside in any drive.

DOSGEN will respond with the following question asking if drive one contains a diskette that can be written to:

DOSGEN DRIVE <dn>?

where <dn> is the logical drive number of the destination drive. The response should be the letter "Y", if the diskette in the specified drive is to be DOSGENed. Any other response will terminate the DOSGEN command and return control to XDOS. In this case, the diskette in drive <dn> 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 XDOS 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 XDOS 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 XDOS information must be in files in order to be accessed by XDOS. 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 cause 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 \$280. 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 XDOS system will have been generated on the destination diskette.

## 8.2 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 10). If a sector can be marked as bad, then the cluster to which the bad sector belongs will be automatically locked out from XDOS 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 XDOS 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 XDOS system files normally are copied (in the first several tracks) the DOSGEN process will fail. Such diskettes cannot be used as XDOS system diskettes unless the FORMAT command (Chapter 10) can be used to correctly rewrite the bad sectors.

### 8.3 Minimum System Generation

---

If the DOSGEN command is invoked with the "U" option, the resultant diskette will not contain any of the XDOS commands from drive zero. Only the XDOS family of system files that must reside on every diskette used in an XDOS 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 drive one.

### 8.4 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 1?

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 XDOS.

#### 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 XDOS 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 XDOS. 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 8.1. 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.

## XDOS.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 XDOS.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 10 should be consulted for making such a diskette usable again.

## 8.5 Examples

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

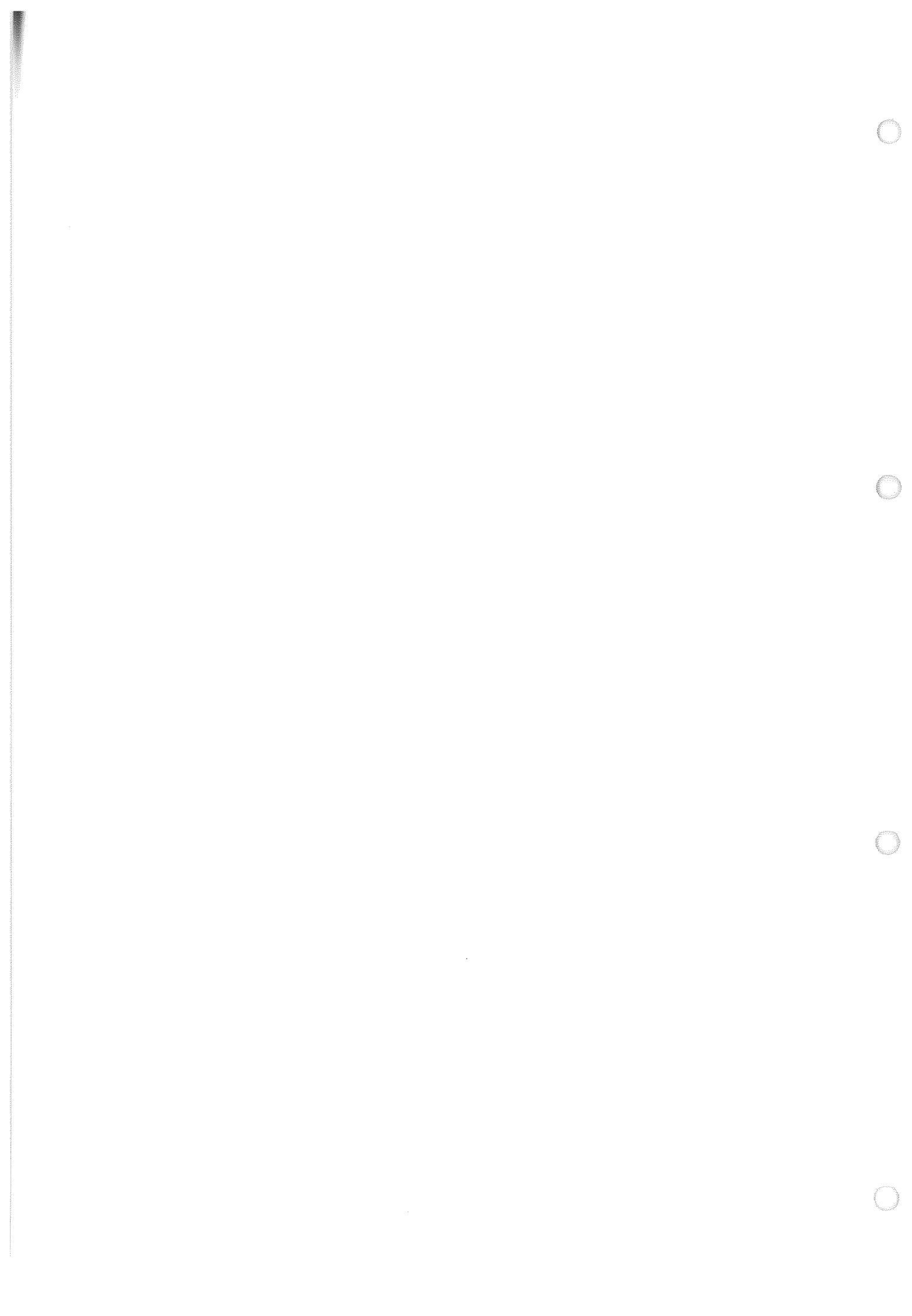
```
=DOSGEN ;TU
DOSGEN DRIVE 1? Y
DISK NAME: USER0001
DATE (MMDDYY): 072578
USER NAME: SYSTEM DEVELOPMENT 1
LOCKOUT ADDITIONAL SECTORS? N
COPYING FILE XDOS .SY
COPYING FILE XDOsov0 .SY
COPYING FILE XDOsov1 .SY
COPYING FILE XDOsov2 .SY
COPYING FILE XDOsov3 .SY
COPYING FILE XDOsov4 .SY
COPYING FILE XDOsov5 .SY
COPYING FILE XDOsov6 .SY
COPYING FILE XDOser .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 XDOS 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 ;T
DOSGEN DRIVE 1? Y
DISK NAME: USER0002
DATE (MMDDYY): 072578
USER NAME: TEST SYSTEM
SECTOR 0030 LOCKED OUT
SECTOR 0031 LOCKED OUT
SECTOR 0056 LOCKED OUT
LOCKOUT ADDITIONAL SECTORS? N
COPYING FILE XDOS .SY
XDOS.SY DOES NOT START AT SECTOR $18
=
```

Three bad sectors were found during the write/read test. When the XDOS family of files was copied, it was detected that the locked out sectors prevented the resident operating system file XDOS.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 10 should be consulted for making such a diskette usable again.



## CHAPTER 9

---

### 9. 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.

#### 9.1 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.

##### 9.1.1 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 (0-1). 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 9.2.2) must be used to restore the diskette drive after the diskette has been inserted.

#### 9.1.2 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.

#### 9.1.3 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 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.

## 9.2 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 XDOS 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.

### 9.2.1 Quit -- Q

The Q command is used to terminate DUMP and return control to XDOS. 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.

### 9.2.2 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 0-1. 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 track zero.

### 9.2.3 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.

### 9.2.4 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.

### 9.2.5 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 9.4.

### 9.2.6 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 9.4.

#### 9.2.7 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.

**9.2.8 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.

#### 9.2.9 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.

#### 9.2.10 Examine/change sector buffer

A special command is used for examining/changed 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/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.

### 9.3 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-\$27F 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 9.4. 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.

#### 9.4 Examples

The following example shows how the Cluster Allocation Table is displayed with the DUMP command.

```
=DUMP
PHYSICAL MODE
: SB
UNIT=0 CLUSTER ALLOCATION MAP
PSN=0001
00 FF FC 00 00 00 .....
10 00 00 00 00 FF .....
20 FF .....
30 FF .....
40 FF .....
50 FF .....
60 FF .....
70 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 XDOsov6 SY.t.....
00 58 44 4F 53 5F 56 36 20 53 59 00 74 F2 00 00 00 00 XDOsov6 SY.t.....
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 .....
PSN=0004 LSN=0001 XDOS SY.....
00 58 44 4F 53 20 20 20 20 53 59 00 18 F2 00 00 00 00 XDOS SY.....
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 .....
```

```

PSN=0005          LSN=0002
00 44 49 52 20 20 20 20 20 43 4D 00 8C F2 00 00 00 DIR CM.....
10 4D 45 52 47 45 20 20 20 43 4D 01 28 F2 00 00 00 MERGE CM.(.....
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=0006          LSN=0003
00 58 44 4F 53 45 52 20 20 53 59 00 7C E5 00 00 00 XDOSER SY.....
10 58 44 4F 53 4F 56 31 20 53 59 00 48 F2 00 00 00 XDOSOV1 SY.H.....
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 XDOSER
** 04 FILE NAME NOT FOUND
: O XDOSER.SY
: S 1,2
UNIT=0 FILE=XDOSER .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 relocate a memory-image position independent file. The load and execution start addresses are located in the RIB of the file. Using the change buffer, the user can read the RIB sector, modify it and write it back at the same place. The load start address is originally \$2800 and the execution start address, \$2804. They will be changed respectively to \$6000 and \$6004.

```
=DUMP RUNTIME.LO
```

```
: R FFFF
: S
CHANGE BUFFER
```

	PSN=0128	LSN=FFFF	
00	6C 4A 80 6C 00 00 00 00	00 00 00 00 00 00 00 00	1J.1.....
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 18 00 6D	28 00 28 04 00 00 00 00	.....m(.(..

```
: 78/
78 28 60
79 00
7A 28 60,4/
: S
CHANGE BUFFER
```

	PSN=00F0	LSN=FFFF	
00	6C 4A 80 6C 00 00 00 00	00 00 00 00 00 00 00 00	1J.1.....
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 18 00 6D	60 00 60 04 00 00 00 00	.....m`^..

```
: W
: Q
=
```



## CHAPTER 10

---

### 10. FORMAT COMMAND

---

The FORMAT command attempts to write the sector addressing information on diskettes. Non-formatted diskettes or diskettes on which the sector addressing information is destroyed must be formatted with this command before they can be used with XDOS.

#### 10.1 Use

---

The FORMAT command is invoked with the following command line:

FORMAT [:n]

The diskette to be formatted must reside in drive n. Since the FORMAT command will destroy all information on the diskette in drive n, the prompt

FORMAT DRIVE n ?

will be displayed. Any response other than "Y" will cause the FORMAT command to be terminated and control returned to XDOS. In this case, the diskette in the drive n is unaffected. If the "Y" response is entered, the operator should have placed a diskette that needs to be formatted into the drive n.

FORMAT will then proceed to:

1. Rewrite the soft sector addressing information on each track (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 track 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 XDOS. 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.

## 10.2 Messages

---

The only messages that the FORMAT command can display are the prompt shown above, asking if the diskette in the drive n is to be formatted, and the standard PROM I/O error message, indicating that a diskette controller error was encountered during the formatting process. In addition XDOS will, in the case that the destination diskette is a minidiskette issue the user is prompted with the following question.

SINGLE- OR DOUBLE-SIDED DISK (S/D) ?

Answering "S" causes side 1 of the minidisk to be formatted, while "D" initiates the formatting of both sides.

## 10.3 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 00D0
=FORMAT
FORMAT DRIVE 1 ?
Y
**PROM I/O ERROR-STATUS=38 AT 2006 ON DRIVE 1-PSN 00F0
=FORMAT
FORMAT DRIVE 1?
Y
**PROM I/O ERROR-STATUS=38 AT 2006 ON DRIVE 1-PSN 0150
=FORMAT
FORMAT DRIVE 1?
Y
**PROM I/O ERROR-STATUS=31 AT 2006 ON DRIVE 1-PSN 0150
=FORMAT
FORMAT DRIVE 1?
Y
**PROM I/O ERROR-STATUS=31 AT 2006 ON DRIVE 1-PSN 0150
=
```

## CHAPTER 11

### DISKETTE MANAGEMENT

#### 11. FREE COMMAND

The FREE command displays the number of unallocated sectors and the number of empty directory entries remaining on a diskette.

##### 11.1 Use

The FREE command program is invoked with the following command line:

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

where <unit> can be the logical unit number, 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. A "C" option can be added to the existing "L" option; both options can be specified in the command line option field, without separating comma.

Option "C" causes the configuration parameters of a diskette to be displayed, following the standard display of the number of unallocated sectors and empty directory entries.

Example :

Display on the line printer the amount of free sectors, empty directory entries, and configuration parameters of the diskette in drive 5.

```
=FREE :5;LC
DRIVE 5 : XDOS4SYS
      0268/$10C SECTORS  136/$88 FILES
      0268/$10C LARGEST CONTIGUOUS BLOCK
CONFIGURATION PARAMETERS :
  . MAIN DRIVES :
    16(SS) OR 32(DS) SCT/CYL,
    0640(SS) OR 1280(DS) USABLE SECTORS.
  . AUXILIARY DRIVES :
    26(SS) OR 52(DS) SCT/CYL,
    2000(SS) OR 4004(DS) USABLE SECTORS.
```

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.
xxxxxxx	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.

## CHAPTER 12

### 12. 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.

#### 12.1 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.

### 12.1.1 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

Smmmm

The "Lnnnnn" form is used to specify a starting physical line number. The value "nnnnnn" must be a 1-5 digit decimal number in the range 1-65535, inclusive. The "Smmmm" form is used to specify a starting logical sector number. The value "mmmm" 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.

### 12.1.2 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 trying to find verify errors from the COPY command (Chapter 5) 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.

#### 12.1.3 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.

#### 12.1.4 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.

### 12.2 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 12.1.4.

**\*\* 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.

**12.3 Examples**  
-----

The XDOS 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 CTL-P key was depressed to terminate the LIST command and return control to XDOS in this example.

```
=LIST EQU
PAGE 001 EQU .SA:0
OPT NOL
PAGE
*
* X DOS VERSION 03.00 -- SYSTEM EQUATE FILE -- AUGUST 14, 1979
*
SPC 3
*
* S Y S T E M   F U N C T I O N   D E F I N I T I O N
*
=
```

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 OPT NOL
00002 PAGE
00003 *
00004 * X DOS VERSION 03.00 -- SYSTEM EQUATE FILE -- AUGUST 14, 1979
00005 *
00006 SPC 3
00007 *
00008 * S Y S T E M   F U N C T I O N   D E F I N I T I O N
00009 *
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
.STCHR EQU .STCHB+1 STORE CHARACTERS
.ALPHA EQU .STCHR+1 CHECK ALPHABETIC CHARACTER
.NUMD EQU .ALPHA+1 CHECK DECIMAL DIGIT
.ADDAM EQU .NUMD+1 INCREMENT MEMORY (DOUBLE BYTE) BY A
.SUBAM EQU .ADDAM+1 DECREMENT MEMORY (DOUBLE BYTE) BY A
.MMA EQU .SUBAM+1 MULTIPLY (SHIFT LEFT) MEMORY BY A
.DMA EQU .MMA+1 DIVIDE (SHIFT RIGHT) MEMORY BY A
.MDENT EQU .DMA+1 ENTER X DOS WITHOUT RELOADING
.LOAD EQU .MDENT+1 LOAD A FILE FROM DISK
.DIRSM EQU .LOAD+1 DIRECTORY SEARCH AND MODIFY
.PFNAME EQU .DIRSM+1 PROCESS FILE NAME
=
```

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 CTL-P key was used in this example to terminate the display of the file.

```
=LIST EQU,S5
PAGE 001 EQU .SA:0
T (TERM W/ EOT, NO CR/LF)
.CKBRK EQU .DSPLZ+1 CHECK CONSOLE FOR BREAK KEY
.DREAD EQU .CKBRK+1 EROM DISK READ
.DWRIT EQU .DREAD+1 EROM DISK WRITE
.MOVE EQU .DWRIT+1 MOVE A STRING
.CMPAR EQU .MOVE+1 COMPARE STRINGS
.STCHB EQU .CMPAR+1 STORE BLANKS
.STCHR EQU .STCHB+1 STORE CHARACTERS
.ALPHA EQU .STCHR+1 CHECK ALPHABETIC CHARACTER
.NUMD EQU .ALPHA+1 CHECK DECIMAL DIGIT
=
```

The following example displays the XDOS 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 CTL-P key was used to terminate the display.

```
=LIST EQU;F20
PAGE 001 EQU .S
OPT NOL
PAGE
*
* XDOS VERSION 03.00
*
SPC 3
*
* S Y S T E M      F
=
=
```

The last example lists the first nine lines of the XDOS 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 XDOS SYSTEM EQUATE FILE

PAGE 001 EQU      .SA:0  THIS IS THE XDOS SYSTEM EQUATE FILE

00001    OPT NOL
00002    PAGE
00003    *
00004    * XDOS VERSION 03.00 -- SYSTEM EQUATE FILE -- AUGUST 14, 1979
00005    *
00006    SPC 3
00007    *
00008    *   S Y S T E M      F U N C T I O N      D E F I N I T I O N
00009    *
=
```



## CHAPTER 13

---

### 13. 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 Alternate Memory Map of the EXORset, and for loading a program over the resident operating system.

#### 13.1 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 XDOS, that load outside range of contiguous memory known to XDOS, or that execute in the Alternate Memory Map of the EXORset 30 correctly 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 XDOS; keep XDOS vectors to allow system function access.
U	Load program into the Alternate Memory Map of the EXORset 30. The Memory Maps must be correctly configured; disable XDOS vectors.
V	Allow program to load over XDOS or anywhere else in memory; disable XDOS vectors.

Other Option	Function
none	Enter debug monitor after loading program.
G	Execute program after loading.
(<str>)	Initialize XDOS 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 13.1.1) are guaranteed that memory exists since the memory was sized at initialization time; however, programs loading into discontiguous areas of memory are not guaranteed that memory exists. Programs loaded in the Alternate Memory Map are only guaranteed the memory exists at their load address. 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.

#### 13.1.1 Command-interpreter-loadable programs

Programs that can be loaded by the XDOS 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 XDOS command interpreter by entering their file names as the first file name specification on an XDOS 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 memory locations at the proper addresses so it loads above XDOS). 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 originated 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 XDOS initialization. Such programs require an additional fifty bytes of memory beyond their highest loaded address to allow room for a stack for the diskette controller. These fifty bytes must be within the contiguous memory block known to XDOS.

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
Y	Highest address loaded into
S	Highest address loaded into +50
U	Highest address loaded into
DP	Zero
A	Zero
B	Zero
CC	\$50 (F and I set, E, H, N, Z, V and C clear)

When the G option is used, the registers are initialized as above, except the condition code: F and I are set, the remainder is indeterminate.

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 100 (decimal) bytes in size.

In addition to setting up the pseudo registers, the LOAD command will change the XDOS variable ENDUS\$ (Chapter 17) 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 20).

### 13.1.2 Non-command-interpreter-loadable programs

---

Programs are not loadable by the XDOS 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 XDOS. Such programs cannot be executed directly via the XDOS 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 first 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 17.2).

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
Y	Highest address loaded into
S	EXORbug stack address
U	Highest address loaded into
DP	Zero
A	Zero
B	Zero
CC	\$50 (F and I set, E, H, N, Z, V and C clear)

When the G option is used, The registers are initialized as above, except the condition code register: The F and I bit are set, the remainder is indeterminate.

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 vector link 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 XDOS functions.

#### 13.1.3 Programs in the Alternate 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 Alternate Memory Map of the EXORset 30.

```
LOAD TESTPROG;U
```

The alternate map configuration is tested prior to load the program. If the alternate memory map is not configured correctly for the program to be loaded, an error message will be displayed.

The only requirement placed on programs loading into the Alternate Memory Map is that the ending load address not be greater than \$FFFF. Otherwise, any memory allocations

(\$0000-FFFF) can be loaded into; however, no check is made to ensure that memory exists where the program is loaded, except at its load address. If the "G" option is omitted, the debug monitor will be entered after the program is loaded. The debug monitor will display the Alternate Memory Map prompt, not the Current Memory Map prompt. The pseudo registers will contain the following values:

#### Pseudo register Contents

---

P	Starting execution address
X	Lowest address loaded into
Y	Highest address loaded into
S	EXORbug monitor stack address
U	Highest address loaded into
DP	Zero
A	Zero
B	Zero
CC	\$50 (F and I set, E, H, N, Z, V and C clear)

When the G option is used, registers are initialized as above, except the condition code register: The F and I bit are always set, the others are indeterminate.

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 "M6809 EXORset 30 User's Guide" should be consulted for a complete discussion of the Alternate Memory Map.

If the "U" option is used (with or without the "G" option), the vector link 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 XDOS functions.

#### 13.1.4 XDOS command line initialization

---

The Other Option (<str>) is used while testing command-interpreter-loadable programs (section 13.1.1). 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 XDOS command line buffer. The command line buffer pointer, CBUFP\$ (Chapter 17), will be adjusted to point to a null character which precedes the string (a valid terminator for the .PFNAM function, Chapter 20). 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).

### 13.1.5 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.

The LOAD command has configured itself so that the ";P" command will cause a normal return to the XDOS command interpreter.

If the "V" option was used without a file name specified on the command line, the ";P" command will cause XDOS to reinitialize as if an "XDOS" command had been given to the debug monitor.

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.

## 13.2 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 and the Alternate Map was not correctly configured.

### \*\* 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 DUMP command (Chapter 9) 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 XDOS (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 \$F000.

In the cases where the ending load address exceeds \$F000, 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 so that the starting load address is a multiple of eight. If this is not the case, the diskette is probably damaged. The BACKUP command (Chapter 3) should be invoked to save the valid data residing on the diskette.

**\*\* 30 INVALID EXECUTION ADDRESS**

The file from which a program is to be loaded has an invalid RIB. The starting execution address lies outside of the block of memory that would be loaded by the program.

**13.3 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 originated to load above the resident XDOS and below the end of contiguous memory. The XDOS 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 XDOS functions:

```
LOAD BLAKJACK;VG
```



## 14. 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 updating memory-image files.

## 14.1 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.
-A <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>,<dirname>

is identical to the command line

COPY <name 1>,<dirname>

assuming that <name 1> is not the same as <dirname>.

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 File format  
shown by DIR

0	User-defined
2	Memory-image
3	Binary record
5	ASCII record

Memory-image files can be merged together. The file <dirname>, 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 <dirname> already exists, MERGE cannot ensure such allocation. For all other file formats that <name i> can assume, <dirname> can already exist. In such cases where <dirname> is different from <name 1> and already exists in the directory (and no "W" option on command line), the message

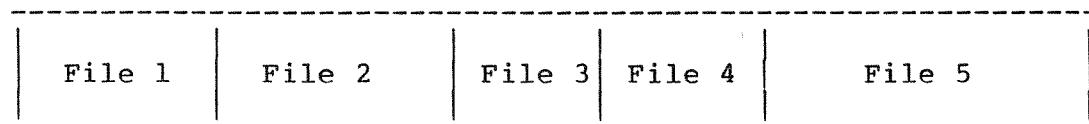
<dirname> EXISTS. OVERWRITE?

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 XDOS.

#### 14.1.1 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 <dirname> 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



:  
..... 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.

#### 14.1.2 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 Location	1	2	3
1	6	8	F
0	0	0	F
0	0	0	F
22222222222222222222222211111111			33333333
22222222222222222222222211111111			33333333
22222222222222222222222211111111			33333333
:	:		:
:	:		:
:	....Overlaid <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 overlaid 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, <dbname> 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 <dbname>.

#### 14.1.3 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.

#### 14.2 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 returned to XDOS.

\*\* 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 DUMP command (Chapter 9) 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>.

**14.3 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,PART2:1,PART3:1,PART4:0,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 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 15

### 15. 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.

#### 15.1 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 protections
S	Set system attribute
N	Remove system attribute

The <options> are discussed in detail in the following sections.

#### 15.1.1 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,BLAJKACK
```

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 *.LX,.SA
```

would change all file names on drive zero that had the suffix "LX" (as would be created by the assembler when requesting EXORbug-loadable file format) so that they had the new suffix "SA". Similarly, the command line

```
NAME TESTPROG.*:1,BLAJKACK
```

would change all files named TESTPROG (any suffix) on drive one to have the new name BLAKJACK. The suffixes would remain the same, preserving the identity of source, EXORbug-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.

### 15.1.2 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 protections), "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.

## 15.2 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>.

## 15.3 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 *.LX,.SA;X
```

This example will take all file names with the suffix "LX", change it to "SA", 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:0,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 16

### 16. ROLLOUT COMMAND

The ROLLOUT command is used for writing the contents of memory to diskette. The ROLLOUT command supports the current and alternate memory maps of the EXORset. Options exist for writing memory directly into a diskette file or for writing to a scratch diskette.

#### 16.1 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 16.1.4). 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 Alternate Memory Map.
none	Write memory into a file. Only memory not overlayed by XDOS 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 4). Since most of the processing is done by a position-independent routine that must work without XDOS being resident, the resident XDOS I/O functions cannot be used. Therefore, the special keyboard keys CTL-X, CTL-D, CTL-W, CTL-P, 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 XDOS 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.

#### 16.1.1 Alternate Memory Map

---

When the ROLLOUT command is invoked with the command line

ROLLOUT <name>;U

the memory from the Alternate Memory Map will be written into the diskette file <name> on the specified logical unit. If the alternate memory map is not configured, ROLLOUT will terminate after displaying the following message:

ALTERNATE MEMORY MAP NOT CONFIGURED

If the alternate 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 Alternate 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 XDOS.

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 XDOS.

### 16.1.2 Non-overlaid memory

If the ROLLOUT command is invoked with the command line

```
ROLLOUT <name>
```

then any block of memory not overlayed by XDOS or the ROLLOUT command 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 16.1.1, 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 XDOS or the ROLLOUT command. If an address range is specified that falls into the overlaid memory, the message

START ADDRESS MUST BE GREATER THAN \$nnnn

will be displayed. The "nnnn" is the last address that has been used by XDOS 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.

### 16.1.3 Overlaid memory

If the ROLLOUT command has been invoked with the command line

```
ROLLOUT ;V
```

then any block of memory can be subsequently 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 formatted 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 XDOS 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 XDOS 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 XDOS 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 XDOS. "nnnn" is the hexadecimal address of the last location in memory occupied by XDOS 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 XDOS.

Caution must be used in locating the position-independent routine in memory. Since XDOS 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 XDOS. 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 EXORbug 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 16.1.1. 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 formatted 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 XDOS. 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 16.1.4) to copy the raw data from the scratch diskette into a file on drive zero.

#### 16.1.4 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 XDOS.

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.

#### 16.2 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 XDOS 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 than "nnnn".

LOAD ADDRESS MUST BE GREATER THAN \$nnnn

The address specified for locating the position-independent routine in memory includes memory occupied by XDOS and/or the ROLLOUT command. The address must be greater than \$nnnn shown in the message.

ALTERNATE MEMORY MAP NOT CONFIGURED

The "U" option has been specified when the map decoding prom was not configured properly (see "EXORset 30 User's Guide" for more information).

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 XDOS.

**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 16.1.3 should also be reviewed.

**16.3 Examples**

The following example shows the operator-system dialogue for writing a block of memory to a file from the Alternate Memory Map:

```
=ROLLOUT AMBLOCK;U  
START ADDRESS: 9000  
END ADDRESS: 97FF  
ARE YOU SURE (Y, N, Q)? Y  
=
```

The file named AMBLOCK.LO will be created on drive zero. It will contain the block of memory from \$9000 to \$97FF, inclusive, from the Alternate Memory Map.

The following example illustrates how a copy of the diskette controller ROM can be written into a diskette file:

```
=ROLLOUT DISKROM:1  
START ADDRESS: E800  
END ADDRESS: EBFF  
ARE YOU SURE (Y, N, Q)? Y  
=
```

The file named DISKROM.LO will be created on drive one.

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 XDOS. A maximum contiguous memory range of 32K is assumed.

```
=ROLLOUT ;V
LOAD ADDRESS: 7F80
** 53 INSUFFICIENT MEMORY
=ROLLOUT ;V
LOAD ADDRESS: 7A00
=LOAD TESTPROG;V
. (User does testing here via EXORbug)
.7A00;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 XDOS command interpreter, an additional block of memory is allowed for the XDOS stack. Thus, the ROLLOUT command had to be invoked again. Then, after loading and testing his program, the operator invoked the routine via the "7A00;G" EXORbug 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 17

### 17. SYSTEM DESCRIPTION

This chapter contains the detailed descriptions of the structure of an XDOS diskette, the structure of XDOS files and their formats, the system overlays, the memory map, the command interpreter, interrupt handlers, the system function handler, and the XDOS equate file. The subsequent three chapters contain the detailed descriptions of the individual system functions and how they are parameterized.

#### 17.1 Diskette Structure

XDOS is based on a single and/or double sided 5.25" and/or 8" flexible disks. The diskettes are 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 XDOS as a complete, self-contained entity. Each diskette has its own system tables, operating system, and files.

Information on an XDOS diskette is stored in sectors 128 (decimal) bytes in size. The number of sectors per cylinder varies according to the diskette size and number of recording surfaces on each diskette.

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. XDOS 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.

Quantity	5.25" SS Dec/Hex	5.25" DS Dec/Hex	8" SS Dec/Hex	8" DS Dec/Hex
Surfaces/diskette	1/1	2/2	1/1	2/2
Bytes/Sector	128/80	128/80	128/80	128/80
Sectors/track	16/10	16/10	26/1A	26/1A
Tracks/cylinder	1/1	2/2	1/1	2/2
Sectors/cylinder	16/10	32/20	26/1A	52/34
Cylinders/diskette	40/28	40/28	77/4D	77/4D
Sectors/surface	640/280	640/280	2002/7D2	2002/7D2
Sectors/diskette	640/280	1280/500	2002/7D2	4004/FA4
Sectors/cluster	4/4	4/4	4/4	4/4
Clusters/diskette	160/A0	320/140	500/1F4	1001/3E9

XDOS accesses sectors on the diskette via a physical sector number (PSN). The diskette controller decodes the PSN into the appropriate track/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 track/sector and physical sector numbers, Appendix A has been provided. It contains the physical sector numbers of the first sector of each track.

A portion of each diskette is reserved for some special system tables. These tables reside in the outermost track of the diskette, track 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, XDOS RIB	\$17,18

### 17.1.1 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 XDOS overlay RIB addresses. The ID is displayed by the DIR and FREE 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	XDOS overlay RIB addresses
\$3A-\$7F	\$46	Zeroes

### 17.1.2 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

XDOS functions.

Bytes 0-\$13 of the CAT correspond to the physical locations on the diskette. Bytes \$14-7F are set to all ones.

#### 17.1.3 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 process. 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 track zero and any user locked out areas (see Chapter 8, DOSGEN command) are flagged as unavailable for normal allocation.

#### 17.1.4 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 XDOS 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 XDOS 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

XDOS default suffixes are:

Suffix	Implied meaning
AL	Assembly listing file
CF	Chain procedural file
CM	Command file file
LO	Loadable, memory-image file
LX	EXORbug-loadable 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 17.3.

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
<hr/>															
:	:	:	:	:	:	:	:								<----- 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
<hr/>															

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
<hr/>							
:	:	:	:	:	:		
:	:	:	:	:			.... Position within sector (0-7)
:	:	:	:	:			
<hr/>							
..... Physical sector number (\$3-\$16)							

### 17.1.5 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.

## 17.2 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.

XDOS 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.

#### 17.2.1 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, 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. It is the first memory location address occupied by the program once loaded into the memory. This value must be greater than a hexadecimal location \$1F if the program is to be loaded via the XDOS loader.

4. The ending load address is calculated from bytes 117-121 in the following manner:

$$EL = (NSL - 1) * 128 + NBLS + SL - 1$$

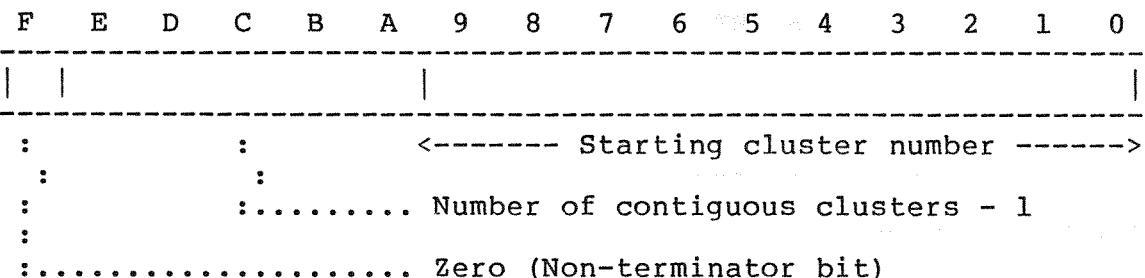
where EL is the ending load address, NSL is the number of sectors to load (bytes 118-119), NBLS is the number of bytes in the last sector (byte 117), and SL is the starting load address (bytes 120-121). The ending load address must be less than 65536.

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).

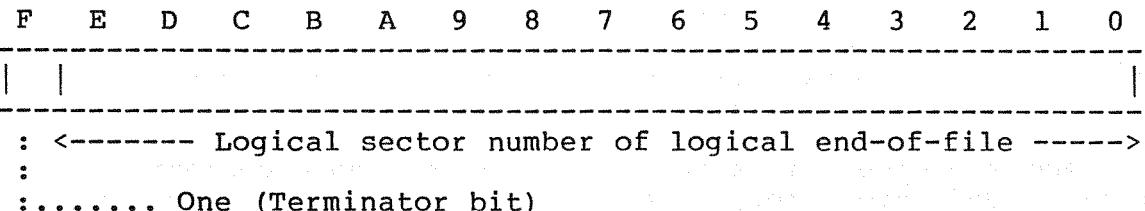
6. Bytes 124-127 are not used and must be zero.

The following diagrams illustrate the format of a segment descriptor word and the terminator.

#### SEGMENT DESCRIPTOR WORD



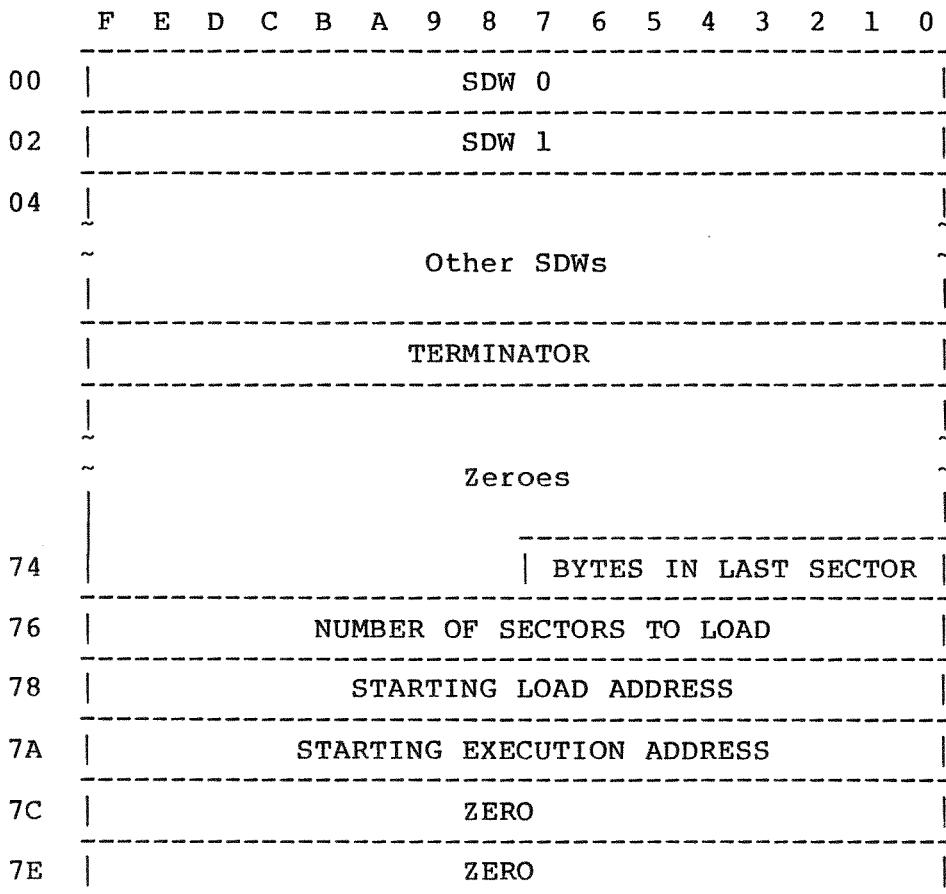
#### TERMINATOR



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.



### 17.2.2 File formats

XDOS 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 XDOS at the sector level. XDOS 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 XDOS 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, X DOS programs can only be loaded in multiples of eight bytes. A further restriction placed on memory-image files is that their content cannot be loaded below memory location \$20.

Binary record files may be used to record structured binary (not listable) information to the diskette. It offers a suitable alternative to the user defined file, since the record i/o is allowed with this files.

ASCII record files are used to contain all other X DOS-supported data. Such files can be in either space-compressed or non-space-compressed form. Normally, X DOS 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, X DOS 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.

### 17.3 Record Structure

---

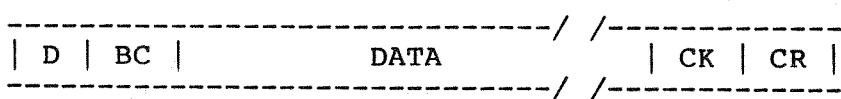
This section describes in detail the two record types supported for diskette files. In addition, a special record type used for copying binary files to a non-diskette device is also discussed. The actual use of such records is fully discussed in Chapter 18 which describes the supported I/O functions. All records supported by X DOS 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 X DOS.

#### 17.3.1 Binary records

---

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.

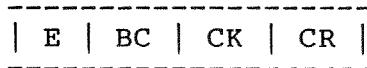
The format of a binary record can be illustrated as follows:



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:



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.

### 17.3.2 ASCII records

---

ASCII records are used primarily for source files on the diskette; however, EXORbug-loadable format files are ASCII even though they are object files output from the assembler.

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 XDOS 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. XDOS automatically expands these characters into spaces when such files are read. XDOS will also automatically create these compressed characters when such files are written.

Since XDOS 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 XDOS, 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.

#### 17.3.3 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 converting them to their ASCII-hexadecimal equivalent. 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.



#### 17.3.4 File descriptor records

---

XDOS 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 XDOS 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, XDOS can search for a named file on a sequential mass storage, 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:

H = High-order byte of the file number  
BC = Block count

NAME = Name of the file, up to 8 characters long.  
SUFX = File suffix, up to 3 characters long.  
NU = Number of the file, up to 2 digits long.  
FDF = File format descriptor, up to 2 digits long.  
CK = Checksum, up to 2 digits long.  
CR = Carriage return character.

The file number is the index of the file in the directory. It is used by the operating system to identify the file. The block count is the number of sectors required to store the file. The file name is the name of the file, and the suffix is the extension of the file. The file format descriptor is used to indicate the type of file, such as text or binary. The checksum is used to verify the integrity of the file.

The carriage return character is used to indicate the end of a line of text. The file number, block count, file name, suffix, and file format descriptor are all stored in ASCII format. The remaining data of the file is stored in binary format.

The file descriptor record is used to identify the file to the operating system. It contains the file number, block count, file name, suffix, file format descriptor, and checksum. The file number is used to identify the file in the directory. The block count is used to determine the size of the file. The file name and suffix are used to identify the type of file. The file format descriptor is used to indicate the type of file. The checksum is used to verify the integrity of the file.

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.

#### 17.4 System Files

---

On every XDOS 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 XDOS.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 XDOS has been initialized since they are referenced by their physical sector numbers.

##### 17.4.1 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 to be loaded from the diskette. The functions contained in the seven overlay files are shown in the following table:

Overlay	Function
XDOSOV0.SY	Diskette space allocation and deallocation.
XDOSOV1.SY	Processing standard file names, allocating contiguous memory, reserving a device, releasing a device, writing standard records, writing FDR's, writing end-of-file records, issuing next command.
XDOSOV2.SY	Reading standard records, reading FDRs.
XDOSOV3.SY	Closing a file/device, rewinding diskette files, changing file names and attributes.
XDOSOV4.SY	Opening a file/device.
XDOSOV5.SY	CHAIN file execution.
XDOSOV6.SY	Command line interpretation.

When X DOS 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, X DOS must be reinitialized each time the diskette in drive zero is changed so that the overlays can be located again.

Overlays XDOSOV0 and XDOSOV1 use overlay region one. Overlays XDOSOV2 and XDOSOV3 use overlay region two. Overlays XDOSOV4 and XDOSOV5 use overlay region three, and overlay XDOSOV6 uses the User Program Area into which the X DOS commands also are loaded. The overlay regions are shown in the memory map diagram of section 17.5.

#### 17.4.2 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 XDOSEN.SY. This file is accessed by the error message function .MDERR (section 20.4). 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.

#### **17.5 Memory Map**

---

The memory mapping of XDOS within the EXORset system is illustrated in the following diagram:

		DISKETTE CONTROLLER VARIABLES
0000		
0020	~	UNUSED DIRECT ADDRESSING
	~	AREA
00AE		COMMAND LINE BUFFER
00FE		COMMAND LINE BUFFER POINTER
0100		XDOS 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
		END OF MINIMUM SYSTEM MEMORY
BFFF		END OF CONTIGUOUS MEMORY
		RAM Discontinuity
	~	NON-XDOS RAM
	~	
E000		ALPHANUMERIC DISPLAY MEMORY
E800		DISKETTE CONTROLLER PROM
EC00		SYSTEM I/O (FDC)
F000		EXORbug MONITOR

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 X DOS functions), then these locations cannot be used by the program for storage. Locations \$00AE-00FD, inclusive, contain the X DOS 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 X DOS 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 X DOS 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 X DOS equate file. The pointers most often required are:

Pointer Name	Content
CBUFPS\$	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 X DOS. 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 X DOS system functions. This vector is set to point to an RTI instruction each time the X DOS command interpreter is given control.
IRQ\$VC	The address of an IRQ handler. This vector must be initialized by a user program if it is using IRQs. This

	vector is set to point to an error routine each time the X DOS command interpreter is given control.
FIR\$VC	The address of a FIRQ handler. This vector must be initialized by a user program if it is using FIRQs. This vector is set to point to an error routine each time the X DOS command interpreter is given control.
NMI\$VC	The address of a NMI handler. This vector must be initialized by a user program if it is using NMIs. This vector is set to point to an error routine each time the X DOS command interpreter is given control.
SW2\$VC	The address of a SWI2 handler. This vector must be initialized by a user program if it is using SWI2s. This vector is set to point to an error routine each time the X DOS command interpreter is given control.
SW3\$VC	The address of a SWI3 handler. This vector must be initialized by a user program if it is using SWI3s. This vector is set to point to an error routine each time the X DOS command interpreter is given control.

## 17.6 X DOS Command Interpreter

---

The X DOS command interpreter is one of the X DOS overlays that gets control whenever X DOS has been initialized or whenever a command has completed and returned control to X DOS. 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 X DOS 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 XDOS variable ENDOS\$ (greater than \$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 XDOS 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 XDOS command interpreter processed the input buffer. After completing its function, the command can return to XDOS through one of the system functions (.MDENT) which will pass control back to the XDOS command interpreter, repeating the cycle.

It should be noted here that commands invoked via the XDOS 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:1

will invoke the directory command from drive one to display the directory of the diskette in drive zero.

Whenever the XDOS 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 XDOS initialization. If these parameters differ, one of the standard error messages EI, ER, EU, EV (Chapter 21) will be displayed and control given to the debug monitor. XDOS will then have to be reinitialized before the XDOS command interpreter will accept further commands.

In addition, the following parameters are reinitialized each time the XDOS command interpreter is given control. The user-defined SWI vector (SWI\$UV) is reset to point to an RTI instruction. Since the user program is no longer resident, the interrupt handlers are deactivated and vectors are reset to point to error routines. 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.

## 17.7 Interrupt Handling

---

When XDOS initializes, it saves the contents of the interrupt vector link required by the debug monitor. The interrupt vector link is then changed to point into the XDOS

interrupt vector table. (See "EQU" file listing, appendix I). SWI vector is initialized to point into the X DOS function handler. The other interrupt vectors are configured to point to error routines. (See chapter 21). User programs, however, can configure the X DOS variables IRQ\$VC, NMI\$VC, SW2\$VC and SW3\$VC so that if one of these interrupts occur, the routine specified by the user will be given control.

Such user-defined interrupt handlers are accessible as long as the X DOS command interpreter is not re-entered. Whenever control is returned to the X DOS command interpreter, the interrupt vectors will be changed to point back into X DOS. Thus, interrupts cannot occur after the user program has terminated. Otherwise, X DOS will output an error message and return to the monitor. This is to be expected, since X DOS has no way of knowing what device generated the interrupt, where the device is, or how to respond to the interrupt. An interrupt must not be pending or occur when the X DOS command interpreter is given control. The X DOS variable SWI\$VC is not to be modified, since it is used for the X DOS functions access. The X DOS variable SWI\$UV is provided to allow the user to implement his own SWI routine.

Certain precautions must be remembered if a user program is to process IRQ's and use the X DOS diskette functions. The X DOS diskette controller uses IRQ as a ready signal from the Floppy Disk Controller. A non-disk IRQ which occurs during a disk access will always cause the disk data transfer to be erroneous. The user's program must then inhibit IRQ's at the peripheral level when the disk must be accessed.

FIRQ is masked during disk controller program execution. If a FIRQ occurs at this time, interruption will be in effect at the completion of the disk operation only. No NMI may occur during a diskette function execution, since it will break the critical delay sequence of the disk data transfer. Thus, a user program cannot capture an interrupt (IRQ, FIRQ or NMI) during a diskette function execution.

The system functions provided by X DOS are accessible through use of the software interrupt or SWI instruction. A full explanation regarding the X DOS SWIs is given in the next section; however, X DOS allows a user-defined SWI vector to be configured through the variable SWI\$UV. The user-defined SWI handler is only accessible as long as the X DOS command interpreter is not reentered. Whenever control is returned to the X DOS command interpreter, the user-defined SWI vector will be changed to point back into X DOS. Thus, user-defined SWIs cannot be processed after the user program has terminated. This is to be expected, since X DOS commands and user programs all load into one area of memory. Thus, the user-defined SWI handler is not resident after the X DOS command interpreter regains control.

## 17.8 System Function Calls

---

All of the system functions that X DOS commands use are also available to the user and can be incorporated into his program development. All X DOS 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. XDOS'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 XDOS 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.

XDOS 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 XDOS assembler, the provided definitions with the names SCALL and UCALL can be used to generate the code for XDOS system functions and user-defined functions, respectively. They require an argument to be passed. This argument is the name or value of the function to be executed. The names of XDOS functions are assigned symbols in the XDOS equate file (next section) so that the use of absolute numbers is not necessary. Use of the SCALL or UCALL pseudo-instructions makes the program a bit easier to read, especially if names are used for the pseudo-instruction arguments.

XDOS system functions receive their parameters in the registers or in tables that are pointed to by the registers. Chapters 18 and 20 contain the detailed entry parameters and exit conditions for all XDOS 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 XDOS 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.

### 17.9 XDOS Equate File

With each XDOS system diskette comes a file, EQU.SA, known as the XDOS equate file. The XDOS equate file contains the definitions of all symbols that are required by the resident XDOS and all of the XDOS 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 XDOS equate file contains the following definitions. The sequence of the descriptions more or less follows the sequence of the file from beginning to end.

First is a list of names that identifies all of the system functions accessible via the SCALL pseudo-instruction (or a SWI instruction followed by a function byte). The first function is given the value of zero. Subsequent functions are assigned a number one higher than the previous function. If the SCALL pseudo-instruction 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 XDOS delimiters used for suffixes, options, logical unit numbers, device names, and family indicators.

Next is a list of XDOS sector equates that defines where the various system tables are located. In addition, the sector size and the sectors/track, 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 sequenced definitions for the various I/O function error statuses.

Following the error statuses, the locations of all of the XDOS 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 EXORbug required by XDOS. The EXORbug equate list is not complete. Thus, users requiring other entry points into EXORbug must provide them within their programs.

If programs are being written that use the resident XDOS

functions, it is suggested that the XDOS equate file be included as a part of the assembly. Symbols within the XDOS equate file may have their values changed by Motorola in subsequent versions of XDOS; however, all attempts will be made to ensure a minimal number of such changes. Therefore, the XDOS equate file should not be copied from one version of XDOS to another. Like the resident system and command files that comprise the operating system, the XDOS equate file is associated with a specific version and revision of the operating system.

A listing of the XDOS equate file is contained in Appendix H.

**CHAPTER 18****18. 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 X
Y	Index register Y
U	User stack pointer
DP	Direct page 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 17.8).

**18.1 Supported Devices**

XDOS provides input and output functions to access the following supported devices:

	XDOS Name	Physical Device
	CN	Console keyboard and/or display
	DK	Diskette drive
	LP	Line printer

The following sections describe the system functions that are available for accessing these devices.

**18.2 Device Dependent I/O Functions**

XDOS 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 XDOS 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 120 bytes (decimal). Each system function call pushes twelve

bytes on the stack. Since function calls may be nested within X DOS, a large stack is required. It should be noted that EXORbug 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 18.3) via parameter tables held in the X DOS 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 X DOS command interpreter. Since X DOS 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 18.3) must be used instead.

#### 18.2.1 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 XDOS 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 XDOS 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, Y, U and DP are unchanged.

CC is indeterminate.

The input buffer contains the entered data, including the terminating carriage return.

#### 18.2.2 Check for BREAK key -- .CKBRK

---

The .CKBRK function examines the system console PIA to see if a CTL-P 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, DP, U, Y and X registers are unchanged.

C = 0, Z = 1 if CTL-P has not been depressed. The remainder of CC is indeterminate.

C = 1, Z = 0 if CTL-P has been depressed.  
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.

If CTL-P is depressed during a console input, it is not considered as a break request, but as an input character.

#### 18.2.3 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:** U, Y, DP, A and B registers are unchanged.

X = The address of the string's terminating character.

CC is indeterminate.

#### 18.2.3.1 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, X DOS is given control via the system function .MDENT. This function is described in Chapter 20. A maximum string length of ten is allowed. The example has been assembled with the X DOS equate file.

It is assumed in this example that the program is originated above location \$1FFF since it is using the resident X DOS functions. The program can either be loaded with the LOAD command or invoked from the X DOS 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   LDB    #10        MAX 10 CHAR
       LDX    #IBUFF
       SCALL  .KEYIN     GET INPUT STRING
       TSTB
       BNE    SWAP       CHECK FOR ZERO INPUT
       SCALL  .MDENT     EXIT IF NO INPUT
*
* INVERT STRING INTO OBUFF
*
SWAP    LDY    #OBUFF     POINT TO OUTPUT BUFFER
       LDX    #IBUFF     POINT TO END OF INPUT BUFFER
       ABX
LOOP    LDA    0,-X      GET CHAR
       STA    0,Y+      STORE CHAR
       DECB
       BNE    LOOP       TALLY COUNTER
       LDA    #CR        LOOP UNTIL ZERO
       STA    0,Y        STORE TERMINATOR
       LDX    #OBUFF     INTO OUTPUT BUFFER
       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
       BSZ    120       STACK SET HERE BY LOAD
*
END    START      BEGIN EXECUTION AT THIS LABEL

```

#### 18.2.4 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:** U, Y, DP, A and B registers are unchanged.

X = The address of the string's terminating character.

CC is indeterminate.

#### 18.2.4.1 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 XDOS. Both of these functions are described in Chapter 20. The example was assembled with the XDOS equate file.

It is assumed in this example that the program is originated above location \$1FFF since it is using the resident XDOS functions. The program can either be loaded with the LOAD command or invoked from the XDOS 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 LDA #SPACE INITIAL CHARACTER
LOOP LDX #OBUFF
      LDB #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 XDOS
*
* WORKING STORAGE
*
OBUFF BSZ 80 OUTPUT BUFFER
FCB CR
BSZ 120 STACK SET HERE BY LOAD
*
END START BEGIN EXECUTION AT THIS LABEL

```

#### 18.2.5 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 18.2.7 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, seek 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 track zero) a maximum of five tracks, repositioned over the track in which the sector to be read resides, and another five read attempts were performed.
3. The read head was stepped inward (towards track 39) a maximum of five tracks, repositioned over the track in which the sector to be read resides, and another five read attempts were performed.
4. The drive is restored (forced seek to track zero), repositioned over the track in which the sector to be read resides, and another five read attempts were performed.
5. The read head was stepped outward (towards track zero) a maximum of five tracks, repositioned over the track in which the sector to be read resides, and another five read attempts were performed.
6. The read head was stepped inward (towards track 39) a maximum of five tracks, repositioned over the track 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 XDOS command interpreter. If an error occurs during the .EREAD function, the EXIT CONDITIONS described below apply (for C = 1).

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 to be read
2	Address of 128 byte sector buffer
3	
4	

EXIT CONDITIONS: C = 0 if no errors occurred. The remainder of the CC is indeterminate.

The A register is indeterminate.

The U, Y, DP and X register are 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 U, Y, DP and X register are 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 21.1 has a complete description of the diskette controller errors.

The contents of the 128 byte sector buffer are indeterminate.

**18.2.6 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 18.2.8 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 18.2.5 (.DREAD, .EREAD) was attempted.

**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 contents of the sector buffer are unchanged after returning to the calling program.

**18.2.7 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 18.2.5 (.DREAD, .EREAD) was attempted.

**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 .ERED; 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).

#### 18.2.8 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 18.2.5 (.DREAD, .ERED) was attempted.

**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.

### 18.2.9 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 XDOS, they are described in a separate section (Appendix D). That appendix also describes some entry points for accessing the line printer on an XDOS-independent basis.

## 18.3 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 XDOS equate file are used. These are noted in the description of the status byte of the IOCB, section 18.3.1.1. 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.

### 18.3.1 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 19.2). 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				
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 CDB address								IOCGDW				
09													
0A		R		LUN						IOCLUN -- Logical unit number			
0B	File name or								IOCNAME / IOCMLN				
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								IOCRIPT				
16													
17		W		D		S		C		N		FMT	IOCFDF - File descriptor flags
18	(reserved; =0)												

	7	6	5	4	3	2	1	0	
19									(reserved; =0)
1A									
1B									IOCDEN - Directory entry number
1C									(reserved; =0)
1D									IOCSBP
1E									Sector buffer pointer
1F									IOCSBS
20									
21									IOCSBE
22									
23									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)

### 18.3.1.1 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 (XDOS 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 21. 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 register equal to zero. Section 20.4 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
1A	I\$IFNM	** 25 INVALID FILE NAME

#### 18.3.1.2 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 18.3.6). 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 IOCMSL) 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 IOCMSL 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 IOCDF).

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 18.3.3.

#### 18.3.1.3 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.

#### 18.3.1.4 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 18.3.3) when dealing with non-diskette devices.

#### 18.3.1.5 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 18.3.3) when dealing with non-diskette devices.

#### 18.3.1.6 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 XDOS device name that is to be accessed (see section 18.1). The device name consists of two ASCII characters. Once the .RESRV function (section 18.3.2) has been called, IOCGDW will contain the address of the controller descriptor block (CDB, section 19.2.1) 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 19.2 contains a description of how to configure the IOCGDW field for non-supported devices.

#### 18.3.1.7 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.

#### 18.3.1.8 IOCNAME -- 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), IOCNAME is not used at all; however, in the file format mode, IOCNAME must contain the name of the file to be accessed. The file name must be in the valid XDOS file name format. Any unused parts of the name must be spaces (\$20). The file name should be placed into IOCNAME before the .OPEN function is invoked. After a file has been opened, the eight bytes will be replaced with the four two-byte fields IOCMIS, IOCSDW, IOCSLS, and IOCLSN (only if the device is diskette).

When dealing with non-diskette devices in the file format mode, the IOCNAME 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 IOCNAME (and IOCSUF).

#### 18.3.1.9 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 XDOS 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 IOCNAME). 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 IOCNAME 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 IOCNAME (and IOCSUF).

#### 18.3.1.10 IOCMSL -- Maximum LSN referenced

---

This two-byte field of the IOCB overlays the first two bytes of the IOCNAME 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. IOCMSL 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 XDOS 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 IOCMSL 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 IOCMSL must be set to a value of \$FFFF and the truncate flag must be set to one.

#### 18.3.1.11 IOCSDW -- Current SDW

---

The IOCSDW field overlays the second two bytes of IOCNAME 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.

#### 18.3.1.12 IOCSLS -- Starting LSN of SDW

---

The IOCSLS field overlays the third two bytes of IOCNAME 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.

#### 18.3.1.13 IOCLSN -- Next LSN

---

The IOCLSN field overlays the fourth two bytes of IOCNAME 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.

#### 18.3.1.14 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.

#### 18.3.1.15 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.

#### 18.3.1.16 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 XDOS 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 the device's default format for binary records. Each device has associated with its CDB (section 19.2) 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 17.2. 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.

#### 18.3.1.17 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 18.3.3) 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)

#### **18.3.1.18 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, XDOS 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 20.4) 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.

#### **18.3.1.19 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.

#### **18.3.1.20 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:

$$\frac{\text{IOCSBE}-\text{IOCSBS}+1}{128} = \text{INTEGER } (\text{Maximum # of Sectors})$$

IOCSBE will not be altered by any of the I/O functions.

### 18.3.1.21 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:

$$\begin{array}{l} \text{IOCSBI-IOCSBS+1} \\ \hline \text{----- = INTEGER (Actual # of Sectors)} \\ \text{128} \end{array}$$

### 18.3.2 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 19.2.4 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, 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.

U, Y, DP and X are 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.

### 18.3.3 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 17.2. 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 17.3.4).

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 17.3.4).

IOCNAM must contain a valid XDOS-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 XDOS-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 18.3.1.20).

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 18.3.1.20).

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.

U, Y, DP and X are 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, I\$EOM, I\$IVDV, I\$NONM, I\$NORV, I\$NRDY, I\$RIB, I\$WRIT, I\$IFNM.

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 IOCMSL, IOCSDW, IOCSLS and IOCLSN.

IOCMSL 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.

IOCRIPT 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 18.3.1.16.

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.

#### 18.3.4 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 17.3), their byte count, and their checksums. Only the data characters between the byte count and checksum fields will be returned. A carriage return will be the last data character in the data buffer. 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 XDOS 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 18.2.1).

**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.

U, Y, DP and X are 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\$BUFO, I\$CKSM, I\$CLOS, I\$DTYP, I\$EOF, I\$FTYP, I\$EOM, I\$NRDY, 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. This character will always be a carriage return.

IOCMLS, IOCSDW, IOCSLS, IOCLSN, IOCEOF, IOCSBP, and IOCSBI contain the system-maintained parameters as described in section 18.3.1 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.

#### 18.3.5 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 17.3), 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.

U, Y, DP and X are 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 18.3.1 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.

### 18.3.6 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 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 IOCMSL) will normally be deallocated on a cluster boundary and returned to the free space pool (assumes that the truncate flag and IOCMSL 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. 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 (IOCMSL) 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 IOCMILS are explicitly set up by the calling program. When IOCMILS 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.

U, Y, DP and X are 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.

#### 18.3.7 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.

U, Y, DP and X are 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.

#### 18.3.8 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 XDOS 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,
        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 XDOS 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 20. 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 XDOS 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). This function is discussed in detail in Chapter 20.

In this example, the assumption is made that the program is invoked from the XDOS command line. Thus, it must be originated 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    #PFNPAK  ADDRESS OF PROCESS FILE NAME PACKET  
       LDD    #OUTPUT+IOCLUN STANDARD FILE NAME AREA ADDRESS  
       STD    2,X      DESTINATION OF FILE NAME  
       LDD    CBUFP$   POINTER INTO CMD BUFFER  
       STD    0,X      SOURCE OF FILE NAME  
       SCALL  .PFNAM  FORMAT STANDARD FILE NAME  
       TSTB   .  
       BEQ    STARTA  CHECK FOR ERRORS  
       EQ    => GOOD NAME  
       ASLB   .  
       BCS    ERR1    CS => NAME MISSING  
       BSR    ERROR   ILLEGAL NAME MSG NUMBER  
       FCB    7  
*  
ERR1   BSR    ERROR   NAME REQUIRED MSG NUMBER  
       FCB    5  
*  
ERR3   BSR    ERROR   I/O ERR MSG NUMBER; DECODED  
       FCB    0  
*  
ERROR  LDB    [0,S++]  FETCH ERROR NUMBER  
       SCALL  .MDERR  OUTPUT ERROR MESSAGE  
       BRA    XDOS   GO EXIT  
*  
* 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
       LDA    0,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
       DECB   CALC END OF RECORD BUFFER
       ABX   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 XDOS
*
EXIT   LDX    #OUTPUT    POINT TO THE IOCB
       SCALL .CLOSE
       BCS    ERR3      CS => ERROR
       SCALL .RELES
       BCS    ERR3      CS => ERROR
XDOS   SCALL .MDENT    RETURN TO XDOS
*
* LEAVE SOME ROOM FOR STACK
*
       BSZ    120        STACK SET HERE BY LOAD
       END    START

```

#### 18.3.9 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.

##### 18.3.9.1 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 18.3.1.20). 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 18.3.1.20). 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.

U, Y, DP and X are 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\$EOF,

I\$SECB, 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 18.3.1. 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:

$$\frac{\text{IOCSBI}-\text{IOCSBS}+1}{128} = \# \text{ SECTORS READ}$$

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.

#### 18.3.9.2 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 18.3.1.20). 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 18.3.1.20).

**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.

U, Y, DP and X are 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, I\$RIB, I\$FSPC, 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 18.3.1. 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.

### 18.3.9.3 Rewind file -- .REWND

The .REWND function resets the pointers of the IOCB so that subsequent I/O functions will access the peripheral file as if it had just been opened, i.e., from the beginning.

XDOS 3 drivers are compatible if the DD\$RWD bit is clear.

Files opened in any mode can be rewound. If the peripheral can perform logical sector I/O (DD\$LOG set), the sector pointers are reset to perform the next I/O at the beginning of the file. Disk files are handled as follows:

- .Do nothing if already at beginning of file.
- .If last transfer was an output:
  - .If record I/O mode, flush data in sector buffer.
  - .If truncate bit set, truncate file.
- .Set IOCLSN=0, IOCSBI=IOCSBE, IOCSBP=0.
- .IOCMLS and DT\$TRU are not changed.

#### 18.3.9.3.1 .REWND entry and exit conditions

**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.

IOCSBS must contain the starting address of a sector buffer if logical sector I/O can be performed by the device involved. The sector buffer must be an integral number of sectors in size (see section 18.3.1.20).

IOCSBE must contain the address of the last byte of the sector buffer.

**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, Y, DP, and U are unchanged.

C = 0 and Z = 1 if no error occurred (B=0)  
C = 1 and Z = 0 if an error occurred (B#0)  
The remainder of CC is indeterminate.

The IOCB is affected in the following manner if an error occurred:

IOCSTA contains the error status: I\$NORV, I\$CLOS, I\$RWND and any error status returned by the driver.

The IOCB is affected in the following manner if no error occurred:

IOCSTA = 0.

IOCSDW contains the first SDW from the file's RIB if the device is DK and the file has been truncated.

IOCSLS contains the value \$FFFF if the device is DK and the file has been truncated.

IOCLSN contains 0 if logical sector I/O is allowed, unchanged otherwise.

IOCEOF has been set up if device is DK and file has been truncated.

IOCSBI contains the value in IOCSBE if logical sector I/O is allowed.

The remainder of the IOCB is unchanged.

#### 18.3.9.3.2 .REWND Error Message

.A standard error message has been added in the error file XDOSER.SY.

HEX INDEX NUMBER    ERROR MESSAGE

-----  
37                \*\* 54 DEVICE MAY NOT BE REWOUND

.The independent I/O error status code that matches the above error message is:

I\$RWND = \$1B (27 dec.)

.The standard name for the device driver rewind entry offset is:

DV\$RWD = 15 (dec.)

#### 18.3.9.4 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 18.3.8; 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 XDOS equate file are used throughout this example.

```
TEXFIL EQU      *          START OF TEXFIL IOCB
              FCB      0          IOCSTA
              FCB      DT$OPU+DT$SIO+DT$CLS IOCDTT
              FDB      0          IOCDBP
              FDB      0          IOCDBS
              FDB      0          IOCDBE
              FCC      2,DK       IOCGDW
              FCB      '0+0       IOCLUN -- DEFAULT = 0
              FCC      8,
              FCC      2,SA       IOCNAM
              FDB      0          IOCSUF -- DEFAULT = SA
              FDB      0          IOCRIB
              FDB      FD$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 XDOS 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 20. 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 XDOS 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, or 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 20.

In this example, the assumption is made that the program is invoked from the XDOS command line. Thus, it must be originated 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    #PFNPAK      PROCESS FILE NAME PACKET ADDRESS  
       LDD    #TEXFIL+IOCLUN STANDARD FILE NAME AREA ADDRESS  
       STD    2,X  
       LDD    CBUFP$        SOURCE OF NAME  
       STD    0,X  
       SCALL  .PFNAM      EXTRACT FILE NAME  
       TSTB   CHECK FOR VALID NAME  
       BEQ    STARTA      EQ => GOOD  
       ASLB  
       BCS    ERR1        CS => NAME MISSING  
       BSR    ERROR       ILLEGAL NAME MSG NUMBER  
       FCB    7  
*  
ERR1   BSR    ERROR       NAME REQUIRED MSG NUMBER  
       FCB    5  
*  
ERR3   BSR    ERROR       I/O FUNCTION ERROR MSG NUMBER  
       FCB    0  
*  
ERROR  LDB    [0,S++]     FETCH ERROR NUMBER  
       SCALL  .MDERR      EXIT  
       BRA    EXIT        DISPLAY ERROR, THEN EXIT PROGRAM  
*  
* 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  
       LDA    0,X          GET CHAR FROM BUFFER  
       BSR    CONVRT  
       STA    0,X+          PUT CHARACTER BACK  
       STX    TEXFIL+IOCSBP SAVE POINTER  
       CMPX   TEXFIL+IOCSBE CHECK FOR LAST CHARACTER  
       BLS    LOOP2        NE => MORE DATA TO CONVERT
```

```

* WRITE LOGICAL SECTOR BACK INTO FILE
*
      LDX    #TEXFIL   IOCB ADDRESS
      LDD    IOCLSN,X  PICK LSN PLUS ONE
      SUBD   #1        POINT BACK TO LAST READ SECTOR
      STD    IOCLSN,X
      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 XDOS 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
      ORA    #SPACE  CONVERT TO L/C
      CONVEX CLR    UCFLG  RESET FLAG TO CONVERT NEXT ALFA
      RTS
*
      CONTRM CMPA   #'.
      BEQ    SETFLG  PERIOD
      CMPA   #'!
      BEQ    SETFLG  EXCLAMATION
      CMPA   #'?
      BNE    CONEX2
      SETFLG INC    UCFLG
      CONEX2 RTS    DONE, RETURN
*
* SAVE SOME ROOM FOR STACK
*
      BSZ    120      STACK POINTER SET HERE BY LOAD
*
      END    START

```

#### 18.3.10 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 20; 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 XDOS.

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 19

---

### 19. INPUT/OUTPUT PROVISIONS FOR NON-SUPPORTED DEVICES

---

It is assumed that the reader is familiar with the device dependent I/O functions described in section 18.3 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.

#### 19.1 Device Dependent I/O

---

The device dependent I/O functions described in Chapter 18 for accessing the console and the line printer cannot be changed to access non-standard devices. These routines are a part of XDOS and its basic environment requirements; however, a user can construct his own device drivers that are accessed by his programs. It is not possible to use a non standard device/driver with the standard XDOS commands. The COPY command (Chapter 5) 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.

#### 19.2 Device Independent I/O

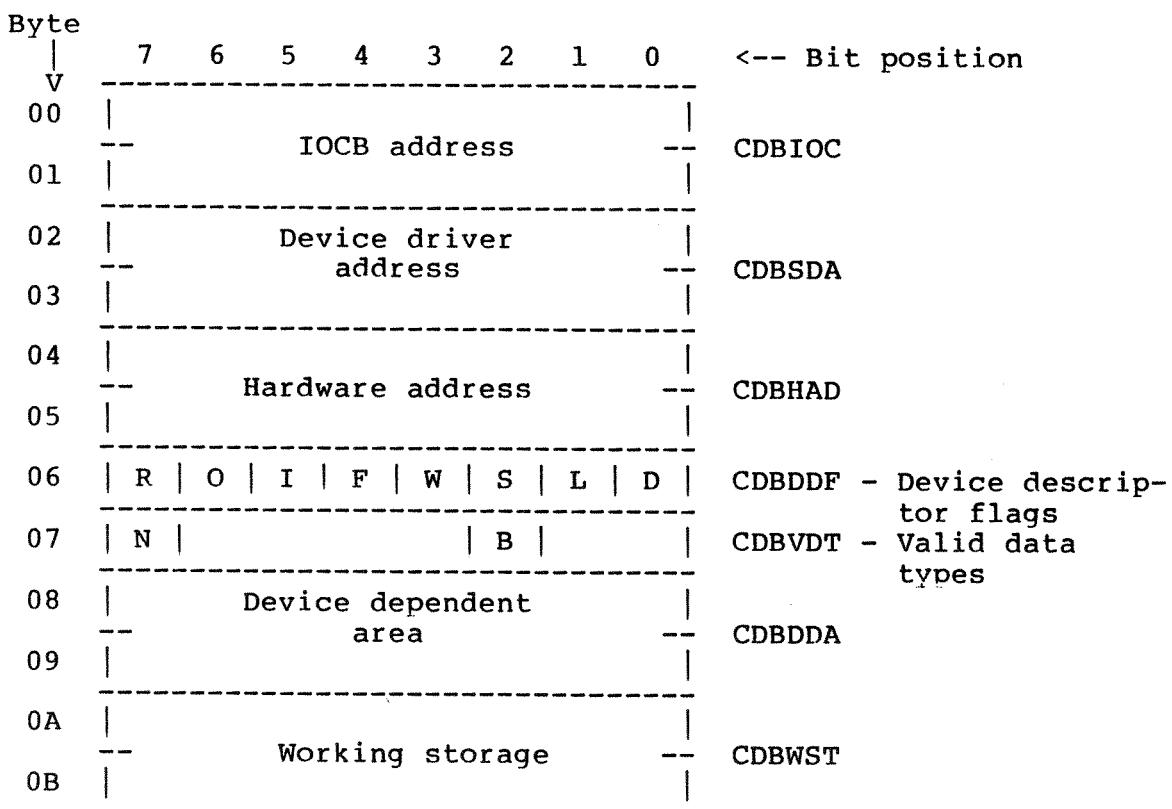
---

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.

##### 19.2.1 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.



## 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)

#### 19.2.1.1 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.

#### 19.2.1.2 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 19.2.2.

#### 19.2.1.3 CDBHAD -- Hardware address

These two bytes of the CDB are intended to contain the lowest address of the hardware device (PIA, ACIA, etc.) 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.

#### 19.2.1.4 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) 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 XDOS, 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 18.3.4. 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.

**19.2.1.5 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 18.3.1.2). 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 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.

#### 19.2.1.6 CDBDDA -- Device dependent area

These two-bytes of the CDB are available to the device drivers as working storage. For the XDOS-supported devices, this field has been provided for future expansion. For other devices, this field can be used for whatever purposes are deemed appropriate.

#### 19.2.1.7 CDBWST -- Working storage

These two-bytes of the CDB are available to the device drivers as working storage.

### 19.2.2 Device drivers

---

Each device type that is to be accessed via the device independent I/O functions (section 18.3) 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
	JMP	DEVIO	CHARACTER I/O ROUTINE
DEVRWD	EQU	*	DEVICE REWIND 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 XDOS 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" bit 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 18.3.1.1). 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", "X", "Y" and "U" 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.

### 19.2.3 Example of device driver

---

The following example illustrates a CDB and its associated device driver for a high-speed paper tape reader. The system symbols from the XDOS 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     $EE04        CDBHAD
FCB     DD$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 rewritable (according to the definition in section 19.2.1.4), 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.

```

*          *PIA EQUATES
*
PTCTL EQU 1          PIA CONTROL REGISTER
PTDTA EQU 0          PIA DATA REGISTER
*
*DEVICE DRIVER ENTRY POINTS
*
HRDRV$ EQU *
        JMP GOODR    TURN DEVICE ON
*
        JMP GOODR    TURN DEVICE OFF
*
        JSR INITR    DEVICE INITIALIZATION
*
        JMP GOODR    DEVICE TERMINATION
*
        BSR GETCP    CHARACTER INPUT
        TFR A,B      RETURN WITH CHAR IN "B"
        BCC RETURN   CC => NO ERROR
        LDX [0,S]    CS => END OF MEDIA (TIMEOUT)
        LDX 0,X      GET ADR OF IOCB
        LDA #I$EOM   SET END OF MEDIA STATUS
        STA IOCSTA,X
RETURN PULS X         RETURN TO CALLER
        JMP 2,X      JUMP TO ADR FOLLOWING FDB
GOODR CLRA    1-BYTE CLEAR CARRY
        BRA RETURN   EXIT DRIVER
*
* READER INITIALIZATION ROUTINE
*
INITR LDX HR$CDB+CDBHAD  GET THE PIA ADDRESS
        CLR PTCTL,X
        CLR PTDTA,X
        LDA #$3C
        STA PTCTL,X
        RTS
*
* INPUT ONE CHARACTER
*
GETCP LDX HR$CDB+CDBHAD  GET THE PIA ADDRESS
        LDA PTDTA,X  CLR INTERRUPT
        LDA #$34     STROBE READER
        STA PTCTL,X
        LDA #$3C
        STA PTCTL,X
        CLR HR$CDB+CDBWST  INIT THE TIMEOUT COUNTER
        CLR HR$CDB+CDBWST+1 AND CLEAR CARRY
GETC1 LDA PTCTL,X      READY TO READ?
        BMI GETC2    MI => YES
        DEC HR$CDB+CDBWST+1 PL => CHECK TIMEOUT
        BNE GETC1    NE => KEEP LOOPING
        DEC HR$CDB+CDBWST
        BNE GETC1    NE => KEEP LOOPING
        COMA      SET CARRY FOR TIMEOUT
GETC2 LDA PTDTA,X      GET CHAR
        BCS GETC4    CS => TIMEOUT
*
* IF ASCII FILE, STRIP PARITY

```

\*

LDX	[2,S]	GET ADR OF IOCB POINTER
LDX	0,X	GET ADR OF IOCB
LDB	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	CLRB	SET STATUS TO OK (CLEAR CARRY)
GETC4	RTS	

#### 19.2.4 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 18.3.2).

Once the IOCB has been configured in this manner, the other I/O functions can be used in the normal fashion.

## CHAPTER 20

### 20. 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 X
Y	Index register Y
S	Stack pointer register
U	User stack pointer register
DP	Direct page register
CC	Condition code register
E	Entire status flag of condition code register (bit 7)
F	FIRQ mask of condition code register (bit 6)
H	Half-carry flag of condition code register (bit 5)
I	IRQ mask of condition code register (bit 4)
N	Negative flag of condition code register (bit 3)
Z	Zero flag of condition code register (bit 2)
V	Overflow flag of condition code register (bit 1)
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

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 17.8).

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.

#### 20.1 Register Functions

The register functions was primarily intended for use as an extension of the M6800 instruction set (in 6800 MDOS III).

It should be noted that some of these functions are useless, since the equivalent hardware instructions are available in the M6809 instruction set. However, these entry points are still available, to preserve compatibility with MDOS III.

Care should be taken when using the double-byte register functions: they consider the B register as the most significant byte and the A register as the least significant. The M6809 D register is the concatenation of the A register as the most significant byte and the B register as the least significant. Using XDOS double-byte register functions in conjunction with the double-byte hardware features requires the A and B register swapping (via "EXG A,B") before and after each function call. It is then recommended to avoid the use of these functions in new programs to the benefit of the hardware instructions; old programs coming from M6800 MDOS III generally use these functions, they need not be adapted to the 6809 hardware and may carry on with the use of these functions.

#### 20.1.1 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.  
U, Y, DP and X are unchanged.  
CC is indeterminate.

Equivalent hardware code :

```
TFR    X,D  
EXG    A,B
```

#### 20.1.2 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: U, Y, DP, A and B are unchanged.  
XH contains B.  
XL contains A.  
CC is indeterminate.

Equivalent hardware code :

```
EXG    A,B  
TFR    D,X  
EXG    A,B
```

### 20.1.3 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.
- U, Y, DP and CC are unchanged.

Equivalent hardware code:

```
EXG    A,B
EXG    D,X
```

### 20.1.4 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:**

- U, Y, DP, A and B are unchanged.
- X has been incremented by the contents of B.
- CC has been set as in a normal unsigned addition.

This function is equivalent to the hardware instruction "ABX". However, unlike this instruction, the condition code is modified.

### 20.1.5 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:**

- U, Y, DP, A and B are unchanged.
- X has been incremented by the contents of A.
- CC has been set as in a normal unsigned addition.

Equivalent hardware code (CC not modified):

```
EXG    A,B
ABX
EXG    A,B
```

## 20.1.6 Add B,A to X -- .ADBAZ

The .ADBAZ function adds the contents of the register pair B,A to the contents of the X register.

ENTRY PARAMETERS: None.

EXIT CONDITIONS: U, Y, DP, A and B are unchanged.  
 X has been incremented by the contents of B,A.  
 CC has been set as in a normal unsigned addition.

Equivalent hardware code (CC: Z tested only):

```
EXG    A,B
LEAX   D,X
EXG    A,B
```

## 20.1.7 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.  
 U, Y, DP, X are unchanged.  
 CC has been set as in a normal unsigned addition.

Equivalent hardware code (r?quires 2 stack bytes):

```
PSHS   X
EXG    A,B
ADDD   0,S++
EXG    A,B
```

## 20.1.8 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: U, Y, DP, A and B are unchanged.  
 X has been decremented by the contents of B.  
 CC has been set as in a normal, unsigned subtraction.

**20.1.9 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: U, Y, DP, A and B are unchanged.  
X has been decremented by the contents of A.  
CC has been set as in a normal unsigned subtraction.

**20.1.10 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: U, Y, DP, A and B are unchanged.  
X has been decremented by the contents of B,A.  
CC has been set as in a normal unsigned subtraction.

**20.1.11 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.  
U, Y, DP and X are unchanged.  
CC has been set as in a normal unsigned subtraction.

**20.1.12 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: U, Y, X, DP, A and B are unchanged.  
CC has been set as in a normal unsigned subtraction.

**20.1.13 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: U, Y, DP, A and B are 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.

#### 20.1.14 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: U, Y, DP, A and B are 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.

#### 20.1.15 Push X on stack -- .PSHX

The .PSHX function pushes the contents of the X register on the current stack.

Since the equivalent hardware instruction PSHS X occupies only 2 bytes, this function has been modified to (try to) replace the function call in memory by the PSHS X op-code value. However, it pushes the content of the X register anyway. The next time that the same sequence will be executed, The hardware instruction will have replaced the call and will be executed.

ENTRY PARAMETERS: None.

EXIT CONDITIONS: U, Y, X, DP, A, B and CC are unchanged.  
S has been decremented by 2. The contents of XL have been pushed on the stack followed by the contents of XH.

Equivalent hardware code:

PSHS X

#### 20.1.16 Pull X from stack -- .PULX

The .PULX function pulls the contents from the stack into the X register.

Since the equivalent hardware instruction PULS X

occupies only 2 bytes, this function has been modified to (try to) replace the function call in memory by the PULS X op-code value. However, it pulls the contents from the stack into the X register anyway. The next time that the same sequence will be executed, the hardware instruction will have replaced the call and will be executed.

**ENTRY PARAMETERS:** None.

**EXIT CONDITIONS:** U, Y, DP, A, B and CC are 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.  
 S has been incremented by 2.

**Equivalent hardware code:**

PULS X

## 20.2 Double-byte Arithmetic Functions

---

The double-byte arithmetic functions are used by some of the other system functions and the XDOS commands as an extension of the M6809 instruction set. These functions are not as general purpose as the register functions, but they are useful in special cases.

### 20.2.1 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.  
 U, Y, X, DP and B are unchanged.  
 CC is indeterminate.  
 The double byte in memory has been incremented by the contents of A.

### 20.2.2 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.  
 U, Y, X, DP and B are unchanged.  
 CC is indeterminate.  
 The double byte in memory has been

decremented by the contents of A.

#### 20.2.3 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: U, Y, X, DP, A and B are 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.

#### 20.2.4 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: U, Y, X, DP, A and B are 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.

### 20.3 Character String Functions

The character string functions are used by some of the more complex system functions and the XDOS commands as macro instructions or subroutines.

#### 20.3.1 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 initially 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 source string	
1			
2		Address of the destination string	

**EXIT CONDITIONS:** **A** is indeterminate.  
**B** = 0.  
**U, Y, X** and **DP** are 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.

#### 20.3.2 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 initially 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 source string	
1			
2		Address of the 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.  
**U, Y, X** and **DP** are 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.

#### 20.3.3 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:** U, Y, X, DP and A are unchanged.  
B = 0.  
CC is indeterminate.  
The string is filled with the character in A.

#### 20.3.4 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.  
U, Y, X, DP are unchanged.  
CC is indeterminate.  
The string is filled with blanks.

**20.3.5 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:** U, Y, X, DP, A and B are 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.

**20.3.6 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).  
U, Y, X, DP and B are 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.

**20.4 Diskette File Functions**

The diskette file functions can be used in conjunction with the device dependent I/O functions (section 18.2) 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 XDOS 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 XDOS 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 17.1.4 and 17.3.1. 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 descriptor 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		

## 20.4.1 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 XDOS 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 0-1, \$30-\$31), 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 20.4.4) should be consulted for a full description of the allocation scheme and the reasons for arriving at this error.

B = 6 indicates that the file name supplied was illegal.

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.

U, Y, X and DP are 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.

#### 20.4.2 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 = 2 indicates that the file name in the old DFT is invalid.

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.

B = 6 indicates that the file name in the U, Y, X and DP are 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.

### **20.4.3 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 Alternate Memory Map of the EXORset if it is properly configured.

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 XDOS are guaranteed that memory exists since the memory was sized during XDOS initialization; however, programs loading beyond the end of contiguous memory known to XDOS or programs loading into the Alternate Memory Map 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.

#### **20.4.3.1 Load from main controller drive**

---

##### **20.4.3.1.1 Load in system available memory**

---

Uses program end address + 64 as stack if control is not to be passed to calling program. If region bit set, updates ENDUSS\$.

##### **20.4.3.1.2 Load in alternate map**

---

Checks that RAM memory exists in alternate map (whole program area); this memory may not be shared with current map. If control is not to be passed to calling program, disables XDOS interrupt vectors, uses EXORbug stack and toggles maps.

##### **20.4.3.1.3 Load anywhere in current map**

---

Disables XDOS interrupt vectors, uses EXORbug stack, transfers control to disk firmware.

#### **20.4.3.2 Load from alternate controller**

---

These functions, although feasible, are slower because

of cross map operations.

#### 20.4.3.2.1 Load in system available memory

---

Same as 10.1.1.1; the EXORbug stack is used to access the alternate drive but the stack pointer is restored at the end of the function to reflect the conditions under 10.1.1.1.

#### 20.4.3.2.2 Load in alternate map

---

Checks that memory in the alternate map (not shared with the current map) exists in the whole program loading area, uses EXORbug stack and executes alternate controller firmware, restores stack pointer if control is to be passed to the calling program, else, disables XDOS interrupt vectors and toggles maps.

#### 20.4.3.2.3 Load anywhere in current map

---

This is a special case: the program must not be loaded above \$E000 (alphanumeric display memory). The screen memory is used for cross map operations. During load time, the screen memory is disabled (screen is blank). Upon completion of the load operation, the screen is erased and the following message is displayed:

LOAD COMPLETE

The XDOS interrupt vectors are disabled and the stack pointer is initialized to point to the EXORbug stack. If control is to be passed to the loaded program, NO REGISTER INITIALIZATION IS PERFORMED.

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 X DOS (location \$1FFF) and below the last location of contiguous memory established during X DOS initialization. Programs loaded in this manner require an additional fifty bytes of memory beyond the last address loaded into by the program. The X DOS variable ENDUS\$ will be changed to reflect the last address loaded into by the program. The X DOS interrupt vector link will be unchanged to allow access to X DOS 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 Alternate Memory Map of the EXORset. The X DOS interrupt link 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 X DOS interrupt vector link will be unchanged. The only requirement placed on programs loading into the Alternate Memory Map is that the ending load address not be greater than \$FFFF. Otherwise, any memory locations (\$0000-FFFF) can be loaded into. It must be avoided to load a program in the monitor ram region, since it will destroy the monitor parameters and stack: That may cause the .LOAD function to blow up. 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 XDOS or for loading into discontiguous memory. As a result, the XDOS interrupt vector link is restored to point back into the debug monitor, making XDOS 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. It must be avoided to load a program in the monitor ram region, since it will destroy the monitor parameters and stack: That may cause the .LOAD function to blow up. 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 0-1, \$30-\$31), 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 Alternate

Memory Map in a system which is not properly configured.

B = 2 indicates that the file name supplied is illegal.

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 file 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 Alternate Memory Map of the EXORset properly 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 Alternate Memory Map 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 21.1.

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 XDOS (Bit 4 set to one in entry value of B), the stack pointer will contain the highest address loaded into (fifty bytes greater than the highest program location). If loading over the resident XDOS or into discontiguous memory (Bit 7 set to one in entry value of B) or into the Alternate Memory Map (Bit 5 set to one in entry value of B), the stack pointer will contain the address of the EXORbug stack area.

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.
U	Ending load address.
Y	Ending load address.
X	Starting load address.
DP	Zero.
A	Zero.
B	Zero.
CC	\$50 (F and I set, E, H, N, Z, V and C clear).

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. If the program is given control, the registers are initialized as above, except the condition code register: the F and I bits are always set, the others are indeterminate.

#### 20.4.4 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 XDOS 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 XDOS equate file, section 18.3.1.1, or section 21.3. 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 17.2).

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.

U, Y, X and DP are 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.

#### 20.4.5 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 XDOS 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 XDOS equate file, section 18.3.1.1, or section 21.3. 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 17.2).

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.

U, Y, X and DP are 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.

#### 20.4.6 Display system error message -- .MDERR

The .MDERR function displays on the system console one of the standard system error messages contained in the X DOS 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 21.4).

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 identifies 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 X DOS file name. Eleven bytes comprise an X DOS 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 X DOS 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 21.

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 FILE NAME
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

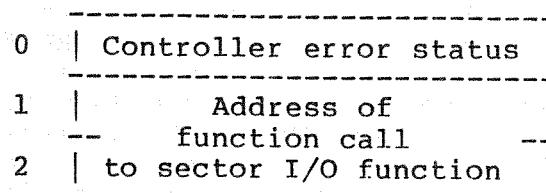
INDEX NUMBER	ERROR MESSAGE
29	** 22 BUFFER OVERFLOW
2A	** 23 CHECKSUM ERROR
2B	** 26 FILE IS WRITE PROTECTED
2C	** 43 INVALID DIRECTORY ENTRY NO. AT \8
2D	** 44 CANNOT DEALLOCATE ALL SPACE, DIRECTORY ENTRY EXISTS AT \8
2E	** 45 RECORD LENGTH TOO LARGE
2F	** 48 CHAIN OVERLAY DOES NOT EXIST
30	** 08 CHAIN ABORTED BY CONTROL-P 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:



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.

U, Y and DP are unchanged.

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 21.4 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 XDOS 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 XDOS 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).

## 20.5 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 XDOS commands and are available for user programs in order to extract file name or device specifications from the XDOS 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 XDOS command interpreter.

### 20.5.1 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 XDOS 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 XDOS 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 XDOS command interpreter in using the XDOS 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 XDOS 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.

U, Y, X and DP are 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 --

**20.5.2 Re-enter resident X DOS -- .MDENT**

The .MDENT function passes control from a calling program to the X DOS 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 been 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 X DOS was initialized via the resident debug monitor.

**EXIT CONDITIONS:** There is no return from this function; however, the following actions are performed:

The interrupt vector link is configured for the X DOS function handler.

The user SWI vector maintained by X DOS (SWI\$UV) is reset to point to an RTI instruction. The user program is no longer resident, thus user-defined SWI interrupt cannot be processed after X DOS regains control.

The end of user memory pointer, END\$US, is reset.

The command line buffer is initialized.

The version/revision numbers of X DOS 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, EXORbug is given control.

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.

**20.5.3 Reload X DOS 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 X DOS diskette since the last time X DOS 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 the EXORbug command "X DOS".

**ENTRY PARAMETERS:** A valid X DOS diskette must be ready in drive zero.

**EXIT CONDITIONS:** This function does not return to the calling program. A new copy of X DOS 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 17.6. Control is given to the X DOS command interpreter after X DOS has been initialized.

#### 20.5.4 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 4).

**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 21.4 describes the format of the error status word.

**EXIT CONDITIONS:** U, Y, X, DP, A and B are 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.

#### 20.5.5 Allocate user program memory -- .ALUSM

The .ALUSM function adjusts the X DOS 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 on 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:** U, Y, DP and A are 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 ENDOSS\$. The allocated memory block must reside completely above the address contained in ENDOSS\$.

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 ENDUSS 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 XDOS variable ENDUSS is unchanged if an error occurred. Otherwise, ENDUSS will contain the following: if the entry value of B was zero, ENDUSS will contain the entry value of the X register; if the entry value of B was one, ENDUSS will have been incremented by the entry value of the X register; and if the entry value of B was two, ENDUSS will contain the value of ENDSY\$.

#### 20.5.6 Issue next command -- .COMND

---

The .COMND function terminates the execution of a program and initializes the command line with the string passed as an argument. The operations are equivalent to the .MDENT function (see section 20.5.2), but the next command to perform is neither input from the console, nor read from the CHAIN intermediate file, but taken from the calling program. This function destroys the contents of locations \$2000 thru \$23FF, as well as the upper 200 bytes of system available memory. When used for program chaining, common data must then not reside in this area.

**ENTRY PARAMETERS:** X = address of the command string. The string must be terminated by a carriage return (\$0D). The maximum string length is 80 characters including the carriage return. Nulls (\$00), line-feeds (\$0A), DC1 (\$11), DC2 (\$12), DC3 (\$13) and DC4 (\$14) characters are ignored. If more than 79 characters are found, the string is truncated and a carriage return is forced into the command buffer.

**EXIT CONDITIONS:** The .COMND function is one of the few which does not return to the calling program. However, if the command line passed in entry parameter is valid, the control is given to the specified program as if it was called from the

console. The system parameters are handled by .MDENT (Section 20.5.2). If the supplied command is not found in the directory, the message

WHAT?

is displayed, the error type of the Error Status Word is set to \$80 and the control is given to the command interpreter.

## CHAPTER 21

---

### 21. 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 XDOS. 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 XDOS 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 3-16.

#### 21.1 Diskette Controller Errors

---

The diskette controller errors can be displayed in two forms depending on the phase XDOS 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 XDOS has been properly initialized, the diskette controller errors are identified by the text "PROM I/O ERROR". Control is returned to the XDOS command interpreter.

##### 21.1.1 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.
- Closing the door, waiting a little bit longer before entering the "XDOS" 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      This error status is returned when the track address has not been found after five attempts.
- E6      The diskette controller has been presented with a track-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 may come from 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 8) and/or FORMAT (Chapter 10) 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 2.8.8).

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 track 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 track zero; however, this problem rarely occurs. The E7 errors can occur if a user-written program accesses drive one 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 XDOS.SY is in error. The operating system cannot be loaded.
	The diskette probably is not an XDOS system diskette, or the system files have been moved from their original places.

- 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 XDOS 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 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 RIBs of the XDOS 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 2.8.8). This error may also occur if the door to drive zero is opened while initialization is in progress.
- EN A NMI has occurred and the XDOS NMI vector (NMI\$VC) was not initialized. This error may also occur after completion of the initialization.

- EQ An IRQ has occurred and the XDOS IRQ vector (IRQ\$VC) was not initialized. This error may also occur after completion of the initialization.
- EF A FIIRQ has occurred and the XDOS FIIRQ vector (FIR\$VC) was not initialized. This error may also occur after completion of the initialization.
- ES A SWI2 has occurred and the XDOS SWI2 vector (SW2\$VC) was not initialized. This error may also occur after completion of the initialization.
- EW A SWI3 has occurred and the XDOS SWI3 vector (SW3\$VC) was not initialized. This error may also occur after completion of the initialization.

### 21.1.2 Errors after initialization

---

If a diskette controller error is detected after XDOS 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 EXORbug; 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, 37, 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 tracks outward from the sector in error, repositioned back over the sector, and another five accesses attempted. Then, the head was positioned a maximum of five tracks inward from the sector in error, repositioned back

over the sector, and another five accesses attempted. If it fails again, the drive is restored, repositioned back over the sector, and another five accesses are attempted. A fifth retry is made by rocking the head five tracks outward. Lastly, five track inward rock is performed.

Occasionally, if the diskette in drive zero was changed without properly reinitializing the system, or if an XDOS 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.

## 21.2 Standard Command Errors

---

The following list contains all of the standard error messages than can be displayed by the XDOS 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 20.4) 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 18.3.1.20) 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 XDOS command interpreter, not by a command. It has no number identifying it. The second error message is only displayed if the XDOS 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, 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 XDOS 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 CONTROL-P KEY**

This message is displayed by the CHAIN command to indicate that the operator depressed the CTL-P 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 13).

**\*\* 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 DUMP command (Chapter 9) 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 14) can display this message if a memory-image file has an invalid RIB. The DUMP command (Chapter 9) 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.

The error code 23 is returned to the application program by the .OPEN function if the file format mode is specified and the device is read in search of an FDR, but no FDR is found.

The error code 23 is returned to the application program by the .OPEN function if the file format mode is specified and the device is read in search of an FDR, but the FDR header character is not found.

The error code 23 is returned to the application program by the .OPEN function if the file format mode is specified and the device is read in search of an FDR, but the FDR header character is found, but the FDR header character is not found.

The error code 23 is returned to the application program by the .OPEN function if the file format mode is specified and the device is read in search of an FDR, but the FDR header character is found, but the FDR header character is not found.

The error code 23 is returned to the application program by the .OPEN function if the file format mode is specified and the device is read in search of an FDR, but the FDR header character is found, but the FDR header character is not found.

The error code 23 is returned to the application program by the .OPEN function if the file format mode is specified and the device is read in search of an FDR, but the FDR header character is found, but the FDR header character is not found.

The error code 23 is returned to the application program by the .OPEN function if the file format mode is specified and the device is read in search of an FDR, but the FDR header character is found, but the FDR header character is not found.

The error code 23 is returned to the application program by the .OPEN function if the file format mode is specified and the device is read in search of an FDR, but the FDR header character is found, but the FDR header character is not found.

**\*\* 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 15) limits the use of the family indicator. Failure to do so may result in this error.

This error can also occur if a diskette function is called with an invalid or null file name in the DFT. Check that .PFNAM is called correctly, that the DFT initialization is correct.

**\*\* 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 XDOS-supported device. This usually occurs if the device name is mistyped. The valid device names for the I/O functions are CN, 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 occur (e.g., specifying units greater than 1 ? for diskette drives or units greater than 0 for other devices).

The COPY command (Chapter 5) will also accept the device name 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 one. 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 DUMP command (Chapter 9) should be used to correct the error.

## **\*\* 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 XDOS 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 17.2. The DUMP command (Chapter 9) 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 12) specified a physical line number whose value was larger than the total number of lines in the file.

**\*\* 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 (User Driver only).

**\*\* 38 INVALID LINE NUMBER OR RANGE**

This error message is not used by the XDOS system or commands. However, it resides in the system error file for compatibility with MDOS III.

**\*\* 39 LINE NUMBER ENTERED BEFORE SOURCE FILE**

This error message is not used by the XDOS system or commands. However, it resides in the system error file for compatibility with MDOS III.

**\*\* 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 XDOS 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 XDOS SWI (system function) handler with a function byte following the SWI instruction that is not defined. The operating system is reloaded when this error occurs.

**\*\* 48 CHAIN OVERLAY DOES NOT EXIST**

The CHAIN overlay's file name does not exist in the directory. The diskette in drive zero must be reconfigured via DOSGEN or BACKUP command in order to use the CHAIN feature.

**\*\* 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:

$$\frac{\text{IOCSBE} - \text{IOCSBS} + 1}{128} = \text{INTEGRAL NUMBER OF SECTORS}$$

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 XDOS commands. Thus, this message indicates a problem with the existing memory, or tampering with the memory map. The same is true for the XDOS-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 XDOS commands in section 1.1.

The ROLLOUT command (Chapter 16) 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).

### 21.3 Input/Ouput Function Errors

---

The XDOS 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 XDOS 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 XDOS 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 XDOS 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
1A	I\$IFNM	** 25 INVALID FILE NAME

## 21.4 System Error Status Word

Within the operating system's resident variables is a two-byte error status word. Each X DOS 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:

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.

## 21.5 Commands Affecting Error Status Word

All XDOS 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 XDOS 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 XDOS 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.

XDOS Function	Error Type	Error Message or Condition
XDOS Command Interpreter	\$80	WHAT?
.MDERR	\$FF	**INVALID MESSAGE mm AT nnnn
BACKUP	\$80 \$81 \$82 \$84 \$85 \$86 \$87	SOURCE FILE COPY ERROR OBJECT FILE CREATION COPY ERROR CANNOT DELETE DUPLICATE NAME DIRECTORY READ/WRITE ERROR SYSTEM SECTOR COPY ERROR SYNTAX ERROR Sector verify error
CHAIN	-	-
COPY	\$80 \$81	Response other than "Y" to overwrite question Verify error
DEL	\$80 \$81	<name> DOES NOT EXIST <name> IS PROTECTED
DIR	\$80 \$81 \$82	NO DIRECTORY ENTRY FOUND NO TERMINATOR FOUND IN FILE'S R.I.B. *NO SDWS*
DOSGEN	\$80 \$81	INVALID SECTOR NUMBER SECTOR xxxx LOCKED OUT
DUMP	\$80 \$81 \$82 \$83 \$84	SYNTAX ERROR MODE ERROR BOUNDARY ERROR INVALID SECTOR ADDRESS WHAT?
FORMAT	-	-
FREE	-	-
LIST	-	-
LOAD	-	-
NAME	-	-
MERGE	\$80	Response other than "Y" to overwrite question
ROLLOUT	-	-



## APPENDIX

### A. Cylinder-Sector/Physical Sector Conversion Table

---

The following table gives the physical sector numbers for the first sector of every cylinder.

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	Double sided diskettes have two recording surfaces. The top surface is called SFC0 and the bottom surface is called SFC1.

## Single-Sided 5.25" Disks

CYLINDER		PSN		CYLINDER		PSN	
DEC	HEX	DEC	HEX	DEC	HEX	DEC	HEX
00	00	000		20	14	140	
01	01	010		21	15	150	
02	02	020		22	16	160	
03	03	030		23	17	170	
04	04	040		24	18	180	
05	05	050		25	19	190	
06	06	060		26	1A	1A0	
07	07	070		27	1B	1B0	
08	08	080		28	1C	1C0	
09	09	090		29	1D	1D0	
10	0A	0A0		30	1E	1E0	
11	0B	0B0		31	1F	1F0	
12	0C	0C0		32	20	200	
13	0D	0D0		33	21	210	
14	0E	0E0		34	22	220	
15	0F	0F0		35	23	230	
16	10	100		36	24	240	
17	11	110		37	25	250	
18	12	120		38	26	260	
19	13	130		39	27	270	

## Double-Sided 5.25" Disks

CYLINDER		PSN		CYLINDER		PSN	
DEC	HEX	SFC0	SFC1	DEC	HEX	SFC0	SFC1
00	00	000	010	20	14	280	290
01	01	020	030	21	15	2A0	2B0
02	02	040	050	22	16	2C0	2D0
03	03	060	070	23	17	2E0	2F0
04	04	080	090	24	18	300	310
05	05	0A0	0B0	25	19	320	330
06	06	0C0	0D0	26	1A	340	350
07	07	0E0	0F0	27	1B	360	370
08	08	100	110	28	1C	380	390
09	09	120	130	29	1D	3A0	3B0
10	0A	140	150	30	1E	3C0	3D0
11	0B	160	170	31	1F	3E0	3F0
12	0C	180	190	32	20	400	410
13	0D	1A0	1B0	33	21	420	430
14	0E	1C0	1D0	34	22	440	450
15	0F	1E0	1F0	35	23	460	470
16	10	200	210	36	24	480	490
17	11	220	230	37	25	5A0	5B0
18	12	240	250	38	26	5C0	5D0
19	13	260	270	39	27	5E0	5F0

## Single-Sided 8" Disks

CYLINDER			PSN		
DEC	HEX	HEX	DEC	HEX	HEX
00	00	000	39	27	3F6
01	01	01A	40	28	410
02	02	034	41	29	42A
03	03	04E	42	2A	444
04	04	068	43	2B	45E
05	05	082	44	2C	478
06	06	09C	45	2D	492
07	07	0B6	46	2E	4AC
08	08	0D0	47	2F	4C6
09	09	0EA	48	30	4E0
10	0A	104	49	31	4FA
11	0B	11E	50	32	514
12	0C	138	51	33	52E
13	0D	152	52	34	548
14	0E	16C	53	35	562
15	0F	186	54	36	57C
16	10	1A0	55	37	596
17	11	1BA	56	38	5B0
18	12	1D4	57	39	5CA
19	13	1EE	58	3A	5E4
20	14	208	59	3B	5FE
21	15	222	60	3C	618
22	16	23C	61	3D	632
23	17	256	62	3E	64C
24	18	270	63	3F	666
25	19	28A	64	40	680
26	1A	2A4	65	41	69A
27	1B	2BE	66	42	6B4
28	1C	2D8	67	43	6CE
29	1D	2F2	68	44	6E8
30	1E	30C	69	45	702
31	1F	326	70	46	71C
32	20	340	71	47	736
33	21	35A	72	48	750
34	22	374	73	49	76A
35	23	38E	74	4A	784
36	24	3A8	75	4B	79E
37	25	3C2	76	4C	7B8
38	26	3DC			

## Double-Sided 8" Disks

CYLINDER		PSN		CYLINDER		PSN	
DEC	HEX	SFC0	SFC1	DEC	HEX	SFC0	SFC1
00	00	000	01A	39	27	7EC	806
01	01	034	04E	40	28	820	83A
02	02	068	082	41	29	854	86E
03	03	09C	0B6	42	2A	888	8A2
04	04	0D0	0EA	43	2B	8BC	8D6
05	05	104	11E	44	2C	8F0	90A
06	06	138	152	45	2D	924	93E
07	07	16C	186	46	2E	958	972
08	08	1A0	1BA	47	2F	98C	9A6
09	09	1D4	1EE	48	30	9C0	9DA
10	0A	208	222	49	31	9F4	A0E
11	0B	23C	256	50	32	A28	A42
12	0C	270	28A	51	33	A5C	A76
13	0D	2A4	2BE	52	34	A90	AAA
14	0E	2D8	2F2	53	35	AC4	ADE
15	0F	30C	326	54	36	AF8	B12
16	10	340	35A	55	37	B2C	B46
17	11	374	38E	56	38	B60	B7A
18	12	3A8	3C2	57	39	B94	BAE
19	13	3DC	3F6	58	3A	BC8	BE2
20	14	410	42A	59	3B	BFC	C16
21	15	444	45E	60	3C	C30	C4A
22	16	478	492	61	3D	C64	C7E
23	17	4AC	4C6	62	3E	C98	CB2
24	18	4E0	4FA	63	3F	CCC	CE6
25	19	514	52E	64	40	D00	D1A
26	1A	548	562	65	41	D34	D4E
27	1B	57C	596	66	42	D68	D82
28	1C	5B0	5CA	67	43	D9C	DB6
29	1D	5E4	5FE	68	44	DD0	DEA
30	1E	618	632	69	45	E04	E1E
31	1F	64C	666	70	46	E38	E52
32	20	680	69A	71	47	E6C	E86
33	21	6B4	6CE	72	48	EA0	EBA
34	22	6E8	702	73	49	ED4	EFE
35	23	71C	736	74	4A	F08	F22
36	24	750	76A	75	4B	F3C	F56
37	25	784	79E	76	4C	F70	F8A
38	26	7B8	7D2				

## APPENDIX

### B. ASCII Character Set

---

BITS 4 TO 6 --		0	1	2	3	4	5	6	7
B	0	NUL	DLE	SP	0	@	P	`	p
I	1	SOH	DC1	!	1	A	Q	a	q
T	2	STX	DC2	"	2	B	R	b	r
S	3	ETX	DC3	#	3	C	S	c	s
	4	EOT	DC4	\$	4	D	T	d	t
O	5	ENQ	NAK	%	5	E	U	e	u
	6	ACK	SYN	&	6	F	V	f	v
T	7	BEL	ETB	'	7	G	W	g	w
O	8	BS	CAN	(	8	H	X	h	x
	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. XDOS 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	CHAIN <command file> CHAIN N* CHAIN *	
5	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
6*	DEL [<file>] [;<options>]	S - System files Y - Yes, delete
7*	DIR [<file>] [;<options>]	A - Allocation information E - Entire entry L - Line printer S - System files

Chapter	Command Line	Options
8	DOSGEN [:<unit>] [;<options>]	T - Write/read surface test U - User diskette (minimum system files)
9	DUMP [<file>]	
10	FORMAT	
11	FREE [:<unit>] [;<options>]	L - Line printer
12	LIST <ASCII file>[,,<start>][,<end>] [;<options>]	F[mmm].[nn] - Page format H - Input heading L - Line printer N - Line numbers
13	LOAD [<memory-image file>] [;<options>]	null - Go to EXORbug null - Load above XDOS G - Load and go U - EXORset Alternate Memory Map V - Overlay XDOS; discontiguous memory (<string>) - Initialize command buffer
14	MERGE <file 1>[,<file 2>,...,<file n>],<destination file> [;<options>]	W - Overwrite <start address>
15*	NAME <old name>[,<new name>] [;<options>]	D - Delete protection N - Non-system file S - System file W - Write protection X - No protection
16	ROLLOUT [<memory-image file>] [;<options>]	null - Memory above XDOS D - Build file from scratch diskette U - EXORset Alternate 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 the EXORset's floppy disk drives 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 of the drive to be selected (zero or one). The starting sector must be between 0 and \$27F, inclusively.
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 \$280.
LSCTLN	\$0005	This byte contains the number of bytes to be read from the last sector during a read operation. This number must be between 1 and 128 (\$80), inclusively.
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 one-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 21. 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 was primarily indicating the type of diskette that is in a drive (MDOS III supports single and double sided diskettes). Since XDOS III uses single sided mini diskettes only, this byte is unused and the sign bit is always set to one (to indicate single sided diskette) for MDOS III program compatibility.

For all of the firmware entry points described below, the content of the registers is meaningless upon entry. Upon exit, the registers are unchanged. 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 bits set to one in the condition code register. No non-disk interrupt must occur during a disk controller routine execution. The routines also modify the interrupt vector link. The interrupt vector link, the interrupt masks and the original register's contents 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 XDOS 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.
FDINIT	\$E822	This entry point initializes the FDC. 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 data located at the address contained in CURADR to be written into bytes of NUMSCT sectors beginning with STRSCT of CURDRV. After NUMSCT sectors are written, 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 track 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 track containing STRSCT (see Appendix A). The only parameter changed is FDSTAT.

- |               |   |
|---------------|---|
| WRTEST \$E87B | This entry point causes the data located at the address contained in CURADR to be written into 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:

$$\text{PSN} = \text{STRSCT} + \text{NUMSCT} - \text{SCTCNT} - 1$$

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	\$EB00	This entry point exists for MDOS III compatibility. It was originally used to initialize the line printer PIA. In the EXORset, this is done by the EXORbug monitor at RESTART time. LPINIT will then return immediately to the calling program.
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 "EXORset User's Guide" should be consulted.



## APPENDIX

E. Mini-Diagnostic Facility

A mini-diagnostic routine is available in the EXORset diskette controller firmware. 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 EXORbug stack is acceptable). Then, the debug monitor command

EB98;G

will give control to the mini-diagnostic routine causing the FDC 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. Then 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 FDC 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.

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.

### Automatic configuration

---

The EXORset floppy disk firmware provides the facility to configure interactively the mini-diagnostic parameters, avoiding the user to deal with the addresses of the parameters. It will ask first for a drive number: the user must type in a single digit indicating the drive number to be tested. Then, the diagnostic will prompt with "S/C"; Typing "S" means that single execution is required, typing "C" tells

the firmware to enter the continuous execution mode. Lastly, "DES" is displayed: answering "Y" enables the destructive test, an "N" will cause the diagnostic to preserve the disk information during test. Configuration is now complete. The common part of the mini-diagnostic program is entered at CLRTOP (\$EB90) : counters are cleared and the test begin.

Mini-diagnostic parameters are configured as follow

Name	Address	Configuration
CURDRV	\$0000	Configured by the first answer: The drive number typed in.
STRSCT	\$0001	Set to zero. The test will apply on the whole disk.
NUMSCT	\$0003	Set to \$280. All sectors will be tested.
LDADDR	\$0020	Set to \$EBFD. This location of the disk driver contains the data \$E5 which is used by the RWTEST routine.
EXADDR	\$0022	Set to RDCRC address (\$E86F) for non-destructive test, to RWTEST (\$E872) for destructive test.
ONECON	\$0024	Cleared if answer to second prompt is "C". Set to \$FF if answer is "S".

Automatic configuration mode is entered by the EXORbug command

EAD2;G

Error processing of the mini-diagnostic program in automatic configuration mode is performed as in the manual mode.



## 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.

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 right edge (seeing the diskette from above with drive head hole at the bottom). This hole is located 1.25 inches from the top edge of 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.

## 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 two bytes 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 XDOS 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 XDOS.

```

* X DOS DIRECTORY HASHING FUNCTION
*
*HASH GETS HASH CODE IN RANGE 0-19 FOR FILE NAME
*
*ENTRY: X = POINTER TO 10 BYTE FILE NAME AND SUFFIX
*
*EXIT: A = HASH CODE VALUE -- RANGE 0-19, DECIMAL.
*      B AND X HAVE BEEN ALTERED
*
  HASH    LDA      #10          COUNT TEN CHARACTERS
         CLRB
  HASH1   PSHS     B,CC        CLR HASH VALUE AND CARRY
         LDB      0,X+
         SUBB     #$25        SAVE HASH VALUE AND CARRY
         BPL      HASH2       GET CHARACTER, BUMP INDEX
         CLRB
  HASH2   PULS     CC          UNIQUE 6 CHARACTER CODE
         ADCB     0,S+
         ROLB
         DECA
         BNE      HASH1       RESTORE CARRY
         RORB
         PSHS     B           UPDATE HASH CODE VALUE
         RORB
         RORB
         RORB
         ADDB     0,S+
         TFR      B,A
         ANDA     #%00011111  TALLY CHARACTER COUNTER
         CMPA     #19
         BLS      HASH3       LOOP TEN TIMES
         SUBA     #20
         CMPA     #9
         BHI      HASH3       ALL CHARACTERS INCLUDED
         ASRB
         ROLA
  HASH3   RTS      RESULT IN A-REG, EXIT

```

## APPENDIX

### H. XDOS Equate File Listing

This appendix contains a modified listing of the XDOS equate file. Only the pertinent parts the assembler outputs are shown. The leftmost column contains the value equated to the system symbol. The XDOS equate file can be assembled on a user's system if the EXORset M6809 Assembler is available. Note that the equate file is not line-numbered and cannot be assembled with a line-numbered program.

```
OPT      NOLIST
PAGE

*
* 6809 XDOS VERSION 4.1X -- SYSTEM EQUATE FILE -- 08/19/80
*
      SPC      3
*
* S K I P 2      M A C R O
*
* THE GENERATED BYTE IS A "COMPARE X 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 "BRANCH NEVER" OP-CODE IS GENERATED. NO REGISTER IS AFFECTED
* BY THIS OPERATION, CONDITION CODES ARE NOT ALTERED.
*
SKIPI  MACR
      FCB      $21
      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
PAGE

*
*
* 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
*
*
SPC 3
.RESRV EQU 0      RESERVE A DEVICE
.RELES EQU .RESRV+1 RELEASE A DEVICE
.OPEN EQU .RELES+1 OPEN A FILE
.CLOSE EQU .OPEN+1 CLOSE A FILE
.GETRC EQU .CLOSE+1 READ A RECORD
.PUTRC EQU .GETRC+1 WRITE A RECORD
.REWND EQU .PUTRC+1 POSITION TO BEGINNING OF FILE
.GETLS EQU .REWND+1 READ LOGICAL SECTOR
.PUTLS EQU .GETLS+1 WRITE LOGICAL SECTOR
.KEYIN EQU .PUTLS+1 CONSOLE INPUT
.DSPLY EQU .KEYIN+1 CONSOLE OUTPUT (TERM W/ CR)
.DSPLX EQU .DSPLY+1 CONSOLE OUTPUT (TERM W/ EOT)
.DSPLZ EQU .DSPLX+1 CONSOLE OUTPUT (TERM W/ EOT, NO CR/LF)
.CKBRK EQU .DSPLZ+1 CHECK CONSOLE FOR BREAK KEY
.DREAD EQU .CKBRK+1 EROM DISK READ
.DWRIT EQU .DREAD+1 EROM DISK WRITE
.MOVE EQU .DWRIT+1 MOVE A STRING
.CMPAR EQU .MOVE+1 COMPARE STRINGS
.STCHB EQU .CMPAR+1 STORE BLANKS
.STCHR EQU .STCHB+1 STORE CHARACTERS
.ALPHA EQU .STCHR+1 CHECK ALPHABETIC CHARACTER
.NUMD EQU .ALPHA+1 CHECK DECIMAL DIGIT
.ADDAM EQU .NUMD+1 INCREMENT MEMORY (DOUBLE BYTE) BY A
.SUBAM EQU .ADDAM+1 DECREMENT MEMORY (DOUBLE BYTE) BY A
.MMA EQU .SUBAM+1 MULTIPLY (SHIFT LEFT) MEMORY BY A
.DMA EQU .MMA+1 DIVIDE (SHIFT RIGHT) MEMORY BY A
.MDENT EQU .DMA+1 ENTER XDOS WITHOUT RELOADING
.LOAD EQU .MDENT+1 LOAD A FILE FROM DISK
.DIRSM EQU .LOAD+1 DIRECTORY SEARCH AND MODIFY
.PFNAM EQU .DIRSM+1 PROCESS FILE NAME
.ALUSM EQU .PFNAM+1 ALLOCATE USER MEMORY
.CHANG EQU .ALUSM+1 CHANGE NAME/ATTRIBUTES
.MDERR EQU .CHANG+1 XDOS ERROR MESSAGE HANDLER
.ALLOC EQU .MDERR+1 ALLOCATE DISK SPACE
.DEALC EQU .ALLOC+1 RETURN DISK SPACE
.EWORD EQU .DEALC+1 SET ERROR STATUS WORD FOR CHAIN
.TXBA EQU .EWORD+1 TRANSFER X TO B,A
.TBAX EQU .TXBA+1 TRANSFER B,A TO X
.XBAX EQU .TBAX+1 EXCHANGE B,A AND X
.ADBX EQU .XBAX+1 ADD B TO X
.ADAX EQU .ADBX+1 ADD A TO X
.ADBAX EQU .ADAX+1 ADD B,A TO X
.ADXBA EQU .ADBAX+1 ADD X TO B,A
.SUBX EQU .ADXBA+1 SUBTRACT B FROM X
.SUAX EQU .SUBX+1 SUBTRACT A FROM X

```

.SUBAX EQU	.SUAX+1	SUBTRACT B,A FROM X
.SUXBA EQU	.SUBAX+1	SUBTRACT X FROM B,A
.CPBAX EQU	.SUXBA+1	COMPARE B,A TO X
.ASRX EQU	.CPBAX+1	SHIFT X RIGHT (ARITHMETIC)
.ASLX EQU	.ASRX+1	SHIFT X LEFT (ARITHMETIC/LOGICAL)
.PSHX EQU	.ASLX+1	PUSH X ON S STACK
.PULX EQU	.PSHX+1	PULL X FROM S STACK
.PRINT EQU	.PULX+1	PRINT-TERMINATE WITH CR
.PRINX EQU	.PRINT+1	PRINT-TERMINATE WITH EOT
.GETFD EQU	.PRINX+1	READ FDR (RESIDENT XDOS ONLY)
.PUTFD EQU	.GETFD+1	WRITE FDR (RESIDENT XDOS ONLY)
.PUTEOF EQU	.PUTFD+1	WRITE EOF (RESIDENT XDOS ONLY)
.EREAD EQU	.PUTEOF+1	DISK READ W/ ERR RETN
.EWRIT EQU	.EREAD+1	DISK WRITE W/ ERR RETN
.MREAD EQU	.EWRIT+1	MULTIPLE SECTOR READ
.MWRIT EQU	.MREAD+1	MULTIPLE SECTOR WRITE
.MERED EQU	.MWRIT+1	MULTIPLE SECTOR READ W/ ERR RETURN
.MEWRT EQU	.MERED+1	MULTIPLE SECTOR WRITE W/ ERR RETURN
.BOOT EQU	.MEWRT+1	RELOAD XDOS
.COMND EQU	.BOOT+1	ISSUE NEXT COMMAND AND EXIT

PAGE

\*

\* A S C I I      C O N T R O L      C H A R A C T E R S

NULL EQU	0	NULL
SOH EQU	1	START OF HEADING
STX EQU	2	START OF TEXT
ETX EQU	3	END OF TEXT
EOT EQU	4	END OF TRANSMISSION
ENQ EQU	5	ENQUIRY (WRU - WHO ARE YOU)
ACK EQU	6	ACKNOWLEDGE
BEL EQU	7	BELL
BS EQU	8	BACKSPACE
HT EQU	9	HORIZONTAL TAB
LF EQU	\$A	LINE FEED
VT EQU	\$B	VERTICAL TAB
FF EQU	\$C	FORM FEED
CR EQU	\$D	CARRIAGE RETURN
SO EQU	\$E	SHIFT OUT
SI EQU	\$F	SHIFT IN
DLE EQU	\$10	DATA LINK ESCAPE
DC1 EQU	\$11	DEVICE CONTROL 1
DC2 EQU	\$12	DEVICE CONTROL 2
DC3 EQU	\$13	DEVICE CONTROL 4
DC4 EQU	\$14	DEVICE CONTROL 4
NAK EQU	\$15	NEGATIVE ACKNOWLEDGE
SYN EQU	\$16	SYNCHRONOUS IDLE
ETB EQU	\$17	END OF TRANSMISSION BLOCK
CAN EQU	\$18	CANCEL
EM EQU	\$19	END OF MEDIUM
SUB EQU	\$1A	SUBSTITUTE
ESC EQU	\$1B	ESCAPE
FS EQU	\$1C	FILE SEPARATOR
GS EQU	\$1D	GROUP SEPARATOR
RS EQU	\$1E	RECORD SEPARATOR
US EQU	\$1F	UNIT SEPARATOR
SPACE EQU	\$20	SPACE (WORD SEPARATOR)
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

```

*
SUF DLM EQU   ' .      SUFFIX DELIMETER
OPT DLM EQU   ' ;      OPTIONS DELIMETER
DRV DLM EQU   ' :      LOGICAL DRIVER DELIMETER
DEV DLM EQU   ' #      GENERIC DEVICE NAME DELIMETER
FAM DLM EQU   ' *      FAMILY NAME/SUFFIX DELIMETER
E$FATL EQU   1!<7    FATAL ERROR BIT
PAGE

*
* X D O S     S E C T O R     E Q U A T E S
*
SC$DID EQU   0      DISK ID PHYSICAL SECTOR NUMBER
SC$CAT EQU   1      CLUSTER ALLOCATION TABLE PHYSICAL SECT.
SC$LOK EQU   2      LOCKOUT CLUSTER TABLE PHYSICAL SECT.
SC$DIR EQU   3      DIRECTORY START PHYSICAL SECTOR NUMBER
SC$DRE EQU   $16    DIRECTORY END PHYSICAL SECTOR NUMBER
SC$BB EQU   $17    BOOT BLOCK PHYSICAL SECTOR NUMBER
SC$DOS EQU   $18    OPERATING SYSTEM PHYSICAL SECTOR NUMBER
SC$SIZ EQU   128   SECTOR SIZE IN BYTES
SC$CLS EQU   4      NUMBER OF SECTORS / CLUSTER
DFCLS$ 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
*
DID$ID EQU   0      OFFSET TO DISK ID (8 BYTES)
DID$VN EQU   8      OFFSET TO VERSION NUMBER (2 BYTES)
DID$RN EQU   10    OFFSET TO REVISION NUMBER (2 BYTES)
DID$DT EQU   12    OFFSET TO DATE (6 BYTES)
DID$NM EQU   18    OFFSET TO USER NAME (20 BYTES)
DID$RB EQU   38    OFFSET TO RIB ADDRESSES (20 BYTES)
DID$CP EQU   112   OFFSET TO DISK CONFIGURATION PARAMETERS ( 12 BYTES )
*
* D I R E C T O R Y     E N T R Y     O F F S E T S
*
DIR$NM EQU   0      OFFSET TO NAME (8 BYTES)
DIR$SX EQU   8      OFFSET TO SUFFIX (2 BYTES)
DIR$RB EQU   10    OFFSET TO RIB ADDRESS (2 BYTES)
DIR$AT EQU   12    OFFSET OF ATTRIBUTES (2 BYTES)
DIR$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
*
RIB$LB EQU   117   NUMBER OF BYTES IN LAST SECTOR
RIB$SL EQU   118   NUMBER OF SECTORS TO LOAD
RIB$LA EQU   120   MEMORY LOAD ADDRESS
RIB$SA EQU   122   START EXECUTION ADDRESS
PAGE
*
* 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
*
IOCSTA EQU   0      ERROR STATUS
IOCDTT EQU   1      DATA TRANSFER TYPE
IOCDBP EQU   2      DATA BUFFER POINTER
IOCDBS EQU   4      DATA BUFFER START ADDRESS
IOCDBE EQU   6      DATA BUFFER END ADDRESS
IOCGDW EQU   8      GENERIC DEVICE TYPE/CDB ADDRESS
IOCLUN EQU   10    LOGICAL UNIT NUMBER

```

IOCNAMEQU	11	FILE NAME
IOCMLSEQU	IOCNAME	MAXIMUM REFERENCED LSN
IOCSDWEQU	IOCNAME+2	CURRENT SEGMENT DESCRIPTOR WORD
IOCSSLSEQU	IOCNAME+4	1ST LOGICAL SECTOR OF CURRENT SEGMENT
IOCLSN EQU	IOCNAME+6	CURRENT LOGICAL SECTOR NUMBER
IOCSUF EQU	19	FILE NAME SUFFIX
IOCEEOF EQU	IOCSUF	LOGICAL END OF FILE
IOCРИB EQU	21	PHYSICAL DISK ADDRESS OF R.I.B.
IOCФDF EQU	23	FILE DESCRIPTOR FLAGS
IOCДEN EQU	27	DIRECTORY ENTRY NUMBER
IOCСBP EQU	29	SECTOR BUFFER POINTER/INITIAL SIZE
IOCСBS EQU	31	SECTOR BUFFER START ADDRESS
IOCСBE EQU	33	SECTOR BUFFER END ADDRESS
IOCСBI EQU	35	SECTOR BUFFER INTERNAL PTR
IOCBLNEQU	IOCСBI+2-IOCSTA	IOCBL LENGTH

\* U N I F I E D      I / O      E R R O R      S T A T U S E S

I\$NOER	EQU	0	NO ERRORS, NORMAL RETURN
I\$NODV	EQU	I\$NOER+1	NO SUCH DEVICE
I\$RESV	EQU	I\$NODV+1	DEVICE RESERVED ALREADY
I\$NORV	EQU	I\$RESV+1	DEVICE NOT RESERVED
I\$NRDY	EQU	I\$NORV+1	DEVICE NOT READY
I\$IVDV	EQU	I\$NRDY+1	INVALID DEVICE
I\$DUPE	EQU	I\$IVDV+1	DUPLICATE FILE NAME
I\$NONM	EQU	I\$DUPE+1	FILE NAME NOT FOUND
I\$CLOS	EQU	I\$NONM+1	INVALID OPEN/CLOSED FLAG
I\$EOF	EQU	I\$CLOS+1	END OF FILE
I\$FTYP	EQU	I\$EOF+1	INVALID FILE TYPE
I\$DTYP	EQU	I\$FTYP+1	INVALID DATA TRANSFER TYPE
I\$EOM	EQU	I\$DTYP+1	END OF MEDIA
I\$BUFO	EQU	I\$EOM+1	BUFFER OVERFLOW
I\$CKSM	EQU	I\$BUFO+1	CHECKSUM ERROR
I\$WRIT	EQU	I\$CKSM+1	FILE IS WRITE PROTECTED
I\$DELT	EQU	I\$WRIT+1	FILE IS DELETE PROTECTED
I\$RANG	EQU	I\$DELT+1	LOGICAL SECTOR NUMBER OUT OF RANGE
I\$FSPC	EQU	I\$RANG+1	NO DISK FILE SPACE AVAILABLE
I\$DSPC	EQU	I\$FSPC+1	NO DIRECTORY SPACE AVAILABLE
I\$SSPC	EQU	I\$DSPC+1	NO SEGMENT DESCRIPTOR SPACE AVAILABLE
I\$IDEN	EQU	I\$SSPC+1	INVALID DIR. ENTRY NO.
I\$RIB	EQU	I\$IDEN+1	INVALID RIB
I\$DEAL	EQU	I\$RIB+1	CAN'T DEALLOCATE ALL SPACE
I\$RECL	EQU	I\$DEAL+1	BINARY RECORD LENGTH TOO LARGE
I\$SECB	EQU	I\$RECL+1	SECTOR BUFFER SIZE ERROR
I\$IFNM	EQU	I\$SECB+1	INVALID FILE NAME
I\$RWND	EQU	I\$IFNM+1	DEVICE MAY NOT BE REWOUND

PAGE

\* \* X 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

```
MDOSS EQU $100      START OF XDOS ASECT
CBLFL$ EQU 80        COMMAND BUFFER LENGTH
CBUFF$ EQU MDOSS-CBLFL$-2 COMMAND BUFFER LOCATION
CBLFPL$ EQU CBUFP$+CBLFL$ COMMAND BUFFER SCAN POINTER
VERS$$ EQU MDOSS      VERSION #
REVSS$ EQU VERS$$+2 REVISION #
KYI$SV EQU REVSS$$+2 SAVE AREA FOR KEYIN$ VECTOR
```

```

ENDOS$ EQU KYI$SV+2 END OF XDOS
ENDUS$ EQU ENDOS$+2 END OF USER PROGRAM AREA
ENDSY$ EQU ENDUS$+2 END OF SYSTEM (MDOS) RAM
RIBBA$ EQU ENDSY$+4 RIB BUFFER ADDRESS
ENDRV$ EQU RIBBA$+2 END OF XDOS ROM VARIABLES
GDBA$ EQU ENDRV$+2 GENERIC DEVICE TABLE ADDREBSS
SYERR$ EQU GDBA$+2 SYSTEM ERROR STATUS WORD
SWI$SV EQU SYERR$+2 SWI VECTOR SAVE AREA
SWI$UV EQU SWI$SV+2 SWI USER VECTOR
CHFLG$ EQU SWI$UV+2 CHAIN FUNCTION FLAG WORD
SYIOCB EQU CHFLG$+2 SYSTEM CONSOLE IOCB
SYPOCB EQU SYIOCB+IOCBLN SYSTEM PRINTER IOCB
SYEOCB EQU SYPOCB+IOCBLN ERR MSG FILE
SW3$VC EQU SYEOCB+IOCBLN SOFTWARE INTERRUPT 3 VECTOR
SW2$VC EQU SW3$VC+2 SOFTWARE INTERRUPT 2 VECTOR
FIR$VC EQU SW2$VC+2 FAST INTERRUPT REQUEST VECTOR
IRQ$VC EQU FIR$VC+2 INTERRUPT REQUEST VECTOR
SWI$VC EQU IRQ$VC+2 SOFTWARE INTERRUPT VECTOR
NMI$VC EQU SWI$VC+2 NON MASKABLE INTERRUPT VECTOR
RES$VC EQU NMI$VC+2 RESTART VECTOR
VECT$ EQU RES$VC+1 TOP OF INTERRUPT VECTOR TABLE
SCTRKS EQU VECT$+1 NUMBER OF SECTORS/TRACK (S.S.) ( CURRENT MAP DRIVES )
SCMAX$ EQU SCTRKS+1 NUMBER OF USABLE SECTORS (S.S.) ( CURRENT MAP DRIVES )
SCTKDS EQU SCMAX$+2 NUMBER OF SECTORS/CYLINDER (D.S.) ( CURRENT MAP DRIVES )
SCMXD$ EQU SCTKDS+1 NUMBER OF USABLE SECTORS (D.S.) ( CURRENT MAP DRIVES )
SATRK$ EQU SCMXD$+2 NUMBER OF SECTORS/TRACK (S.S.) ( ALTERNATE MA$P DRIVES )
SAMAX$ EQU SATRK$+1 NUMBER OF USABLE SECTORS (S.S.) ( ALTERNATE MAP DRIVES )
SATKD$ EQU SAMAX$+2 NUMBER OF SECTORS/CYLINDER (D.S.) ( ALTERNATE MAP DRIVES )
SAMXD$ EQU SATKD$+1 NUMBER OF USABLE SECTORS (D.S.) ( ALTERNATE MAP DRIVES )
ADKPR$ EQU SAMXD$+2 ALTERNATE DISK PARAMETERS STORAGE
PAGE
*
*L O G I C A L   U N I T   N U M B E R - - B I T   D E F .
*
LU$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
*
DT$OPP EQU %00000000 OPEN UPDATE/INPUT
DT$OPI EQU %00000001 OPEN INPUT MODE
DT$OPO EQU %00000010 OPEN OUTPUT MODE
DT$OPU EQU %00000011 OPEN UPDATE MODE
DT$NFF EQU %00000100 NON-FILE FORMAT I/O FLAG
DT$TRU EQU %00001000 TRUNCATE FLAG
DT$CLS EQU %00010000 FILE OPEN/CLOSE FLAG
DT$SIO EQU %00100000 SECTOR I/O FLAG
DT$OUT EQU %01000000 OUTPUT TRANSFER TYPE
DT$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
*
FD$FMU EQU %00000000 USER DEFINED FORMAT (SECTOR I/O ONLY)
FD$FMD EQU %00000001 DEFAULT OBJECT REC'D FORMAT
FD$FML EQU %00000010 BINARY LOAD FORMAT
FD$FMB EQU %00000011 BINARY RECORD FORMAT
FD$FMA EQU %00000101 ASCII RECORD FORMAT
FD$FMC EQU %00000111 ASCII-CONVERTED-BINARY REC'D FORMAT
FD$CMP EQU %00001000 SPACE COMPRESSION FLAG
FD$CON EQU %00010000 CONTIGUOUS ALLOCATION FLAG
FD$SYS EQU %00100000 SYSTEM FILE ATTRIBUTE

```

```

FD$DEL EQU      $01000000 DELETE PROTECTION ATTRIBUTE
FD$WRT EQU      $10000000 WRITE PROTECTION ATTRIBUTE
*
* U N I F I E D   I / O   C O N T R O L   D E S C R I P T O R
*
*               B L O C K   O F F S E T S
*
CDBIOC EQU      0      ADDRESS OF IOCB
CDBSDA EQU      2      SOFTWARE DRIVER ADDRESS
CDBHAD EQU      4      HARDWARE ADDRESS
CDBDDF EQU      6      DEVICE DESCRIPTOR FLAGS
CDBVDT EQU      7      VALID DATA TYPE
CDBDDA EQU      8      DEVICE DEPENDENT AREA
CDBWST EQU     10      WORKING STORAGE
CDBLEN EQU      CDBWST+2 CDB LENGTH
*
* C D B D D F   --   B I T   D E F I N I T I O N S
*
DD$FMC EQU      $00000001 ASCII-CONVERTED-BINARY IS DEFAULT
DD$LOG EQU      $00000010 LOGICAL SECTOR I/O FLAG
DD$CNS EQU      $000000100 CONSOLE FLAG
DD$RWD EQU      $00001000 REWIND FLAG
DD$OCF EQU      $00010000 OPEN/CLOSE FLAG
DD$INP EQU      $00100000 INPUT DEVICE FLAG
DD$OUT EQU      $01000000 OUTPUT DEVICE FLAG
DD$RES EQU      $10000000 RESERVABLE DEVICE FLAG
*
* C D B V D T   --   B I T   D E F I N I T I O N S
*
VD$BIN EQU      $000000100 BINARY OBJECT FLAG
VD$GDB EQU      $000001000 TEMP GDB POINTER FLAG
VD$SDA EQU      $000100000 TEMP SDA POINTER FLAG
VD$NFF EQU      $100000000 NON-FILE FORMAT FLAG
*
* D E V I C E   D R I V E R   E N T R Y   O F F S E T S
*
DV$ON EQU      0      DEVICE ON OFFSET
DV$OFF EQU     3      DEVICE OFF OFFSET
DV$INT EQU     6      DEVICE INITIALIZATION OFFSET
DV$TRM EQU     9      DEVICE TERMINATION OFFSET
DV$IO EQU    12      DEVICE CHARACTER INPUT/OUTPUT OFFSET
DV$RWD EQU    15      DEVICE REWIND OFFSET
PAGE
*
* D I S K   E R O M   E Q U A T E S
*
CURDRV EQU      0      CURRENT DRIVE NUMBER
STRSCT EQU      1      STARTING PHYSICAL SECTOR NUMBER
NUMSCT EQU      3      NUMBER OF SECTORS TO OPERATE UPON
LSCTLN EQU      5      # OF BYTES TO READ FROM LAST SECTOR
CURADR EQU      6      MEMORY ADDRESS FOR DISK TRANSFER
FDSTAT EQU      8      DISK TRANSFER STATUS
SCTCNT EQU      $B      SECTOR COUNT USED IN DETERMINING ERRORS
SIDES EQU      $D      - ->SINGLE; + -> DOUBLE SIDED
FREQ EQU      $1A      TIMING CONSTANT VRS FREQUENCY
*
* E R O M   E N T R Y   P O I N T S
*
OSLOAD EQU      $E800    BOOTSTRAP THE OPERATING SYSTEM
FDINIT EQU      $E822    INITIALIZE THE FLOPPY DISK CONTROLLER

```

CHKERR EQU	\$E853	CHECK AND PRINT ERROR FROM FDSTAT
PRNTER EQU	\$E85A	PRINT ERROR FROM FDSTAT
READSC EQU	\$E869	READ SECTOR(S)
READPS EQU	\$E86D	READ PARTIAL SECTOR
RDCRC EQU	\$E86F	READ AND CHECK FOR CRC
RWTEST EQU	\$E872	WRITE/READ TEST
RESTOR EQU	\$E875	MOVE HEAD TO TRACK 0
SEEK EQU	\$E878	POSITION HEAD TO TRACK OF "STRSCT"
WRTEST EQU	\$E87B	WRITE TEST
WRDDAM EQU	\$E87E	WRITE DELETED DATA MARK
WRVERF EQU	\$E881	WRITE AND VERIFY CRC
WRITSC EQU	\$E884	WRITE SECTOR(S)
CLOCK EQU	\$E887	COMPUTE TIMING CONSTANTS

\* E R O M      E R R O R      E Q U A T E S  
 \*

ER\$CRC EQU	'1	DATA CRC ERROR
ER\$WRT EQU	'2	WRITE PROTECTED DISK
ER\$RDY EQU	'3	DISK NOT READY
ER\$MRK EQU	'4	DELETED DATA MARK ENCOUNTERED
ER\$TIM EQU	'5	TIMEOUT
ER\$DAD EQU	'6	INVALID DISK ADDRESS
ER\$SEK EQU	'7	SEEK ERROR
ER\$DMA EQU	'8	DATA ADDRESS MARK ERROR
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  
 \*

RETRY\$ EQU      5      RETRY COUNT FOR DISK READ/WRITE ERRORS

\* L I N E      P R I N T E R      E R O M      E Q U A T E S  
 \*

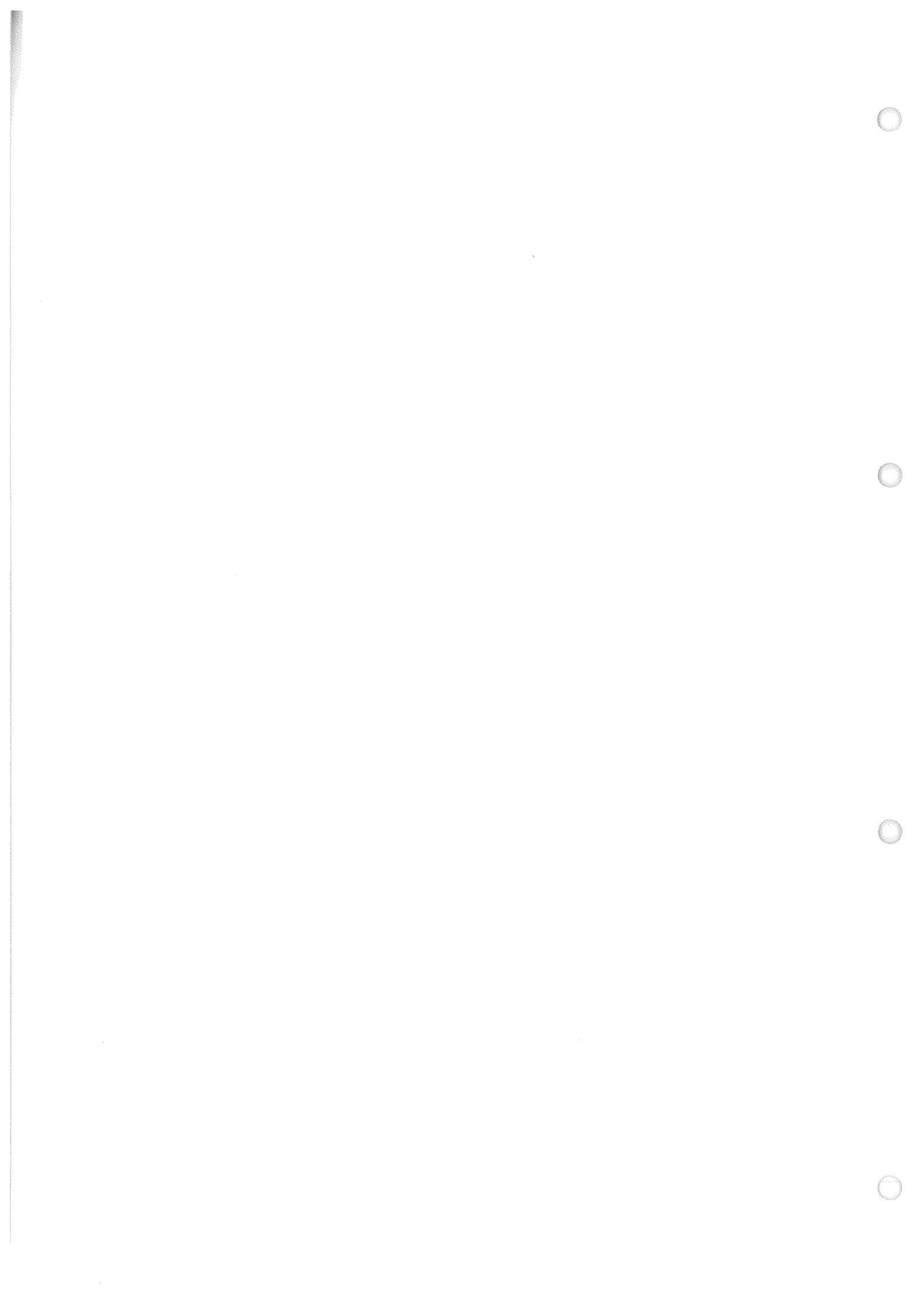
LPINIT EQU	\$EB00	INIT PRINTER PIA
LIST EQU	\$EBCC	PRINT CONTENTS OF 'A'
LDATA EQU	\$EBE4	PRINT STRING, CR/LF
LDATA1 EQU	\$EBF2	PRINT STRING, NO CR/LF
PAGE		

\*

\* E X O R B U G      E Q U A T E S      F O R      X D O S  
 \*      (INCLUDES ALL REFERENCES BUT ROLLOUT)

INCHNP EQU	\$F015	INPUT CHARACTER (NO PARITY)
OUTCH EQU	\$F018	OUTPUT ONE CHARACTER
OCHAR\$ EQU	\$F018	OUTPUT CHAR ROUTINE WITHOUT NULL PADDING
PCRLF EQU	\$F021	PRINT LF/CR
PDATA EQU	\$F024	PRINT STRING
MAID\$ EQU	\$F02D	EXORBUG ENTRY POINT
XLDA EQU	\$F030	CROSS MAP LOAD A-REGISTER
XSTA EQU	\$F033	CROSS MAP STORE A-REGISTER
XTOGL EQU	\$F036	CROSS MAP TOGGLE ROUTINE
ZAPBRK EQU	\$F039	CLEAR ALL BREAKPOINTS ROUTINE
CKBRK EQU	\$F045	CHECK BREAK ROUTINE
AECHO EQU	\$E714	INPUT CHARACTER ECHO FLAG (0=>ECHO)
ATOP\$ EQU	\$E72E	INTERRUPT VECTOR TABLE TOP ADDRESS
XSTAK\$ EQU	\$E703	EXORBUG STACK
XREG\$P EQU	\$E738	EXORBUG P-REG.
XREG\$S EQU	\$E73A	EXORBUG S-REG.
XREG\$U EQU	\$E73C	EXORBUG U-REG.
XREG\$Y EQU	\$E73E	EXORBUG Y-REG.
XREG\$X EQU	\$E740	EXORBUG X-REG.

```
XREG$D EQU    $E742      EXORBUG DP-REG.  
XREG$B EQU    $E743      EXORBUG B-REG.  
XREG$A EQU    $E744      EXORBUG A-REG.  
XREG$C EQU    $E745      EXORBUG C-REG.  
KEYBD$ EQU    $EF82      KEYBOARD PIA  
LINES$ EQU    $E74C      SEARCH/LOAD/VERIFY BUFFER  
XPEED$ EQU    $E736      TERMINAL SPEED FLAG  
CAS$ET EQU    $E72C      PUNCH ON FLAG  
*  
OPT      LIST,LLEN=120
```



## APPENDIX

### I. XDOS 4.00 Differences

---

The following appendix contains a description of the differences between XDOS 4.00 and MDOS 3.00.

1. A program accessing logical unit 1 without first using the system calls will have to be changed so that the read head is restored before the unit is accessed. XDOS restores both logical units 0 and 1 each time the system is initialized. However, if there is no diskette in drive 1 when booting XDOS, this drive is not restored. Seek errors may result when failing to care of this. System calls always restore the accessed drive when more than three recoverable controller errors occur.
2. BINEX, BLOKEDIT, ECHO, EMCOPY, EXBIN, PATCH and REPAIR commands have been evicted to preserve mini-diskette space.
3. The Alternate Memory Map of the EXORset has replaced the Dual Memory Map of the EXORciser II. Care should be taken when transferring MDOS programs that perform cross map operations.
4. Since the EXORbug monitor features hardware breakpoints, the ABORT or RESTART functions have no longer to be activated between a LOAD command and the "XDOS" or "E800;G" EXORbug commands.
5. In the system functions that use the file name provided in a DFT (.OPEN, .LOAD, .CHANG, .DIRSM), a check is made to ensure that the file name is legal. If not, an error status is returned (see 20.4).
6. The console reader (CR) and console punch (CP) standard generic device names are no longer supported since the EXORset system console does not include such peripheral devices.
7. The CHAIN command has been downgraded. Compilation operators are not supported.
8. An additional system call is available (.COMND) which is not implemented in MDOS III.



## APPENDIX

### J. 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.

	R E S R V	O P E N C	G E T R C	P U T R C	C L O S E	R E L E S	G E T L S	P U T L S	R E W N D
VALID CALL IOCB ENTRY	Y	Y	Y	N	Y	Y	N	N	Y
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-'1 (\$30-\$31)  
 IOCNAM = File name of existing file  
 IOCSUF = Suffix  
 IOCSBS = Sector buffer start  
 IOCSBE = Sector buffer end  
 IOCDBS = Data buffer start  
 IOCDBE = Data buffer end

R	O	G	P	C	R	G	P	R
E	P	E	U	L	E	E	U	E
S	E	T	T	O	L	T	T	W
R	N	R	R	S	E	L	L	N
V		C	C	E	S	S	S	D

**VALID CALL  
IOCB ENTRY**

 IOCSTA  
 IOCDTT  
 IOCDBP  
 IOCDBS  
 IOCDBE  
 IOCGDW  
 IOCLUN  
 IOCNAME/MLS  
 /SDW  
 /SLS  
 /LSN  
 IOCSUF/EOF  
 IOCRIB  
 IOCFFDF  
 -----  
 IOCDEN  
 IOCSBP/SIZ  
 IOCSBS  
 IOCSBE  
 IOCSBI

	Y	Y	N	Y	Y	Y	N	N	N
IOCSTA	.	.	.	.	.	.	.	.	.
IOCDTT	.	X	.	.	.	.	.	.	.
IOCDBP	.	.	.	.	.	.	.	.	.
IOCDBS	.	.	.	X	.	.	.	.	.
IOCDBE	.	.	.	X	.	.	.	.	.
IOCGDW	X	.	.	.	.	.	.	.	.
IOCLUN	X	.	.	.	.	.	.	.	.
IOCNAME/MLS	.	X	.	.	.	.	.	.	.
/SDW	.	X	.	.	.	.	.	.	.
/SLS	.	X	.	.	.	.	.	.	.
/LSN	.	X	.	.	.	.	.	.	.
IOCSUF/EOF	.	X	.	.	.	.	.	.	.
IOCRIB	.	.	.	.	.	.	.	.	.
IOCFFDF	.	X	.	.	.	.	.	.	.
-----	.	.	.	.	.	.	.	.	.
IOCDEN	.	.	.	.	.	.	.	.	.
IOCSBP/SIZ	.	X	.	.	.	.	.	.	.
IOCSBS	.	X	.	.	.	.	.	.	.
IOCSBE	.	X	.	.	.	.	.	.	.
IOCSBI	.	.	.	.	.	.	.	.	.

**Diskette Device -- Record Processing, Output (New file)**

```

IOCDTT = DT$CLS + DT$OPO
IOCGDW = DK
IOCLUN = '0-'1 ($30-$31)
IOCNAME = File name of new file
IOCSUF = Suffix
IOCFFDF = FD$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
  
```

	R	O	G	P	C	R	G	P	R
	E	P	E	U	L	E	E	U	E
	S	E	T	T	O	L	T	T	W
	R	N	R	R	S	E	L	L	N
	V	C	C	E	S	S	S	D	
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	.	.	.	.	.	.	.	.
IOCNAME/MLS	.	X	.	.	.	.	.	.	.
/SDW	.	X	.	.	.	.	.	.	.
/SLS	.	X	.	.	.	.	.	.	.
/LSN	.	X	.	.	.	.	.	.	.
IOCSUF/EOF	.	X	.	.	.	.	.	.	.
IOCRIPTB	.	.	.	.	.	.	.	.	.
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-'1 (\$30-\$31)  
 IOCNAME = 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

	R	O	G	P	C	R	G	P	R
	E	P	E	U	L	E	E	U	E
	S	E	T	T	O	L	T	T	W
	R	N	R	R	S	E	L	L	N
	V		C	C	E	S	S	S	D
VALID CALL IOCB ENTRY	Y	Y	Y	Y	Y	Y	N	N	Y
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	.	.	.	.	.	.	.
IOCРИB	.	.	.	.	.	.	.	.	.
IOCDFDF	.	.	.	.	.	.	.	.	.
----	.	.	.	.	.	.	.	.	.
IOCDEN	.	.	.	.	.	.	.	.	.
IOCSBP/SIZ	.	.	.	.	.	.	.	.	.
IOCSBS	.	X	.	.	.	.	.	.	.
IOCSBE	.	X	.	.	.	.	.	.	.
IOCSBI	.	.	.	.	.	.	.	.	.

## Diskette Device -- Record Processing, Update (Existing file)

IOCDTT = DT\$CLS + DT\$OPP  
 IOCGDW = DK  
 IOCLUN = '0-'1 (\$30-\$31)  
 IOCNAM = File name  
 IOCSUF = Suffix  
 IOCSBS = Sector buffer start  
 IOCSBE = Sector buffer end  
 IOCDBS = Data buffer start  
 IOCDBE = Data buffer end

	R E S R V	O P E N C	G E T R C	P U T R C	C L O S E	R E L E S	G E T L S	P U T L S	R E W N D
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--'1 (\$30-\$31)  
 IOCNAM = File name of existing file  
 IOCSUF = Suffix  
 IOCLSN = Starting logical sector number to be read  
 IOCSBS = Sector buffer start  
 IOCSBE = Sector buffer end

	R	O	G	P	C	R	G	P	R
	E	P	E	U	L	E	E	U	E
	S	E	T	T	O	L	T	T	W
	R	N	R	R	S	E	L	L	N
	V		C	C	E	S	S	S	D
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-'1 ($30-$31)
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

```

	R	O	G	P	C	R	G	P	R
	E	P	E	U	L	E	E	U	E
	S	E	T	T	O	L	T	T	W
	R	N	R	R	S	E	L	L	W
	V	C	C	E	S	S	S	D	N
VALID CALL	Y	Y	N	N	Y	Y	Y	Y	Y
IOCB ENTRY									
IOCSTA	.	.	.	.	.	.	.	.	.
IOCDTT	.	X	.	.	.	.	.	.	.
IOCDBP	.	.	.	.	.	.	.	.	.
IOCDBS	.	.	.	.	.	.	.	.	.
IOCDBE	.	.	.	.	.	.	.	.	.
IOCGDW	X	.	.	.	.	.	.	.	.
IOCLUN	X	.	.	.	.	.	.	.	.
IOCNAME/MLS	.	X	.	.	.	.	.	.	.
/SDW	.	X	.	.	.	.	.	.	.
/SLS	.	X	.	.	.	.	.	.	.
/LSN	.	X	.	.	.	.	X	X	.
IOCSUF/EOF	.	X	.	.	.	.	.	.	.
IOCRIPTB	.	.	.	.	.	.	.	.	.
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-'1 ($30-$31)
IOCNAME = 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

```

	R E S R V	O P E N C	G E T R C	P U T R C	C L O S E	R E S E S	G E T L S	P U T L S	R E W N D
VALID CALL IOCB ENTRY	Y	Y	N	N	Y	Y	Y	Y	Y
IOCSTA	.	.	.	.	.	.	.	.	.
IOCDTT	.	X	.	.	X	.	.	.	.
IOCDBP	.	.	.	.	.	.	.	.	.
IOCDBS	.	.	.	.	.	.	.	.	.
IOCDBE	.	.	.	.	.	.	.	.	.
IOCGDW	X	.	.	.	.	.	.	.	.
IOCLUN	X	.	.	.	.	.	.	.	.
IOCNAME/MLS	.	X	.	.	X	.	.	.	.
/SDW	.	X	.	.	.	.	.	.	.
/SLS	.	X	.	.	.	.	.	.	.
/LSN	.	X	.	.	.	.	X	X	.
IOCSUF/EOF	.	X	.	.	.	.	.	.	.
IOCRIB	.	.	.	.	.	.	.	.	.
IOCDFDF	.	.	.	.	.	.	.	.	.
----	.	.	.	.	.	.	.	.	.
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-'1 (\$30-\$31)  
 IOCNAME = File name of existing file  
 IOCSUF = Suffix  
 IOCLSN = Starting logical sector number  
 IOCSBS = Sector buffer start  
 IOCSBE = Sector buffer end  
 IOCSBI = Sector buffer end

	R E S R V	O P E N C	G E T R C	P U T R C	C L O S E	R E L E S	G E T L S	P U T L S	R E W N D
VALID CALL	Y	Y	Y	N	Y	Y	N	N	N
IOCB ENTRY									
IOCSTA	.	.	.	.	.	.	.	.	.
IOCDTT	.	X	.	.	.	.	.	.	.
IOCGDW	.	.	.	.	.	.	.	.	.
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  
 IOCLUN = '0 (\$30)  
 IOCFDF = FD\$FMA  
 IOCSUF = Display prompt if device is CN  
 IOCDBS = Data buffer start  
 IOCDBE = Data buffer end

	R E S R V	O P E N C	G E T R C	P U T R C	C T O S E	R E L E S	G E T L S	P U T L S	R E W N D
VALID CALL IOCB ENTRY	Y	Y	N	Y	Y	Y	N	N	N
IOCSTA	.	.	.	.	.	.	.	.	.
IOCDTT	.	X	.	.	.	.	.	.	.
IOCDBP	.	.	.	.	.	.	.	.	.
IOCDBS	.	.	.	X	X	.	.	.	.
IOCDBE	.	.	.	X	X	.	.	.	.
IOCGDW	X	.	.	.	.	.	.	.	.
IOCLUN	X	.	.	.	.	.	.	.	.
IOCNAME/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 or CN  
 IOCLUN = '0 (\$30)  
 IOCFDF = FD\$FMA  
 IOCDBS = Data buffer start  
 IOCDBE = Data buffer end

	R E S R V	O P E N C	G E T R C	P U T R C	C L O S E	R E L E S	G E T L S	P U T L S	R E W N D
VALID CALL IOCB ENTRY	N	Y	Y	N	Y	Y	N	N	N
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 = CDB address IOCLUN = \$70-\$79 IOCDBS = Data buffer  
 start (used for FDR processing)  
 IOCDBE = Data buffer end  
 IOCNAM = File name of existing file  
 IOCSUF = Suffix

	R E S R V	O P E N C	G E T R C	P U T R E	C T O S E	R E L S S	G E L S S	P U T L S	R E W N D
VALID CALL IOCB ENTRY	N	Y	N	Y	Y	Y	N	N	N
IOCSTA	.	.	.	.	.	.	.	.	.
IOCDTT	.	X	.	.	.	.	.	.	.
IOCDBP	.	.	.	.	.	.	.	.	.
IOCDBS	.	X	.	X	X	.	.	.	.
IOCDBE	.	X	.	X	X	.	.	.	.
IOCGDW	.	X	.	.	.	.	.	.	.
IOCLUN	.	X	.	.	.	.	.	.	.
IOCNAME/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 = CDB address IOCLUN = \$70-\$79 IOCDBS = Data buffer  
 start (used for FDR processing)  
 IOCDBE = Data buffer end  
 IOCNAME = File name  
 IOCSUF = Suffix  
 IOCFDF = FD\$FMA, FD\$FMB, FD\$FMC, or FD\$FMD (only)

