DISK OPERATING SYSTEM DOS. User's Guide

Version 2 (upgraded to 2.3)

March, 1977

Model Code No. 50216

DATAPOINT CORPORATION

The leader in dispersed data processing™

DISK OPERATING SYSTEM
DOS.

User's Guide

Version 2 (Upgraded to 2.3)

March, 1977

Model Code No. 50216

PREFACE

The purpose of this User's Guide is to provide the user of a Datapoint DOS that information required to generate a system, make effective use of the available commands, and to make user-written programs compatible with the DOS.

This manual applies to all Version 2.3 and above "dot-series" Disk Operating Systems, such as DOS.A, DOS.B, etc. This manual replaces the previous Version 2 User's Guide and Version 2 System's Guide.

TABLE OF CONTENTS

	page
1. INTRODUCTION	1-1
1.1 Hardware Support Required	1-1
1.2 Software Configurations Available	1-2
1.3 Program Compatibility	1-2
2. OPERATOR COMMANDS	2-1
3. EQUIPMENT CARE	3-1
3.1 Environment	3-1
3.2 Processor	3-2
3.3 Disks and Disk Drives	3-2
3.4 Other Peripherals	3-2
4. DISK FILES	4-1
4.1 File Names	4-1
4.2 File Creation	4-2
4.3 File Deletion	4-3
4.4 File Protection	4-3
5. SYSTEM GENERATION	5-1
5.1 Initial Generation	5-1
5.1.1 Formatting	5-1
5.1.2 Cassette System Generation	5-2
5.2 Partial Generation	5-3
5.3 UPGRADE/X	5-4
5.4 Scratch Disk Generation	5-5
5.5 Generation Cassettes and Emergencies	5-5
6. GENERAL COMMAND CHARACTERISTICS	6-1
6.1 General Command Format	6-1
6.2 Signon Messages	6-1
6.3 Common Error Messages	6-2
7. APP COMMAND	7-1
7.1 Purpose	7-1
7.2 Use	7-1
8. AUTO COMMAND	8-1
9. AUTOKEY COMMAND	9-1
9.1 Introduction to AUTOKEY	9-1
9.2 The Hardware Auto-Restart Facility	9-1
9.3 Automatic Program Execution Using AUTO	9-2

9.4 Auto-Restart Facilities Using AUTOKEY 9-2
9.5 A Simple Example 9-3
9.6 A More Complicated Example 9-4
9.7 Special Considerations 9-7
9.8 AUTOKEY and DATASHARE 9-7

10. BACKUP COMMAND 10-1
10.1 Purpose 10-1
10.2 Use 10-1
10.3 Mirror Image Copy 10-2
10.4 Reorganizing Files 10-3
10.4.1 Copying DOS to Output Disk 10-3
10.4.2 Deleting Named Files 10-3
10.4.3 Copying Named Files 10-4
10.5 Use of KEYBOARD and DISPLAY Keys 10-4
10.6 Error Messages 10-4
10.7 Reorganizing Files for Faster Processing 10-5
10.8 BACKUP with CHAIN 10-6
10.9 Clicks during Copying 10-6
10.10 Special Considerations for BACKUP 10-6

11. BLOKEDIT COMMAND 11-1
11.1 Purpose 11-1
11.2 File Descriptions 11-1
11.2.1 Command Statement Lines 11-2
11.2.2 Source File 11-3
11.2.3 New File 11-3
11.3 Using BLOKEDIT 11-4
11.4 Messages 11-4
11.4.1 Informative Messages 11-6
11.4.2 Fatal Errors 11-6
11.4.3 Selectively Fatal Errors 11-7

12. BUILD COMMAND 12-1
12.1 Purpose 12-1
12.2 Use 12-1
12.3 A Simple Example 12-2

13. CAT COMMAND 13-1
13.1 Purpose 13-1
13.2 Use 13-1

14. CHAIN COMMAND 14-1
14.1 Introduction 14-1
14.2 Tag Definition 14-2
14.3 Compilation Phase Directives 14-3
14.3.1 IF Directive 14-3
14.3.2 ELSE/XIF Directives 14-4

14.4 Tag Value Substitution 14-5
14.5 BEGIN/END Directives 14-6
14.6 ABORT Directives 14-7
14.7 Comments 14-8
14.8 Complex CHAIN Example 14-10
14.9 Resuming An Aborted CHAIN 14-13
14.10 Notes On Usage of CHAIN 14-14

15. CHANGE COMMAND 15-1

16. COPY COMMAND 16-1
16.1 Purpose 16-1
16.2 Use 16-1

17. DOSGEN COMMAND 17-1
17.1 Purpose 17-1
17.2 Use 17-1
17.3 Special Considerations 17-2

18. DUMP COMMAND 18-1
18.1 Purpose 18-1
18.2 Use 18-1
18.3 Informational Messages Provided 18-2
18.4 Level One Commands To DUMP 18-4
18.5 Level Two Commands To DUMP 18-4
18.6 Level Three Commands To DUMP 18-5
18.7 Level Four Commands To DUMP 18-6
18.8 Level Five Commands to DUMP 18-6
18.9 Error Messages 18-7

19. THE DUMP93X0 COMMAND 19-1
19.1 Use 19-1
19.2 The primary command handler 19-3
19.3 Using DUMP93X0 with a Local Printer 19-3
19.4 Screen Display format 19-5
19.5 The Screen Dump Command Handler 19-6
19.6 Cassette Operations 19-8
19.7 Drive Numbers 19-10
19.8 Error Messages 19-10

20. EDIT COMMAND 20-1
20.1 Introduction 20-1
20.2 Operation 20-1
20.2.1 DOS Initialization 20-1
20.2.2 Files 20-1
20.2.3 Parameter List 20-2
20.2.3.1 Margin Bell 20-2
20.2.3.2 Tab Key Character 20-3

20.2.3.3 Mode 20-3
20.2.3.4 Update 20-3
20.2.3.5 Key-click 20-4
20.2.4 Examples 20-4
20.2.5 Data Entry 20-5
20.2.6 Data Retrieval 20-6
20.2.7 EDITOR Command Format 20-6
20.3 Basic EDITOR Commands 20-7
20.4 Modification Commands 20-9
20.4.1 DELETE Command 20-9
20.4.2 MODIFY Command 20-10
20.4.2.1 Line Modification 20-10
20.4.2.2 Field Modification 20-11
20.5 File Search Commands 20-13
20.6 Miscellaneous Commands 20-14
20.7 Recovery Procedures 20-15
20.7.1 Bypassing Errors or End of File 20-16
20.7.2 File Recovery 20-16
20.8 Glossary 20-16
20.9 Command List 20-20

21. ENCODE/DECODE COMMANDS 21-1
21.1 Purpose 21-1
21.2 Use 21-1

22. FILES COMMAND 22-1
22.1 Command Description 22-1
22.2 Default Messages 22-2
22.3 File Descriptions 22-3
22.4 Error Messages 22-4

23. FIX COMMAND 23-1
23.1 Purpose 23-1
23.2 Operation 23-1
23.3 Commands 23-1
23.4 Error Messages 23-3

24. FREE COMMAND 24-1
24.1 Purpose 24-1
24.2 Use 24-1

25. INDEX COMMAND 25-1
25.1 Introductionn 25-1
25.2 System Requirements 25-1
25.3 Operation 25-2
25.3.1 Parameters 25-2
25.4 Choosing A Record Key 25-3
25.5 Preprocessing the File 25-4

25.5.1 Invoking Reformat 25-4
25.5.2 Considerations for Unattended Indexing 25-4
25.6 INDEX Messages 25-5
25.7 ISI File Formats 25-7
25.8 Examples of the Use of INDEX 25-9

26. THE INIT9370 COMMAND 26-1
26.1 Use 26-1
26.2 Error messages 26-2

27. KILL COMMAND 27-1

28. LIST COMMAND 28-1
28.1 Purpose 28-1
28.2 Parameters 28-1
28.3 INPUT File Specification 28-2
28.4 Starting Point 28-2
28.5 OUTPUT File Specification 28-3
28.6 Output Device 28-3
28.7 Output Format 28-4
28.8 Format Control 28-4
28.9 Operator Controls 28-5
28.10 Error Conditions 28-5

29. MANUAL COMMAND 29-1

30. MIN COMMAND 30-1
30.1 Purpose 30-1
30.2 Tape Formats 30-1
30.2.1 Single File Tapes 30-1
30.2.2 Double File Tapes 30-1
30.2.3 Multiple Numbered-File Tapes 30-2
30.2.4 Multiple Named-File Tapes 30-2
30.3 Parameters 30-2
30.3.1 Single File Tapes 30-2
30.3.2 Double File Tapes 30-4
30.3.3 Multiple Numbered-File Tapes 30-4
30.3.4 CTOS Tapes 30-5
30.3.5 MOUT With Directory Tapes 30-5
30.3.6 Options 30-6
30.4 Errors 30-8

31. MOUT COMMAND 31-1
31.1 Purpose 31-1
31.2 Parameters 31-1
31.3 Options 31-2
31.4 File Names 31-5
31.5 Writing 31-7

31.6 Verifying 31-8

32. NAME COMMAND 32-1

33. PUTIPL COMMAND 33-1

34. PUTVOLID COMMAND 34-1

35. REFORMAT COMMAND 35-1
35.1 Introduction 35-1
35.2 Operation 35-1
35.3 Output File Formats 35-3
35.4 Reasons for Reformatting 35-3
35.5 Reformat Messages 35-4
35.6 Text File Formats 35-7

36. THE REPAIR COMMAND 36-1
36.1 Applications of REPAIR 36-2
36.2 When to use REPAIR 36-2
36.3 Understanding REPAIR 36-3
36.3.1 Preliminary reading 36-4
36.4 Minimal Operator Interface 36-4
36.4.1 Executing REPAIR 36-4
36.4.2 Sign-on and drive number specification 36-5
36.4.3 Cylinder Lockout 36-5
36.4.4 Directory check monitor 36-6
36.4.5 Directory Errors 36-7
36.4.6 Retrieval Information Blocks check 36-7
36.4.7 Retrieval Information Blocks Errors 36-8
36.4.8 End of RIB check 36-8
36.4.9 Cluster allocation phase, Pass 1 36-9
36.4.10 Cluster allocation phase, Pass 2 36-9
36.4.11 Cluster allocation phase, pass 3 36-10
36.4.12 Cluster Allocation Conflicts 36-10
36.4.13 System Table Replacement 36-11
36.4.14 Termination of REPAIR 36-12
36.5 Medial Operator Interface 36-12
36.5.1 Executing REPAIR 36-13
36.5.2 Sign-on and drive number specification 36-14
36.5.3 Cylinder lockout 36-14
36.5.4 Directory check monitor 36-14
36.5.5 Directory errors 36-15
36.5.5.1 Delete errors 36-16
36.5.5.1.1 One entry deleted 36-17
36.5.5.1.2 Delete Incomplete 36-19
36.5.5.2 RIB Address Errors 36-20
36.5.5.2.1 RIB Address Invalid 36-20
36.5.5.2.2 RIB Addresses not equal 36-22

36.5.5.3 File protection not same 36-23
36.5.5.4 Name-Extension not equal 36-24
36.5.6 Retrieval Information Blocks check 36-25
36.5.7 Retrieval Information Blocks errors 36-26
36.5.7.1 A simple case 36-30
36.5.7.2 A Complex Case 36-31
36.5.8 End of RIB check 36-32
36.5.9 Cluster allocation phase, Pass 1 36-32
36.5.10 Cluster allocation phase, Pass 2 36-32
36.5.11 Cluster allocation phase, Pass 3 36-32
36.5.12 Cluster allocation conflicts 36-33
36.5.12.1 Cluster allocation phase, Pass 3 Messages 36-34
36.5.12.1.1 Left side of screen 36-34
36.5.12.1.2 Right side of screen 36-36
36.5.13 System table replacement 36-38
36.5.14 Termination of REPAIR 36-39
36.6 Cylinder Lockout with REPAIR 36-39
36.7 CAT errors and directory read/write errors 36-43
36.7.1 Cluster allocation table read errors 36-44
36.7.2 Cluster Allocation Table is destroyed 36-46
36.7.3 Cluster Allocation Table Copies Do Not Match 36-48
36.7.4 Directory Read Errors 36-49

37. REWIND COMMAND 37-1

38. SAPP COMMAND 38-1

39. SORT COMMAND 39-1
39.1 Introduction 39-1
39.2 General Information 39-1
39.3 Fundamental SORT Concepts 39-2
39.3.1 File Formats 39-2
39.3.2 The Key Options 39-3
39.3.3 How to Sort a File 39-4
39.4 The Other Options 39-4
39.4.1 Generalized Command Statement Format 39-4
39.4.2 Keys-overlapping and in Backwards Order 39-10
39.4.3 Collating Sequence File 39-10
39.4.4 Ascending and Descending sequences 39-12
39.4.5 Input/output File Format Options 39-12
39.4.6 Limited output format option 39-12
39.4.7 TAG file output format option 39-16
39.4.8 KEYTAG File Output Format Option 39-19
39.4.9 HARDCOPY output option 39-19
39.4.10 Primary/Secondary sorting considerations 39-20
39.4.11 Key File Drive Number 39-21
39.4.12 Disk space requirements 39-21
39.4.13 LINK into SORT from programs 39-22

39.5 The use of CHAIN with SORT 39-26
39.5.1 How to Set up a chain file for SORT 39-26
39.5.2 Naming a repetitive SORT procedure 39-27
39.5.3 Using CHAIN to cause a merge 39-27
39.6 SORT Execution-Time Messages 39-28

40. SUR COMMAND 40-1
40.1 Purpose 40-1
40.2 About Subdirectories 40-1
40.2.1 Creation of Subdirectories 40-2
40.2.2 Deletion of Subdirectories 40-2
40.2.3 Being "in a Subdirectory" 40-3
40.2.4 Scope of a File Name 40-3
40.2.5 About Subdirectory SYSTEM 40-4
40.2.6 Files vs. the User Being "in a Subdirectory" 40-4
40.2.7 Getting a File into a Subdirectory 40-5
40.3 Usage 40-5
40.3.1 Establishing a "Current Subdirectory" 40-5
40.3.2 Creating a Subdirectory 40-5
40.3.3 Deleting a Subdirectory 40-6
40.3.4 Renaming a Subdirectory 40-6
40.3.5 Displaying Subdirectories 40-6

41. UBOOT COMMAND 41-1

42. UTILITY/SYS 42-1

43. SYSTEM DESCRIPTION 43-1
43.1 System Philosophy 43-1
43.2 System Structure 43-1

44. SYSTEM STRUCTURE 44-1
44.1 Disk Structure 44-1
44.1.1 Introduction 44-1
44.1.2 Disk Space Management: CAT and Lockout CAT 44-2
44.1.3 Files: HDI, Directory Mapping Bytes, Directory, R44-3
44.1.4 Sector Identification 44-4
44.1.5 Addressing Byte Structures 44-5
44.1.5.1 PDA - Physical Disk Address 44-5
44.1.5.2 RIB Address/Protection 44-5
44.1.5.3 Segment Descriptor - used in RIB to define a segment. 44-6
44.1.5.4 Physical File Numer - used to access directory and HDI 44-6
44.2 Disk Data Formats 44-7
44.3 Memory Mapping 44-8
44.4 Memory Tables 44-9
44.4.1 Entry Point Tables 44-9

44.4.2 Logical File Table 44-9
44.5 Disk Overlays 44-11
44.6 The Command Interpreter 44-12

45. INTERRUPT HANDLING 45-1
45.1 Interrupt Mechanism 45-1
45.2 Interrupt Scheduler 45-1
45.3 Active Processes 45-3
45.4 Timing Considerations 45-4
45.5 DOS Interrupt Routines 45-5
45.5.1 SETI$ 45-5
45.5.2 CLRI$ 45-5
45.5.3 CS$ 45-5
45.5.4 TP$ 45-6
45.6 Programming Considerations 45-6
45.6.1 Background Code 45-6
45.6.2 Foreground Code 45-7

46. SYSTEM ROUTINES 46-1
46.1 Parameterization 46-1
46.2 Exit Conditions 46-1
46.3 Error Handling 46-2
46.4 Foreground Routines 46-2
46.4.1 CS$ - Change Process State 46-2
46.4.2 TP$ - Terminate Process 46-3
46.4.3 SETI$ - Initiate Foreground Process 46-3
46.4.4 CLRI$ - Terminate Foreground Process 46-3
46.5 Loader Routines 46-4
46.5.1 BOOT$ - Reload the Operating System 46-4
46.5.2 RUNX$ - Load and Run a File by Number 46-4
46.5.3 LOADX$ - Load a File by Number 46-5
46.5.4 INCHL - Increment the H and L Registers 46-5
46.5.5 DECHL - Decrement the H and L Registers 46-5
46.5.6 GETNCH - Get the Next Disk Buffer Byte 46-6
46.5.7 DR$ - Read a Sector into the Disk Buffer 46-6
46.5.8 DW$ - Write a Sector from the Disk Buffer 46-7
46.5.9 DSKWAT - Wait for Disk Ready 46-8
46.6 File Handling Routines 46-8
46.6.1 PREP$ - Open or Create a File 46-9
46.6.2 OPEN$ - Open an Existing File 46-10
46.6.3 LOAD$ - Load a File 46-10
46.6.4 RUN$ - Load and Run a File 46-11
46.6.5 CLOSE$ - Close a File 46-11
46.6.6 CHOP$ - Delete Space in a File 46-13
46.6.7 PROTE$ - Change the Protection on a File 46-13
46.6.8 POSIT$ - Position to a Record within a File 46-14
46.6.9 READ$ - Read a Record into the Buffer 46-14
46.6.10 WRITE$ - Write a Record from the Buffer 46-15

46.6.11 GET$ - Get the Next Buffer Character 46-16
46.6.12 GETR$ - Get an Indexed Buffer Character 46-16
46.6.13 PUT$ - Store into the Next Buffer Position 46-17
46.6.14 PUTR$ - Store into an Indexed Buffer Position 46-18
46.6.15 BSP$ - Backspace One Physical Sector 46-18
46.6.16 BLKTFR - Transfer a Block of Memory 46-18
46.6.17 TRAP$ - Set an Error Condition Trap 46-19
46.6.18 EXIT$ - Reload the Operating System 46-21
46.6.19 ERROR$ -- Reload the Operating System 46-22
46.6.20 WAIT$ -- DOS Wait-a-While "NOP" Routine 46-22
46.7 Keyboard and Display Routines 46-23
46.7.1 DEBUG$ - Enter the Debugging Tool 46-23
46.7.2 KEYIN$ - Obtain a Line from the Keyboard 46-26
46.7.3 DSPLY$ - Display a Line on the Screen 46-27

47. DOS FUNCTION FACILITY (DOSFNC) 47-1
47.1 FUNC1 - Retrieve Directory and C.A.T. Addresses 47-2
47.2 FUNC2 - Retrieve Directory Sector or Filename 47-5
47.3 FUNC3 - Retrieve R.I.B. Information 47-7
47.4 FUNC4 - Retrieve DOS Configuration Information 47-9
47.5 FUNC5 - Request Access to System Tables 47-10
47.6 FUNC6 - Keyboard / Display Interface Routines Function 47-11
47.7 FUNC7 - Test the Disk Buffer Memory 47-14
47.8 FUNC8 - Timed Pause 47-15
47.9 FUNC9 - Non-Sharable Resource Status Request 47-16
47.10 FUNC10 - Qualify for Execution in Fixed Partition 47-18
47.11 FUNC11 RAM Screen Loader 47-19
47.12 FUNC12 - Unassigned DOS Function 47-20
47.13 Overlay Loader (FUNC-13,14,15) 47-21
47.14 FUNC-13 Overlay Lookup By Name 47-23
47.15 FUNC-14 LOAD ABSOLUTE LIBRARY MEMBER 47-24
47.16 FUNC-15 RELOCATABLE LOADER 47-25

48. CASSETTE HANDLING ROUTINES 48-1
48.1 TPBOF$ - Position to the Beginning of a File 48-2
48.2 TPEOF$ - Position to the End of a File 48-2
48.3 TRW$ - Physically Rewind a Cassette 48-3
48.4 TBSP$ - Physically Backspace One 48-3
48.5 TWBLK$ - Write an Unformatted Block 48-3
48.6 TR$ - Read a Numeric CTOS Record 48-4
48.7 TREAD$ - TR$ and Wait for the Last Character 48-4
48.8 TW$ - Write a Numeric CTOS Record 48-5
48.9 TWRIT$ - TW$ and Wait for the Last Character 48-5
48.10 TFMR$ - Read the Next File Marker 48-6
48.11 TFMW$ - Write a File Marker Record 48-6
48.12 TTRAP$ - Set an Error Condition Trap 48-7
48.13 TWAIT$ - Wait for I/O Completion 48-8
48.14 TCHK$ - Get I/O Status 48-8

49. COMMAND INTERPRETER ROUTINES 49-1
49.1 CMDINT - Return & Scan MCR$ line 49-1
49.2 DOS$ - Return & Display Sign On 49-2
49.3 NXTCMD - Return & Say "READY" 49-2
49.4 CMDAGN - Return & Give Message 49-2
49.5 GETSYM - Get Next Symbol from MCR$ 49-3
49.6 GETCH - Get the Next Character from MCR$ 49-3
49.7 GETAEN - Get Auto-Execute Physical File Number 49-4
49.8 PUTAEN - Set or Clear a File to be Auto-Executed 49-4
49.9 GETLFB - Open the User-Specified Data File 49-5
49.10 PUTCHX - Store the Character in "A" 49-5
49.11 PUTCH - Alternate Version of PUTCHX 49-6
49.12 PUTNAM - Format a Filename from Directory 49-6
49.13 MOVSYM - Obtain the Symbol Scanned by GETSYM 49-7
49.14 GETDBA - Obtain Disk Controller Buffer Address 49-7
49.15 SCANFS - Scan Off File Specification 49-7
49.16 TCWAIT - Test controller memory & wait 49-8

50. USER SUPPORTED INPUT/OUTPUT 50-1

51. ERROR MESSAGES 51-1

52. ROUTINE ENTRY POINTS 52-1

53. DOS QUESTIONS AND ANSWERS 53-1

54. 5500 ROMGUIDE 54-1
54.1 System ROM Functions 54-1
54.1.1 Introduction 54-1
54.1.2 Startup Procedure 54-1
54.1.3 Saving the Machine State 54-2
54.1.4 Display Format 54-2
54.1.5 The Command Interpreter 54-3
54.1.6 Command Syntax 54-3
54.1.7 Input Command List 54-4
54.1.8 DEBUG Command Summary 54-8

Appendix A. DOS.A AND DOS.E A-1
A.1 Planning for DOS.A/DOS.E A-1
A.1.1 DOS.A Physical Configuration A-1
A.1.2 DOS.E Physical Configuration A-2
A.2 Disk Drives A-2
A.3 Disk Media A-2
A.4 Loading and unloading Disk Cartridges A-2
A.5 Switches and Indicators A-3
A.6 Care and Handling of Disk Cartridges A-4
A.7 Care and Maintenance of the 9350 Drives A-5

A.8 Head Crashes A-5
A.8.1 Prevention of Head Crashes A-6
A.8.2 Recognition of a Head Crash A-6
A.8.3 What to Do if You Have a Head Crash A-7
A.9 Preparing Disk Packs for Use A-7
A.10 Disk Organization under DOS.A/DOS.E A-8
A.10.1 Logical Drive Mapping A-8
A.10.2 Size of a Logical Drive A-8
A.10.3 Cluster Mapping A-8
A.10.4 Segments under DOS.A A-9
A.10.5 Maximum File Size A-9
A.10.6 Cluster Allocation Table and Directory A-10
A.11 Internal DOS Parameterization A-11
A.11.1 Physical Disk Address Format A-11
A.11.2 Hardware Address Structure A-11

Appendix B. DOS.B B-1
B.1 Planning for DOS.B B-1
B.2 File Storage Capacity under DOS.B B-1
B.3 Disk Drives B-2
B.4 Disk Media B-2
B.5 Loading and unloading Disk Packs B-2
B.5.1 Models 9370-9373 B-2
B.5.2 Model 9374/9375 B-3
B.6 Switches and indicators B-4
B.6.1 Models 9370-9373 B-4
B.6.1.1 Memorex Drives B-4
B.6.1.2 "Telex" Drives B-5
B.6.1.3 Common Features B-5
B.6.2 Model 9374/9375 B-6
B.7 Care and Handling of Disk Packs B-7
B.8 Care and Maintenance of the 9370 Drives B-7
B.9 Head Crashes B-9
B.10 Preparing Disk Packs for Use B-9
B.11 Disk Organization under DOS.B B-10
B.11.1 Logical Drive Mapping B-10
B.11.2 Size of a Logical Drive B-11
B.11.3 Cluster Mapping B-11
B.11.4 Segments under DOS.B B-12
B.11.5 Maximum File Size B-12
B.11.6 Cluster Allocation Table and Directory B-13
B.12 Internal DOS Parameterization B-14
B.12.1 Physical Disk Address Format B-14
B.12.2 Hardware Address Structure B-14

Appendix C. INTRODUCTION TO DOS.C C-1
C.1 Planning for DOS.C C-1
C.2 Performance of DOS.C C-2

C.3 Disk Drives C-3
C.4 Disk Media C-3
C.5 Loading and Unloading Diskettes C-3
C.6 Drive Numbering and Switches C-5
C.7 Care and Handling of Diskettes C-5
C.8 Preparing Diskettes for Use C-6
C.9 Suggested Disk Organization Techniques C-7
C.10 Disk Organization under DOS.C C-8
C.10.1 Radius Spiraling and Sector Skewing C-8
C.10.2 Size of a Diskette C-10
C.10.3 Cluster Mapping C-10
C.10.4 Segments under DOS.C C-11
C.10.5 Maximum File Size C-11
C.10.6 Cluster Allocation Table and Directory C-12
C.11 Internal DOS Parameterization C-13
C.11.1 Physical Disk Address Format C-13

Appendix D. DOS.D D-1
D.1 Planning for DOS.D D-1
D.2 File Storage Capacity under DOS.D D-1
D.3 Disk Drives D-2
D.4 Disk Media D-2
D.5 Disk Organization under DOS.D D-2
D.5.1 Logical Drive Mapping D-2
D.5.2 Size of a Logical Drive D-3
D.5.2.1 Models 9370-9373 D-3
D.5.2.2 Models 9374/9375 D-4
D.5.3 Cluster Mapping D-4
D.5.4 Segments under DOS.D D-4
D.5.5 Maximum File Size D-5
D.5.6 Cluster Allocation Table and Directory D-5
D.6 Internal DOS Parameterization D-6
D.6.1 Physical Disk Address Format D-6

Appendix E. COMPARSION CHART FOR DOS'S E-1

Appendix F. DISK DATA FORMATS F-1
F.1 Disk Data Formats F-1
F.2 OBJECT File Format for Disk F-1
F.3 Relocatable Code Formats F-3
F.3.1 Directory F-4
F.3.2 Program Identification F-5
F.3.3 Object Text F-5
F.3.3.1 Memory Location F-6
F.3.3.2 Absolute Text F-6
F.3.3.3 Complex Relocatable References F-7
F.3.3.4 Simple Relocatable References F-8
F.3.4 External Definitions F-10

F.3.5 External and Forward References (4096 maximum) F-11
F.3.6 Transfer Address F-11
F.4 Format of Library Files F-12
F.4.1 Directory F-12
F.4.2 Members F-13
F.4.3 Library Type Chart F-14
F.5 DATABUS Code File Format F-14
F.6 DATAFORM Data File Format F-14
F.7 MULTIFORM File Format F-15
F.8 TEXT File Format F-16
F.9 ISI File Format F-17
F.10 SORT TAG File Format F-19

CHAPTER 1. INTRODUCTION

Datapoint Corporation's Disk Operating System (DOS) is a comprehensive system of facilities for sophisticated data management.

DOS provides the operator with a powerful set of system commands by which the operator can control data movement and processing from the system console. These commands allow the system operator to accomplish things which could be substantially more difficult on other computing systems. Sorting a large file, for instance, can generally be accomplished in one single command line. In spite of the simplicity of operation, a wide range of features is provided.

To the programmer, DOS offers a set of facilities to simplify and generalize his task and file management problems. Concepts like dynamic disk space allocation allow programs to efficiently operate without regard to the amount of space required for the data files they are using. In addition, the disk file structure used by DOS allows for direct random access to data files. DOS also makes use of fully space-compressed text files.

These features, combined with the ability to support up to 200 million bytes of high-speed random access disk storage, provide a full range of data processing capabilities.

1.1 Hardware Support Required

The minimal configuration required to run DOS is a Datapoint 1100, 2200, or 5500 computer, with a minimum 16K of memory, and one (9350, 9370, or 9380 series) disk storage unit. For backup and support purposes, users with the Diskette 1100 computer are required to have at least one system with more than one diskette drive. Configurations based on the other processors can operate with only a single disk drive unit in conjunction with the integral tape cassettes, but for backup and system support purposes a two-drive system is a recommended minimum.

The two 5500-only DOS, DOS.D and DOS.E, support a minimum of two physical disk drives.

Users running single physical drive 9350, 9370, and 9380 configurations are supported under DOS.A, DOS.B, and DOS.C

respectively.

1.2 Software Configurations Available

DOS is provided in several different versions. Different versions are used depending upon the type of disk in use at an installation. Specific versions are indicated by a letter after a period in the name of DOS. As an example, the following versions of DOS are currently defined:

DOS.A -- Supports 9350 series disk drives on Datapoint 2200 and 5500 series computers.

DOS.B -- Supports 9370 series disk drives on Datapoint 2200 and 5500 series computers.

DOS.C -- Supports 9380 series disk drives on Datapoint 1100, 2200 and 5500 series computers.

DOS.D -- Supports 9370 series disk drives (with 16 buffer disk controller) on 48K Datapoint 5500 series computers.

DOS.E -- Supports 9350 series disk drives (with 16 buffer disk controller) on 48K Datapoint 5500 series computers.

1.3 Program Compatibility

This manual describes the compatible set of facilities available to the DOS user within the Disk Operating System. Programs written in any of the supported higher level languages (DATASHARE, RPG II, BASIC, etc.) will generally run unmodified on any of the DOS. Most programs written in assembler language will also run under any of the dot-series DOS, without reassembly.

Basically, in only a few cases will a program need to be changed when it is transferred from one DOS to another. The need for program modification will usually stem from one or more of the following types of situations, which should be avoided whenever possible:

1) Programs which make assumptions regarding the size of files. For example, programs originally written for the 9350 series disks might assume that the size of the biggest possible file could be expressed as four ASCII digits. Under DOS.D, this assumption is invalid since files under DOS.D may be over 38,000 data sectors long.

2) Programs which make assumptions regarding the physical structuring of the data on the disks. For example, each DOS allocates space on the disk in segments of different sizes, and places its system tables in different locations on the disk.

3) Programs which generate or modify physical disk addresses themselves. Since the disks are each organized somewhat differently to take advantage of the particular characteristics of the specific type of drives involved, the physical disk address formats naturally vary among different DOS.

4) Programs which rely upon other characteristics of a DOS which are not documented in this manual. A possible situation would be where a programmer might look at the values in the registers following the return from a system routine and determine, for instance, that some routine always seemed to return with the value "1" in one of the registers. If he then constructs his program in such a manner that it will not function correctly if the "1" is not present upon return from the routine, then he is likely to find that his program may not work properly on a different DOS.

All of the above situations, except for the first, will usually only occur in assembler language programs operating at the very lowest levels. Programmers who require this level of detailed knowledge about the DOS will find the information specific to each DOS, in the Appendix for the DOS they are using.

CHAPTER 2. OPERATOR COMMANDS

All Datapoint computers include, as a standard feature, an integral CRT display through which the internal computer communicate with the operator. The system console also includes a typewriter-style keyboard which the operator uses to communicate with the computer. The DOS is normally controlled by commands entered at this system console.

When DOS first becomes ready for commands, it displays a signon message on the CRT and says "READY". Upon completion of any job the DOS generally again displays "READY". Whenever the ready message is shown, the operator may key in a command, which will be displayed on the bottom line of the CRT as it is keyed in. While typing a command, the BACKSPACE key will erase one character for correction, and the CANCEL key will erase the entire line.

A command line specifies first what job is to be performed, then any disk files or special system directives, then options for the job. The command programs provided with DOS are described in this manual; the information that must be entered for each command is specified in the chapter about that command. A command line is always terminated with the ENTER key.

In general, a command line is entered as:

<field>,<field>,<field>,<field>;[options]

Each <field> indicates a DOS file name specification (see the Disk Files chapter) or possibly a special field such as a subdirectory name. The first <field> on the line always specifies the program that will be run. Special attention must be given to the separators between fields on the command line. The most common separators are space and comma. For readability the first two fields are usually separated by a space and subsequent fields are separated by a comma. A command then usually looks like:

SORT ACCTFILE,SRTFILE,:DR3;2-11

In this example the first field, the program to be executed, is "SORT". The second field is "ACCTFILE", the third is "SRTFILE", and the fourth is ":DR3". All of these fields provide information to the SORT program. A semi-colon (;) is a special separator which always separates <field> entries from [options]. In the above example the options field is "2-11". Slash (/) and colon

(:) are special separators used within a file name.

Aside from the separators noted above, most special characters (#,=,?,$,@,etc.) act as separators just like space or comma. In general, any character that is not a syntactically valid part of a file name will be interpreted as a field separator. The command example above could have been entered as:

```
SORT@ACCTFILE=SRTFILE$:DR3;2-11
```

The use of special characters is not recommended since the resulting command line is very confusing for human interpretation.

As already noted, the first field on the command line specifies the program to be executed. For any command this first field must be given, any other fields may or may not be needed for a particular command. The command program must be a loadable object file, loading above 01400, or the program load will fail and the DOS will simply return to "READY" condition. If the program specified to be run cannot be found, the DOS displays the message "WHAT?" and waits for another command. If desired, the program name specification can be preceded by an asterisk (*) or a colon (:), indicating the command is to be located in UTILITY/SYS in preference to a separate command file (See Command Interpreter section).

Fields on the command line are often order dependent. If a command is being used which accepts several fields, one of which is not wanted, skip that field by entering two separators with nothing between them.

```
SORT ACCTFILE,,:DR3;2-11
```

By using two commas, ":DR3" is recognized as the fourth field on the line, with the third field being null.

When the command line is discussed in this manual, the first field is called the "command"; subsequent fields before the semi-colon are called "<filespec>" or some similiar term; characters following the semi-colon are called "options" or "parameters".

CHAPTER 3. EQUIPMENT CARE

Computers, disk drives, printers, and other data processing equipment are delicate devices. They must be operated correctly and given a degree of care to continue to perform correctly. Datapoint prints "A Guide for Operating Datapoint Equipment", model code #60252, which gives detailed instructions on the operation of Datapoint equipment. It is recommended that any installation without trained computer operators obtain this manual.

3.1 Environment

Datapoint systems must be installed in an area with adequate air conditioning. Datapoint processors can stand a fairly wide range of temperatures, but disk drives should have a temperature range of 60 to 80 degrees F. (15.5 to 26.7 degrees C.). The temperature tolerance varies with the type of drive in use (diskette drives can stand a much wider temperature range) but the 60-80 degree range is safest. Humidity must be kept low enough to avoid condensation (below 80%) but high enough to avoid excessive static electricity problems.

The machine area must be reasonably clean and dust-free. Fanatic cleanliness is not necessary, but dust, cigarette ashes, spilled liquids, and so forth can seriously affect machine operation.

Processors and peripherals require fairly "clean" power to avoid erratic operation. Machine room power should be supplied from a completely separate transformer if possible. Be sure devices such as adding machines and power tools are not connected to the same power leads as computer equipment. The electric motors in these devices cause severe power line noise and will seriously affect machine operation. If necessary, isolation transformers are available to supply clean power for Datapoint equipment.

3.2 Processor

The only user maintenance on the processor is to dust and clean the cabinet, CRT screen, and keyboard occassionally and to clean the cassette decks. The cassette decks are especially sensitive to grime: dirty decks can cause read/write errors and can even destroy tapes. The decks are cleaned in the same way audio cassette decks are cleaned. Use tape head cleaner and a cotton swab to clean the tape heads and capstans; use a dry, lint-free cloth or swab to clean the pinch rollers. The cassette decks should be cleaned as necessary depending on use; normally every two or three months, as often as weekly if the decks get very heavy use.

Be sure the ventilation slots on the top and rear of the processor are never blocked, as impeded air flow will cause overheating.

3.3 Disks and Disk Drives

Be sure all operators know how to insert and remove disks in the disk drives. Disks must be stored properly in an environment similar to that for the equipment. Consult the appendices of this manual, or the Guide for Operating Datapoint Equipment, or the Datapoint Product Specifications (green sheets) for details on disk handling.

The disk drives must not be subjected to bumps or jolts or head misalignment can occur. Physical location of the drives must allow adequate air circulation for cooling purposes.

3.4 Other Peripherals

All peripherals should be dusted occasionally in keeping with the necessary environment cleanliness. Aside from printers, most Datapoint peripherals require practically no user maintenance. For any necessary care, consult the Guide for Operating Datapoint Equipment, the green sheets, or your Datapoint service representative.

Printer ribbons must be changed periodically to maintain print quality. Cloth ribbons left in use for too long can disintegrate, requiring a very messy clean-up of inky lint when the ribbon is finally changed, so check the ribbon occasionally. To avoid paper jams on printers, be sure the paper is aligned correctly when loaded, and be sure the paper has a free path into the printer and as it emerges to the paper tray.

CHAPTER 4. DISK FILES

On all DOS-supported disks, information is stored in sectors, each of which contains 256 bytes of information. Sectors containing related information are organized in a single structured group called a file. All information on a disk will generally be organized in files, except for certain system tables.

4.1 File Names

From the console, files are identified by a NAME, EXTENSION, and LOGICAL DRIVE NUMBER. The NAME consists of up to eight alphanumeric characters (no special characters). Typical file names would include:

```
EDIT      PAYROLL
EMPLOYEE  JUL1075
23NOV76   X1
```

The EXTENSION must start with a letter and may be followed by up to two alphanumeric characters. If an extension is used in a file name, it is separated from the NAME by a slash (/). The extension further identifies the file and usually indicates the type of information contained in the file. A "TXT" extension means text and usually implies data or program source code. "ABS" implies program object code (absolute code) loadable by the system loader. "CMD" implies an object code file to be used as a command program from the system console. Other common extensions are: REL, ISI, DBC, OVn, SYS, PRT, BAS, and LEX.

The LOGICAL DRIVE NUMBER specifies on which logical drive the file is (or will be) located. The drive specification is identified by a leading colon (:) and has the form ":DRn" or ":Dn" or ":<volid>". When the ":DRn" or ":Dn" forms are used, the "n" is a number indicating the logical drive number as assigned at system installation. The ":<volid>" form allows logical volume identification, regardless of the physical drive on which the disk is located. "<volid>" is an eight character identifier placed on a disk by the PUTVOLID program.

The complete form of a file name is thus

```
NAME/EXTENSION:DRIVE
```

When a file name is entered as part of a command, all three parts of the name are not usually needed, though they can be specified. The presence or absence of a part of the file name is determined by the special separators "/" and ":". Syntactically correct file name entries are:

```
NAME/ABS:DR0    /ABS:DR1
NAME/REL        /TXT
NAME:D0         :D2
NAME            NAME:DOSD1
```

If a portion of the file name is not used, DOS applies default values; the default value used depends on the location of the name on the command line, and on the command in use.

The first field on any command line is the command program to be run. For this field, a NAME must be given, the default extension is CMD, and the default drive is any drive. (An "any drive" default usually means a search of all drives, starting with drive 0). If the command name is preceeded by an asterisk (*) or a colon (:), the default extension and all-drive search do not apply, as the leading character indicates the given name is to be located as a member of UTILITY/SYS (an "absolute library"), rather than searched for as a file.

The default values for file names given as parameters to a command are described separately for each command.

4.2 File Creation

Files are always created implicitly. That is, the operator never specifically instructs the system to create a given file. Any command that writes to an output file will write into an existing file or will automatically create a new file if necessary.

A file to be created will be created on the drive specified in its file name field or specified in default values applied to its name. When a file is being created on a specific drive, files with the same name and extension on other drives are unaffected. If no drive is specified in the name or by default, the file is created on any drive which has free space, the search for available space starting on drive 0. "Available space" means one free space in the drive's directory, in which to place the name of the new file, and at least one cluster of free space on the disk, in which to place the data the file will contain. (A "cluster" is the smallest unit of disk space that can be assigned to a file;

clusters are defined in the chapter on System Structure).

4.3 File Deletion

Deletion of a file is performed explicitly by operator command, using the KILL command described later. No other programs delete an existing file, although procedures such as system generation and backup naturally destroy all files on the output disk.

4.4 File Protection

DOS files can be given three types of protection: write protection, delete protection, and no protection. If a file is write protected, it can be neither written upon nor deleted. If a file is delete protected it cannot be deleted, although it can be written over, effectively destroying any data previously in it. If a file has no protection it can of course be modified in any manner. The CHANGE command is used to set the protection of a file.

CHAPTER 5. SYSTEM GENERATION

Before a disk can be used with DOS it must first be prepared by writing onto it basic system tables. Also, a surface verification must be performed so any bad areas of the disk surface will not be used. On a new installation, the system utility programs must be placed onto the disk for use. All these operations constitute system generation.

5.1 Initial Generation

Datapoint distributes DOS in two forms: as a set of cassette tapes or as a completely generated disk. Users who receive the complete disks need not perform the cassette generation described below, as it has already been performed on their disk. Anyone requiring additional working disks should generate them as outlined in Scratch Disk Preparation.

5.1.1 Formatting

Before a disk can be written or read on any drive, it must be appropriately formatted. Cartridge disks for use on Datapoint drives (9350 series) require no formatting because they use hardware formatting -- the sector formatting is inherent to the disk. Datapoint diskettes (9380 series) are formatted when received and do not require a special formatting process before they can be used. The mass storage disks (9370 series) require a special formatting process before they can be used.

The first tape of the DOS generation cassettes for mass storage operating systems (DOS.B and DOS.D) is a formatting program. Simply insert the cassette in the rear cassette deck and depress RESTART (on a 5500 processor, RUN must be depressed simultaneously). The tape will rewind and then load the formatting program INIT9370. This program will ask for a specific physical (not logical) drive number containing the disk to be formatted. After receiving a reply, the program will ask if the operator is certain the drive number is correct and the disk in it is scratch, since formatting destroys any information previously on the disk. Formatting will then proceed. When finished the program will display a message indicating the pack is completely formatted.

For additional information on the formatting program, see the chapter on INIT9370.

5.1.2 Cassette System Generation

The first tape of the DOS generation cassettes (second tape, for DOS.B and DOS.D) is the actual generation cassette. To use this cassette load it into the rear cassette deck and depress RESTART (on 5500 processors, RUN must be depressed simultaneously). The tape will rewind and then load the DOS generation program. Loading takes about a minute. When the program has loaded it will display a sign-on message and ask what logical drive is to be generated. The drive specified must be on-line with a ready disk in it.

Following drive selection the program will ask if a full generation is desired. To get a full DOS generation, answer Y; for a partial gen (useful only for upgrades from an older version DOS) answer N. Partial generation is described below. Following selection of full generation, the program will ask to be sure the disk in the selected drive is scratch, containing no valuable files that would be destroyed by generation.

After the verification question, the program performs a surface test on the cylinders used by DOS for its system tables and operating files. If this test fails, the disk is considered unusable and error messages will so indicate. After a short pause for the above test, the program will ask if any cylinders are to be locked out. The normal answer to this question is N, since locked-out cylinders cannot be used by DOS. If it is desired to lock out any cylinders for special use, consult the DOSGEN chapter for a description of cylinder lockout.

The next step in system generation is a quick surface verification of the entire disk. The program clicks once for each cylinder tested and passed. If an error is encountered, the program displays the cylinder number in which the error occurred, beeps, and flags the cylinder in the Lockout CAT so the DOS will not use it.

Following surface verification the basic system tables are built on disk and the system programs are loaded from the tape. Programs loaded are SYSTEM0/SYS - SYSTEM7/SYS, CAT/CMD, MIN/CMD, and UBOOT/CMD.

NOTE: For initial generation of mass storage disks, be sure to repeat the above procedure twice, once for each logical drive

on the disk.

After loading the system programs, system generation is complete except for loading utilites, and the new DOS is brought up ready for commands. As soon as the system is ready (easy to tell since the message on the CRT is "READY") enter the command UBOOT to produce a boot tape for the DOS. UBOOT will ask for a blank tape in the front cassette deck and will then write and verify a boot block on that tape. It is wise to make at least two boot tapes at this time, since the boot tape is the only way to start up DOS. Any time it is necessary to start DOS (after the processor has been turned off, after loading a different set of disks, etc.) simply place the boot tape in the rear deck and depress RESTART (and RUN on 5500's) to boot the operating system.

To completely finish system generation, the system programs and utilities must be loaded. These files are contained on the second and third tapes of the system generation cassettes (third and fourth tapes for DOS.B and DOS.D). To load the commands simply place each cassette in turn into the front cassette deck and enter the command

```
MIN;AO:Dn
```

where n is the drive number being generated. When the files on these two tapes have been copied to disk, generation is finished.

The generation cassettes for DOS.C include a fourth tape of system commands, containing all the programs in UTILITY/SYS (see the appropriate chapter in this manual) as separate files. These files are provided as a convenience so that only desired programs can be placed on a system diskette, leaving free space on the diskette for other use.

5.2 Partial Generation

The DOS generation tape program has an option to perform a partial generation for purposes of upgrading an older version of DOS to the present version. To use partial generation load the gen tape and specify the drive to be generated. When the program asks if a full generation is desired answer N. The program will ask a couple of verification questions to be sure it should just replace the system and command files, and will then do so.

During partial generation, new system tables are built on the disk being upgraded and the eight system files SYSTEM0/SYS through SYSTEM7/SYS are replaced by new files from the tape. The old

utility programs must be deleted and new programs loaded from tape before partial generation is complete.

When performing a partial generation on a DOS. 1.1, 2.1, or 2.2 disk, it will be necessary to replace the old MIN/CMD with the new command from the generation tape before the utilities tapes can be loaded. (The old MIN cannot recognize the file format of UTILITY/SYS.) To replace MIN, load the generation cassette in the front deck and run MIN (the old command already on disk). MIN will identify the tape as "CTOS SYSTEM TAPE FORMAT" and will scan the tape to find the CTOS catalog. When the catalog is located, the files on the tape will be displayed and MIN will ask

LOAD B?

Skip the file named B by answering "N", skip CAT in the same manner, then answer "Y" to load MIN. The program will ask for a DOS file name; the name given should be "MIN/CMD". MIN will ask to be sure the existing command should be overwritten, answer "Y" to the OVERWRITE? question. Once MIN/CMD has been loaded, enter an asterisk to end the program when it asks if file number 20 should be loaded.

After MIN/CMD has been replaced, use the new MIN to load the utility tapes in the normal manner.

Following a partial generation, it is a good idea to BACKUP the upgraded disk with reorganization. The reorganization removes any fragmentation in system files and allows an operator to easily delete undesired old files. Until the old command files have been deleted, be sure to enter a leading * on each command so as to use the new utilities from UTILITY/SYS.

Partial generation is not valid between some versions of DOS (notably Version 1 DOS.B and any newer version). Check with your Datapoint System Engineer before attempting an upgrade by partial generation.

5.3 UPGRADE/X

A disk-based upgrade facility is available in a file called UPGRADE/X, X being the letter specification of the DOS in use. UPGRADE is a standard text file to be used as a chain procedure by the command

```
CHAIN UPGRADE/X;OUTPUT=:Dn
```

where n is the drive number containing the disk to be upgraded.

The UPGRADE procedure copies the eight system files from the new version disk (which should be in drive zero) to the specified drive. SYSTEM7/SYS is copied by use of MOUT and MIN to preserve the subdirectory structure on the old disk. The other files are copied by the COPY utility. After the system files are copied, old utilities on the output disk are deleted and new utilities are copied from the input disk. The program PUTIPL is then run to place the necessary IPL blocks on the output disk.

Since UPGRADE is a text file, it can be edited to modify the chain procedure followed, to adjust to special needs. Any modifications performed should be very carefully considered to assure a good upgrade. System conversions are a complex process and any errors can result in an unusable disk or lost data.

As with partial generation from cassette, use of UPGRADE is not valid for all possible versions of DOS. Check with your Datapoint System Engineer before using UPGRADE for a disk conversion.

5.4 Scratch Disk Generation

Any disk to be used in a DOS system must be generated to contain the necessary system tables and basic system files. Scratch disks or new system disks are best produced by use of the DOSGEN program described later in this manual. DOSGEN is a totally disk based program and performs much more quickly than cassette generation. If necessary or desired, the DOS generation cassette can be used to produce a new disk, as described above in Initial Generation.

5.5 Generation Cassettes and Emergencies

If all boot tapes at an installation are lost or destroyed, there is suddenly no way to access perfectly good disks. New boot tapes can be made by loading the DOS generation cassette in the rear deck and pressing RESTART, then holding down the KEYBOARD key while the tape loads. After about 30 seconds a READY message will appear on the screen from the CTOS (Cassette Tape Operating System), which has just been loaded. Enter the command "RUN B" and CTOS will load and run the program called "B", which is UBOOT, producing a new boot tape for the DOS.

The generation tapes also provide an excellent backup copy of all system utilities and of the system files themselves. The system files are on the DOS generation tape as files #21 through #30 (SYSTEM0/SYS through SYSTEM7/SYS respectively). The availability of such backups can be invaluable in event of massive data loss on system disks.

CHAPTER 6. GENERAL COMMAND CHARACTERISTICS

Some features of the commands supplied with the DOS apply to most DOS commands. These characteristics and messages are discussed briefly in this chapter.

6.1 General Command Format

As mentioned in a previous chapter, DOS commands are entered as a command line. The general format of the command line is:

command [<file spec>][,<file spec>][,<file spec>]...[;options]

The item referred to as "command" is always required on a command line. This defines the command being issued to the system.

The items referred to as "<file spec>" represent one or more specifications for files. These files generally are input, output, scratch, or other files to be used by the command program. Usually the first such specification represents input file(s), and the following specifications represent output or scratch file(s).

A square bracket convention is used here, as well as elsewhere throughout most Datapoint documentation, to indicate fields whose presence is optional. The corner bracket convention (as in <file spec>) represents replacement fields where the replacement field name is contained within the corner brackets. After the replacement is made, the corner brackets themselves do not appear in the resulting line.

The field indicated by "options", separated from the file specification fields by a semicolon, generally contains one or more option letters, which are defined for each specific command.

6.2 Signon Messages

Upon entering a system command, the command program being invoked will generally display a message identifying itself. If the command is specific to one single DOS, the signon message will also identify which DOS the command is designed to execute under. The main purpose of the signon message is to allow the operator to determine, in the event of some difficulty, whether a superceded

version of the command is in use.

6.3 Common Error Messages

Several error messages are common to many of the DOS commands. These error messages, and their meanings, include the following.

WRONG DOS. This message indicates that the version of the command program being run was intended to run on a specific version of the DOS, and that version is not the same as the DOS that is running. This message generally occurs either as a result of accidentally copying a command from one DOS to a different one, or attempting to use an obsolete version of a command under a newer DOS (usually caused by an incomplete upgrade).

INVALID DRIVE. This message appears when one of the drive specifications given by the operator is invalid. Either the drive specification was not of the correct format, or the drive number specified exceeds the range available under the resident DOS.

NAME IN USE. This message occurs when the command's continued execution would necessarily result in a conflict of file name with an already existing file.

NAME REQUIRED. This message generally occurs when one of the file names required on the command line was not specified by the operator.

NO SUCH NAME. This message indicates that a file specified on the command line could not be found. Generally the name as specified is simply misspelled or otherwise incorrectly entered. However, sometimes this message will occur because the file desired is not in the current subdirectory (described later).

NO! THAT FILE IS PROTECTED. This message indicates that a request was made to modify a file that was write or delete protected.

WHAT? This message means that the command name (the first item on the command line being processed) is illegal. This usually indicates that either it is not a valid command, or that the command specified is not in the current subdirectory.

CHAPTER 7. APP COMMAND

7.1 Purpose

The APP command appends two object files together creating a third. Object files are files containing absolute object code in a format that can be loaded by the DOS loader.

7.2 Use

APP <file spec>,[<file spec>],<file spec>

The APP command appends the second object file after the first and puts the result into the third file. Note that neither of the input files are disturbed. If extensions are not supplied, ABS is assumed. The first two files (if a second is specified) must exist. If the third file does not already exist, it will be created. The first file's transfer address is discarded and the new file is terminated by the transfer address of the second file. The transfer address of an object file is defined as the entry point of the program contained in the file.

Omitting the second file specification causes the first file to be copied into the third file. For example:

APP DOG,,CAT

will copy the file DOG/ABS into the file CAT/ABS.

The first and third file specifications are required. If either is omitted the message

NAME REQUIRED

will be displayed. The second and third file specifications must not be the same.

Because the APP command recognizes the actual end of the object module contained in a file, APPing an object file, similiar to the example above, is one technique for releasing excessive unused space at the end of an object file.

CHAPTER 8. AUTO COMMAND

AUTO - Set Auto Execution

AUTO <file spec>

The AUTO command establishes the indicated program to be automatically executed upon the loading of DOS. (Specifically, upon execution starting at the DOS$ entry point.) If no extension is supplied, ABS is assumed. If there is already a file set for auto execution, the message

AUTO WAS SET TO NAME/EXTENSION (PFN).

will be displayed (where PFN is the physical file number). Regardless, the name specified will be recorded in the DOS table location reserved for the auto-execution information. A check is made to see if the file is an object file and if the file is on drive zero. If the specified file does not exist, the message

NO SUCH NAME

will be displayed. Note that if a program has been set to auto-execute, its execution can be inhibited by depression of the KEYBOARD key when DOS is loaded.

If no file spec is given in the commmand line, then the setting of the file to be auto-executed is not changed. However, if a file spec was present, then the message:

AUTO NOW SET TO NAME/EXTENSION (PFN).

will be displayed after the new auto-execution setting has been made.

If no <file spec> is entered and AUTO is not set, the message

NAME REQUIRED

will be displayed.

Note that the AUTO command does not make provision for file specifications to be given to the program which is to be automatically executed. This makes it impossible to use AUTO for programs requiring or accepting such parameters. AUTO also does

not place anything in MCR$ (defined later). Therefore, programs which use overlays with the same name (but different extension) as the program will not run. For more information, refer to the chapter describing the AUTOKEY command.

Auto-execution mode is cleared with the MANUAL command, described in a later chapter.

Programs contained in absolute libraries (UTILITY/SYS for example) cannot be "AUTO'd" directly. Use the AUTOKEY command described below, then "AUTO AUTOKEY/CMD".

CHAPTER 9. AUTOKEY COMMAND

9.1 Introduction to AUTOKEY

Many users allow their Datapoint computers to run in an unattended mode. This allows large data processing tasks, perhaps running via the DOS command chaining facility (see CHAIN), to be run during the evening hours when no operator is present. (An example might be the creation of several new index files for one or more large, ISAM-accessed data bases). However, the momentary power failures which data processing users are being forced to contend with during times of shortage, thunderstorms and the like can bring down any computer not having special, uninterruptible power supplies. When this happens to a computer running in unattended mode, the office staff will generally return the next morning to find their computer sitting idle and its work unfinished.

The Datapoint computers are all equipped with an automatic-restart facility which can be used to cause them to automatically resume their processing tasks following such an interruption. The purpose of the AUTOKEY (and AUTO) commands is to provide a software mechanism for users who wish to handle such unusual circumstances and provide for the restarting of a processing task.

9.2 The Hardware Auto-Restart Facility

There are two small tabs on the back edge (directly opposite from where the tape is visible) of each cassette tape. The leftmost of these (as you look at the top side of the cassette) is the write protect tab, which prevents writing on the topmost side of the tape. The right-hand tab is the auto-restart tab.

Users who frequently use both sides of cassettes will probably immediately notice that if one turns over the tape, the assignments of these two tabs switch around, the tab which had been write protect now being auto restart and vice versa. This in fact is precisely what happens.

If the auto-restart tab on the rear cassette is punched out (or slid to the side), then the computer will automatically

re-boot, just like it does when RESTART is depressed, whenever the processor goes to STOP. Assuming that the rear cassette drive contains a DOS boot tape, this will cause DOS to come up and give its familiar message, "READY".

Diskette 1100 and 1150 users are provided with switch-selectable auto restart. The computer will either halt or automatically restart upon being stopped, depending upon the setting of an internal switch. This switch can be set by a Datapoint representative (System Engineer or Customer Engineer) upon request.

9.3 Automatic Program Execution Using AUTO

In order to provide a mechanism for programs to resume automatically following an interruption (such as a DATASHARE system, for instance, which might be running unattended) DOS has a comparable facility to enable a program to be automatically executed whenever DOS comes up. (Note that any loading and running the DOS, whether by an auto-restart, pressing the RESTART key, or under program control, will activate this facility).

The AUTO command is used to establish a program to receive control when DOS comes up. This setting can be cleared with the MANUAL command. For some applications, the AUTO and MANUAL commands are adequate to allow a programmed restart of a lengthy data processing task. However, some programs require parameters be specified on the command line, and these are obviously not present if no command line has been provided.

9.4 Auto-Restart Facilities Using AUTOKEY

AUTOKEY is simply a command program which can be AUTO'd. The way in which it works is very simple. If it is run via the DOS auto-restart facility, AUTOKEY supplies a command line just as if the same one line were entered at the system console. If AUTOKEY is run from the system console (or likewise from an active CHAIN), it simply displays the command line it is currently configured to supply and offers the user the option of changing that stored command line.

The command line supplied to AUTOKEY could do anything specifiable in one command line to the DOS; DATASHARE could be brought up, a SORT invoked, a user's own special restart program started or even a CHAIN begun. AUTOKEY, when used with AUTO, MANUAL, and CHAIN can therefore provide a very powerful facility.

9.5 A Simple Example

To specify a command line to be used during automatic system restart, simply enter:

```
AUTOKEY
```

at the system console. AUTOKEY will display a signon message and display the current autokey line if there is one. It then asks if this line is to be changed. If "N" is answered, AUTOKEY simply returns to the DOS and the DOS "READY" message is displayed. If "Y" is answered, AUTOKEY requests the new command line to be configured and then returns to the DOS and "READY".

Alternatively, if the user wishes to simply specify a new command line to be configured regardless of the current setting of the AUTOKEY command line, he can merely place the new command line after the "AUTOKEY" that invokes the AUTOKEY command.

As a simple example, assume that XYZ Company has several of their sales offices on-line to their home office DATASHARE system, which is running completely unattended. Lightning strikes a powerline outside of XYZ Company's home office, and power is cut off for 15 seconds. As soon as power is restored, their Datapoint 5500 computer re-boots its DOS (since the right-hand tab on the boot tape has been punched out) and warmstarts the DATASHARE system. One command sequence to accomplish this would look like the following:

```
AUTOKEY
DOS.nn AUTOKEY COMMAND
NO AUTOKEY LINE CONFIGURED.
CHANGE THE AUTOKEY LINE? Y
ENTER NEW AUTOKEY LINE:
DS3
READY
AUTO AUTOKEY/CMD
AUTO NOW SET TO AUTOKEY/CMD (nnn)
READY
```

An alternate form of the above would be the following:

```
AUTOKEY DS3
DOS.nn AUTOKEY COMMAND
NO AUTOKEY LINE CONFIGURED.
ENTER NEW AUTOKEY LINE:
DS3  <--- (this is supplied automatically)
READY
```

```
AUTO AUTOKEY/CMD
AUTO NOW SET TO AUTOKEY/CMD (nnn)
READY
```

Once a program has been set for auto-execution, the only way one can bypass it is to hold down the KEYBOARD key while the DOS is coming up. This action bypasses the auto-executed program and enters the normal command interpreter. The user then can use the MANUAL command to clear the auto-execution option.

9.6 A More Complicated Example

The following example uses many of the features of other facilities in the Datapoint system besides simply AUTOKEY. Explaining all of these in detail is beyond the scope of this section. The intention here is just to demonstrate the sophistication possible using AUTOKEY in conjunction with the other facilities within the DOS.

Let's assume that XYZ Company is running an eight-port Datashare system. Each of the company's seven sales offices around the country has a Datapoint 1100 computer which is connected up to the home office Datashare system as a port. (The eighth port is used by the home office's secretary, Susie.) During the day, each of the seven sales offices makes inquiries of the central inventory, price, and model code files through a system of Datashare programs, and another Datashare program lets them key orders into a file called "ORDERSn" where n is their port number. At the end of each business day, XYZ Company wants to process these orders. First they put the seven files all into one large file, sort it, and use a Datashare program to make corresponding entries into the master order file. The master order file is then reformatted and the index reconstructed. The final step is to create a second copy of the master order file onto magnetic tape, which will then be saved for backup purposes.

Since the operation just described is fairly lengthy, one of the programmers at XYZ Company decided to allow it to run unattended after everyone has gone home. They even set up Susie's MASTER program so that it automatically takes down the Datashare system and starts up the end-of-day processing one-half hour after the company's Los Angeles sales office (two time zones behind the Chicago main office) closes for the afternoon. When the daily processing is completed, Datashare is brought back up again so that it will be up by the time the first people start arriving at the New York sales office the next morning, an hour before the Chicago main office opens.

In the event of an unanticipated power failure, the system will recover and bring itself back up, resuming operations at the last checkpoint established by AUTOKEY. Notice that the system is also left in a state such that after the chain completes, Datashare will automatically restart in the event of any possible system failure. (NOTE: Datapoint 9350 disk systems using Diablo disk drives will initialize with hardware in "WRITE PROTECT" mode after power interruption.)

The following chain file ("OVERNITE/TXT") accomplishes the preceding, assuming that subdirectory "SYSTEM" is used throughout the chain. The chain file could be modified easily to eliminate this assumption. However, the chain file can be made almost arbitrarily complicated; the point here is simply to show one of many possible techniques for handling unattended operations which wish to restart automatically in the case of some failure. Notice that the chain file might have to be modified depending on the particular version of DSCON an installation is using.

```
// IFS S1
//. FIRST SET UP FOR AUTO RESTART IF REQUIRED.
AUTOKEY CHAIN OVERNITE;S1
AUTO AUTOKEY/CMD
BUILD NULL;!
!
//. NEXT APPEND TOGETHER THE SEVEN FILES.
SAPP ORDERS1,ORDERS2,SCRATCH
SAPP SCRATCH,ORDERS3,SCRATCH
SAPP SCRATCH,ORDERS4,SCRATCH
SAPP SCRATCH,ORDERS5,SCRATCH
SAPP SCRATCH,ORDERS6,SCRATCH
SAPP SCRATCH,ORDERS7,SCRATCH
//. NOW SCRATCH CONTAINS THE DAILY FILES.
AUTOKEY CHAIN OVERNITE;S2
// XIF
// IFS S1,S2
//. PHASE TWO SORTS FILE "SCRATCH" INTO "ORDERDAY".
SORT SCRATCH,ORDERDAY;1-5
//. NEXT CHECKPOINT HAVING BUILT "ORDERDAY".
AUTOKEY CHAIN OVERNITE;S3
// XIF
// IFS S1,S2,S3
//. PHASE THREE PROCESSES THE FILE WITH A DS3 PROGRAM.
DSCON
Y
N
Y
Y
```

```
1
DS3 PROCESS
    .
    .
    .
```
The program PROCESS/DBC ends with ROLLOUT "CHAIN NULL" to end the program and continue the chain.
```
    .
    .
    .
//. THE MASTER ORDER FILE "ORDERMAS" NOW IS UPDATED.
AUTOKEY CHAIN OVERNITE;S4
// XIF
// IFS S1,S2,S3,S4
//. PHASE FOUR REFORMATS THE MASTER ORDER FILE.
REFORMAT ORDERMAS,SCRATCH:WORK2;R
//. "SCRATCH" NOW IS A REFORMATTED COPY OF "ORDERMAS".
AUTOKEY CHAIN OVERNITE;S5
// XIF
// IFS S1,S2,S3,S4,S5
//. PHASE FIVE COPIES "SCRATCH" BACK TO "ORDERMAS"
COPY SCRATCH:WORK2,ORDERMAS
//. "ORDERMAS" IS NOW READY FOR INDEXING.
AUTOKEY CHAIN OVERNITE;S6
// XIF
// IFS S1,S2,S3,S4,S5,S6
//. PHASE SIX RECREATES THE INDEX FOR "ORDERMAS"
INDEX ORDERMAS;1-16
//. THE INDEX HAS NOW BEEN REBUILT.
AUTOKEY CHAIN OVERNITE;S7
// XIF
// IFS S1,S2,S3,S4,S5,S6,S7
//. NOW DUMP MASTER FILE TO 9-TRACK MAGNETIC TAPE.
TAPE ORDERMAS/TXT,I/E
B
0
200X4
X
*
//. NOW THE BACKUP COPY OF "ORDERMAS" IS ON TAPE.
AUTOKEY CHAIN OVERNITE;S8
//XIF
//IFS S1,S2,S3,S4,S5,S6,S7,S8
DSCON
Y
N
N
8
```

```
Y
AUTOKEY DS3
//. AND START UP DATASHARE FOR NEXT DAY.
DS3
// XIF
```

9.7 Special Considerations

When building long chain files that allow for automatic restart, several considerations must be made. Among these are that a file must not be changed in such a way that the change cannot be repeated if the previous checkpoint is actually used. To accomplish this, frequently the file being updated must be copied out to a scratch file, and the scratch file then updated. Following the completion of the update is when another checkpoint would be taken: following that the next phase would copy the updated file back over the original. Note that a checkpoint (i.e. resetting the AUTOKEY command line) would have to be before the creation of the dummy copy to be updated; putting a checkpoint between the creation of the copy to update and the actual updating process could result in the updating of a partially updated copy. A little thought when choosing places to update the AUTOKEY command line is called for to ensure that the chain may be resumed from any of them without incorrect results.

9.8 AUTOKEY and DATASHARE

Some users who make frequent use of the DATASHARE ROLLOUT feature will notice that AUTO-ing AUTOKEY with the AUTOKEY command line set to DSBACK will mean that whenever any port rolls out to any program or chain of programs, Datashare is automatically brought back up when that program or chain of programs finishes, regardless of whether or not DSBACK was included at the end of the port's chain file.

CHAPTER 10. BACKUP COMMAND

10.1 Purpose

The BACKUP command provides for making copies of DOS disks. The user can make either an exact mirror image copy of the input disk or can select reorganization, which will group files by extension and file name, remove unnecessary segmentation and allow deletion of unnecessary files. Reorganization also allows copying of DOS disks onto disks with locked out cylinders that differ from those on the input disk. Some special considerations apply for specific disk configuration.
NOTE: BACKUP always copies the volume-id (VOLID) to the output disk.

10.2 Use

A disk backup is initiated by the operator entering the following command:

BACKUP <input drive>,<output drive>

Input drive and output drive are specified as :DRn, or :Dn, or :<volid>. The drive selected as the INPUT DRIVE MUST BE WRITE PROTECTED; that is, it must be in "read only" mode or have its "protect" light on for 9370 and 9350 series drives respectively. The requirement for the input drive to be write protected is absent on the 9380 series flexible diskettes. The program will respond by displaying the message:

DRIVE n SCRATCH?

If the disk on drive n is scratch (note that BACKUP deals with logical drives), enter a "Y" . Any other reply will cause the program to return to DOS. If you do reply "Y", the program will display the message:

ARE YOU SURE?

If you are absolutely sure that you want to write over the output disk, type "Y" again and press the enter key. Any other reply will cause the program to return to DOS. If the output

(logical) disk has not been DOSGENed or the DOS file structure on it has been damaged, the message:

DOSGEN YOUR DISK FIRST

will appear and control returns to DOS. If the output (logical) disk has been DOSGENed and seems in reasonable shape, the following message is displayed:

FILE REORGANIZATION?

If different cylinders are locked out on the input and output disks (if the disks' lockout CATs do not match), a mirror image BACKUP is not possible so the "FILE REORGANIZATION?" question is bypassed. Instead, a message appears specifying that reorganization is required and BACKUP with reorganization proceeds as described below.

If you wish to reorganize the files being transferred to the output disk, enter a "Y" in response to the reorganization question. In this case, see the section on reorganizing files for further instructions.

If you do not wish to reorganize your files and desire a mirror image copy of your input disk, enter an "N" in response to the reorganization question.

10.3 Mirror Image Copy

If you have typed "N" in response to the file reorganization question, the program will ask the question:

DO YOU WANT THE OUTPUT COPY VERIFIED?

This question should always be answered "Y". At present the answer given has no effect the output is always write-verified. The question is maintained so chain procedures invoking BACKUP do not need to be modified.

The program then asks:

DO YOU WANT TO COPY UNALLOCATED CLUSTERS?

Type "Y" and press the enter key if you want all data on the disk copied regardless of whether or not it is in an area allocated by DOS. This option is preferred in cases where you suspect that your DOS files may be partially destroyed or the

output disk has never been fully initialized with data. Also use this mirror image copy if you have the 9374 disk system and one of the drive's heads gets misaligned. Backup will use the offset feature to try and retrieve your data. If BACKUP uses the track offset it will slow the program down but it could save your data.

Type "N" and press the enter key if you wish to copy your disk as quickly as possible without copying unused areas of the input disk. "Y" and "N" are the only replies allowed.

10.4 Reorganizing Files

If you have typed "Y" in response to the file reorganization question, the program will copy the System files, sort the Directory names, and allow the operator to delete files before copying the files to the disk copy.

Backup with reoganization to drive 0 is not possible.

10.4.1 Copying DOS to Output Disk

Various program status messages will appear during the copying of DOS. System tables are initialized and then the SYSTEMn/SYS files are copied to the output disk.

10.4.2 Deleting Named Files

When all directory names have been sorted into file extension followed by file name sequence the following question will be displayed:

DELETE ANY FILES DURING REORGANIZATION?

Type "N" and press the enter key if all files are to be copied. Type "Y" and press the enter key if you wish to delete any files. If you reply "Y" a message asking which files are NOT to be copied will appear. The lower screen will be filled by a numbered list of files for you to choose from. Type the number or range of numbers (nn or nn-nn) found next to names of individual files you wish deleted. Type "ALL" and press the enter key if you wish to delete all of the files in the list. The files selected for deletion will be erased from the list. When all desired deletions have been made from a list, type "." and press the enter key to advance to the next list of file names.

When all file name lists have been examined, the program will advance to the copy named files phase.

10.4.3 Copying Named Files

Files with names in the system directory are copied in alphameric file extension, file name sequence. The name of each file is displayed as it is copied. All files are written as close together as possible with a minimum of segmentation.

10.5 Use of KEYBOARD and DISPLAY Keys

The KEYBOARD and DISPLAY keys may be pressed any time messages are being displayed. Depressing the DISPLAY key will hold the current display until the key is released. Depressing the KEYBOARD key will cause the program to terminate and return to DOS.

10.6 Error Messages

During the execution of BACKUP the following error messages may appear:

******* PLEASE PROTECT YOUR INPUT DISK *******

Action: Write-disable the input drive.

INVALID DRIVE SPECIFICATION!

Action: Retype the BACKUP command with correct <input-drive> and <output-drive> specification.

ILLEGAL OUTPUT DRIVE!

Action: <input-drive> and <output-drive> have been specified as the same drive! Retype BACKUP command with correct specification.

BAD CLUSTER ALLOC TABLE!

Action: A bad Cluster Allocation Table has been detected on the input disk. The Cluster Allocation Table may be able to be fixed using the REPAIR command.

CYLINDER 0 OF BACKUP DISK IS UNUSABLE!

Action: Your scratch disk cannot be used for a system disk due to surface defects in cylinder 0. Use another output disk and start over.

SYSTEMn /SYS IS MISSING!

Action: Your DOS disk cannot be reorganized due to a missing system file. Catalog the missing system file on your input disk and start over.

PARITY- :DRn address

Action: An irrecoverable parity error has been detected on drive n during the BACKUP operation. The address is shown for each error. If drive n is your output disk, DOSGEN must be rerun to lockout the bad addresses or use a different scratch disk for mirror image copy. If drive n is your input disk, new parity will be computed and the record will be copied. Note the error address and check for errors when copy is complete.

FORWARD OFFSET TRACK BEING USED

REVERSE OFFSET TRACK BEING USED

Action: On a 9374 disk system a parity error has been detected on the input drive and offset tracking is being used to try to recover the data. There will be 10 attempts on both sides of the track.

10.7 Reorganizing Files for Faster Processing

After a DOS disk has been used for awhile, the file structure becomes fragmented and related files become scattered. The more the disk is used the more total system performance is degraded due to increased disk access time. System degradation is especially noticeable when DATASHARE is being used. File reorganization using the BACKUP program is one way to clean up DOS disks and improve their efficiency.

BACKUP reorganization improves system efficiency by making the following changes:

. File segments are consolidated

. Files are packed more closely together

. Related files are clustered together

. Unused trash files are removed (optionally)

. Files are rewritten reducing marginal parity errors

10.8 BACKUP with CHAIN

Because BACKUP requires that its input drives be write protected, does not abort if parity errors occur during the backup, and may ask different questions depending upon the condition of the input and output disks, BACKUP generally should not be invoked from a CHAIN. Since the BACKUP operation is so critical to the protection of important files, an operator should monitor the entire backup operation.

10.9 Clicks during Copying

A click occurs each time an unused sector is copied (reorganization mode only). A file which, when copied, results in a lot of clicks (more than a dozen, perhaps) can probably be reduced in size, without any data loss, by using APP or SAPP as appropriate.

10.10 Special Considerations for BACKUP

When using BACKUP on the 11-platter 9370 disk packs, it is important to remember that each disk is two logical drives. Since BACKUP deals with logical drives, BACKUP must be run twice, once from each logical drive, to backup an entire physical disk.

Also, BACKUP will not allow backing up from one logical drive to the other one on the same disk. There is no real backup value, since the two copies would be physically on one pack.

With the 9374 and 9354 disk drives,it is important to remember that the drive contains a fixed platter that is a separate logical drive. BACKUP between the fixed and removable platters is possible.

CHAPTER 11. BLOKEDIT COMMAND

11.1 Purpose

The BLOKEDIT command provides for DOS text file manipulation. The command copies lines of text from any DOS text file(s) to create a new text file.

The BLOKEDIT command is useful for such things as:

New program source file generation by copying routines from existing program source files;

Existing program source file re-arranging by copying the lines of source-code into a new sequence (into a new source file).

Re-arranging lines or paragraphs of a SCRIBE file into a new file.

In this Chapter, the following applies:

Text file means a DOS text file as defined in the REFORMAT chapter.

Line means one line of a text file as displayed by the DOS LIST program.

11.2 File Descriptions

BLOKEDIT deals only with text files. For any given application there will be one text file called the COMMAND FILE which will hold the controlling commands for BLOKEDIT. Optionally the controlling commands may be entered directly to BLOKEDIT via the keyboard by defaulting the command file parameter. There will be one or more text files called SOURCE FILES from which lines of text will be copied. And there will be one text file called the NEW FILE which will be the desired end result for the application.

11.2.1 Command Statement Lines

The command statements are the controlling factor for a BLOKEDIT execution. The command statements specify which source files will be used and which lines of text will be copied from them. If the command statements are to be read from a command file it must be generated by the DOS. EDIT command, or DOS. BUILD command, etc., before BLOKEDIT can be used.

There are three kinds of statement lines that are meaningful to BLOKEDIT: COMMENT lines, COMMAND lines, and QUOTED lines.

A COMMENT line is a line which has a first character of period.

This is an example of COMMENT LINES:

```
.
. THESE THREE LINES ARE COMMENT LINES.
.
```

As in program source files, a comment line may have explanatory notes or nothing at all following the period.

A COMMAND LINE is a line which has a SOURCE FILE NAME and/or source file LINE NUMBERS, or begins with a double quote symbol (").

The following are some example command lines:

```
FILENAME/EXT:DR0          NAME THE SOURCE FILE
1-100                     COPY LINES 1 THRU 100
350-377                   COPY LINES 350 THRU 377
150/TXT                   NAME THE SOURCE FILE
```

A command line must have a first character of an upper-case alphabetic character, or a digit, or a double quote symbol.

A command line that begins with an upper-case alphabetic character indicates that a new SOURCE FILE is being named. A new source file can be named only by putting the name of the file at the very beginning of the command line. Optionally, the extension and/or drive number for the file may be included with the source file name. If the source file name begins with a digit the file extension must be given.

A command line that begins with a digit indicates that the command line will have one or more numbers, which are the numbers

of the lines to be copied from the source file previously specified into the new file.

A command line that begins with a double quote symbol indicates the beginning/ending of QUOTED LINES. The only information used by BLOKEDIT in a command line that begins with a (") is the (") itself, therefore the rest of the line can be used for comments.

A QUOTED LINE is a line between a pair of command lines which begin with a double quote symbol.

This is an example of QUOTED LINES:

```
" THIS IS THE BEGINNING OF QUOTED LINES COMMAND LINE.
INCMNT  HL    COUNT           POINT TO COUNTER
        LAM                   LOAD TO "A" REGISTER
        AD    1               INCREMENT BY 1
        LMA                   RESTORE TO MEMORY
" THIS IS THE ENDING OF QUOTED LINES COMMAND LINE.
```

There may be more than one quoted line between the command lines that begin with ("). A quoted line will be copied directly from the command file or keyboard to the new file. Quoted lines enable a BLOKEDIT user to include original lines of text in a new file along with lines copied from source files.

11.2.2 Source File

The SOURCE FILE is a text file from which lines will be copied. Source files are named in the command lines for a BLOKEDIT application, and the lines to be copied from the source file will also be specified in the command lines. It will be useful to have a listing of a source file with line numbers, as produced by the LIST command, when creating the command statement lines for a BLOKEDIT application.

11.2.3 New File

The NEW FILE is a text file produced by the BLOKEDIT command. The new file is named at BLOKEDIT execution time by the second file specification entered on the command line.

11.3 Using BLOKEDIT

Before the BLOKEDIT command can be used one must create a command file, unless the command statements are to be entered via the keyboard. When the BLOKEDIT command is to be executed, the operator must enter the following command line:

BLOKEDIT [<file spec>],<file spec>

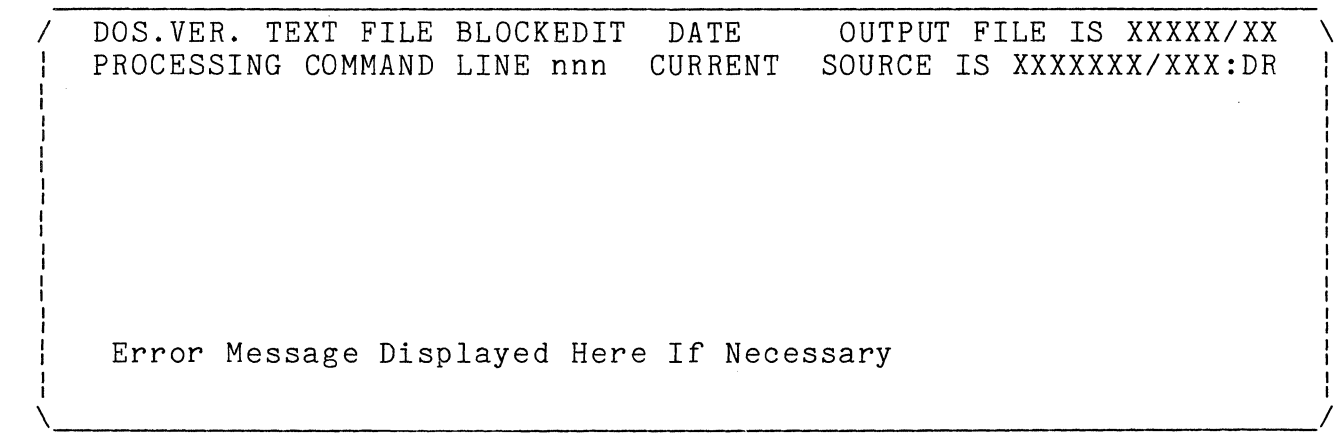
The first file specification refers to the command file, if not specified the commands will be entered via the keyboard. The second file specification names the new (output) file. If no extension is supplied with the first file specification, TXT is assumed. If no extension is supplied with the second file specification, the extension given or assumed for the first file is used. If no drive is given for the first file, all drives are searched. If no drive is given for the second file, the drive given or assumed for the first file is used. The specified output file must not exist on any drive on line.

11.4 Messages

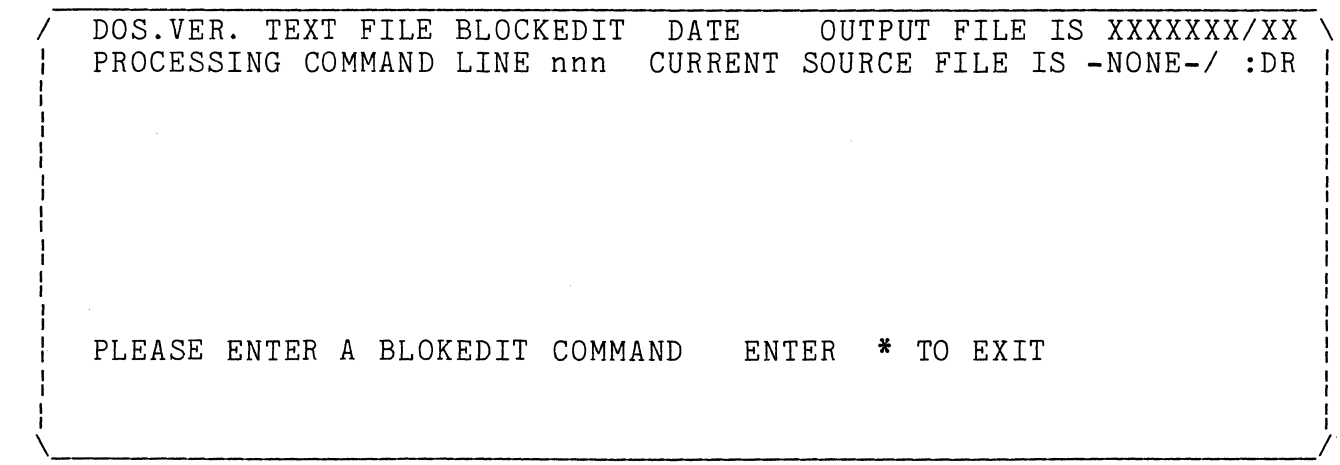
This section describes the operator messages that BLOKEDIT may display on the CRT screen during execution. Some of the messages are monitor messages to keep the operator informed of the progress of the program, while other messages are error messages. If the keyboard was selected as input to BLOKEDIT, the user will be prompted by the "Please enter BLOKEDIT command Enter * to exit." message when input is required. The character * will terminate BLOKEDIT and return to DOS.

The general format of the CRT display screen varies depending on the source of the BLOKEDIT command statements.

If the command statements are being read from a command file the format of the display is:



If the command statements are being entered via the CRT keyboard, the format is:



As BLOKEDIT commands are entered on line 12, they are rolled thru lines 9-4.

11.4.1 Informative Messages

PROCESSING COMMAND LINE .. CURRENT SOURCE FILE IS ../..:DR.

This message is the BLOKEDIT monitor message. This message is displayed while BLOKEDIT is writing lines of text to the new file. The monitor message displays the command file line number currently being processed and the name, extension, and drive number of the last named source file.

SOURCE FILE WENT TO E.O.F.

This messge is displayed if the source file from which lines were being copied ended before the specified lines were finished.

BLOKEDIT TRANSFER COMPLETE
OUTPUT FILE WAS name LINE COUNT WAS nnn

This message is displayed when all of the command file lines have been executed. The number of lines in the new file is displayed following the second line.

11.4.2 Fatal Errors

If BLOKEDIT detects a fatal error in the command statement line the monitor message is rolled up the screen, an appropriate error message is displayed, and the DOS entry ERROR$ is called.

NEW FILE NAME REQUIRED

This message is displayed if the operator did not name a new file when the BLOKEDIT command was called.

COMMAND FILE DRIVE INVALID

This message is displayed if the operator specified for the command file a drive number that is invalid.

NEW FILE DRIVE INVALID

This message is displayed if the operator specified for the new file a drive number that is invalid.

COMMAND AND NEW FILE NAMES MUST NOT BE IDENTICAL

This message is displayed if the operator specified command file and new file names the same and the extension and the drives

for the files were specified or assumed to be the same. Defaulting of extensions and drives is described in an earlier paragraph.

COMMAND FILE NOT FOUND

This message is displayed if the command file name was not found on the drive(s) specified or assumed.

NEW FILE NAME IN USE

This message is displayed if the specified output file was found on the drive(s) specified or assumed. BLOKEDIT will not write into an existing file if commands are being read from a command file. If commands are being entered to BLOKEDIT via the KEYBOARD, the operator is given the option to overwrite the existing file.

***NEW FILE NAME IN USE, OVERWRITE IT? ***
[Answer with a Yes or No]

If the operator answers Yes (Y) the file is overwritten.
If the reply is No (N) BLOKEDIT returns control to DOS.

BAD FILE SPECIFICATION

This message is displayed if the first character of a command file line, other than a quoted line, is an upper-case alpha character but the DOS file specification was not recognizeable.

11.4.3 Selectively Fatal Errors

These errors are fatal when BLOKEDIT is reading a command file, and informative when commands are being entered via the keyboard.

SOURCE FILE NOT FOUND

This message is displayed if the source file specified could not be found. It is probably either misspelled or in a different subdirectory.

BAD LINE NUMBER SPECIFICATION

This message is displayed if a command file line other than a quoted line began with a digit but contained an unrecognizable line number specification.

Here are some examples of valid line numbers:

4	A single digit is acceptable.
999999	A line number may have up to six digits.
100-364	First and last line to be selected are separated by a dash.
34,55-78,100-147	Commas separate line specifications.

Here are some examples of invalid line numbers:

1A	Only "-", ",", or space after a digit, unless the line is a source file name beginning with a digit. If it is, an extension must be given.
1234567	Number has more than six digits.
17-34-77	Only two numbers separated by "-".

LINE NUMBER ZERO IS NOT VALID

This message is displayed if a line number of zero is specifed in a command line. It is ignored if entered via the keyboard.

START LINE NO. > END LINE NO

This message is displayed if the first number of a line number pair is larger than the second number of the pair, as in: 235-176. It is ignored if entered via the keyboard.

BAD DATA IN SOURCE FILE LINE nnn *

This message is displayed if BLOKEDIT discovers non-ASCII characters in a source file. The line number will be displayed following the message. If commands are being entered via the keyboard the source file is reselected, and next command is requested.

NO VALID SOURCE FILE FOR TRANSFER

This message is displayed if BLOKEDIT discovers line numbers to be transfered from a source file when there is no open source file.

FORMAT OR RANGE ERROR ON SOURCE FILE

This message is displayed if DOS discovers a file which can not be read. If commands are being entered via the keyboard the source file will be de-selected, and next command requested.

CHAPTER 12. BUILD COMMAND

12.1 Purpose

BUILD provides an alternative means to create a text file without having to use the standard DOS editor. BUILD is useful for rapid generation of very short text files, such as two and three line CHAIN files. Also, BUILD is usable from within a CHAIN.

12.2 Use

The BUILD command is invoked by entering the command line:

BUILD <file spec>[;<end character>]

The file specification defines the ou ‘ file. This output file specification is always required. If the named file does not exist, it is created. The default extension is /TXT.

The end character is optional. If no end character is specified on the command line, BUILD terminates upon receiving a null input line (a null input line is a line consisting of only an ENTER; a blank line is not a null line).

BUILD accepts input lines from the keyboard and writes each one to the output file. When BUILD is ready to accept an input line it displays a colon (:) as a prompting character. Each input line BUILD receives is tested for the presence of the specified end character, if any, as the first character entered. If the end character is present as the only character of the entered line, the end line is discarded (it is not written to the output file), and an end of file mark is written to the output file and the output file closed by returning to DOS.

Entering an end character followed by a string will pass the string to the output line without the end character ... and will not terminate BUILD. This action allows entering CHAIN commands into a chain file being written by BUILD from within an active CHAIN.

12.3 A Simple Example

Suppose that the operator wishes to construct a simple CHAIN file to establish a program to be auto-executed, so that the auto-execute request can be accomplished later with a single command line entered at the keyboard. All that is required is to enter at the system console:

```
BUILD <chain file spec>;*
AUTOKEY <program name>
AUTO AUTOKEY/CMD
*
```

Upon receiving the "*" input line, BUILD closes the output file and terminates. Note that in the two places where the "*" appears, any enterable character could have been used. (This allows nesting calls to BUILD, which can be very useful in the BUILDing of chain files). After the BUILD command is finished, the output file named on the BUILD command line contains the following two lines:

```
AUTOKEY <program name>
AUTO AUTOKEY/CMD
```

It is also possible, through BUILD nesting, to create chain files which during execution of the chain construct other chain files and execute them automatically upon completion of the first chain (since any statement of a chain file <u>is</u> allowed to be a CHAIN command).

The references to CHAIN made here may be premature, since CHAIN is discussed in a later chapter, but are included because BUILD and CHAIN can be of great usefulness when used together in this manner.

The KEYBOARD and DIS[illegible] keys may be pressed any time messages are being displa[illegible] The keys will be effective just prior to the display of th[illegible]ompting ":". Depressing the DISPLAY key will hold the current display until the key is released. Depressing the KEYBOARD key will cause the program to terminate and return to DOS.

CHAPTER 13. CAT COMMAND

13.1 Purpose

The CAT command selectively displays filenames in the DOS directory or in a library directory. One may choose to display all cataloged filenames on all drives online or specific filenames on specific drives.

13.2 Use

The CAT command is invoked by entering the command line:

CAT [<name>][</ext>][:DR<n>][,L]

where: <name> specifies the filename or a portion of the filename, <ext> specifies the extension or a portion of the extension, <n> specifies the logical disk drive number, and L specifies list only those files in the current subdirectory.

To display a library directory enter:

CAT <library name>*

To display the UTILITY/SYS directory enter:

CAT *

Directory entries are displayed in the form:

NAME/EXTENSION (PFN)P

where PFN is the physical file number in octal (0-0377) and P is the protection on the file; D for deletion, W for write, and blank for none. If the file displayed is in a subdirectory other than system, the directory entry is displayed in the form

NAME/EXTENSION-(PFN)P

with the dash indicating a subdirectory entry. All drives are searched, unless a specific drive is requested, and as each drive is scanned, the line

---- DRIVE n VOLUME ID (volid) SUBDIRECTORY (subdirectory name):

is displayed. This line is not displayed if the drive is not on line, or if no files from it are to be displayed.

Depressing the DISPLAY key causes the catalog display to pause as long as the key is held. Depressing the KEYBOARD key causes the catalog display to terminate. If the CAT command is parameterized by only an extension, only files of that extension will be displayed. If the CAT command is parameterized by only a name, only files of that name (all extensions) will be displayed. If the CAT command is parameterized by a name and an extension, only files of that root name and extension (all drives) will be displayed. If the CAT command is parameterized by only the drive number, only files on that drive will be displayed. If only a portion of the filename is entered, all files beginning with the letters specified will be displayed. For example, entering:

```
CAT /T
```

would cause the display of all files on all on-line drives whose extensions start with "/T".

Entering:

```
CAT MA:WORK2
```

would cause the display of all files on symbolic drive "WORK2" whose file names start with "MA".

CHAPTER 14. CHAIN COMMAND

14.1 Introduction

The CHAIN command enables a user to create and execute procedure files. The chain file should contain the commands to invoke all required programs, and all inputs required by the invoked programs. Basically, CHAIN replaces the DOS Keyboard Entry Routine with one that reads lines from a procedure file each time the Keyboard Entry Routine (KEYIN$) is called. Each time any program would normally request a line to be entered from the keyboard, it will be read from the Procedure File. This reading of lines from a Procedure File is transparent to the executing program. When the last line of the Procedure File has been read, and a new DOS command is desired by the system, DOS is reloaded and commands are accepted from the keyboard.

The CHAIN command has two separate functions which are performed at different times. They are Compilation and Execution phase.

Compilation Phase

CHAIN executes a compilation phase in which statements are read from the chain input file. During this phase all compile time decisions are made and micro substitution is done. The result of this phase is a Procedure File named CHAINP/SYS. This Procedure File consists only of statements needed for the execution phase. The procedure file name will contain the partition number, instead of a "P", if CHAIN is run under PS.

Execution Phase

The execution phase of CHAIN is the interface to the operating system and the control of retrieval of information from the Procedure File. During the execution phase, CHAIN/OV1 overlays the DOS KEYIN$ routine with a disk read routine that fits in the same space. After CHAIN/OV1 has been loaded the first line is read from the Procedure File and given to DOS as input.

When a routine calls KEYIN$ for a line, CHAIN/OV1 fetches a line from the Procedure File. The return from CHAIN/OV1 appears the same as it would from KEYIN$. The HL and DE registers are the

same as if the line had been entered by the user from the keyboard. If the line read is longer than the maximum specified by the calling program the program is aborted and the chain abondoned. The same is true if a program is requesting a line and the procedure file is at end of file.

When a program invoked by the CHAIN Procedure File terminates by jumping to the DOS EXIT$ or NXTCMD the CHAIN/OV1 routine reads the next statement, if present, from the Procedure File. If the end of file is reached when DOS is requesting another command, the CHAIN is determined to be finished. At this time, normal termination of CHAIN, the procedure file is deleted and commands can be entered via the keyboard.

14.2 Tag Definition

The CHAIN command line can contain both tag names and/or tag names and values for the tags. These parameters follow the semicolon (;) on the command line which invokes CHAIN. The tag names can be from one to eight characters in length and may have values from one to seventy characters in lenght. A tag must contain only letters or digits. The value of a tag may contain any valid character except comma (,), equals (=) or pound sign (#). The character restriction depends on the syntax being used.

A tag is defined by just its presence on the CHAIN command line. Tags may have a value given to them by one of the following syntaxes:

```
CHAIN DOIT;LIST,DATE=30NOV76,TIME=1500hr      (New Syntax)

CHAIN DOIT;LIST,DATE#30NOV76#,TIME#1500hr#    (Old Syntax)
```

Both syntax structures are supported and the results of the two CHAIN commands is identical. The tag LIST has been defined but has a null value; DATE has the value of 30NOV76 and TIME has the value of 1500hr.

CHAIN allows two uses to be made of tags:

1.) A tag can be tested to determine whether it was defined on the CHAIN command line.

2.) The value of the tag can be substituted on CHAIN input statements before the line is written to the Procedure File.

14.3 Compilation Phase Directives

All CHAIN directives are denoted by the characters // as the first two on a line. Any number of spaces (including zero) are scanned until the CHAIN directive is reached. The first thing after the // must be a valid CHAIN directive else an error message is issued and CHAIN is aborted. The following is a list of these statements.

//IFS	IF SET (TAG DEFINED)
//IFC	IF CLEAR (TAG NOT DEFINED)
//XIF	END OF IF
//ELSE	REVERSE EFFECT OF IF
//BEGIN	BRACKETS A GROUP OF
//END	IF/ELSE/XIF STATEMENTS
//.	EXECUTION TIME COMMENT
//*	EXECUTION TIME BREAKPOINT
//ABORT	ABORT CHAIN COMPILATION
//ABTIF	CONDITIONALLY ABORT CHAIN EXECUTION
.	COMPILATION TIME COMMENT. (Note that the //'s are not present)

14.3.1 IF Directive

The IF directive has two variations, IFS and IFC, which are IF SET and IF CLEAR. The IFS directive proves positive if the tag named appeared on the CHAIN command line, and negative if the tag was omitted.

For example:

```
//IFS LIST
```

will prove positive if LIST was mentioned in the CHAIN command line, and negative if the tag does not exist. The opposite of this is true for the IFC directive.

For example:

```
//IFC LIST
```

will prove positive if LIST was omitted and negative if it appeared on the CHAIN command line.

Simple logical operations can be performed on IF directives. The tags to be used are separated by logical operators. The logical OR is indicated by '¦' (vertical bar) or ',' (comma). The logical AND is indicated by '&' (ampersand) or '.' (period). For example the following lines are in the file DOIT:

```
//IFS DATE&TIME|QUICK   or   //IFS DATE.TIME,QUICK
SNAP TEST                    SNAP TEST
```

If DATE AND TIME OR QUICK are defined on the CHAIN command line the SNAP test line will be included in the procedure file. CHAIN DOIT;DATE=30NOV76,TIME=1500hr or CHAIN DOIT;QUICK or CHAIN DOIT;DATE,TIME will all result in a true logical condition and the SNAP line will be included.

IF directives are only evaluated if lines are being included. If one IF directive has proven negative and has inhibited the use of lines, all following IF directives will be ignored until either an ELSE or XIF statement is found.

For example:

```
//IFS DATE
//IFS TIME
SNAP TEST
//XIF
```

If DATE was not defined, all lines until the // XIF will be ignored. In this example the //IFS TIME statement would not be evaluated and the SNAP TEST would not be included even if TIME was defined.

14.3.2 ELSE/XIF Directives

CHAIN has two directives that will alter the inclusion of lines from an IF directive. The first is the XIF directive. It will unconditionally terminate the range of the last IF directive. The second is the ELSE directive; it will reverse the results of the last IF directive; that is to say, if lines were being skipped because the last IF proved negative, an ELSE would cause lines to be included.

Example, the DOIT file contains the following lines:

```
//IFS LIST
SNAP TEST;L
//ELSE
SNAP TEST
//XIF
//IFS TAPE
MOUT;D,30NOV76,V
TEST/ABS
*
//XIF
```

If CHAIN is invoked by 'CHAIN DOIT;LIST' the procedure file will contain

```
SNAP TEST;L
```

If invoked by 'CHAIN DOIT;TAPE', the procedure file will contain

```
SNAP TEST
MOUT;D,30NOV76,V
TEST/ABS
*
```

14.4 Tag Value Substitution

A tag value is substituted whenever a pair of '#' symbols are found with a syntactically valid tag name between them. The value substituted is the tag value given in the CHAIN command line.

An example: Contents of a file called DOIT

```
SNAP TEST;XL
TEST PROGRAM ASSEMBLED ON #DATE# -- #TIME#

SNAP #NAME#;XL
#NAME# PROGRAM ASSEMBLED ON #DATE# -- #TIME#
```

If CHAIN is invoked by "CHAIN DOIT;TIME=2400hr,DATE=29NOV76,NAME=TEST2" the procedure file will contain

```
SNAP TEST;XL
TEST PROGRAM ASSEMBLED ON 29NOV76 -- 2400hr

SNAP TEST2;XL
TEST2 PROGRAM ASSEMBLED ON 29NOV76 -- 2400hr
```

If a tag is mentioned in the CHAIN command line but given no value and if the value is to be used for substitution, a null value is substituted for the #tag# within the line. The effect is that the #tag# characters disappear from the line. For example, if CHAIN was invoked by "CHAIN DOIT;DATE=29NOV76,NAME=TEST2" the procedure file will contain

```
SNAP TEST;XL
TEST PROGRAM ASSEMBLED ON 29NOV76 --
SNAP TEST2;XL
TEST2 PROGRAM ASSEMBLED ON 29NOV76 --
```

14.5 BEGIN/END Directives

The BEGIN and END statements allow groups of IF/ELSE/XIF statements to be parenthesized. A counter called the BEGIN/END counter is initialized to zero when compilation of a procedure begins. If the use of procedural lines is turned off and a BEGIN operator is encountered, then the BEGIN/END counter is incremented. If an END operator is encountered, then the BEGIN/END counter is decremented unless it is already zero. The ELSE and XIF operators have no effect if the BEGIN/END counter is not equal to zero. For example:

```
// IFS FLAG1
ASM TEST1;XL
TEST PROGRAM ONE
// ELSE
// BEGIN
// IFS FLAG2
ASM TEST2;XL
TEST PROGRAM TWO
// ELSE
ASM TESTTEST;XL
TEST TESTER
// XIF
// END
// XIF
// IFS FLAG3.FLAG27
LIST SCRATCH;L
THE SCRATCH FILE AT FLAG 27
// XIF
```

The 6th through the 12th lines will not be used if FLAG1 exists, notwithstanding the fact that there is an ELSE and XIF operator within those lines, because the BEGIN/END pair prevented

these statements from having any effect.

14.6 ABORT Directives

The //ABORT statement will cause CHAIN to return to DOS if it is processed. For example:

```
//IFC TIME¦DATE
.**** TIME AND DATE ARE BOTH REQUIRED
//ABORT
//XIF
.
.
.
```

If the Procedure File is invoked with TIME or DATE missing, the error message comment line would be displayed, and the compilation of the input file would ABORT.

The //ABTIF statement will conditionally cause the execution phase of CHAIN to ABORT. This statement causes DOSFLAG to be examined and if bit 7 (ABTIF) is on, the chaining will abort. Bit 7 of DOSFLAG is the abnormal program completion bit. If errors have been found during the execution of the last program the ABTIF bit should be set. For example, the procedure file contains:

```
KILL TESTFILE/CMD
Y
//ABTIF
KILL OUTPUT/TXT
Y
.
.
```

If the file TESTFILE/CMD is not found by KILL, it will set the ABTIF bit. When the //ABTIF statement is processed the abnormal program completion bit will be checked, and in this case it will be on, so the CHAIN will be aborted.

14.7 Comments

CHAIN allows for two types of comment lines within the procedural file. One type is the execution time comment. This type may appear only before a DOS command entry and will not appear until just before that command is to be executed. An execution time comment can appear only just before a command because at any other place in a procedure file, the comment would be presented as keyboard response to an executing program. Comments can be placed at the end of a procedure, since this location is equivalent to immediately prior to a command. For example, the procedure file containing:

```
//. ASSEMBLY OF THE TEST PROGRAM
ASM TEST;XL
TEST PROGRAM
```

would cause the first line to be displayed before the assembly was executed. A variation on the execution time comment is the operator break point. For example, the procedure file containing:

```
//* INSERT TAPE Z12548 INTO THE FRONT CASSETTE DECK
MOUT ;LV
TEST
DATA/TXT
*
```

would cause a BEEP and the first line to be displayed. At this point the machine would wait for the operator to depress either the KEYBOARD or DISPLAY key and then continue with the MOUT process.

The second type of comment line is a compilation time comment. This line is not included in the procedure but is displayed on the screen immediately after it is read from the procedural file. This is useful in communicating to the operator what procedure is about to be followed by CHAIN.

Both types of comment lines will be ignored (not displayed or written) just as other procedure lines if a test has proven negative and an ELSE or XIF operator has not been reached. For example, if the following procedure file MAKETEST was created:

```
.  ASSEMBLY OF TEST PROGRAM
// IFS LIST
.  YOU ARE GOING TO GET A LISTING
ASM TEST;XL
TEST PROGRAM
```

```
// ELSE
.  YOU AREN'T GOING TO GET A LISTING
ASM TEST
```

and the CHAIN command:

```
CHAIN MAKELIST;LIST
```

was given, then only the lines:

```
. ASSEMBLY OF TEST PROGRAM
. YOU ARE GOING TO GET A LISTING
```

will appear on the screen before the procedure is executed. If, however, the CHAIN command:

```
CHAIN MAKETEST
```

was given, then only the lines:

```
. ASSEMBLY OF TEST PROGRAM
. YOU AREN'T GOING TO GET A LISTING
```

will appear on the screen before the procedure is executed.

14.8 Complex CHAIN Example

As an example of a complex CHAIN operation, consider the following procedure file, RUNTEST. This file is part of a series of CHAIN procedures for program generation and testing. RUNTEST builds a procedure file for program assembly; the resulting procedure file would be run by a later CHAIN.

RUNTEST recognizes several tags:

P5500 - mention of this tag indicates a 5500 processor will be used for assembly.

REL - mention of this tag, along with P5500 will cause the relocatable assembler to be used.

FLAG - the substitution value for this tag will be tag tested for list control on the output procedure file.

PROG - the substitution value for this tag will be a tag to provide program name in the output procedure file.

DATE - the substitution value for this tag will provide the assembly date in the output procedure file.

Complex CHAINing example:

```
.
. TEST FOR 5500 PROCESSOR FLAG
.
//IFC P5500
//BEGIN
.
. BEGIN PROCEDURE FOR 2200 ASSEMBLY
.
BUILD ASMIT;*
.
. NOTE HOW BEGINNING INPUT LINE TO BUILD/CMD WITH THE TERMINATION CHARACTER
. ALLOWS ENTERING CHAIN COMMANDS TO THE OUTPUT FILE.  THE LINE IMMEDIATELY
. BELOW IS WRITTEN OUT AS "//IFS #FLAG#"; IF IT HAD NOT BEGUN WITH "*", IT
. WOULD HAVE BEEN INTERPRETED AS A CHAIN DIRECTIVE FOR THE CURRENT CHAIN.
.
*//IFS #FLAG#
*//* ASSEMBLY LISTING  -  BE SURE PRINTER IS READY
ASM5 ##PROG##;LX
##PROG## ASSEMBLY     #DATE#
*//ELSE
ASM5 ##PROG##
*//XIF
*
//END
.
. THIS "//ELSE" INSTRUCTION REVERSES THE EFFECT OF THE "//IFC P5500" ABOVE
.
//ELSE
//BEGIN
.
. BEGIN PROCEDURE FOR 5500 ASSEMBLY USING SNAP/1 OR SNAP/2 BASED ON "REL"
. FLAG.
. THE "BEGIN" ABOVE CAUSES THE "XIF"S AND "ELSE"S IN THE FOLLOWING SECTION
. TO AFFECT ONLY DIRECTIVES AT THE SAME BEGIN/END LEVEL, AND NOT THE
. "//ELSE" DIRECTIVE ABOVE, WHICH CONTROLS THE ENTIRE 5500 ASSEMBLY
. SECTION.
.
BUILD SNAPIT;*
*//IFS #FLAG#
*//* ASSEMBLY LISTING  -  BE SURE PRINTER IS READY
.
. THE FOLLOWING DIRECTIVES ARE RECOGNIZED DURING CHAIN COMPILATION AND
. CONTROL SELECTION OF LINES TO FOLLOW THE BUILD COMMAND ABOVE.
.
//IFS REL
```

```
SNAP2 ##PROG##;LX
//ELSE
SNAP ##PROG##;LX
//XIF
##PROG## ASSEMBLY #DATE#
*//ELSE
//IFS REL
SNAP2 ##PROG##
//ELSE
SNAP ##PROG##
//XIF
*//XIF
//IFS REL
.
. AGAIN TEST IF RELOCATABLE ASSEMBLER IS DESIRED.  IF SO, ADD LINK COMMAND.
.
*//IFS #FLAG#
LINK ##PROG##;L
LINK MAP FOR ##PROG##
*//ELSE
LINK ##PROG##
*//XIF
//XIF
*
.
. PROCEDURE IS EFFECTIVELY FINISHED AT THIS POINT, BUT IT IS ESSENTIAL TO
. PROVIDE AN "END" DIRECTIVE TO MATCH THE UNMATCHED "BEGIN" ABOVE, AND
. AN "XIF" TO TERMINATE THE "ELSE" IMMEDIATELY PRIOR TO THE "BEGIN".
.
//END
//XIF

RESULT OF "CHAIN RUNTEST;P5500,REL,FLAG=LIST,PROG=NAME,DATE=21JAN77"

//IFS LIST
//* ASSEMBLY LISTING  -  BE SURE PRINTER IS READY
SNAP2 #NAME#;LX
#NAME# ASSEMBLY 21JAN77
//ELSE
SNAP2 #NAME#
//XIF
//IFS LIST
LINK #NAME#;L
LINK MAP FOR #NAME#
//ELSE
LINK #NAME#
//XIF
```

```
RESULT OF "CHAIN RUNTEST;FLAG=PRINT,PROG=PROG,DATE"

//IFS PRINT
//* ASSEMBLY LISTING  -  BE SURE PRINTER IS READY
ASM5 #PROG#;LX
#PROG# ASSEMBLY
//ELSE
ASM5 #PROG#
//XIF
```

14.9 Resuming An Aborted CHAIN

Before the CHAIN overlay fetches the next DOS command it stores in the CHAINP/SYS file pointers for the line to be used. If something goes wrong during the DOS command which follows and the procedure is aborted, CHAIN still knows where it was in the CHAINP/SYS file when the problem occurred. Since CHAIN does not delete the CHAINP/SYS file unless the procedure completes successfully, it can pick up where it stopped in the CHAINP/SYS file if the operator can correct the condition which caused the procedure to abort in the first place. Often, the reason for the abort is something correctable like the disk running out of files. In this case, the operator need only correct the condition and then enter:

```
CHAIN *
```

and the procedure will pick up with the command which failed before. This action can generally be applied even if the RESTART key has been depressed. Thus, one can recover from jammed paper in a printer half way through a listing by simply depressing RESTART, fixing the printer, and then entering the CHAIN * command.

If the failing command cannot ever succeed, it may be bypassed by entering the command:

```
CHAIN/OV1
```

This simply restarts the chain with the next available line in the procedure. If the next line had been intended as a keyin line for the failed program (as opposed to a DOS command line) the chain will generally immediately abort again. However, by restarting the chain in this manner, repeatedly if necessary, the invalid step can usually be bypassed and chaining resumed.

NOTE: CHAIN/OV1 only works if the area from MCR$+80 through

MCR$+100 has not been disturbed.

14.10 Notes On Usage of CHAIN

CHAIN only replaces the DOS keyboard entry routine (KEYIN$). Therefore, only programs that use this routine for input will receive their input from the chain file. Programs which have their own input routines, like the Editor, can be invoked from a chain file but editing must be done manually by the operator. The CHAIN program itself can be called from within a CHAIN file. The chain is aborted when a CHAIN-invoked program makes an exit to DOS that implies that an error of some kind has been made. The error message given by the program will generally remain on the screen after the chain is aborted.

Some programs can go through a rather complex set of requests for input which can make them hard to use with the CHAIN program without making a mistake. For this reason, most DOS programs allow almost all options to be specified on the command line and keep the variation in the number of keyin requests to a minimum. It is good practice for all programs to be written with this in mind to facilitate their use with CHAIN.

An additional item to keep in mind is the fact that some DOS programs use their own keyboard entry routine as well as the one provided by the DOS. This enables the program to avoid the use of the CHAIN procedural lines when special operator intervention is required.

CHAPTER 15. CHANGE COMMAND

CHANGE - Change a file's protection

CHANGE <file spec>;p

The CHANGE command enables one to write protect, delete protect, or clear the protection of a disk file. If a file is delete or write protected, a KILL command (or program generated KILL) cannot affect it. If a file is write protected, it cannot be written into by the standard system routines.

The option parameter "p" is used above to indicate the protection for the file specified. Protection can be specified as:

D - delete protect
W - write protect
X - clear protection.

For example:

```
CHANGE NAME/EXTENSION;D
CHANGE NAME/EXTENSION:DR2;X
```

will delete protect the file in the first case, and remove all protection in the second case. If a first specification is not given, the message

NAME REQUIRED.

will be displayed. If the file indicated by the first file specification cannot be found, the message

NO SUCH NAME.

will be displayed. If the option parameter does not follow the above syntax rules, the message

INVALID PROTECTION SPECIFICATION.

will be displayed.

CHAPTER 16. COPY COMMAND

16.1 Purpose

It is frequently useful to make a copy of a disk file. It may be desired, for example, to make a copy on a separate volume for backup or distribution purposes.

Another feature of the COPY command will optionally allow a user to selectively update (replace) an existing file, or create (add) a new file to receive the copy. These options used in combination with the CHAIN utility provide an easy method of updating and maintaining DOS disks and diskettes.

The COPY command does not make assumptions about the format of the sectors being copied, but merely copies the file sector-for-sector. It can copy most types of disk files which previously were not possible to copy using the SAPP and APP commands. Some particular types of files are still unmovable, however. The outstanding example are INDEX files, usually with extension /ISI. These cannot be moved because index files contain, internal to themselves, pointers indicating their actual physical location on the disk volume, which are made invalid when the file is moved to another place on the disk.

16.2 Use

The COPY command is invoked by entering at the system console:

```
COPY <file spec>,<file spec>       Unconditional copy
COPY <file spec>,<file spec>;R     Replace only
COPY <file spec>,<file spec>;A     Add only
```

UNCONDITIONAL:
This option will cause the first specified file to be copied into the second one. Attributes of the first file, such as its protection, are copied to the second file as well.

REPLACE:
This option will copy a file only to an existing file. If the output file does not exist no copy of data takes place and an informative message is given, "file-name NOT COPIED".

ADD:
This option will copy a file only if the output file does not already exist. If the output file does exist no copy of data takes place and an informative message is given "file-name NOT COPIED".

The only portion of the operands that is specifically required is the name of the input file. The extension of the input file, if none is specified, is assumed to be /TXT. If a drive specification is entered for the input file, then only that specific drive is searched for the indicated file. If no drive specification for the input file is given, all drives are searched. If the name of the output file is omitted, it is assumed to be the same as that of the input file. If the output file's extension is not given, it is also assumed to be the same as that of the input file. All drives are searched for the output file unless a particular drive is specified.

Example, to copy file PAYROLL/TXT from symbolic drive "WORK2" to symbolic drive "WORK1"

```
COPY PAYROLL:WORK2,:WORK1
```

Example, to make another copy of PROGRAM/ABS on drive zero, but to be named MYPROG.

```
COPY PROGRAM/ABS,MYPROG:DR0
```

Example, to make another copy of PAYROLL/TXT drive 0, on drive 1 only if it does not already exist on drive 1.

```
COPY PAYROLL:DR0,:DR1;A
```

Example, to update (replace only) TREK/ABS, a file on drive 0 from a newer version on drive 1.

```
COPY TREK/ABS:DR1,:DR0;R
```

People who experience parity errors in one of their data files can frequently recover their data using COPY. Since the COPY program merely comments about parity errors encountered and does not abort when one occurs, the data copied will occasionally

be correct (or almost correct) even if a parity error occurs and can be used to recover the data in the original file. Alternatively, using the COPY program to write the file on top of itself (therefore without changing the file) by simply specifying the input file and no output file, a user can frequently clear soft (and occasionally what seem to be hard) parity errors occurring in an important data file. (Of course, no important file should be updated in place unless a copy of the file exists somewhere for recovery purposes in the event of a failure).

The COPY command issues a click each time an unused sector is copied. If more than a dozen or so clicks occur at the end of copying a file, it usually indicates that the file is larger than necessary to contain the data in it. In this case, moving the file using APP or SAPP can sometimes help to reduce its size. Clicks ocurring during the copying (before the end of the file) indicate sectors containing DOS format errors, possibly implying a sector accidentally destroyed by some faulty program.

CHAPTER 17. DOSGEN COMMAND

17.1 Purpose

Before any disk can be used by DOS, certain tables and other information must be placed onto it to establish the basis that DOS requires for the support of its file structure. These tables include the skeleton of the DOS directory, (where the names of the files contained on the disk are stored), as well as a map showing which places on the disk are bad and should not be used.

The purpose of the DOSGEN command is to provide the user with a simple way of accomplishing this preparation.

17.2 Use

To DOSGEN a disk enter:

DOSGEN <drive spec>

The drive spec is a standard DOS drive specification which specifies which drive contains the disk to be prepared for DOS use. Since the directory initialization process will effectively KILL any files that might be on the disk, the command asks several times to make sure that the operator is aware of the potential seriousness of the operation he has invoked.

After the operator has acknowledged that he does not mind the overwriting of the new disk, the command asks if any cylinders on the volume are to be locked out. Normally, the answer to this question is NO. However, by answering YES, it is possible to cause the DOS to lock out one or more cylinders of the disk from DOS access. This can be useful in some special applications where it is desired to not allow DOS programs access to a file stored in unusual format. If the user does wish to lock out any cylinders, he may do so by specifying one or more cylinder numbers, in the format:

12,14,16,25-28,40

The above example would cause cylinders 12, 14, 16, 25, 26, 27, 28, and 40 to be locked out. Note that the cylinder numbers

to be locked out are given in decimal as opposed to octal.

After the operator has specified that no, or which, cylinders are to be locked out, the DOSGEN command checks for bad sectors on the disk and issues a message indicating any cylinders it finds which contain bad sectors. Any cylinders found bad are automatically locked out and will not be used by DOS. The remainder of the operation is completely automatic and indicates its completion with the familiar DOS message, "READY".

Upon completion of the DOS generation process, the only files on the new disk are the eight system files SYSTEM0/SYS through SYSTEM7/SYS and the CAT command.

17.3 Special Considerations

It is important to remember that on disk packs for use with DOS systems recognizing more than one logical drive per physical disk pack, for example the 9370 series disk system, two DOSGENs must be done before the physical pack is fully initialized. This allows the user to DOSGEN either logical disk on the pack without disturbing files he wishes to keep that may be stored on the other logical disk.

Another important thing to remember is that both the 9370 and 9380 series disks must be formatted before DOSGEN can be used on them. Diskettes (for the 9380 series drives) come pre-formatted from the manufacturer; disk packs for the 9370 series drives do not. It is therefore necessary to format all disk packs for the 9370-series drives using the program INIT9370 before attempting to use DOSGEN on them. A diskette that has been formatted with tracks locked out (error mapped) cannot be DOSGENed.

CHAPTER 18. DUMP COMMAND

18.1 Purpose

Occasionally while writing into files on disk (in particular, during the program debugging stage) it is useful to be able to verify that the formatting of the information into the standard text format is being done correctly. Or, perhaps an assembler language program (/ABS file) that previously loaded correctly no longer will, as indicated by DOS just coming back up when the program is run.

The DUMP command provides a simplified mechanism for examining the entire contents of physical sectors on the disk. The display includes both the octal and ASCII contents of every byte on the sector. No examination for control bytes of any kind is made, allowing the user to see the precise contents of every physical location in the disk sector.

18.2 Use

The DUMP command is invoked by entering:

```
DUMP
```

or

```
DUMP <file spec>
```

The DUMP command operates with basically five separate levels of control. These levels are:

LEVEL ONE - Logical drive level
LEVEL TWO - File level
LEVEL THREE - Logical record number level
LEVEL FOUR - Physical disk address level
LEVEL FIVE - Disk directory level

The (optional) entry file and/or drive specifications on the command line allow the first one or two input levels in DUMP to be automatically bypassed.

When the DUMP command is used, the top line on the display is the primary control line. Input is accepted on this line. This line is broken into four basic areas, one corresponding with each of the first four control levels. The primary control level at any given time during the operation of the DUMP command can be determined by the position of the flashing cursor on the control line.

For example, if the flashing cursor is positioned after the "DRIVE:" legend on the control line, the DUMP command is operating at level one. If the cursor is positioned after the "FILE:" legend on the control line, the DUMP command is operating at level two, etc.

18.3 Informational Messages Provided

The second line on the display is primarily used for sector informational messages. These serve both to indicate any special significance of the sector just read and to describe any unusual occurrences associated with reading the sector. These messages are generally self-explanatory. Among the messages that can be displayed are the following, along with an explanation of the meaning of each.

RETRIEVAL INFORMATION BLOCK (RIB). This message indicates that the sector being displayed is the primary RIB for the currently opened file.

RETRIEVAL INFORMATION BLOCK BACKUP. Each RIB is maintained in duplicate for backup purposes and to allow recovery in the event of a program erroneously destroying the primary RIB. This message indicates that the sector being displayed is the secondary RIB for the currently opened file.

CLUSTER ALLOCATION TABLE. This message indicates that the sector being displayed is the primary Cluster Allocation Table (normally referred to as the CAT) for the current logical drive.

CLUSTER ALLOCATION TABLE BACKUP. This message indicates that the sector being displayed is the secondary, backup CAT for the current logical drive. The CAT is also maintained in duplicate just as is the RIB.

LOCKOUT CLUSTER ALLOCATION TABLE. Associated with each logical drive is a sector that indicates which areas have been locked out, prohibiting their use by DOS. This message indicates that the sector being displayed is the Lockout CAT for the current

logical drive.

LOCKOUT CLUSTER ALLOCATION TABLE BACKUP. This message indicates that the sector being displayed is the secondary, backup copy of the sector.

SYSTEM DIRECTORY SECTOR. This message indicates that the sector being displayed is one of the DOS directory sectors. The directory sector number (in decimal and in octal) immediately follows the message.

USER DATA SECTOR. This message indicates that the sector is not recognized as one of the above special system sectors.

DISK SECTOR CRCC ERROR. This message indicates that the sector requested for display either was not found on the disk or that a CRCC error repeatedly occurred during the read operation. The sector displayed is the data as it was read from the disk, unless the sector was not found.

DISK OFFLINE. This message indicates that the currently specified logical drive is not on line.

DISK SECTOR FORMAT ERROR. This message is displayed when DUMP notices that the sector being displayed does not correspond to standard DOS file conventions (the first byte of each sector is its physical file number, and the two following bytes are the logical record number). The appearance of this message does not necessarily indicate that the sector of the file has been destroyed, since unwritten sectors at the end of a file and older version DATASHARE object code files normally will fall into this class. It merely means that if the sector were read with the DOS READ$ routine, a format trap would occur.

SECTOR OUT OF RANGE. This message is displayed if the sector requested (by logical record number) is not within the range of the currently opened file.

FILE NOT FOUND. This message indicates that the file requested could not be found. This does not necessarily mean that the file does not exist. For example, the file could be in a non-current subdirectory. If the user has not requested non-specific volume mode (to be described), this message might mean simply that the file desired is on a different logical drive.

INVALID PHYSICAL ADDRESS. This message indicates that the physical disk address specified is invalid.

The remainder of the display contains the contents of the current half of the sector most recently read. The display is arranged as eight groups of sixteen bytes each. Each of these groups is preceded by the three octal digit offset of that group within the sector. Each sixteen byte group consists of the octal and ASCII contents of each of the sixteen bytes in that group. Each byte's contents form a column one character wide and four lines high, where the first three lines are the value of the byte, in octal, and the fourth line is the ASCII value of that character. Notice that the character is not examined for special significance before it is displayed, so that computers having the high speed RAM display option (which is strongly recommended for all DOS systems) may display characters other than the normal ASCII set.

18.4 Level One Commands To DUMP

When the flashing cursor indicates that DUMP is functioning at level one, the following commands are accepted:

<enter> - The CAT on the current drive is displayed and control is transferred to level two. In addition, the non-specific drive mode is enabled.

number - The drive number indicated becomes the currently selected drive. The CAT from that drive is displayed and control is transferred to level two. Non-specific drive mode is disabled.

* - DUMP command returns control to the DOS.

> - The second half of the current sector is displayed.

< - The first half of the current sector is displayed.

18.5 Level Two Commands To DUMP

When the flashing cursor indicates that the DUMP command is functioning at control level two, the following commands are accepted:

<enter> - If a file is currently opened, the secondary RIB for the file is displayed and control is transferred to level three. If no file is opened, control is transferred to level four.

name/ext - The named file is opened on the current drive, or any drive if non-specific drive mode is enabled. The primary RIB for the file is displayed and control is transferred to level three.

pfn - The file indicated by the octal physical file number given is opened on the current drive. The primary RIB for the

file is displayed and control transfers to level three.

I - The current physical file number is incremented and the new file thus indicated is opened. If no file corresponding to that physical file number exists on the current drive, the PFN is incremented repeatedly until a file corresponding to the PFN is found. The primary RIB for the file is displayed and control is transferred to level three.

D - D works just like the I command above except that instead of incrementing the PFN, it is decremented.

#pfn - The directory sector containing the entry corresponding to the file indicated by the specified physical file number is displayed; then control is transferred to level five. Since only the last four bits of the PFN are relevant, the pfn specifier is equivalent to a relative directory sector number. These directory sector numbers are always specified in octal.

* - Return control to level one.

> - Show the second half of the current sector.

< - Show the first half of the current sector.

18.6 Level Three Commands To DUMP

When the cursor indicates that DUMP is functioning at level three, the LRN level, the following commands are accepted.

<enter> - The current sector is shown and control is transferred to level four.

number - Access and display the record indicated by the LRN specified. If the number given has a leading zero, it is assumed to be octal; otherwise it is assumed to be decimal. The number specified is the user (as opposed to system) LRN. The system LRN, the value in bytes one and two in the sector, is always two greater than the user LRN. The two numbers displayed at level three in the control line are the user LRN in decimal (the one with leading zeros suppressed) and octal (the one in parentheses, with leading zeros).

I - Increment the current logical record number, access it and display the sector.

D - Decrement the current logical record number, access it and display the sector.

* - Return to the File level of control (level two).

> - Show the second half of the current sector.

< - Show the first half of the current sector.

18.7 Level Four Commands To DUMP

Level four of the DUMP command requires more detailed understanding of DOS physical disk addresses, and as such is not usually as useful as the LRN level. However, when access to a specific sector on the disk is desired, it can be achieved using DUMP level four. It is important to realize that the physical disk addresses specified are logical physical disk addresses, i.e. the same format as is given to the DR$ and DW$ routines in the DOS. They are not necessarily the same as actual physical locations on the disk. For example, with DOS.C for the 9380 series diskettes, the logical disk addresses are remapped onto the diskette into different hard physical sector numbers than those indicated by the logical physical disk address. The important thing to understand here is that the disk addresses used in the level four control of DUMP are those that would be used to parameterize DR$ and DW$.

The commands accepted at level four of DUMP are as follows.

msb,lsb - Access and display the sector indicated at the given physical disk address on the current logical drive. The first field (most significant byte) is assumed to be in decimal unless a leading zero is supplied. The second field (least significant byte) is always considered to be in octal, regardless of whether a leading zero is supplied or not. The second field is separated from the first by a comma. The physical disk address given by the user is assumed to be valid. If it is not of the proper format, undefined results may occur. Users who are not sure of their understanding of DOS internal physical disk addresses should not use level four of DUMP.

* - Return control to level two if no file is opened, or level three otherwise.

> - Show the second half of the current sector.

< - Show the first half of the current sector.

18.8 Level Five Commands to DUMP

When the flashing cursor indicates that the DUMP command is operating at control level five (system directory sector level), the following commands are accepted:

number - Show the directory sector indicated by the low order four bits of the number specified. Since only the low order four bits of the number are used, it is not an error to specify simply the physical file number (PFN) of the file whose directory entry is to be examined. A leading zero indicates the number is in

octal, otherwise decimal is assumed.

I - The current directory sector number is incremented and the corresponding directory sector is displayed.

D - The current directory sector number is decremented and the corresponding directory sector is displayed.

* - Return control to level two.

> - Show the second half of the current directory sector.

< - Show the first half of the current directory sector.

18.9 Error Messages

Only one error message is issued by the DUMP command. It is:

ERROR IN DOS FUNCTION. DUMP ABORTED.

If this error message occurs, it means that the DOS FUNCTIONs are probably incorrect on the disk, generally indicating that the disk in the booted drive has not been completely (or correctly) DOSGENed. If this is the case, SYSTEM7/SYS should be loaded using the latest copy of DOS as distributed by Datapoint.

CHAPTER 19. THE DUMP93X0 COMMAND

DUMP93X0 represents one of three programs: DUMP9350, DUMP9370, DUMP9380. Each program functions on only one of the Datapoint type disks, 9350 series, 9370 series, or 9380 series repectively. In the following chapter, characteristics of a particular program or disk will be indicated by the specific drive type. Features common to all programs will be indicated by reference to "DUMP93X0", so the "X" can be at any time read as "5", "7", or "8". The examples that follow are primarily set for DUMP9370 use, since the 9370 disk uses the most complex address format. In general, the examples apply equally well to 9350 or 9380 disks, ignoring the head address used in the 9370 command. The DUMP93X0 command enables the programmer to inspect, record, or load physical disk sectors. DUMP93X0 is intended to be used only for extremely low-level disk examination and by trained systems personnel. Most users will find the facilities provided by the DUMP command to be more useful for general disk examination purposes.

19.1 Use

DUMP93X0 can be invoked from an active DOS by keying in at the system console:

```
DUMP93X0
```

Since DUMP93X0 is a completely self-contained program, it can be run from an LGO cassette tape (unlike most DOS commands which rely on one or more of the DOS routines for their execution). In this mode, DUMP93X0 can occasionally be useful in helping to determine the problem when the DOS will not boot up from some disk. If a user intends to use DUMP93X0 in this way, he should take care to make an LGO tape and store it safely away somewhere, before he needs it.

DUMP93X0 can output physical disk records (sectors) to a local printer, the cassette deck, or to the screen, and can load sectors to disk from the cassette deck.

There are two command handlers in DUMP93X0. The primary command handler controls all DUMP93X0 functions except the screen dump. The screen dump requires its own syntax because it is an interactive, and more flexible, facility.

All commands to DUMP93X0 employ the same conceptual structure, though elements of commands may be implicit as well as explicit. The full explicit format for commands is:

```
DUMP9370:    Z AAA,BBB,CCC  DDD,EEE,FFF
DUMP9350:    Z AAA,CCC  DDD,FFF
DUMP9380:    Z AAA,CCC  DDD,FFF
```

where
- Z is the command
- AAA is the starting cylinder number
- BBB is the starting head on cylinder AAA(DUMP9370 only)
- CCC is the starting sector on that track
- DDD is the ending cylinder number
- EEE is the ending head on cylinder DDD(DUMP9370 only)
- FFF is the ending sector on that track

Notice that all disk addresses are "hard" physical disk addresses, as opposed to DOS standard-format (or "logical") physical disk addresses. All numbers input to DUMP93X0 are octal. Consult the appropriate appendix for a description of the physical addressing of the type of disk in use.

The command codes of the primary command handler are:

- P Print on the local printer
- S Screen dump
- CD Cassette dump
- CL Cassette load
- # Jump to DOS DEBUG
- * Return to DOS command interpreter
- A ASCII mode (for printer or screen dump)
- E EBCDIC mode (for printer or screen dump) (DUMP9380 only)
- O Octal mode (for printer or screen dump)
- @ Physical drive number

The command codes of the screen dump command handler are:

- * Return to the primary command handler
- # Jump to DOS DEBUG
- I Increment the (cylinder,head,sector) address
- D Decrement the (cylinder,head,sector) address
- C Cylinder address mode
- H Head address mode (9370 only)
- S Sector address mode
- A ASCII display mode
- E EBCDIC display mode (9380 only)
- O Octal display mode

The following operating instructions discuss the commands and their applications, with some examples, in more detail.

19.2 The primary command handler

As soon as DUMP93X0 has fully loaded, it displays its signon message on the screen. When the cursor appears at the lower left corner of the screen the primary command handler is ready to accept commands.

19.3 Using DUMP93X0 with a Local Printer

P - Print on the local printer

DUMP93X0 will print only to a 132 column local printer, address 0303. The 256 byte disk records (sectors) are listed 32 bytes per line, 8 lines per sector. Preceding each 8 line block of print is a short line giving the physical disk address of the printed sector. One sector or the entire disk may be dumped to the printer by a P command. After the last sector is printed the page is ejected to top of the next page.

Unless otherwise specified, the bytes are printed in octal, with a space separating each byte, except every eighth byte is delimited by a period. If the DUMP93X0 command is in the ASCII mode (set with the A command) characters that are valid ASCII characters will be printed in ASCII. Lower-case ASCII alphabetic characters are indicated by a preceding underscore (_). If the DUMP9380 command is in the EBCDIC mode, bytes that are valid EBCDIC characters will be printed in EBCDIC, lower case characters preceded by an underscore.

COMMAND EXAMPLES:

P 000,000,000 000,000,000

would dump to the printer the disk records from cylinder 000, head 000, sector 000, thru cylinder 000, head 000, sector 000. In other words, print only the one sector with the disk address 000,000,000.

Note from the following examples that the parameter fetching subroutine will make certain assumptions about information not explicitly given.

P 0,0,0 0,23,27

would dump to the printer the disk records from cylinder 000, head 000, sector 000, thru cylinder 000, head 023, sector 027. In other words, dump to the printer all of the sectors on cylinder zero. Note that it is not necessary to supply leading zeros in an address.

For 9350 series disks, the equivalent command, dump all of cylinder 0, is

```
P 0,0 0,67
```

For 9380 series disks, the equivalent command is

```
P 0,0 0,14
```

```
P 0 0
```

would do exactly the same thing as the previous example. When only the first number is given between spaces, it is taken to be a cylinder address, with a sector and head address of 000 assumed for the beginning cylinder. For 9370 disks, a head address of 023 and a sector address of 027 are assumed for the ending cylinder address. For 9350 disks, a sector address of 067 is assumed for the ending address. For 9380 disks, a sector address of 014 is assumed for the ending address.

```
P 4
```

would dump to the printer the disk records from cylinder 004, head 000, sector 000, thru cylinder 004, head 023, sector 027. In other words, all of the sectors on cylinder 4. When only one cylinder address is given, it is taken to be both the beginning and ending cylinder address. For 9350 series, the command would dump from cylinder 004, sector 000, through 004, sector 067. For 9380 series, the command would dump from cylinder 004, sector 000, through cylinder 004, sector 014.

```
P 67 70,7
```

would be assumed to mean: P 067,000,000 070,007,027 ,

```
or for 9350's    P 067,000  070,007
or for 9380's    P 067,000  070,007
```

19.4 Screen Display format

S - Screen dump

DUMP93X0 can display on the CRT one disk physical record (sector) at a time, in octal or ASCII (or EBCDIC for 9380). The address of the sector displayed is controlled in a manner analogous to the display of bytes in memory by the DOS debugging facility.

A special display format is utilized to enable all 256 bytes of a sector to be displayed on the screen at one time. Below is a diagram of what a screen dump of a sector would look like; given the CYL,HED,SEC address = 44,0,6 and each byte in the example sector is its location within the sector; (i.e., starting at the beginning of the sector, the bytes are (in octal) 000, 001, 002, 003,. . ., 0377:

```
044_000001002003004005006007 010011012013014015016017 020021022023024025026027
000 030031032033034035036037 040041042043044045046047 050051052053054055056057
006 060061062063064065066067 070071072073074075076077
  _100101102103104105106107 110111112113114115116117 120121122123124125126127
   130131132133134135136137 140141142143144145146147 150151152153154155156157
   160161162163164165166167 170171172173174175176177
  _200201202203204205206207 210211212213214215216217 220221222223224225226227
   230231232233234235236237 240241242243244245246247 250251252253254255256257
   260261262263264265266267 270271272273274275276277
  _300301302303304305306307 310311312313314315316317 320321322323324325326327
   330331332333334335336337 340341342343344345346347 350351352353354355356357
   360361362363364365366367 370371372373374375376377
```

Note from the diagram that:

The displayed sector address is in the upper left-hand corner of the screen. For 9350 disks, the cylinder and sector address is shown. For 9370 disks, the cylinder, head, and sector address is shown. For 9380 disks, the cylinder, physical sector, and logical sector address is shown. Each portion of the address is on one line; stated sequence above is top to bottom.

Each group of 10(octal) bytes is displayed in a contiguous block of digits.

Each block of 100(octal) bytes begins at the left side of the screen, preceded by an underscore (_).

Each block of 100(octal) bytes consists of 10(octal) groups of 10(octal) contiguous bytes; 3, 3, and 2 groups to a screen line, for the three lines required to display 100(octal) bytes.

The screen displays 400(octal) bytes, which is one disk sector, 256(decimal) bytes.

To further break down the screen and enable quick location

and reading of individual bytes, the first digit of every second byte is flashed on and off. Thus, each group of eight bytes is divided into four units of two bytes.

COMMAND EXAMPLES:

S 044,014,006

would mean: display cylinder 44, head 014, sector 6 on the screen. This command can only be given to the primary command handler, and after it is executed DUMP93X0 will be under the control of the screen dump command handler.

19.5 The Screen Dump Command Handler

Note that as in the DOS debugging facility, the command codes entered are not displayed, the command is merely immediately executed.

* Return to the primary command handler. The screen will be rolled up, the cursor turned on, and keyed commands will be displayed as they are entered at the lower left corner of the screen.
NOTE that the SHIFT key must be depressed at the same time as the asterisk (*) key.

\# Jump to the DOS debugging facility. # will not work if DUMP93X0 was loaded from an LG0 tape.
NOTE that the SHIFT key must be depressed at the same time as the pound sign (#) key.

I Increment the cylinder, head, or sector address and display the sector at the new address. The new disk address will be displayed at the top left corner of the screen.

If the C (Cylinder address mode) command is in force when an I command is given, the cylinder address will be incremented by one, the head and sector addresses will not change. Cylinder address wrap-around occurs at 0312->000 (0114->000 for DUMP9380). Incrementing by cylinder address is useful for scanning quickly thru a large file by steps of 4 (9380) or 8 (9350,9370) clusters per increment.

If the H (Head address mode) command is in force when an I command is given, the head number will be incremented by one. If the head address was 023, it will wrap around to head zero and the cylinder address will be incremented by one. Note that

the head address will increment across both the two logical packs on the physical drive. H is operative only under DUMP9370.

If the S (Sector address mode) command is in force when an I command is given, the sector address will be incremented by one. If the sector was the last on the track (014 for 9380, 067 for 9350, 027 for 9370), then the head and/or cylinder address is incremented by one and the sector address is set to zero. If the cylinder address was the last on the disk, it will be set to zero. Incrementing by sector enables scanning sector by sector thru a file and inspection of the exact data on each disk record. Files which span logical cylinders or are non-contiguous on the disk (which includes most large files) will require more detailed understanding by the user of the DOS file structure (in order to avoid incrementing out of the file's allocated space) and are usually better examined using the DUMP command.

D Decrement the cylinder, head, or sector address and display the sector at the new address. Except for the direction of address change, the D command is functionally like the I command.

C Cylinder address mode. This command causes subsequent I or D commands to alter the cylinder address. Optionally, a cylinder address may be keyed in before striking the C key; the current cylinder address will be replaced by the entered value before the disk record is read and displayed. The entered digits will be displayed at the lower left corner of the screen. Note that the address must be an octal address. If more than three digits are entered DUMP93X0 will BEEP and the procedure must be re-begun. If the address entered is not a valid cylinder address (e.g., greater than 0312) the C command will be in force but the cylinder address will not be changed. Also note that only the eight least significant bits of the value entered will be taken for the address (an entered value of 444 would be interpreted as 044).

H Head address mode. This command causes subsequent I or D commands to alter the head number. Except for the fact that the H command modifies head addresses and sets head mode, it is similar to the C command. (DUMP9370 only.)

S Sector address mode. This command causes succeeding I or D commands to alter the sector address. Optionally, a sector address may be keyed in before striking the S key. The address option is functionally similar to the C command.

Sector address mode is the assumed mode of operation when the program is started.

A ASCII display mode. This command causes the bytes to be displayed in ASCII instead of OCTAL on the screen, for all bytes that have valid ASCII bit configurations. This is useful for examining text files on disk. Note that the ASCII mode will carry over to the P (print) command of the primary command handler unless changed by a subsequent O command.

E EBCDIC display mode (9380 only). This command causes the bytes to be displayed in EBCDIC instead of OCTAL on the screen, for all bytes that have valid EBCDIC bit configurations. This is useful for examining the index track (track zero) on a diskette, and for text files on IBM formatted diskettes. While DUMP9380 is in EBCDIC mode, sector addresses used are taken as physical sector numers. During ASCII or Octal modes the addresses are taken as logical sector numbers and are re-mapped to take sector skewing and radius spiraling into account (see Appendix C).

O OCTAL display mode. This command causes the bytes to be displayed in OCTAL instead of ASCII. OCTAL mode is the assumed mode of operation when the program is started.

19.6 Cassette Operations

CD - Cassette Dump
CL - Cassette Load

DUMP93X0 can write to the front cassette deck the contents of specified disk sectors, and can read DUMP93X0 tapes from the front deck to load specified sectors.

COMMAND EXAMPLES:

CD 000,000,000 000,002,027

would mean: dump the sectors from cylinder 000, head 000, sector 000, thru cylinder 000, head 2, sector 027 to the cassette in the front deck. In other words, dump the first three tracks of the disk to cassette. The CD command will dump from one sector to 500 sectors (all that will fit on a cassette), in contiguous sectors. The disk addresses given (explicitly or implicitly) must be from lesser to greater (e.g. CD 40,0,0 36,0,27 would be invalid because the second address is less than the first address). If any fault is found in the addresses given, the message:

PARAMETER ERROR

will be displayed and the machine will BEEP. Refer to the discussion of the P (print) command for examples of explicit and implicit addresses in commands. If the command is correct, the message:

FRONT DECK SCRATCH ?

will be displayed. A reply of "Y" will cause the cassette dump to proceed, while a reply of "X" will cause an exit to the primary command handler. Any other reply will cause the question to be repeated. When the front deck is ready, the cassette dump will rewind the tape and begin dumping the specified sectors to tape as individual 256-byte records. When all of the sectors have been written, the tape is rewound and checked sector by sector against the sectors on disk. If the tape data does not match the disk data exactly, the cassette dump will abort with the message:

TAPE/DISK VERIFY FAILURE

and exit to the primary command handler. If the tape is correct, it is rewound and control is returned to the primary command handler.

CL 0 ,2

means: load the disk sectors addressed 000,000,000 thru 000,02,027 from the front cassette. Not more than 500 sectors may be specified to be loaded from a cassette. The cassette load read routines expect to find records of exactly 256 bytes on the tape for at least as many records as there are sectors to be loaded. If a record that does not meet the specifications is encountered before the last sector has been loaded, the cassette load will abort with the message

BAD DUMP TAPE

and return control to the primary command handler. It is not necessary that the records on the tape be written to the same disk addresses as from which they were read. Therefore, the CD and CL commands provide a means of moving sectors from place to place on one disk, or from one disk to another.

WARNING: Loading these sectors does not affect the C.A.T. Directory, or RIBs on a disk. Therefore, if the sectors are not loaded carefully into a matching file, they will be unallocated, unreferenced and probably cause FORMAT errors if read.

It is not necessary that a CL read all of the records that may be on a cassette, only that there are at least as many records on the cassette as there are sectors to be loaded. When the specified sectors have been loaded, the tape is rewound and the tape records are re-read and matched against the loaded sectors on the disk. If the data on the tape does not match the data on the disk, the cassette load routine will abort with the message:

TAPE/DISK VERIFY FAILURE

and exit to the primary command handler. If everything is correct, the cassette load routine rewinds the front tape and returns control to the primary command handler.

19.7 Drive Numbers

When DUMP93X0 begins execution it assumes that it is to deal with the disk in drive zero. The @ command instructs DUMP93X0 to deal with the disk in the specified physical drive.

COMMAND EXAMPLE:

@ 1

would mean: succeeding commands will refer to the disk in physical drive 1. The @ 1 command will remain in force until another @ command addresses a different physical drive. Note that the address parameter for the @ command consists of one and only one digit.

19.8 Error Messages

Some of the error messages produced by DUMP93X0 and their meanings are explained below.

PARAMETER ERROR

Occurs if an invalid command and/or disk address is given to the primary command handler. Note that all disk addresses must be expressed in octal.

SO MUCH ?

Occurs if a command is given to dump more than 10 cylinders to the printer. Note that one cylinder will fill 32 printer pages

(8 pages for 9350, 2 pages for 9380), and ten cylinders would represent a very large file. Respond "N" if you really don't want the printer to print out that many pages of paper. Otherwise, "Y" will cause the printing to proceed.

CASSETTE TOO SMALL

Occurs if a command is given to dump too many cylinders to cassette.

TAPE/DISK VERIFY FAILURE

Occurs during the tape-against-disk check phase of a cassette dump or cassette load if the data on the tape does not match exactly the data on disk. The tape is rewound and the dump or load should be retried.

BAD DUMP TAPE

Occurs if a tape record is read that does not conform to the DUMP93X0 tape record format. If it occurs during a cassette load, no data from the bad tape record is written to disk.

DISK NOT ON LINE

This message is self-explanatory.

DISK PROTECTED

Occurs if the disk is protected and a cassette load command is given. Nothing will be written to the disk as long as the READ ONLY indicator is on.

C.R.C. ERROR

Occurs if a hardware read or write error persists after three attempts to accomplish the read/write unless the read error occurs during a printer dump command (so that data on bad sectors can be hard-copy recorded and examined). If a C.R.C. error occurs during a printer dump, the machine will beep.

BEEP (Audio signal)

The machine will BEEP if an invalid command is entered from the keyboard. Also see C.R.C. ERROR.

SEEK INCOMPLETE

(9370 only)

This occurs if the disk controller SEEK INCOMPLETE status bit is set. This bit is set if a cylinder seek operation does not finish within 100 milliseconds. When this occurs, it generally indicates a hardware malfunction.

COMMAND ERROR

(9350 only)

This occurs if the disk controller COMMAND ERROR status bit is set. The DUMP9350 program should be reloaded if this happens. If it happens again, something is wrong with the processor, the I/O bus, the disk controller, or the disk drive.

SECTOR NOT FOUND

(9370 only)

This occurs if the disk controller SECTOR NOT FOUND status bit is set. This usually occurs as a result of the formatting information on a disk (as written by INIT9370) being incomplete or incorrect, but could also indicate a software or hardware malfunction.

(9350 only)

Same as COMMAND ERROR.

(9380 only)

Occurs if the disk controller SECTOR NOT FOUND status bit is set. This usually occurs as a result of the formatting information on a disk being incomplete or incorrect, but could also indicate a software or hardware malfunction.

CHAPTER 20. EDIT COMMAND

20.1 Introduction

The DOS Editor is used to create and to update source data files on the disk. The editor, through the use of initialization parameters, will enable the creation of files in a variety of formats: text files, assembler code files, DATABUS source code files, or many user designed data files.

A GLOSSARY of the many terms and phrases used throughout this chapter is provided in the Glossary at the end of the chapter. A list of commands and brief definitions is provided in the Command List Section. Caution: Although virtually any Datapoint format file may be "edited", files structured with respect to physical records or those containing strings longer than 79 characters may have this organization collapsed as the editor compresses the file into sequential format. In such cases the editor should not be used.

The editor does not truncate trailing blanks at the end of lines unless it is in "COMMENT" mode.

20.2 Operation

20.2.1 DOS Initialization

The EDIT program, is parameterized as follows:

EDIT <f1>[,<f2>][,<f3>][;parameter list]

20.2.2 Files

<f1> is the source file, [<f2>] is the scratch file and [<f3>] is the configuration overlay file. The source file <f1> is assumed to have an extension of 'TXT' if none is provided. If there is no file of the specified name, one will be created. If no scratch file [<f2>] is specified, a file 'SCRATCH/TXT' will be used. The configuration file [<f3>] is assumed to be EDIT/OV1

unless otherwise specified. The default extension for the configuration file is 'OV1'.

If parameters are indicated by the presence of the semi-colon, the question:

RECORD PARAMETERS?

will be displayed. If 'N' is entered, the editor will begin execution with the indicated parameters and the configuration file will not be changed. If 'Y' is entered, the question:

NEW TABS?

will be asked. If 'Y' is entered, the standard tab initialization line of numbers will be displayed (see :T command description). After the new tabs are entered, the parameter information and tabstops are recorded in [<f3>].

If no parameter list is provided, [<f3>], if present, is automatically loaded, causing the recorded parameters to be used.

20.2.3 Parameter List

A parameter list, indicated by the SEMI-COLON (;) following the file specifications may be included. That list may include up to seven parameters which are order independent. The possible parameters are:

[;[margin][tab key][mode][shift][line][update][key-click]

If no parameter list is provided, Assembler mode with a margin at 75 and SPACE bar for tabbing is assumed.

20.2.3.1 Margin Bell

A number in the parameter list will be taken to be the margin designator; this causes the margin 'bell' to ring at the designated margin. (Text may always be input up to column 79 regardless of the margin setting.)

For Example ;30 will cause the bell to ring in column 30.

20.2.3.2 Tab Key Character

A tab key character encountered in the parameter list, i.e., a non-alpha, non-numeric, non-colon, will replace the assumed tab key character. (SPACE in Assembler, DATABUS and Comment mode, SEMI-COLON in Text mode.)

For example, ^ will cause the caret key (^) to replace the assumed character as the tab key.

20.2.3.3 Mode

A new set of assumptions will be used if one of the 'mode' parameters is set. If no mode is listed or 'A' is typed, Assembler mode will be used. DATABUS or DATAFORM (D) mode simply changes the tab stops. Comment mode (C) changes the nature of the DELETE and SCRATCH commands to facilitate adding or changing comments on assembly code files and also truncates trailing spaces.

Text mode (T) sets no tabstops, does no shift inversion and enables the word wrap around feature (see the glossary). To activate line truncation instead of word wrap around in Text mode, enter 'L' in the parameter list. To enable shift key inversion (see glossary) in Text mode, enter the parameter 'S' in the list. Text mode is especially useful for generating SCRIBE input files.

See the glossary for complete definitions of the various modes.

20.2.3.4 Update

During editing, the source file is transferred into the scratch file as the text is updated. The physical source file may be used as the scratch file as the edit proceeds. When the edit is terminated, the physical source file is updated.

To inhibit source file update, the 'ONE-PASS' parameter 'O' may be set in the parameter list. A flag is set which prevents writing on the physical source file. Then, at the completion of the edit, the scratch file will contain the updated information and the source file will be unchanged.

20.2.3.5 Key-click

If the 'K' parameter is set, a click will sound each time a key is struck.

20.2.4 Examples

To perform standard Assembler code editing, enter the command:

```
EDIT <source>
```

To edit a file for input to the text processor, SCRIBE, enter the command:

```
EDIT <source>;T
```

To change the margin bell to ring at column 35 (e.g. for labels) enter the command:

```
EDIT <source>;35T
```

The parameters would set the bell and use the Text mode assumptions. Note that the parameters are order independent; therefore, the command:

```
EDIT <source>;T35
```

would achieve the same results.

To generate a second, slightly different, file (without updating the original file), enter the command:

```
EDIT <source>,<new    e>;0T
```

If the file is Assembler instead of text, simply omit the 'T'; if DATABUS, replace y 'D'.

A second file, with the same name as <f1> but with a different extension, may be used as the scratch file by entering:

```
EDIT <f1>,/<extension>
```

Once the initial command (and parameter list) has been entered, the DOS Editor signon message will appear on the screen. This message will be rolled up and the screen cleared with the cursor left on the 'command line'. From this position data may be

entered, lines may be fetched from the source file, or editor commands may be entered.

20.2.5 Data Entry

To enter text, simply type on the bottom line; when the ENTER key is pressed the screen rolls up one line. The command line is once again blank and the cursor is at the beginning of the command line, ready to accept more input.

If word wrap around is enabled, when a SPACE is typed within the last 10 columns of the line or typing proceeds past the end of the line, the editor automically will roll up the screen and begin a new line. If a non-space character is typed into the last column, the last word on the line is removed and, after the screen is rolled up, that word is placed on the command line, where data entry may proceed.

When typing on a 'screen line' (as the result of a command), the ENTER key causes the cursor to return to the command line. To continue data entry at the same screen area, the Pseudo-ENTER key may be used. This key (DEL shifted) causes (in all but command mode), a new blank line to be inserted at that point on the screen so that data entry may proceed.

If word wrap around is enabled, and data is being entered on a screen line, a new line will automatically be inserted at that point when, as on the bottom line, a space is entered within the last 10 columns or a character is typed past column 79.

The BACKSPACE key erases the last character and moves the cursor back one position. The CANCEL key erases the line back to the previous tabstop (this action would erase the entire line if no tabs are set).

Typing the tab key character causes the cursor to move to the next tab stop to the right. If there are no tab stops to the right of the cursor, the tab key character is accepted as a normal data character.

20.2.6 Data Retrieval

To fetch data from the source file, press the KEYBOARD and DISPLAY keys simultaneously. As long as the two keys are depressed, data will be fetched, displayed on the command line and rolled up the screen. If end of file is reached, no more data is fetched and the machine beeps.

To fetch a single line, the shifted DEL key may be pressed (in the first column of the command line). Using this key insures that only one input line will be fetched.

20.2.7 EDITOR Command Format

The text appearing on the eleven screen lines (i.e. the lines above the command line) may be edited using a set of 'commands'. A 'pointer' (>) in the left hand column of the screen indicates the line which the command will affect.

To move the pointer up, press the KEYBOARD key. To move the pointer down, press the DISPLAY key. The pointer wraps around from the top to the bottom and vice versa.

Commands allow the user to delete a single line (:D) or part of the screen (:SC and :SB), insert (:I) a new line between the current lines on the screen and modify (:M) parts of a line by replacing text or inserting new text. Commands are also available to search the file for specific text (:F and :L) or for the end of the file (:EO or :E*).

An editor command is always preceeded by a COLON (:). To enter a command, type, in the first column of the command line, a colon and the appropriate command character and any necessary parameters. The command is always typed with the machine in lower case; thus, with shift inversion on (as in Assembler, Databus and Comment modes), the command character will appear upper case; while with shift inversion off (as in Text mode), it will appear lower case.

20.3 Basic EDITOR Commands

The following commands are a few basic editor commands. The user can get started without worrying about complex command forms. Remember that the 'pointer' on the screen indicates the line affected by the command.

:D - DELETE - in all but Comment mode this command deletes the entire pointed line. (In Comment mode, only the comment field is deleted. The CANCEL key may however be used to delete the preceeding fields in the line.)

The cursor is left on the now null line where new text may be entered. If no replacement text is needed, pressing the ENTER key in the first column of the pointed line returns the cursor to the command line. Trailing blanks will not generally be truncated.

Pseudo-ENTER may be used to generate additional lines at this area of the screen. Word wrap around, if enabled, will apply to text entered on a deleted line. Pressing the ENTER key will return the cursor to the command line.

See the section on modification for more information about the pseudo-ENTER key.

:E* - EOF without display - searches for the end of the file and, when it is reached, displays the last eleven lines of text. The search may be aborted by pressing the KEYBOARD and DISPLAY keys simultaneously.

:EO - EOF with display - causes the data to be displayed on the screen continuously until end of file is reached. The search may be aborted at any time by pressing the KEYBOARD and DISPLAY keys simultaneously.

:F <old text> - FIND match - the screen is cleared and the input file is searched for a line starting with the specified <old text>. Leading spaces in <old text> are significant and should be entered if needed (note that this command should be typed exactly :F<SPACE><old text>).

A FIND will wrap entirely around the file (or up to the end of file if the one-pass option is set). If the requested text is not found, the last line on the screen when the FIND was executed will be displayed. A FIND may be aborted by pressing the KEYBOARD and DISPLAY keys simultaneously.

:I - INSERT - Perform a line insert at the pointed line.

This command causes the lines from the top of the screen to the pointed line, inclusive, to be rolled up and a blank line to be inserted. The cursor is left at the beginning of the new blank line where data entry may proceed.

If the pointed line or the line immediately below it is empty no insert will occur, and the null line will be used as the inserted line where data entry may proceed.

To make complex changes to a line already on the screen, the operator may INSERT a line immediately below the original and then retype the line - with changes. The original line may then be DELETED.

The pseudo-ENTER key may be used to generate additional lines at the same point on the screen.

:L - LOCATE next - typed exactly :L<ENTER>, clears the screen and finds the next line of text. If positioned at the end of the file, the 'next' line will be the first line of the file.

:L <old text> - LOCATE match - similar to FIND match except that the locate command searches for imbedded text matching <old text>. Leading spaces should be supplied if meaningful.

For additional forms of the FIND and LOCATE commands see the 'FILE SEARCH' section.

:M <old text><command separator><new text> - MODIFY - a modify command allows the operator to replace <old text> by <new text>, insert <new text> after <old text> or append (i.e., truncate and add) <new text> after <old text>. For the various forms of this command see the MODIFY Command section.

:SC - SCRATCH above - in all but Comment mode this command erases the lines from the top of the screen down to the pointed line, inclusive. (In Comment mode, only the comment fields are erased.)

The cursor is left on the pointed line where data entry may proceed.

:SB - SCRATCH below - in all but Comment mode this command erases the lines from the pointed line to the bottom of the screen, inclusive. (In Comment mode, only the comment fields are erased.)

The cursor is left on the pointed line, where data entry may

proceed.

:E - END - the end command causes the remainder of the logical source file to be copied to the logical scratch file and then, if the logical scratch is not the physical input file, the scratch file is copied back to the source file.

The command line will be left on the screen as long as the copy from source to scratch is in progress; it is erased during the final copy from scratch back to source.

The end may be aborted as long as the command line is still displayed, by pressing the KEYBOARD and DISPLAY keys simultaneously. When the final copy is completed, control is returned to DOS.

Note that if the one-pass option was selected in the parameter list, no copy from scratch back to source will be performed.

:E/ - END/DEL - this command causes the remainder of the source file to be deleted (the lines currently on the screen will be written out), and, if the logical scratch file is not the physical source file, the scratch file is copied back to the source file. When the file is completely updated, the system is reloaded.

No copy back is done if the one-pass option is set.

20.4 Modification Commands

Modification of a line may be achieved in a variety of ways. The DELETE command enables the user to remove leading information while the MODIFY command may be used to replace imbedded information, insert text into a line or field, or truncate and add new text at a specified point or in a specified field.

20.4.1 DELETE Command

:D <old text> - DELETE through - this command deletes all characters from the left edge of the pointed line through (and including) the specified <old text>. The remaining characters will be left justified and re-displayed. The cursor returns automatically to the command line.

20.4.2 MODIFY Command

The general form of the MODIFY command is:

:M[#] [old text]<sep>[new text]

where [#] is an optional number which extends the meaning of the command (see Field Modification below) and <sep> is the command separator which defines the action of the command. Both [old text] and [new text] fields are optional. If [old text] is omitted, the command will take effect at the left most edge of the pointed line (or at the left edge of the specified field). If the [new text] field is omitted, a null field will be used to execute the modification.

20.4.2.1 Line Modification

The following descriptions are of the line modification version of the MODIFY command

:M [old text] < [new text] - MODIFY (replace) - replace the specified [old text] by the specified [new text]. The less than character (<) is a command separator which indicates replacement and, therefore, the [old text] may not contain this character. If [new text] field is omitted, the old text will simply be deleted and the line will be compressed to the left.

For example, to modify the text line:

THE QUICK BROWN FOX JUMPED OVER THE LAZY DOG'S BACK.

The command: :M BROWN<RED would cause the line to be redisplayed like this:

THE QUICK RED FOX JUMPED OVER THE LAZY DOG'S BACK.

The command: :M .< 1234 TIMES. to the original line would generate a line like:

THE QUICK BROWN FOX JUMPED OVER THE LAZY DOG'S BACK 1234 TIMES.

If the replacement causes the line to become longer than 79 characters, the trailing word, in text mode only, will be wrapped around and a new line will be inserted containing the entire last word. If the [new text] is shorter than the [old text] it replaces, the line will be shortened.

After the pointed line is redisplayed, the cursor is returned to the command line.

:M [old text] > [new text] - MODIFY (insert) - the command separator greater than (>) causes the [new text] to be inserted in the pointed line immediately after the [old text].

If the line becomes longer than 79 characters, and word wrap around is not in effect, the trailing characters are truncated. If, however, word wrap around is on, the trailing characters and last word are inserted on a new line.

:M [old text] \ [new text] or

:M [old text] ¦ [new text] - MODIFY (append) - the vertical bar (¦) or backslash (\) command separators cause everything in the pointed line, past the [old text], to be replaced by the [new text].

As in all MODIFY commands, if the pointed line becomes longer than 79 characters, truncation occurs if word wrap around is not enabled.

:M - MODIFY repeat - typed exactly :M<ENTER>, uses the <old text> <sep> <new text> from the last MODIFY command. This is useful when making the same change repeatedly.

:M* - MODIFY display - display the expression entered for the last MODIFY. After the saved command is displayed, the cursor is turned off and the operator must press ENTER to proceed. No MODIFY is actually performed.

20.4.2.2 Field Modification

In field modification mode, the MODIFY command acts only on a specific field and does not expand or contract the entire line but maintains the integrity of all fields before and after the affected field.

A field is the area between two consecutive tabs. Field one is between the left margin and the first tab.

:M<#> [old text]<sep>[new text] - MODIFY field - where the pound sign <#> is a number from 1 to 10 designating the field to be modified (or the starting point to search for matching [old text]). In Assembler mode, field 1 is the label, field 2 is the op code, field 3 is the expression and field 4 is the comment. This

command may be executed in any of the previous Modify forms. However, modification is performed within the specified field only. As long as the text being modified is unique, field 1 may be specified, since the field number indicates only where to start looking for matching text. (Note that if the field number is omitted, line modification is assumed.)

Thus, a replacement or append shorter than the original field will be blank filled and subsequent fields will maintain their position and content. An insertion longer than the specified field will be truncated (with the exception of the last field whenever word wrap around is in effect).

For example, in Assembler mode, the line:

```
LABEL     OP       EXP             COMMENT
```

the label may be deleted by the command:

```
:M1 \
```

with the resultant line:

```
          OP       EXP             COMMENT
```

Or, the expression field (EXP) could be changed to EXP+1 without disturbing the comment field position, by the command:

```
:M3 EXP>+1
```

which generates:

```
LABEL     OP       EXP+1           COMMENT
```

To add a comment to a line previously containing none or to replace an existing comment field, enter:

```
:M4 \<new comment>
```

NOTE: When using the repeat form of the MODIFY command, the field number may need to be supplied. The field number is not saved with the rest of the modify expression, as can be seen from the :M* display.

20.5 File Search Commands

The FIND and LOCATE commands have several forms and have been separated from the basic command set to better describe them.

Manual, operator controlled, searches may be performed by depressing the KEYBOARD and DISPLAY keys simultaneously to cause data to be fetched from the file and displayed (as long as the keys are pressed) on the screen. To fetch a single line use the Pseudo-ENTER key (DEL shifted). The :EO command performs the same function automatically, i.e., it causes lines to be fetched and displayed until the end of file is reached. To abort a :EO command, press the KEYBOARD and DISPLAY keys simultaneously.

To find the end of a file without displaying the entire file (since the display is time consuming) use the :E* command. This will search for the end of file and display the last eleven lines of data.

:F <old text> - FIND match - the screen is cleared and the input file is searched for a line starting with the specified <old text>. Leading spaces in <old text> are significant and should be entered if needed. (Note that this command should be typed exactly :F<SPACE><old text>).

A FIND will wrap entirely around the file (or up to the end of file if the one-pass option is set). If the requested text is not found, the last line on the screen when the FIND was executed will be displayed. A FIND may be aborted by pressing the KEYBOARD and DISPLAY keys simultaneously.

The <old text> specified for a FIND (or LOCATE) command is saved. The saved match may be redisplayed or used again.

:F<SPACE> - FIND same match - if the FIND command is followed by exactly one space and the ENTER key, the previous FIND (or LOCATE) <old text> will be used for this FIND. Several occurrences of the same text may be searched out in this manner.

:F* - FIND display - the asterisk (*) after the FIND command causes the <old text> of the previous FIND or LOCATE command to be displayed. The cursor is turned off and the operator must press ENTER to proceed. No FIND is performed.

:L - LOCATE next - typed exactly :L<ENTER>, clears the screen and finds the next line of text. If positioned at the end of the file, the 'next' line will be the first line of the file.

:L <old text> - LOCATE match - similar to FIND match except that the locate command searches for imbedded text matching <old text>. Leading spaces should be supplied if meaningful.

:L<SPACE> - LOCATE same match - typed exactly :L<SPACE><ENTER>, uses the <old text> specified by either the previous LOCATE or FIND command to perform a search.

:L* - LOCATE display - display the <old text> entered for the previous LOCATE or FIND command. As in the FIND display, the cursor is turned off and the operator must press ENTER to continue. No LOCATE is actually performed.

20.6 Miscellaneous Commands

:A - APPEND - copies the pointed line to the bottom of the screen and rolls the screen up one line.

:B - BYPASS - fetch a line from the file, bypassing end of file or record format error (which would normally be treated as an end of file). Subsequent lines (if not also record format errors) may then be fetched by the normal mechanisms. This command is intended as a recovery tool for use only if the file has been accidentally shortened or contains badly formatted records.

:C - COPY - copies the pointed line to the bottom of the screen, deletes the pointed line and rolls the screen up one line. This command cannot be executed on the top screen line.

The cursor is left on the now null pointed line. Text may be entered at this point (the Pseudo-ENTER and word wrap around, if enabled, will apply). When the ENTER key is finally pressed, the pointer is automatically moved to the following screen line so that a group of lines may be easily copied to another part of the screen.

:T - TAB set - this command enables the user to reset the tab stops during execution. (Not available in Comment mode.) The command causes a line of numbers to be displayed across the bottom of the screen.

The operator should space over to each position where a tabstop is desired and type any non-blank character. These tab stops are meaningful during data entry and field modification (:M#) since data within a field may be modified without disturbing the rest of the line. A maximum of 10 tab stops may be set.

:RH - RPG Header - sets tab stops for RPG header specification at columns 6 and 15.

:RF - RPG File - sets tab stops for RPG file description specification at columns 6, 15, 24, 33, 40, 54, 66 and 70.

:RE - RPG Extension - sets tab stops for RPG extension specification at columns 6, 11, 19, 27, 33, 36, 40, 46, 52 and 58.

:RL - RPG Line - sets tab stops for RPG line counter specification at columns 6, 15 and 20.

:RI - RPG Input - sets tab stops for RPG input specification at columns 6, 15, 21, 44, 53, 59 and 65.

:RC - RPG Calculation - sets tab stops for RPG calculation specification at columns 6, 18, 28, 33, 43, 49, 54 and 60.

:RO - RPG Output - sets tab stops for RPG output specification at columns 6, 15, 23, 32, 38, 40 and 45.

:RS - RPG Summary - sets tab stops for RPG summary specification at columns 6, 14 and 23.

:X - TEXT - this command enables word wrap around and disables shift key inversion and space insertion after leading periods. It automatically enters the tab set command (:T), so that tab stops may be cleared by the operator. The tab key character is not changed; therefore, the :<tab key> command must be used to set a new tab key character if one is desired.

:<tab key> - change tab key character to any non-alpha, non-numeric, non-COLON, non-ENTER character typed after a leading colon on the command line.

20.7 Recovery Procedures

A 'FORMAT TRAP' occurs when a record not belonging to the current file is encountered. This can be caused either by a physical misalignment of the disk read head or because a record has erroneously been written into that file by some other program.

A 'RANGE TRAP' occurs when the physical limit of the file is reached and no end of file is present.

20.7.1 Bypassing Errors or End of File

When a format or range error occurs, an appropriate message appears on the command line and the cursor is turned off. In order to proceed, the operator must first press the DISPLAY key. The effect of either a format or range trap is the same as an end of file and no further data will be read from the file.

To read past a format error or past an end of file, use the BYPASS command, :B, repeatedly if necessary.

20.7.2 File Recovery

If the source file is lost (e.g., erroneously KILLed), the scratch file may contain a useful copy. Since the scratch file (SCRATCH/TXT) usually contains a copy of the last file edited, it may be used to recover only that file.

20.8 Glossary

Assembler mode - assumed mode of execution. Tab stops at 9, 15 and 30 (may be changed during execution). The space bar is assumed as the tab key character (this may be changed in parameter list or during execution). Shift key inversion and no word wrap around are assumed. Leading period (.) generates period space (.) for comment lines. Pseudo-ENTER does line-insert.

Command - characters typed at the left edge of the command line following a COLON (:) which have special meaning to the editor.

Command line - the twelfth line of the screen where most data is entered, lines are fetched and commands are typed.

Command separator - the character in a MODIFY command which indicates what is to be done (> means insert, < means replace and \ or ¦ mean append).

Comment field - in assembler code the area of the screen from columns 30 to 79 which is generally used for programmer comments.

Comment mode - assumed if 'C' in parameter list. Facilitates changing or adding comments to assembler code. Tab stops

at 9, 15 and 30 (may not be changed during execution). The space bar is assumed to be the tab key character (this may be changed in parameter list or during execution). Shift key inversion and no word wrap around are assumed. Leading period (.) generates period space (.) for comment lines. Pseudo-ENTER positions to comment field of following line and deletes the comment. Delete and Scratch commands affect only the comment field. Trailing blanks are truncated when data is output.

CONFIGURATION FILE - A file, default name of EDIT/OV1, which automatically provides default options to EDIT.

DATABUS mode - assumed if 'D' in parameter list. Tab stops at 9 and 15 (may be changed during execution). The space bar is assumed to be the tab key character (this may be changed in the parameter list or during execution). Shift key inversion and no word wrap around are assumed. Leading period (.) generates period space (.) for comment lines. Pseudo-ENTER does line-insert. Input lines are blank filled and trailing blanks are truncated on output.

Field number - a digit used in the MODIFY command to designate characters between two tab stops. Field '1' is always from column 1 to the first tabstop; thus, in Assembler mode, '1' designates the label field, '2' the opcode field, '3' the expression field and '4' the comment field. During field modification, trailing fields are preserved.

Format trap - bad record encountered on disk. See 'Recovery Procedures'.

Line insert - results from an INSERT command, data entry or modification when word wrap around is in effect or a Pseudo-ENTER key in any mode other than Comment. The lines above the pointed line are rolled up and a new, blank line is generated at the pointed line.

Logical scratch file - current output file.

Logical source file - current input file.

New text - a group of characters, typed immediately after a command separator in a modify command, which will become part of the line being modified.

Old text - a group of characters, including spaces, which are searched for, either in the pointed line (as in the MODIFY command) or in the file (as in the FIND or LOCATE commands).

One-pass option - assumed if 'O' in parameter list. The one-pass option does not update the physical source file. The FIND, LOCATE and END, END/DEL commands will not write back into the input file if this option is set.

Parameter list - initialization information provided when the editor is first executed. Following file specifications, a SEMI-COLON (;) indicates the presence of a parameter list. The mode, one-pass option, tab character, margin bell column and (in text mode) 'no shift inversion' (S) and 'no word wrap around' (L) may be set.

Pointed line - a pointer (>) in the left hand margin is used to reference lines for modification by command. The line to the right of the pointer is the pointed line.

Physical scratch file - specified (or implied SCRATCH/TXT) output file.

Physical source file - specified input file.

Pseudo-ENTER - the key marked DEL (always shifted) is referred to as the Pseudo-ENTER key. If pressed in the first column of the command line, one line of text will be fetched from the source file.

In comment mode, if pressed on any but the bottom screen line or command line, it will cause the cursor to be positioned to the comment field of the following line and that field will be erased.

In all other modes, the Pseudo-ENTER key causes a new line to be inserted so that data entry may proceed in the same area of the screen. If pressed on the last screen line, the Pseudo-ENTER key simply places the cursor on the command line.

Range trap - attempt to read past the end of allocated space on the input file - see 'Recovery Procedures' in the previous section.

Scratch file - at any point in time, the logical scratch file is the output file. It may, however, physically be the

original input or the assigned 'scratch' file.

Screen line - any of the eleven lines on the screen which may be referenced by the command pointer. The command line is not, therefore, included.

Shift key inversion - reverse the function of the shift key for all alpha characters so that, in lower case, alpha characters will appear upper case.

Source file - originally this is the input file specified at initial execution. The term source file refers to the current input file; thus, at any point in time, the logical source file may be either the specified input file or the file specified as the scratch file.

Text mode - assumed by a 'T' in the parameter list. No tab stops are set (tabs may be set during execution). The SEMI-COLON (;) is the assumed tab character (the tab key character may be changed in the parameter list or during execution). No shift key inversion is performed (this may be selected in the parameter list). Word wrap around is performed (this feature may be turned off by an 'L' in the parameter list).

Word - a word is defined as any group of less than 50 characters preceeded by a space.

Word wrap around - a feature of text mode. During data entry a space within the last 10 columns of the screen causes an immediate carriage return. If this occurs on a screen line, a line insert is performed so that data entry may proceed at the same area of the screen. If a character is typed over the last column of the screen, the last word is removed, a line insert performed and the removed word is placed at the beginning of the inserted line where data entry may proceed. If a modify command causes the line to become longer than 79 characters, the trailing characters, including the last word on the line, will be moved to a new line which will be inserted below the original line. Control will then return to the command line.

20.9 Command List

:A	APPEND pointed line to command line and roll up
:B	BYPASS end of file
:C	COPY pointed line to command line and roll up
:D	DELETE entire line
:D <old text>	DELETE from left thru <old text>
:E	END edit - copy remainder of file and update source
:EO	EOF display - fetch and display data until end of file
:E/	END/DELETE update without copying remainder
:E*	EOF search - find end of file and display last full screen
:F <old text>	FIND match - search file for matching leading text
:F<SPACE>	FIND repeat - use previous find/locate <old text>
:F*	FIND display - display previous find/locate <old text>
:I	INSERT a blank line below pointed line
:L	LOCATE next - clear screen and get next line
:L <old text>	LOCATE match - search file for matching imbedded text
:L<SPACE>	LOCATE repeat - use previous find/locate <old text>
:L*	LOCATE display - display previous find/locate <old text>

LINE MODIFICATION

:M [old text]<[new text] - MODIFY replace old text by new text, adjusting the entire line

:M [old text]>[new text] - MODIFY insert new text after old text, adjusting the entire line

:M [old text]\[new text] or :M [old text]¦[new text] - MODIFY append new text after old text adjusting the entire line

FIELD MODIFICATION

:M<#> [old text]<[new text] - field MODIFY replace old text within specified field with new text without disturbing the remainder of the line.

:M<#> [old text]>[new text] - field MODIFY insert old text after new text within specified field, without disturbing the remainder of the line.

:M<#> [old text]\[new text] or :M<#> [old text]¦[new text] - field MODIFY append the new text after the old text within the specified field without disturbing the remainder of the line.

:M* MODIFY display the previous modify [old]<sep>[new]

:M[#] field MODIFY repeats the previous modify [old]<sep>[new]

:RH RPG HEADER - sets tab stops at columns 6 and 15.

:RF RPG FILE - sets the stops at columns 15, 24, 33, 40, 54, 66, and 70.

:RE RPG EXTENSION - sets tab stops at columns 6, 11, 19, 27, 33, 36, 40, 46, 52, and 58.

:RL RPG LINE - sets tab stops at columns 6, 15, and 20.

:RI RPG INPUT - sets tab stops at columns 6, 15, 21, 44, 53, 59, and 65.

:RC RPG CALCULATIONS - sets atab stops at columns 6, 18, 28, 33, 43, 49, 54, and 60.

:RO RPG OUTPUT - sets tab stops at columns 6, 15, 23, 32, 38, 40, and 45.

:RS	RPG SUMMARY - sets tab stops at columns 6, 14, and 23.
:SB	SCRATCH BELOW deletes the pointed line and all screen lines below it
:SC	SCRATCH ABOVE deletes the pointed line and all screen lines above it
:T	TAB SET permits the user to set up to ten tab stops
:X	TEXT mode switches to text mode with word wrap around and no shift key inversion.
:<character>	changes the tab key character to <character>.

CHAPTER 21. ENCODE/DECODE COMMANDS

21.1 Purpose

The ENCODE command is used to convert disk files containing data in any format into 79 character records containing only ASCII characters. Data in encoded format can be copied or transmitted by all Datapoint programs.

The DECODE command is used to translate encoded data files back into an exact duplicates of the original disk files.

21.2 Use

ENCODE <file spec>,[<file spec>]

The ENCODE command converts the first file into encoded format and writes the data into the second file. If extensions are not supplied, ABS is assumed for the first file and ENC is assumed for the second file. If the second file is not specified, the name of the first file with an extension of ENC is assumed. The second file will be created if it does not already exist. Encoded data creates a file 50 percent larger than the original.

DECODE <file spec>,[<file spec>]

The DECODE command converts the first file from encoded format back into binary and writes the data into the second file. If extensions are not supplied, ENC is assumed for the first file and ABS is assumed for the second file. If the second file is not specified, the name of the first file with an extension of ABS is assumed. The second file will be created if it does not already exist.

INPUT FILE MUST BE SPECIFIED!

will be displayed if the first file specification is omitted.

INPUT FILE DOES NOT EXIST!

will be displayed if the first file specified cannot be found in the DOS directory.

OUTPUT WOULD DESTROY INPUT FILE!

will be displayed if the first and second file specifications are identical.

INPUT FILE CONTAINS BAD DATA!

will be displayed if an encoded data file cannot be decoded into its original binary form.

ENCODE reads and converts binary data until either a valid text end-of-file is read or allocated file space is exausted. Data in encoded form is always terminated with a valid text end-of-file.

CHAPTER 22. FILES COMMAND

FILES is a program which selectively prints or displays DOS file descriptions in file name sequence.

One may select information pertaining to all DOS files or to only those files with names and/or extensions beginning with the characters specified by the operator. Selected directory entries are sorted into ascending file name sequence. If desired, information from associated Retrieval Information Blocks (described in the chapter on System Structure) is also extracted for each directory entry. Extracted data is interpreted and displayed on the screen, listed on a Local or Servo printer, or written to a disk file.

22.1 Command Description

To execute the FILES program, type in the name FILES followed by selection criteria and display options (if option codes are to be used):

FILES [<filename>][/<ext>][:DRn],[<subdir>] [,<output-file>][;optio

<filename>	Select entries for files with names beginning with the 1-8 characters specified.
<ext>:	Select entries for files with name extensions starting with the 1-3 characters specified.
:DRn	Specifies the disk drive to be selected. If this field is omitted, drive 0 will be selected.
<subdir>	Specifies the named subdirectory from which to select entries.
<output-file>	Specifies the disk file to which the selected entries will be written, if disk file output is specified.

options: The following option codes are available, and may be entered in any order:

N - Suppress file allocation map.
D - Display on CRT.
L - List on local printer.
S - List on servo printer.
F - Write output to disk as DOS text-type file.

If options are keyed and D, L, S and F are omitted, then D is assumed. D, L, S, and F options are mutually exclusive; output can be sent to only one device. If F is keyed and the <output file spec> is not present in the command line, one is requested by the message:

DOS OUTPUT FILE SPEC:

22.2 Default Messages

If no option codes are entered, the following messages will be displayed on the CRT:

SUPPRESS FILE ALLOCATION MAP?

If "Y" or "YES" is entered in response to this message, the display of file allocation information from Retrieval Information Blocks (RIB) will be suppressed. If any other response is entered, file allocation information will be displayed for each selected file.

After the user has replied to the map selection message, the program will test to see if the there is a servo printer connected to the processor. If a servo printer is attached and ready, the following message will be displayed:

LIST ON SERVO PRINTER?

If the user enters a "Y" or "YES" in response to this message, the servo printer will be selected to display output. If any other response is entered or the program cannot find an available servo printer, the program will test to see if a local printer is connected and ready for printing. If the program finds

that a local printer is available, the following message will be displayed:

```
LIST ON LOCAL PRINTER?
```

If the user enters "Y" or "YES" in response to this message, the local printer will be selected for output. If a printer has been selected for output, the following message will be displayed:

```
ENTER HEADING:
```

Up to 32 characters can be entered, which will be displayed at the top of each page of printed output.

If no printer is available, or if the operator has rejected printer output, the program will ask for disk output:

```
WRITE OUTPUT ON DISK?
```

If the user enters "Y" or "YES", output will be written to a disk file, otherwise output will be displayed on the CRT. If disk output is selected, an output file name will be requested unless one was provided on the command line.

22.3 File Descriptions

File descriptions are sorted into ascending file name sequence for easy reference and displayed or printed in the following format:

```
FILENAME/EXT (PFN) DW
```

DW flags following the Physical File Number (PFN) indicate if the file is delete protected (D), or write protected (W). If the file allocation map was not suppressed, messages describing the file's size and location will be included in the file description. When allocation map information is printed or displayed, the program displays totals lines specifying the total number of files listed and the total number of sectors in those files. Disk output never has totals lines.

Depressing the DISPLAY key during display or printing of file descriptions will cause the program to pause until the key is released. Depressing the KEYBOARD key will cause the program to terminate and return control to the operating system.

Allocation map information describes each segment in the file

by giving the cylinder and cluster starting address of the segment and its length in sectors. One line is displayed for each segment. See the chapter on System Structure and the Appendix for the appropriate DOS for a description of disk space allocation.

22.4 Error Messages

* PARITY ERROR *

FILES can not continue due to an irrecoverable parity error encountered while trying to read data from the disk.

* DRIVE OFFLINE *

FILES is unable to connect to the disk drive selected by the operator (drive 0 if not otherwise specified).

FILE(S) NOT FOUND.

No Directory entries have been found that meet the user's selection criteria.

INVALID DRIVE

An invalid drive specification was entered.

CONFLICTING OPTIONS SPECIFIED

Options specify output on more than one device.

UNRECOGNIZABLE OPTION CODE

An unrecognizable code has been entered in the option field.

PRINTER NOT AVAILABLE

An option code specifies a printer that does not respond when tested for status.

CHAPTER 23. FIX COMMAND

23.1 Purpose

The FIX program can be used to modify bytes of DOS-loadable object code in an absolute code file. This program can be very dangerous and should be used only by qualified assembler language programmers or by someone folowing specific directions provided by Datapoint.

23.2 Operation

To invoke FIX, enter the command:

FIX <file spec>

The program will display a sign-on message and will then display an initial line of six zeros, two spaces, and three more zeros on the bottom CRT line. (The zeros represent the current address and its contents.)

000000 000

The screen is then rolled up. The program then waits for a command from the operator. The <file spec> must specify a DOS-loadable object file. If no extension is provided, /ABS is assumed.

Commands are in the form [number][character] where the number is assumed to be octal. If the number is omitted, a value of zero is used. Commands are terminated by the enter key. Following a command, the current address and its contents are re-displayed.

23.3 Commands

The following is a list of command characters with their effect:

ENTER - Set current address.

- If no block of object code is currently in

memory (as at the beginning of execution or after a block has been rewritten), search the object file forward until a block containing the given location is found, then display the contents of that location. If the address does not exist in the object file, the current address is left at zero.

- If a block of code is in memory and the location given is within the limits of the block, the contents of the location will be displayed.

- If a block is in memory and the location given is not within the block limits, the current address will be set to the minimum or maximum address of that block, its contents will be displayed and a beep will sound. To access the desired address the current block must first be aborted (A) or transferred (T).

M - Change the contents of the current address to the number given.

I - Increment the current address (up to the maximum address in the current block).

. - Change contents of current address to number given and automatically increment the current address and display the contents of the resulting location.

D - Decrement the current address (down to the minimum address in the current block).

T - Transfer the modified block back to disk - rewriting it in place. After the block is written, the current address is set back to zero, so that all searches always start from the beginning of the file. No modification is made to the stored file until a T command is executed.

A - Abort processing the current block, set the current address back to zero.

O OR * - Return to the operating system - if there is a block of object code in memory, it is <u>not</u> written back into the file.

If the command character is not one of the above, it is ignored and regarded as if only the ENTER KEY had been pressed.

23.4 Error Messages

If the <filespec> is not an absolute object code file, the message

RECORD FORMAT ERROR

is displayed.

If the file specified on the command line is not found, the message

NO SUCH NAME

is displayed.

CHAPTER 24. FREE COMMAND

24.1 Purpose

As a disk becomes full, it is useful to know how many 256-byte sectors remain available for allocation. Another useful bit of knowledge on the larger disks is how many empty slots in the directory remain for the allocation of file names. The FREE command displays these two values.

24.2 Use

The FREE command accepts a drive specification. It may be entered simply as:

```
FREE
```

which will cause the FREE space and files for all the on-line drives to be displayed. It may also be entered as:

```
FREE :<drive spec>
```

which will display the FREE space and files for only drive n.

The command scans all drives that it finds on-line and displays (1) the number of available file names (representing possible files to be created) and (2) the number of available sectors that it finds on each.

Holding down the DISPLAY key will cause FREE to pause. Pressing the KEYBOARD key will cause FREE to terminate and return to the operating system.

CHAPTER 25. INDEX COMMAND

25.1 Introductionn

The DOS INDEX command (with the DOS SORT Command) is used to create the tree structure required by programs using the indexed sequential access method (ISAM), to create a Keytag file from the INDEX file, to create the INDEX file from a Keytag file, or to recreate the INDEX file.

The INDEX command has the capability of creating index files from any DOS text-type files. The indexed access method can then rapidly access records in this file either in sequential or random order. Records in files to be indexed must contain a record key up to 118 characters long contained in the first 249 bytes of each record.

It is possible to build many independent indices to permit access to records of the same file by many separate, unrelated keys. There are no restrictions on the number of indices that may be built, or on the relationship or lack of relationship among the various keys used.

25.2 System Requirements

INDEX runs under the DOS operating system. In addition, INDEX uses the DOS SORT command, which must be resident on an online disk at the time INDEX is used. If the INDEX command is to pre-process the text file, the REFORMAT command must be available. (See the Section on PREPROCESSING the file). If the INDEX command is to be used to recreate the tree structure file, the NAME command must be available.

If possible, INDEX will invoke the FASTSORT program, instead of the normal DOS SORT. Only if a 5500 processor is in use and PS is not active will INDEX look for FASTSORT. Under any other conditions it loads SORT. If FASTSORT is not available, INDEX uses SORT. FASTSORT is released separately.

25.3 Operation

When the Index command is to be executed, the operator must enter:

INDEX <filespec>[,<filespec>][,<filespec>][,<drive>];<parameters>

where only the first file specification and key field description are mandatory, and specify the file to be indexed. Default extension is /TXT. The second file specification is the name of the INDEX file to be created. If no file is specified, the name of the first file is used with default extension of /ISI. If no drive is specified, the INDEX file will be placed on the same drive as the file to be indexed. INDEX files may have any names at all - and be located on physically different drives from the file being indexed. However, high-level languages using ISAM files (DATABUS, for example) assume the INDEX file will have the normal /ISI extension, and if the file open is drive directed the /ISI and /TXT files must be on the same drive.

The third file specification is for the intermediate tag file. The third file name will also default to the name of the first file with a default extension of /TAG. The fourth file specification, which may only specify drive, tells SORT where to put its intermediate work files. Otherwise, SORT will attempt to optimize drive selection.

25.3.1 Parameters

In addition to the parameters that INDEX itself recognizes, the user may specify any parameters acceptable to the REFORMAT utility (if preprocessing is to be done), or a primary record specification to be passed to SORT, or Mnnn or Q options to FASTSORT. Parameters recognized by INDEX are as follows:

K -- Create a Keytag file from the /ISI file.
I -- Create an /ISI file from the Keytag file.
X -- Recreate the /ISI file, handling insertions and deletions.
F -- Preprocess the input file with REFORMAT.
E -- Index in EBCDIC collating sequence.
mmm-nnn -- Key specification

The Keytag file is a standard text file containing the pointer and key of each record to be indexed. The format is explained in the SORT chapter. The file may be LISTed, EDITed or transmitted. This last feature allows the /ISI file to be created

at a remote site without invoking SORT.

The format of the key is mmm-nnn [,mmm-nnn] [,mmm-nnn]... where mmm is the beginning character position of the key field in each logical record and nnn is the ending position of the key field. Note that each record must have a unique key.

The primary record specification is an option that allows the user to create the ISAM index file from a subset of the data file. The format of the primary record specification is PNNNTC. The P must always appear. The field following P, denoted by NNN, represents the column in each logical record where a one position field exists that differentiates records in the file. The location of this one character field must be less than or equal to 249. The T can have one of two values. It can be either an equal sign (=) or a pound sign (#). If the former, it means create the ISAM index file from all records that contain the ASCII character C in position NNN. If it is a pound sign, it means that the ISAM file will be created from all records that do not contain the value of C in position NNN.

In general the parameters for INDEX can be specified in any order and may optionally be separated from each other by a blank or a comma. The only exception to this is when a primary record specification exists, it must precede the key field specification and be separated from the key by a blank or a comma.

25.4 Choosing A Record Key

Since the speed of access to an indexed file varies according to how much file space and thus how many levels of index are required for the index tree, the choice of what to use for a record key becomes highly important. Of course, you must choose a key which will uniquely determine the record you wish to access, but you should scrupulously avoid including information in the key which is not absolutely necessary. For example, a file could be keyed according to automobile license plate numbers. Typically, these numbers will include a hyphen or other punctuation, which could easily be excluded from the record's key. The indexed access method will perform more efficiently if all non-significant characters are removed from the record's key.

25.5 Preprocessing the File

In file structures such as an indexed file where records are randomly inserted and deleted, the file tends to become non-optimum for searching. In addition, due to the method with which the indexed access method inserts records, each inserted record exists in a separate disk sector. This means that for records that are 80 characters long, two-thirds of the disk space for each additional record is wasted. This results in a reduction of the performance of the indexed access method.

In order to reclaim space vacated by deleted records and padding bytes in inserted records, the file may be processed by the REFORMAT utility prior to indexing.

25.5.1 Invoking Reformat

The INDEX utility will automatically invoke REFORMAT if the "F" option is present when INDEX is invoked. You must have specified the options that REFORMAT will need to process the file.

Note that if multiple indices are to be created, reformatting need only be specified for the first INDEX step, and MUST not be specified later if it was not specified in the first step. Although REFORMAT will not destroy the file, specifying reformatting may invalidate any previously built indices.

Basically, you must tell REFORMAT what format the records of the file are to have after preprocessing. You may select record compression, space and record compression, or blocking. Since the reformatting is done in-place, the REFORMAT option cannot enlarge the file which is to be indexed. For additional details on the REFORMAT utility, see the REFORMAT section of this guide.

25.5.2 Considerations for Unattended Indexing

Users who use the INDEX command from a CHAIN file (see the section on the CHAIN command for more details) and used AUTOKEY to restart their chain in the event of a failure should generally avoid using REFORMAT directly from INDEX. The reason why is that REFORMAT as invoked by INDEX uses the REFORMAT-in-place mode of the REFORMAT command. (The reason for this is that it is faster to do so, and also allows the indexing with reformatting of a file which is too big to REFORMAT in the available scratch space on a single-drive, almost full disk). Although REFORMAT is very careful not to damage the file being processed, if the file is

actually in the process of being reformatted when a power failure occurs, the results can be undesirable.

This potential problem during unattended INDEX chaining can be avoided by setting a checkpoint (see the AUTOKEY command description for details), copying the original file to a scratch file, setting another checkpoint, reformatting the scratch file back into the original (using the COPY mode of REFORMAT), setting a further checkpoint, and finally INDEXing the file using INDEX. In this way there is always an undamaged file with which execution can resume if necessary.

25.6 INDEX Messages

The Index command displays several messages on the operator's console. They are listed below with explanations, in the sequence in which they may appear.

DOS. VER 2 INDEX COMMAND - date
: This is the signon message that gives the user the version of DOS required and the date of the INDEX command.

INFILE NAME MISSING.
: This indicates that the user has omitted the first, and required, file specification.

KEYTAG FILE BEING BUILT.
: This indicates that INDEX is now creating the ASCII KEYTAG file requested with the "K" option.

SORT COMMAND MISSING.
: This indicates that INDEX needs to invoke the SORT command but could not find it on any of the on-line drives.

FILE PREPROCESSING WILL BE DONE BY REFORMAT COMMAND.
: This indicates that the user has requested preprocessing of his file by the REFORMAT command.

REFORMAT COMMAND MISSING.
: This indicates that INDEX needs to invoke the REFORMAT command but could not find it on any of the on-line drives.

REFORMAT COMMAND LINE:

This is the parameter list passed to the REFORMAT command.

INDEX WILL USE EBCDIC SORT.
The user has requested an index using the EBCDIC collating sequence.

SORT COMMAND LINE:
This is the parameter list passed to the SORT command.

REFORMAT UNLOADABLE!
This indicates that there is something wrong with the REFORMAT command object file. It needs to be reloaded.

SORT UNLOADABLE!
This indicates that there is something wrong with the SORT command object file. It needs to be reloaded.

BUILDING LOWEST LEVEL INDEX.
This indicates that INDEX is now creating the lowest level of the index file.

NULL INDEX FILE CREATED.
This indicates that an empty tag file was created by SORT. The index file created is usable by programs using ISAM for adding records.

LONG KEY ENCOUNTERED AND TRUNCATED.
This indicates that the tag file contained a key that was longer than 118 characters. It was truncated to 118 characters.

DUPLICATE KEY: <key>
Two keys in the tag file were found to be identical and the first 60 characters of the key are displayed. INDEX will continue so as to display any other duplicate keys that may be found.

INDEX TERMINATED BY DUPLICATE KEYS.
Duplicate keys have been found and so the index file has been deleted. The tag file is not deleted and since it is in standard text format, it may be EDITed to remove or modify the duplicate key and tag. Or a program (e.g. in DATABUS) may be written to display the records containing the duplicate keys so the user may resolve the ambiguity. INDEX may then be reinvoked using the "I" option.

BUILDING -NEXT- LEVEL INDEX.
This indicates that the lower level of the index file has been completed and the next level is now being created.

DONE.
The creation of the index file is now completed.

Other error messages may be generated by REFORMAT or SORT. See the appropriate chapter for an explanation.

25.7 ISI File Formats

The DOS indexed file structure consists of a multi-level radix tree structure based on the record keys, and contains pointers to the location of the keyed records. Note that since many of these pointers are physical disk addresses, the ISI file cannot be moved without re-invoking INDEX. The text file may be moved so long as it is unchanged in any way. Moving the ISI file will destroy it.

The different levels of indices all have the same content, except for the lowest level index. Index levels are built up until an intermediate level of index will fit in a single disk sector. This becomes the highest level of index. This requirement is the reason for the 118 character limitation on key length.

The ISI files have the following format:

Offset	Length	Description
000	003	PFN and LRN bytes as per DOS convention - see the chapter on SYSTEM STRUCTURE.
003	0nn	This is a KEY entry where nn is key length+7 for a lowest level index, and key length+3 for a higher level index. The first sector of an ISI file after the RIBs is a special header record.
0nn+4	0nn	This is the second KEY entry in the sector. There must be at least two KEY entries per sector.
		.
		.

Note that as many key entries are put in a sector as will fit without splitting across a sector boundary.

Each KEY entry for an intermediate level index has the following format:

Offset	Length	Description
000	KEYLEN	The highest key in the next lower level index sector.
KL	001	Octal 012 - This indicates the end of the key and that this is a higher level index entry.
KL+1	002	PDA (MSB,LSB) of the entry in the next lower level of index.
KL+3	001	Octal 0377 - This indicates that this is the last entry in this sector.

Each KEY entry for a lowest level index entry has the following format:

Offset	Length	Description
000	KEYLEN	The key for this particular record.
KL	001	Octal 015 - This indicates that this is a lowest level index entry and delimits the end of the key.
KL+1	003	Buffer Offset, and the physical disk address for the logically next lowest level index entry.
KL+4	003	Buffer Offset, and logical record number of the text file record having this key.
KL+7	001	Octal 0377 - Indicates that this is the end of the lowest level index.

The first data sector in an ISI file is a header record used to locate the file from which the index was built. In this way, it is only necessary to specify the name of the index to DATASHARE.

Offset	Length	Description
000	003	PFN and LRN indicators as per DOS convention. See the System Structure Chapter.

003	013	Name of the data file that goes with this index file.
016	003	PFN, and RIB PDA of this file. This field is used to check that the index file has not been moved.
021	003	PFN, and RIB PDA of the file indexed.
027	003	Buffer address and LRN of the last record used in the data file.
032	003	Buffer address and LRN of the first free index entry.

25.8 Examples of the Use of INDEX

First, a simple example in which only a single ISI file is created, with the same name and on the same device as the text file it indexes. The file is a list of bad checks presented at a local grocery chain, and now each store has a DATASHARE terminal to inquire on the current status of each deadbeat. Thus, while the file is accessed often, additions and deletions are fairly infrequent, so the file will not be reformatted. The file is keyed by bank number (8 digits) and account number (7 digits) concatenated and in positions 1 to 15 of each record.

In order to create the index file, the operator must type:

```
INDEX DEADBEAT;1-15
```

The INDEX program will then create a file DEADBEAT/ISI which DATASHARE can use to access the DEADBEAT/TXT file.

Now, this same grocery chain has expanded its operations, so it desires to include more information on the location and date of each NSF check presented. Therefore, they have expanded the file to include the old key in positions 1 to 15, a store location number in positions 16 to 18, and a date field in positions 19 to 24. As an afterthought, the manager decides to tack on the name of the person passing the bad check in positions 193 to 216.

In order to create the indices required for access by any of these keys, the operator must type:

```
INDEX DEADBEAT,BANK;1-15
INDEX DEADBEAT,DATE;19-24
INDEX DEADBEAT,STORE;16-18
INDEX DEADBEAT,NAME;193-216
```

The INDEX program will create four files with names BANK/ISI, DATE/ISI, STORE/ISI, and NAME/ISI. Each file is logically separate, yet all are on the same volume as DEADBEAT/TXT.

Now the store owners have uncovered a hitch - first, the number of bad checks is becoming so large, there is no room on one disk for all the index files and the text file. In addition, access has been slowing way down as the frequency of additions and deletions increases. The store owners have called Datapoint to complain, and their local systems engineer has told them they need to reformat the files when they re-index, and has sold them another disk drive.

The operator now types:

```
INDEX DEADBEAT,BANK/ISI:DR1;FR1-15
INDEX DEADBEAT,DATE/ISI:DR1;19-24
INDEX DEADBEAT,STORE/ISI:DR1;16-18
INDEX DEADBEAT,NAME/ISI:DR1;193-216
```

Note that the reformatting is done only once at the beginning. If reformatting had not been done when the first index was built, it could not be correctly done later without invalidating the previously built indices.

Now, several years later, the grocery chain has expanded and has a large disk system at their main store. The owners are doing so much processing that there is not the time to run the above INDEX programs as each one invokes SORT. However, they wish to keep access time to the minimum. Also, the DEADBEAT file is so large that numerous additions and deletions hardly affect the size.

Every night the operator now types:

```
INDEX BANK;X
INDEX DATE;X
INDEX STORE;X
INDEX NAME;X
```

which recreates the index files. Then during weekly processing, the operator does the processing above which invokes REFORMAT.

The store owners have wisely dispersed some of their data processing to their branch stores. So each night the operator also types:

```
INDEX BANK;K
INDEX DATE;K
INDEX STORE;K
INDEX NAME;K
```

which creates tag files of the four indices. The operator then transmits DEADBEAT/TXT, BANK/TAG, DATA/TAG, STORE/TAG, and NAME/TAG to each of the branch stores. The operator at the branch store, after receiving these files, types:

```
INDEX DEADBEAT,BANK,BANK;I
INDEX DEADBEAT,DATE,DATE;I
INDEX DEADBEAT,STORE,STORE;I
INDEX DEADBEAT,NAME,NAME;I
```

which creates a local set of indices without invoking SORT.

Note: In the above example that created a BANK tag file, the command line with defaults is:

```
INDEX BANK/TXT,BANK/ISI,BANK/TAG;K
```

As only the /ISI and /TAG files are needed for creation of the tag file, the same results could have been achieved by typing:

```
INDEX ,BANK,BANK;K
```

CHAPTER 26. THE INIT9370 COMMAND

When a new disk pack is received, it is not immediately usable in the 9370 series drives until it has been formatted. The formatting process, which causes track and sector identifying information to be written over the entire disk surface, is performed by the INIT9370 command. This command is useful only on 9370 series disks!

26.1 Use

The INIT9370 command is unusual among DOS commands in that it is one of the few that can be run in a stand-alone mode: that is, without being run under the DOS. This feature is required in particular for the first time a user generates a DOS system disk, when he has no alternative but to start with INIT9370 running from an LGO cassette.

To invoke INIT9370 from a working DOS (normally useful only for users with two-drive systems), the operator enters at the system console:

INIT9370

With an LGO cassette, the operator places the cassette in the rear cassette deck and presses the RESTART key (and RUN key on a 5500 processor). Once the program has initially loaded, it functions the same regardless of whether it has been loaded from cassette or disk.

After being loaded, INIT9370 asks which physical (not logical) drive contains the disk to be formatted and asks the user for confirmation that it is all right to destroy the previous contents of the disk, if any. After the command is satisfied that the user knows what is about to happen, it proceeds to format the disk. The process takes about three and a half minutes.

26.2 Error messages

If the INIT9370 command encounters any sort of error indication before or during the formatting process, it will wait for a while to see if the problem will go away on its own. (A typical example would be if the disk to be formatted has not yet come on line when the INIT9370 command begins execution). If the problem persists, the program will display a comment on the CRT display indicating that it is waiting on the disk, describe the status of the disk as indicated by the controller, attempt some corrective actions that may help to clear the situation, and inform the operator of what corrective action has been taken. This is repeated until the problem is successfully cleared up.

CHAPTER 27. KILL COMMAND

KILL - Delete a file from the directory

KILL [<file spec>]

The KILL command deletes the specified file from the system if the file is not protected. If the file is protected in any way, the message

NO!

will be displayed. If the file specification is not given on the command line (file names which contain special characters cannot be given on the command line), the request for the file name:

```
WHAT FILE?  EXAMPLE:  SCRATCH /TXT:DR1  #143      :DR1
          /   :DR
```

will appear. The user must keyin an eight character filename (including trailing spaces), a slash, a three character extension (including trailing spaces), a colon, the letter "D" and the drive number on which the file resides. If the entire filename specification is not entered properly, the message:

NO SUCH NAME.

will appear. A file can be specified by physical file number by entering "#", followed by the octal PFN, followed by 8 spaces and the drive specification. If the specified file cannot be found (both a name and an extension must always be supplied unless using PFN), the message:

NO SUCH NAME.

will be displayed. If the file is found and is not protected, the message:

THAT FILE IS <filename> ON DRIVE n

will appear. Then the operator must additionally answer the message:

ARE YOU SURE?

with a 'Y' before the actual deletion of the file is achieved. After the deletion has occurred the following message is displayed:

```
* FILE DELETED *
```

CHAPTER 28. LIST COMMAND

28.1 Purpose

The LIST command will list any DOS standard format text file on the screen or a local or servo printer.

The command can be used for such things as:

A quick scan of a file by displaying it on the screen (LISTing a file is faster than EDITing it);

Producing a hardcopy listing of a file for permanent records;

Listing a file for use in preparation of a BLOKEDIT COMMAND FILE.

In this chapter, the following terms apply:

Text file means a file with records containing only ASCII characters, except for space-compression bytes and the End-Of-Record and End-Of-File marks. Files created by EDIT and those produced by DATASHARE are normally in the class of text files.

Line means one record of a text file. When displayed on the screen, only the first 72 characters of a record will be displayed; when listed on a local or servo printer only the first 124 characters will be printed. (The remaining eight characters contain a line number.)

Record means the user logical record number (LRN). The first LRN of a file is zero.

28.2 Parameters

When the LIST program is to be executed, the operator must type:

LIST <filespec> [,<spec2>][,<filespec2>][;options]

Available options are:

L - list on Local printer
S - list on Servo printer
D - Display on CRT
X - suppress line numbers
F - list Formatted print file
P - output formatted Print file
Q - Queue formatted print file, appending to an existing file
Nn - set Number of lines per page to n
I - list in Indexed sequence

Options may be entered in any sequence and should be separated by commas.

28.3 INPUT File Specification

The file specification (<filespec>) must refer to a DOS text file. If no extension is supplied with the file name, an extension is assumed depending on the options given. A default extension of TXT is assumed unless the option "I" or "F" is used. The option "I" (list a file using its index) causes a default extension of ISI and the option "F" (list a file with format control bytes) causes a default extension of PRT. If no drive is supplied with the file specification, all drives will be searched for the filename/ext. If <filespec> is omitted, the message

NAME REQUIRED.

is displayed. If the file indicated by <filespec> is not found on an online volume, the message

NO SUCH NAME.

is displayed.

28.4 Starting Point

The operator may specify a line number, or logical record number, in the file at which the list should begin by including an optional second parameter <spec2>. For example:

LIST <filespec>,L400

would list the specified file beginning with line 400 of the file.

If the line number specification exceeds the number of lines in the file, LIST returns to DOS after displaying the message:

```
FILE EXHAUSTED BEFORE LINE FOUND.
```

```
LIST <filespec>,R18
```

would directly access logical record 18 of the specified file and list, starting at line number 1. If range or format errors occur, the error type is indicated and another record number is requested.

For instance, if the record number specification exceeds the number of records, the message

```
RANGE - NEXT RECORD NUMBER:
```

is displayed.

The DEFAULT value for the second parameter is line 1 and record 0.

28.5 OUTPUT File Specification

If the options "P" (write to a print file on disk) or "Q" ('QUEUED' write to a disk print file starting at the end-of-file mark) are used, then the third parameter (filespec2) may be used to specify the output file. If the filename is not given, it is assumed to be the same as the input file name. If the extension is not given, it is assumed to be PRT.

Output from either the "P" or "Q" option is a text file with print control characters as described in the Format Control section. The file will be paged with headings; line numbers will be included unless suppressed by the "X" option.

28.6 Output Device

The operator may specify an output device other than the CRT display by including an optional parameter of "S" (servo printer), "L" (local printer), "P", or "Q". For example:

```
LIST <filespec>,L400;S
```

would list the specified file on the Datapoint servo printer

starting at line 400 or

LIST <filespec>;L

would list the specified file on a Datapoint local printer beginning at line number one.

For either print or disk output LIST will request a heading, which will be placed at the top of every page of output.

The DEFAULT output device is the CRT display which may be specified by entering a "D".

28.7 Output Format

A parameter is available to suppress line numbers. If the 'X' is entered, lines of up to 132 characters will be printed. For example:

LIST <filespec>;SX

would put the output on the servo printer without line numbers,

LIST <filespec>;X

would display the listing, showing 80 characters per line on the CRT.

Any paged output (from the "L","S","P", or "Q" options) is normally listed at 54 lines per page. The "Nn" option can be used to change the number of lines per page, n being the desired lines/page count.

28.8 Format Control

The parameter "F" is available to allow the handling of print files (those with a format character in the first column of each line). If "F" is entered, the file will be listed without line numbers, page numbers, or headings, since all these items should already be in the print file. The following format characters cause the indicated action to be taken before the line is printed.

1 - Skip to top of form

+ - Suppress line feed

(space) - Single line feed

0 - Double line feed

\- - Triple line feed

Any other character in the first column will be handled as a space (single line feed) and discarded.

28.9 Operator Controls

The listing consists of a continuous stream of the listed file's text. To cause the listing to pause, the operator may hold down the DISPLAY key. To abort the listing, the operator may depress the KEYBOARD key.

28.10 Error Conditions

If printer output was specified and the requested printer is not available, LIST beeps and displays the message:

PRINTER NOT READY

If the printer is made ready, listing will proceed. The KEYBOARD key may be depressed to abort the LIST at this point if necessary.

LIST checks to be sure the text end-of-file is exactly six zeroes and a three (see Text File Formats in the REFORMAT chapter).

If the EOF is not exactly correct, LIST displays the message:

INVALID END OF FILE.

LIST can be used to test for a bad EOF since most text-handling programs are not so particular about EOF format.

When <spec2> has been entered to start LIST at a particular record number, LIST traps FORMAT or RANGE errors and allows a new starting location to be entered. In any other usage, LIST does not trap FORMAT or RANGE errors and any such errors are fatal.

CHAPTER 29. MANUAL COMMAND

MANUAL - Clear Auto Execution

MANUAL

If the auto-execution name has not been set the message

AUTO NOT SET.

will be displayed. Otherwise, the System Table location reserved for the auto-execution information will be cleared and the message

AUTO CLEARED.

will be displayed.

CHAPTER 30. MIN COMMAND

30.1 Purpose

The Multiple In (MIN) command is useful for reading multiple files (source, object, and Datashare object) from the front cassette drive to disk. It will handle all standard single file (OUT and SOUT), double file (SOBO), and multiple file (LGO, CTOS, and MOUT with or without a directory) tape formats.

30.2 Tape Formats

Multiple In will automatically process the tape format by the following conventions if an option is given.

30.2.1 Single File Tapes

An OUT (object out) tape format has a file mark zero, a file mark one, an object file with entry point, and a file mark 0177. An object file has an address with the MSB and LSB in the fourth and fifth bytes of each record. Their complements are in the sixth and seventh bytes. The remainder of each record is filled with octal characters (ranging from 0 to 0377).

A SOUT (source out) tape format has a file mark zero, a source file, a file mark one, and a file mark 0177. A source file consists of records containing only ASCII characters, except for space compression bytes, physical end-of-record bytes, and logical end-of-record bytes.

30.2.2 Double File Tapes

A SOBO (source and object out) tape is the combination of a SOUT and OUT tape. It has a file mark zero, a source file, a file mark one, an object file with entry point, and a file mark 0177.

30.2.3 Multiple Numbered-File Tapes

An LGO (load and go) tape has a loader, a file mark zero, a string of files (the first being an object file and the rest may be source, object, Databus Code, and Relocatable Code intermixed) separated by sequential file marks, and a file mark 040.

A MOUT (multiple out) tape without directory has a file mark zero, a string of files (may be source, object, and Datashare object intermixed) separated by sequential file marks, and file marks 040 and 0177. Single and double file tapes are included in this category if options are not used.

30.2.4 Multiple Named-File Tapes

A CTOS (cassette tape operating system) tape has a loader, a file mark zero, a CTOS object file with entry point, a file mark one, a catalog object file, a string of files separated by sequential (though not necessarily contiguous) file marks, and a file mark 040.

A MOUT (multiple out) tape with directory has a file mark zero, a tape directory, a string of files separated by sequential file marks, and file marks 040 and 0177. The directory is a source format file containing a date entry seven bytes long (DDMMMYY) and 31 file name entries each eleven bytes long (eight bytes for the name and three bytes for the extension). The entries are separated by end-of-string bytes (octal 015). This makes it convenient for display under CTOS LIST or to load to disk and list.

30.3 Parameters

30.3.1 Single File Tapes

For OUT, and SOUT tape formats, the file specifications may be included on the command line in the following manner:

MIN [<file spec>];<option>

where <option> is an 'S' for SOUT tape formats.

File specifications are of the form FILENAME/EXT:DR#. If the drive is not given, all drives online will be searched starting at

drive zero. If the extension is not given, the assumed extension (TXT, ABS, DBC, or REL) will depend on the file format. MIN will identify the tape format. If the file name has not been entered on the command line, the program will ask:

LOAD FILE #XX (format)?

where XX indicates the file number on the cassette and format indicates the type of file (SOURCE, OBJECT, DATABUS CODE, or RELOCATABLE CODE). If the file is to be loaded, the response Y (yes) will cause the message:

DOS FILE NAME:

to be displayed on the same line. If the response is N (no), the operator will be asked for the next file (if any). If the response is *, control is returned to DOS. If no name is entered, the message:

NAME REQUIRED

will appear. If the filename specified already exists, the message:

NAME IN USE. WRITE OVER?

will appear. The answer N (no) will cause the filename request to be displayed again. The answer Y (yes) will cause the disk resident file to be overwritten. If the file to be overwrittten is write protected, the message:

WRITE PROTECTED OVERWRITE?

will appear. If the response is not Y, the filename request will be displayed again. If the response is Y, the protection is changed from write protect to delete protect and the disk resident file is overwritten. When a file has been loaded from the cassette the message:

LOADED

will appear to the right of the filename. The message:

MULTIPLE IN COMPLETED

indicates the successful completion of the program.

30.3.2 Double File Tapes

The file specifications for a SOBO tape may be entered on the command line in the following manner:

MIN [<file spec>][,<file spec>];B

File specifications are of the form discussed above. If the second file name is not given, the first name with the assumed extension of ABS will be used. If the extension is not given with the first name, TXT will be assumed. If the filename has not been entered on the command line, MIN will operate in the same manner as described in the section on single file tapes above for each file on the cassette, displaying the messages in the same order for both files.

30.3.3 Multiple Numbered-File Tapes

LGO tapes and MOUT tapes without a directory are both handled in the same manner. MIN is first executed as:

MIN

An LGO tape will then be identified as:

LGO TAPE FORMAT

In the case of multiple files, MIN will operate in the same manner as described in the section on single file tapes above for loading a file without entering the name on the command line. The questions described will be asked for each file on the tape until end of file has been encountered on the tape or an * is entered in response to the "load" question. MIN bypasses the loader on a LGO tape before searching for the file. If the file is not found, the message:

FILE NOT FOUND

will appear and MIN will be terminated. If the file is found and the file name is not entered on the command line, the file name will be requested as in single-file tapes.

30.3.4 CTOS Tapes

A CTOS tape will be identified as:

CTOS SYSTEM TAPE FORMAT

The system then searches for the catalog (tape file #1). The CTOS file is fairly long so it takes a while. If the catalog file is not an object file or is an object file that loads into memory somewhere other than 017406 or 017410, the message:

BAD CATALOG

will appear and the remainder of the tape will be processed as a multiple numbered-file tape starting at tape file #2. If a good catalog is found, it will then be displayed as:

CATALOG: <file 1> <file 2> <file 3> <file 4>. . .

Then the operator will be asked:

DO YOU WANT TO LOAD <file 1> ?

The entire process is identical to the multiple numbered-file tapes above except the first fourteen files are referred to by name. The filename may be expanded by the operator from the six character name allowed by CTOS to the eight character name allowed by DOS plus the extension. A filename is requested if the reply is 'Y'.

30.3.5 MOUT With Directory Tapes

These tapes are processed in a manner very similar to CTOS tapes. The tape is first identified as:

MOUT TAPE FORMAT

Next the date will be displayed:

DATE: DD MMM YY

Then the directory will be displayed:

DIRECTORY: <file 1/ext> <file 2/ext> <file 3/ext>. . .

Then the operator will be asked:

LOAD <file 1/ext> ?

All the responses are the same as above except that the file name will not be requested. A drive response is also available. Entering "DRn" or "<VOLID>" will imply "YES" and force the file to drive n. The program will cycle until the end-of-tape file mark (040 or 0177) is read at which point the message:

MULTIPLE IN COMPLETED

will be displayed.

30.3.6 Options

Tape file modifications may prevent MIN from automatically determining the tape format. In this event, the options 'L' (for LGO), 'C' (for CTOS), or 'D' (for Directory) are available. Also, option 'N' (for No directory) will tell the system that it is handling a MOUT tape without a directory, which allows entering the file names manually if the directory entry names are not desired. This option also allows entering the directory to disk. Options are entered following a semi-colon.

These options are merely test overrides. If, for instance a tape, starts with a recognizable file mark, a loader won't even be tested for and therefore entering the 'L' option is meaningless.

Unfortunately, MIN cannot differentiate an OUT, SOUT, or SOBO tape from a MOUT without directory tape. To speed the processing, the options 'S' (for SOUT) and 'B' (for SOBO) are available. Once again, if the tape doesn't resemble a SOUT tape, for instance, entering an 'S' is meaningless.

MIN accepts a drive specification option ":DRn" or ":Dn" to force the disk files to a specific drive. Note that this drive specification is an option appearing in the option list following the semicolon, not part of any file specification. Drive specification may be necessary to avoid overwriting existing files on other drives or to force MIN to place the files on a drive other than drive 0.

If the tape is a MOUT tape with a directory, the options 'A' (for All), 'O' (for Overwrite), 'Q' (for modifying the extension with Q's) are available. Using the option 'A' will load all files on the tape. However, if the file already exists, the operator will be asked if overwriting is desired and if not, for a new file name. Entering the 'O' option in conjunction with the 'A' will

force overwriting of existing files (unless write protected). If while processing in the 'All Overwrite' mode a write protected file is encountered, the message:

WRITE PROTECTED

will appear and processing will continue with the next file. Entering the 'Q' option in conjunction with the 'A' will put as many Q's into the directory extension as necessary to create a new filename/ext if the original one already exists. If the original filename/ext exists, the message:

EXISTING FILE

will appear to the right before the modification to the extension is performed. If the filename/QQQ already exists, the message:

Q OPTION EXHAUSTED

will appear to the right and the file will be skipped.

The option 'N' followed by an octal number allows that specific file to be loaded. For example, entering:

MIN FILE/TXT;N12

will load the tape file following file mark 12 (octal) to disk as 'FILE/TXT'. The default extension will be 'TXT' for source, 'ABS' for object, 'DBC' for Databus Code object files, and 'REL' for Relocatable Code files depending on the tape file format. If a non-octal number is entered (e.g. N8) the message:

NUMBER NOT OCTAL

will appear and MIN will be terminated. If an unrecognizable record format is encountered, the message:

UNRECOGNIZABLE TAPE RECORD FORMAT

will appear and MIN will be terminated. MIN bypasses the loader on a LGO tape before searching for the file. If the file specified is not found, the message:

FILE NOT FOUND

will appear and MIN will be terminated. If the file is found and the file name was not entered on the command line, the file name will be requested as in single-file tapes.

The options 'L', 'C', 'N', 'S', and 'B' are mutually exclusive. Only one may be entered. The 'A' may be entered with or without the 'D' and with none of the other above options. 'O' and 'Q' are mutually exclusive and may only be entered in conjunction with the 'A'. If any of these restrictions is violated or a character other than those above entered, the message:

BAD OPTION PARAMETER

will appear and the program will be aborted.

30.4 Errors

If the tape format is not one of the eight standard formats outlined above in the Tape Formats section (e.g. it starts with a file mark two) the message:

INVALID TAPE FORMAT

will appear and the processing will be aborted. If the end of tape is detected while processing, the message:

END OF TAPE

will appear and the processing will be aborted. If a parity error is encountered in an object or Datashare file on tape, the message:

PARITY ERROR-FILE DELETED

will appear, the file name will be removed from the disk directory, and processing will skip to the next file. If a parity error is encountered in a source file on tape, the message:

PARITY ERROR-RECORD MODIFIED

will appear, a 253 byte disk record will be written with percent signs in the first five positions of the record data, and processing will be continued with the next record.

CHAPTER 31. MOUT COMMAND

31.1 Purpose

The Multiple Out (MOUT) command is useful for writing multiple (up to 32, or 31 if a directory is used) disk files (source, object, and Datashare) out to the front cassette drive.

An additional feature is the ablity to create a tape file directory as file #0 on the tape. The directory is a source format file, that is, it consists entirely of ASCII characters except for space compression bytes, physical end-of-record marks, and logical end-of-record marks. The directory contains a date entry seven bytes long (DDMMMYY) and 31 file name entries each eleven bytes long (eight bytes for the name and three bytes for the extension). The entries are separated by end-of-string bytes (octal 015). This makes it convenient to list under CTOS LIST or to load to disk and list. The directory is also used by the MIN program to enter files to disk. MOUT creates the directory in memory before the tape writing starts even if it is not to be written to tape. The writing of a full tape (over 500 records) takes about 10 minutes, which shows the advantage of entering all the names before writing begins.

Another feature is the option to automatically verify a tape following its creation. Or a previously written directory tape may be verified in an 'only verify' mode. If this mode is requested, the system will read the directory on the cassette tape in the front drive (if a valid directory is not found, the system will request file names from the operator) and verification will be performed against the indicated files.

31.2 Parameters

File specifications and/or options may be entered on the command line in the following manner:

MOUT [<file spec>,<file spec>,...][;options]

File specifications are of the form FILENAME/EXT:DR#. If the drive is not given, all online drives will be searched starting at drive zero. If the extension is not given, ABS is assumed. File

specs are separated by anything (including multiple spaces) except letters, numbers, slash (/), or colon (:).

31.3 Options

Options (which follow a semi-colon and may be spaced or separated by commas) are 'L' for a loader format tape, 'D' for a directory format tape, 'V' for verification of the created tape, and 'X' for verification only.

If a loader is to be written, the first file (file #0) must be an object file. There are no restrictions on files other than #0.

The directory option ('D') will write a tape directory as file #0. The first item within the directory is the date entered DDMMMYY. Note: the month is entered as three alpha characters. The date may be entered following the option letter (e.g. D12JAN74). If the date is not entered, it will be requested.

The verify option ('V') will verify all the files on the created tape. Verification consists of making a byte for byte comparison between the data on the disk and the data on the tape. If verification fails, the tape will be rewritten and verification tried one more time.

The verify only option ('X') will cause the first tape file to be read from the front deck. If it is a directory (first seven characters of DDMMMYY format), the remaining files will be automatically verified using the directory entries. If it is a loader, it will be verified and file names requested for the remaining files as they are verified. An 'N' may be entered immediately preceding the 'X' to force the system not to recognize the directory. This would be done if manually entering file names is desired (for instance, the directory names don't match the disk file names). If there is neither a directory or loader, file names are requested as the files are verified.

If the semi-colon is entered with no entry following, it will be interpreted that the tape will not have a loader, a directory, or any verification.

Entering 'D' and 'L' together or entering anything with 'X' other than 'N', or entering some letter other than 'D', 'L', 'V', 'X', or 'N' will result in the message:

BAD OPTION PARAMETER. MOUT DISCONTINUED.

and the Multiple Out will be aborted.

If file names and/or options are not entered on the command line, MOUT will ask for them as required. If options were not entered, the first question will be:

DO YOU WANT A LOADER?

Replies other than 'Y' or 'N' will be answered by:

WHAT?

and a repeat of the question. If the reply is 'N', the next question is:

DO YOU WANT A DIRECTORY?

Again, if the reply is other than 'Y' or 'N', it will be answered by:

WHAT?

and a repeat of the question. If the reply is 'Y', the next request is:

ENTER THE DATE (DDMMMYY):

where the month is entered as three alpha characters. If the day is not in the range of 00 to 39, the month not alpha, or the year not in the range of 70 to 99, the response:

BAD DATE

will appear and again the request for the date. The next question is:

DO YOU WANT TO VERIFY THE TAPE?

If the reply is not 'Y' or 'N', the response:

WHAT?

will appear followed by a repeat of the question. If the reply is 'Y' and the replies to the loader and directory questions were 'N', the question:

DO YOU WANT TO ONLY VERIFY THE TAPE?

will then be asked. If the reply is other than 'Y' or 'N', the response

WHAT?

will appear followed by a repeat of the question. If only verification is requested, the first tape record on the front tape deck is read in. If it is a directory (the first seven characters of DDMMMYY format), the remaining tape files will be automatically verified using the directory entries. If it is a loader, the message:

LGO TAPE FORMAT

will appear. The message:

LOADER IS BEING VERIFIED

will then appear as the loader is being verified. If the loader verifies correctly, the message:

LOADER OK

will appear to the right. Otherwise, the message:

BAD LOADER

will appear. After checking the loader, or if the tape has neither a loader or directory, the message:

CASSETTE FILE #XX (format) DOS FILE NAME:

will appear where XX is the file number and (format) is (SOURCE), (OBJECT), (DATABUS CODE), or (RELOCATABLE CODE) depending on the file format. If nothing is entered, the message:

NAME REQUIRED

will appear and the request will be repeated. If an asterisk (*) is entered, MIN will terminate and return to DOS. If a greater-than sign (>) is entered, the program will skip to the next file. If a less-than sign (<) is entered, the program will backspace to the prior file (bypassing null files). If the program finds the beginning of the tape, it will beep and then move forward to the first file. If a name is entered, the default extension is 'TXT' for source, 'ABS' for object, and 'DBC' for Datashare object depending on the file format. If the drive

number is not entered, all online drives will be searched starting at drive zero. If a drive number greater than DOS allows is given, the message:

BAD DRIVE

will appear and the request repeated. If the file is not found, the message:

FILE NOT FOUND

will appear and the request repeated. If the disk file is found, it will be matched byte by byte against the disk file. If the files completely match, the message:

FILE OK

will appear to the right and processing continues with the next file. If an error is detected, the appropriate message will appear and processing continues with the next file. Null files are bypassed. Processing continues until an end-of-tape mark (file mark 040 or 0177) is read at which time the message:

VERIFICATION PHASE COMPLETED

will appear and MOUT will be terminated.

31.4 File Names

If the file names are not given in the command line, the operator will be asked for the file names one at a time. The request is of the form:

CASSETTE FILE XX DOS NAME:

where XX is the file number. Possible replies to the file name query include:

a) the file specifications as discussed above,
b) a pound sign (#) which will bump the file number to 20 octal if not already there (only allowed on loader tapes to initiate numbered files on a CTOS tape),
c) a dollar sign ($) which will cause a null file (tape file mark only) to be written to tape and the file spec of NULL/NUL to be entered in the directory,
d) an asterisk (*) which will indicate no more files are to be entered and the tape writing started (writing is postponed

until the directory is complete), and
e) OS which will abort the program. The message:
MULTIPLE OUT DISCONTINUED will appear and control is returned to DOS. (To dump OS/ABS, enter 'OS/ABS').

If the operator fails to enter a name, the message:

NAME REQUIRED

will appear and the name request will be repeated. If the drive is given and is not in the range valid for DOS, the message:

BAD DRIVE

will appear followed by a re-request of the name. If the file is not found, the message:

FILE NOT FOUND

will appear followed by a re-request of the name. If the file is found, the format (object, source, or Datashare) will be determined by the system. If the tape is a loader tape and file #0 is not an object file, the message:

FILE FOLLOWING LOADER NOT OBJECT

will appear along with a re-request of the file name. This message may also be displayed if the reply to the file name query for file #0 is a pound sign. Otherwise the messages:

OBJECT FILE

or:

SOURCE FILE

or:

DATABUS CODE FILE

or:

RELOCATABLE CODE FILE

or:

NULL FILE

will appear to the right of the file name. If the pound sign is entered for a tape that does not have a loader, the message:

NOT LGO TAPE

will appear with a re-request of the file name. If 32 files (or 31 on a directory tape) are entered, the message:

THAT'S THE END OF THE LINE

will appear and the tape writing is started automatically.

31.5 Writing

Once the tape writing has started, the system will keep the operator informed of its progress. As a loader is being written, the message:

LOADER IS BEING WRITTEN

will appear. As a directory is being written, the message:

DIRECTORY IS BEING WRITTEN

will appear. While files (including null files) are being written, the message:

FILE <filename/ext> IS BEING WRITTEN

will appear. When the writing is completed, the message:

WRITING PHASE COMPLETED

will appear.

If a non-object record is sensed in an object file while writing to tape, the message:

FILE CONTAINS NON-OBJECT RECORD

will appear and the next file is written over the bad tape file including the file mark. This will leave a directory entry without a file. If this should happen, it will cause verification to display the message:

NON-SEQUENTIAL FILE MARK

and the tape rewritten.

If a non-source record is sensed in a source file while writing to tape, the message:

INCORRECTLY FORMATTED SOURCE RECORD

will appear. The file is ended at this point without writing the bad record and the next tape file will start immediately following. If this should happen, it will cause verification to display the message:

INCORRECTLY FORMATTED DISK RECORD

or:

TAPE EOF BEFORE DISK EOF

and the tape rewritten.

If MOUT runs out of tape, the message:

END OF TAPE ENCOUNTERED WHILE WRITING filename/ext

will appear, an end of tape marker written at the end of the previous tape file, and the unwritten files will be removed from the directory (if there is one). Processing then will be continued with verification.

31.6 Verifying

If verification is requested, the system will keep the operator informed of its progress. As a loader is being verified, the message:

LOADER IS BEING VERIFIED

will appear. As a directory is being verified, the message:

DIRECTORY IS BEING VERIFIED

will appear. While files (including null files) are being verified, the message:

FILE filename/ext IS BEING VERIFIED

will appear. When the verification is completed, the message:

VERIFICATION PHASE COMPLETED

will appear. If verification is requested for a tape having no directory, the message:

NOT DIRECTORY TAPE

is displayed. Then the message:

CASSETTE FILE #XX(format) DOS FILE NAME:

will appear. The filename should be entered. Responses are discussed in the section under OPTIONS.

A variety of error messages may be displayed during the verification phase. Most of them are self-explanatory. They include:

BAD LOADER

BAD DIRECTORY

TAPE FILE DOES NOT MATCH DISK FILE

INCORRECTLY FORMATTED DISK RECORD

DISK FILE CONTAINS NON-OBJECT RECORD.

DISK FILE CONTAINS NON-TEXT RECORD.

NON-SEQUENTIAL FILE MARK.

TAPE FILE MARK READ BEFORE TAPE OBJECT EOF.

TAPE OBJECT EOF NOT FOLLOWED BY TAPE FILE MARK.

DISK EOF BEFORE TAPE EOF

TAPE EOF BEFORE DISK EOF

If an error is detected, the program will then either rewrite the tape (if it has just been created) or skip to the next file (if in the 'verify only' mode). If it rewrites the tape, the message:

I'M NOW REWRITING THE TAPE

will appear. The system will rewrite once before quitting completely at which point the message:

VERIFICATION UNSUCCESSFUL

will appear and the processing terminated.

If a problem arises that causes an abnormal end (e.g. end of tape), the message:

MULTIPLE OUT DISCONTINUED

will appear, otherwise the message:

MULTIPLE OUT COMPLETED

will signal the successful end of the program.

ERROR D ON DECK 2

will signal parity errors on the cassette and control is returned to DOS.

CHAPTER 32. NAME COMMAND

NAME - Change the name of a file

NAME <file spec1>,[<file spec2>][,<subdirectory name>]

NAME will allow the user to change the name of a file, the extension of a file, or the subdirectory in which a file resides. The content of the file is unchanged. The first file specification refers to the current file name and the second file specification is the new name and/or extension to be assigned. If no extension is supplied in the first file specification, ABS is assumed. If no extension is supplied in the second file specification, the extension of the first file is assumed. If no extensions are supplied, both files will be assumed to have extensions of ABS. The drive number should only be specified in the first file specification.

If the NAME command is used to move a file from one subdirectory to another the second file specification may be omitted (unless the filename and/or extension are to be changed) and the subdirectory name denoting the subdirectory into which the file is to be placed is the third specification:

NAME <file spec1>,,<subdirectory name>

In both uses of the NAME command, two specifications are required. If either name is not given, the message

NAME REQUIRED.

will be displayed. If the second name is already defined on the drive that contains the first file, the message

NAME IN USE.

will be displayed. Note that the drive specification on the second name is ignored. If the first name is not found on an online disk, the message

NO SUCH NAME.

will be displayed. If the subdirectory name keyed is not found

on the disk containing the file to be renamed, the message

NO SUCH SUBDIRECTORY.

will be displayed. If the third parameter is not specified, the file is "brought into" the current subdirectory at the completion of the renaming process.

CHAPTER 33. PUTIPL COMMAND

The PUTIPL command writes an IPL (Initial Program Loader) block and DOS boot blocks to the disk.

PUTIPL <:DRIVE>

If the drive number is not specified in the command line, PUTIPL will display the following:

LOGICAL DRIVE TO BE WRITTEN (0-max OR "*" TO EXIT TO DOS):

Respond with the drive number that you want to write to.

CHAPTER 34. PUTVOLID COMMAND

The PUTVOLID command writes a symbolic volume identification (VOLID) onto a disk.

PUTVOLID <VOLID><:DRIVE>;<OWNER ID>

Where VOLID is 1 to 8 characters in length, DRIVE is the logical drive to be written to, and OWNER ID is any information the user wants.

If only a drive number is entered, the existing VOLID for that drive will be displayed.

CHAPTER 35. REFORMAT COMMAND

35.1 Introduction

The DOS REFORMAT command is used to change the internal disk format of text-type (non-object) files. Additionally, it can recover disk space left unused when files are updated by the DATASHARE indexed sequential access method. REFORMAT can compress a file in place on disk provided that such compression does not entail the writing of a physical disk sector prior to the time that sector is read. REFORMAT maintains logical consistency in such cases and will not write on a disk file until it has checked to be sure it can complete its job successfully.

35.2 Operation

When the REFORMAT program is to be executed, the operator must type:

REFORMAT <file-spec>[,<file-spec>][;<parameters>]

where only the first file specification is mandatory, and specifies the file to be reformatted. If the second file specification is given, it must be distinct from the first. Reformatting in place is requested by omitting the second file specification.

The parameter list describes the format the output file is to take, and whether REFORMAT is to free any disk space that might be vacated by the reformatting process. In addition, the user can specify that REFORMAT is to pad short records, and either truncate or segment long records. REFORMAT will produce three different kinds of output files: record compressed, space and record compressed, or blocked records (see the section on TEXT FILE FORMATS). Note that REFORMAT will not produce blocked space compressed records or space compressed non record compressed files although such files can be used as input to the REFORMAT program. If no parameters are given, the output file is blocked one record per sector.

Parameters passed to REFORMAT may be separated by spaces or commas. The valid parameters are as follows:

Parameter	Description
B<n>	The output file will be blocked. This implies no space or record compression, with <n> logical records per physical sector.
C	The output file will be space and record compressed. The number of logical records per physical sector will be indeterminate.
R	The output file will be record compressed, but no space compression will be done. In general, the number of logical records per physical sector will be indeterminate.
L<n>	The length of each logical record will be adjusted to <n> characters. Note that if the logical records are space compressed, this will not make the physical length of the records <n> characters. If the logical record is shorter than <n> characters, it will be padded with blanks to the proper length. If the logical record is longer than <n> characters, the action taken depends on the T and S parameter.
T	(Only valid if L parameter is given) Truncate the logical record if it is longer than <n> characters.
S	(Only valid if L parameter is given) If the length of the logical record is greater than <n> characters, segment it into (q) logical records each of length <n>, padding if necessary. The number (q) is defined as input length divided by <n> rounded upward to the next integer. If neither S or T is specified, and an input record of length greater than <n> is found, a message is issued and REFORMAT gives up.
D	If reformatting is done in place and this parameter is specified, any disk space vacated by the reformatting process will be returned to the operating system for re-use.

35.3 Output File Formats

The REFORMAT utility permits you to select essentially three different output file formats. It will produce blocked files that are not space compressed, record compressed files that are not space compressed, and files that are both record and space compressed. In addition, it has a subcommand to permit you to specify the logical length of the output records. Use of this subcommand will guarantee that each record has exactly the same logical length. Note that if the output format does not specify space compression, the physical length of each record will be identical. This is especially useful for telecommunications disciplines that require records of fixed length.

If you have set a fixed logical length for output records, there are two subcommands available to tell REFORMAT what to do with records whose logical length exceeds the specified output length. You may select either truncation of the input record, or you may segment it into two (or more) output records, each of the logical length specified.

35.4 Reasons for Reformatting

Several uses of REFORMAT deserve special mention. First, a random disk file is structured to have one logical record per physical sector. Often, however, it is convenient to create a random file through the use of the general purpose editor - which record and space compresses its output. REFORMAT can then reprocess the file into the correct format for DATASHARE or DATABUS random access.

Secondly, when a file is accessed with DATASHARE indexed sequential access method, any additions or deletions result in an increase in the physical size of the file. The reason for this is that any inserted records are placed at the physical end of the file, and each one consumes at least one entire physical sector, regardless of its logical length. Similarly, deleted records are simply overstored with octal 032 (logical delete) characters, and the space they vacate is not reused. REFORMAT recognizes this condition, and will recover such vacated space. Note that ISAM read-only or update-only (no additions or deletions) files do not usually need reformatting.

35.5 Reformat Messages

The REFORMAT utility program produces several messages on the operator's console. The contents and where necessary, meaning of those messages follow:

DOS. VER 2 REFORMAT COMMAND - date
Self-explanatory sign on message.

COMMAND LINE ERROR: 015 missing
This is an internal error and should be reported to Datapoint.

PROGRAM ERROR - EXCESS FILE SPACE NOT DEALLOCATED TO PREVENT POSSIBLE LOSS OF DATA
REFORMAT has detected an invalid end of file mark. In order to prevent the possible loss of data which might be after the invalid end of file indicator, space allocated but unused is not freed.

EXCESS FILE SPACE NOT DEALLOCATED; OUTPUT FILE IS DELETE PROTECTED.
Self-explanatory.

OUTPUT FILE IS WRITE PROTECTED AND CANNOT BE WRITTEN INTO OR SHORTENED.
You have requested REFORMAT to output to a write-protected file.

INVALID OPTIONS SPECIFIED
You have given REFORMAT an invalid parameter list. This message is followed by the valid options you may specify.

ILLEGAL, CONFLICTING OR DUPLICATE OPTIONS
You have specified two mutually exclusive options.

YOU SPECIFIED BOTH SEGMENTATION AND TRUNCATION. YOU CANNOT HAVE BOTH
Self-explanatory.

BLOCKING FACTOR CONTAINS INVALID NON-NUMERIC DIGITS
Self-explanatory.

BLOCKING FACTOR REQUIRED BUT MISSING OR ZERO FOUND
You specified blocking but omitted the blocking factor.

LOGICAL RECORD LENGTH REQUIRED BUT MISSING OR ZERO FOUND
   You must specify the logical record length of the output file if you wish to have fixed length output records.

YOU HAVE ILLEGALLY ENTERED A SPECIFICATION FOR A THIRD FILE
   REFORMAT recognizes only two file specifications.

HOW DO YOU EXPECT TO FIT THAT MANY RECORDS IN A 256 BYTE SECTOR?
   Self-explanatory.

LOGICAL RECORD LENGTH, IF SPECIFIED, MUST BE <= 250 BYTES.
   Self-explanatory.

YOUR BLOCKING FACTOR IS TOO LARGE FOR THE SIZE OF THE RECORDS YOU HAVE.
   Self-explanatory.

YOUR LOGICAL RECORD LENGTH IS TOO SMALL FOR THE SIZE OF THE RECORDS YOU HAVE
   While processing the input file, REFORMAT came across a record that was larger than the specified logical record length. Since you specified neither segmentation nor truncation, this is recognized as an error.

SPECIFIED OUTPUT FILE FORMAT ENLARGES PRESENT INPUT FILE. FILES CANNOT BE ENLARGED DURING REFORMAT-IN-PLACE. REFORMAT IN-PLACE REQUEST REFUSED.
   Self-explanatory.

YOU SPECIFIED AN OUTPUT FILE THAT ENDED UP BEING YOUR INPUT FILE. TO REFORMAT IN-PLACE DO NOT SPECIFY ANY OUTPUT FILE.
   Self-explanatory.

OUTPUT FILE NOT FOUND ON DRIVE X.
OUTPUT FILE FOUND ON DRIVE Y.
OUTPUT FILE WILL BE CREATED ON DRIVE Z.
These messages only occur if no specific drive was indicated for the output file. The first message appears followed by either the second or third. REFORMAT could not find the output file on the same drive as the input file. It either found one on a different drive, or created one on the displayed drive. If the output file is created, it is always created on the same drive as the one the input file is on.

REFORMAT IN-PLACE REQUESTED.
PRESCAN IN PROGRESS.
REFORMAT is checking to make sure it can properly process the file inplace.

FILE ALREADY WAS IN THE SPECIFIED FORMAT
Self-explanatory.

COPYING WITH REFORMATTING IN PROGRESS
Self-explanatory.

REFORMAT-IN-PLACE IS IN PROGRESS.
DO NOT DISTURB!!!
Self-explanatory.

NAME REQUIRED
Either you gave only an extension or drive for the input file, or you specified the output file first, followed by the input file.

INVALID DEVICE
An invalid drive was specified for the input file.

NO SUCH NAME
The input file specified cannot be found.

INVALID DRIVE SPECIFICATION
The drive specification entered for one of the file specifications was not in a valid format.

35.6 Text File Formats

Under Datapoint Corporation's Disk Operating System, text files consist of legal ASCII characters, which make up the text itself, and various control characters with special meanings. It is illegal to have the control characters in the text portion of the file. According to DOS convention, any character between 000 and 037 is considered a control character.

Each physical record of a text file is a logical disk sector, and contains 256 characters. The first three and last two characters are reserved for control functions; hence, the maximum space available in a single physical record is 251 bytes. The format of a logical sector is as follows:

Offset	Length	Description
000	001	Physical file number of this file. For a detailed description of physical file organization, see the chapter on System Structure.
001	002	Logical record number. This refers to logical physical records, and is not related to text records within the file.
003	373	Text. 251 bytes of text and control characters, depending upon the format of the file.
376	002	Two characters reserved.

The text part of each file is considered a logical stream, crossing sector boundaries without being logically discontinuous. Demarcations of logical record boundaries are made solely by control characters imbedded within the text itself. There are essentially five control characters found in files generated by DOS: 000 <NUL> used for end of file indication, 003 <EM> used to denote the end of medium (a sector boundary) but not the end of a logical record, 011 <CMP> used to denote space compression, 015 <ENT> used to denote the end of a logical record, and 032 <DEL> used to denote deleted data.

Under DOS each file is treated as a single, continuous stream of data. Physical records bear no relation to the logical structure of the data contained in them. In this way, a proliferation of different file structures, and the special routines needed to treat such special cases has been avoided. This does not mean that there cannot be a relation between physical and logical structure, it simply means that such a relationship is incidental to a particular file, and need not be treated as a special case. For example, random access to a data file is defined in the DATABUS language. Files to be accessed in this manner are structured in such a way that one logical record corresponds exactly with one physical record. This structure is not inherent in the makeup of a random file, in fact, such files can be treated exactly as any other text file.

The basis for this treatment of text files is the logical record. A logical record starts at the beginning of a file, or immediately after the end of a previous logical record. It consists of ASCII data and is of no pre-determined length. Instead, the record is terminated with a single ENT character. In this way, complications arising from a multitude of record types are entirely avoided.

If the logical record contains any CMP characters, it is said to be space-compressed. The character immediately following the CMP character is a space count, and the pair represent the number of ASCII blanks removed when the record was compressed. Since the character following CMP is always assumed to be a space count, CMP can never occur as the next-to-last text character in a physical sector, since the EM character following it would be lost.

If the file is organized so that each physical sector contains exactly the same integral number of logical records, with no logical record spanning an EM character, the file is said to be blocked. If the file is not blocked, then it is said to be record compressed. Note that for a blocked file all sectors except possibly the last one in the file contain the same number of logical records while for record compressed files the number of logical records per physical sector is indeterminate.

Under DOS conventions, a valid end of file mark consists of exactly six NUL characters, followed by an EM character:

000 000 000 000 000 000 003

This mark must begin at a sector boundary. All information after a valid end of file mark in the sector is indeterminate.

CHAPTER 36. THE REPAIR COMMAND

The purpose of REPAIR is to repair a malfunctioning or non-functioning DOS disk pack. The performance of the DOS is directly related to the correctness of disk-resident system tables. Errors in these tables can cause DOS difficulties ranging from occasional mysterious losses of data to complete inability of the DOS to function on the pack. REPAIR finds, identifies to the operator, and attempts to correct errors in the system tables.

REPAIR, once activated by an operator, is capable of seeking errors and determining corrective measures on its own. However, there are operator interfaces which exist to give a human operator the power to monitor and control the program's progress. The program constantly displays on the screen information about what it is doing. If errors are discovered, the operator will be asked if the error should be corrected on disk. Thus, the operator has control over any changes made to disk and may suppress any correction suggested by the program.

REPAIR consists of three phases: the Cluster Allocation Table and Directory check phase, the Retrieval Information Blocks check phase, and the Cluster Allocation Table regeneration phase. In general terms, the program progresses from simple error analysis to quite involved error analysis during its execution. Beginning with the cylinders-to-be-locked-out information supplied by the Lockout CAT on disk and supplemented by the operator, each program phase progresses according to information developed or verified during preceding checks.

The amount of interface and systems expertise required of the operator ranges from almost zero to very much, and is directly proportional to two things: how badly the pack is damaged, and whether the operator wants to try to save files with errors. If the operator merely permits REPAIR to delete every file found to be in error, the result would be guaranteed to be error-free disk-resident system tables, and the operator would not need to understand any details of the DOS. Sometimes, however, it will be easier for the operator to take notes and refer to the appropriate DOS documentation in order to save a file, rather than delete the file and then have to re-create it.

REPAIR is a completely self-contained program and does not require a working DOS to run. REPAIR can be executed as a COMMAND from the DOS or from an LGO cassette. REPAIR carries its own

copies of the standard basic DOS I/O routines (DR$, DW$, KEYIN$, DSPLY$), the DOS interrupt handler, and the DOS DEBUG$ routine.

36.1 Applications of REPAIR

There are three general classes of errors that can cause a DOS to work improperly:

1. Errors in the data within a file. Example: An incorrectly written object code record in a program object file will make the program unloadable and thus unexecutable.

2. Errors in the DOS system files. Example: If one of the DOS system files were inadvertently damaged, as by being partially overwritten, then sooner or later some part of the DOS would not function properly.

3. Errors in the disk system tables. Example: The Cluster Allocation Table is overwritten.

Far and away the most commonly occurring class of error is class 3. (Incidentally, the most common error is the one given for the example: a destroyed C.A.T.). Also, class 1 and class 2 errors most often occur because of previously existing class 3 errors.

REPAIR will not find or fix class 1 or 2 errors. Once those errors have occurred the file with the error should be reloaded to disk. If the user is interested in fixing these kinds of errors he should refer to later sections in this chapter and other appropriate DOS documentation.

REPAIR can fix almost all class 3 errors, and thus can fix almost all of the problems that commonly occur with a disk pack.

36.2 When to use REPAIR

There are three times to run the REPAIR program:

1. Regular disk-pack checking. It never hurts to run REPAIR after every few hours of disk use, in order to catch errors that may be developing that haven't been noticed.

2. Unexplained strange things start happening. If you ever see the message:

FAILURE IN SYSTEM DATA

it is time to run REPAIR. If other error messages are displayed by the DOS, such as:

RECORD FORMAT ERROR

and there seems to be no reason that the error should have occured, REPAIR may find the reason. If files or records in files disappear or get scrambled, it is probably a good idea to run REPAIR to see if errors have developed in the system tables.

3. The DOS will not run at all. Many times if the DOS will not "boot" it is because 1) The CAT has been destroyed - specifically, the auto-execute PFN (the last byte in the sector) is not 000 (REPAIR will always reset the auto-execute PFN to 000 when it writes the regenerated CAT to disk); or 2) The directory (one or more sectors) has been destroyed; or 3) One or more of the RIBs for the system files have been destroyed.

36.3 Understanding REPAIR

This chapter is divided into two major sections for two levels of reference:

1. Minimal operator interface.

The first major section is for users who wish to use REPAIR to make their pack work again as quickly and with as little effort as possible. To use REPAIR, one does not have to understand very much about the DOS or the structure of the data on disk.

2. Medial operator interface.

The second major section is a rather comprehensive discussion of the various messages and options provided by the REPAIR program, and is for users who wish to be able to take advantage of the file-saving options available with REPAIR.

The second major section also discusses a variety of things that can go wrong on a disk pack and how REPAIR can be used to deal with those problems. This is for users who are interested in understanding the DOS system disk data structure for its own sake, with emphasis, of course, on problems that can occur.

36.3.1 Preliminary reading

At absolute minimum, anyone who wants to use the REPAIR program must understand some basic DOS concepts. The REPAIR user must have a concept of what a DOS FILE is, and should be acquainted with the use of the OPERATOR COMMANDS (entered at the DOS system console) and FILE NAMES. The user must understand the concept of FILE DELETION. The user must also know what DRIVE NUMBER means.

If possible, the REPAIR user should read and understand the section "Disk Structure". To use and understand REPAIR to the maximum extent, the user should understand terms such as: cylinder, sector, cluster allocation table, retrieval information block, segment descriptor, and cluster.

36.4 Minimal Operator Interface

This section is for those who wish to use REPAIR to make their disk work again as quickly and with as little effort as possible. To use this section requires no knowledge of the DOS beyond the concept of files. It does require the ability to read through and understand the following step-by-step instructions.

In the most ultimately simple case, the user will not want to lock out any cylinders (a cook-book process -- you don't have to know what a cylinder is), and the REPAIR program will not find any errors. The main structure of the following example is built on such a case: however, places in the example where there may be variations are noted and where in the chapter to find explanations of the variations is also noted.

36.4.1 Executing REPAIR

If REPAIR is catalogued on the disk (as REPAIR/CMD), and if the DOS is capable of loading and executing it, the fastest and easiest way to get REPAIR started is by simply keying the command at the system console:

```
REPAIR
```

REPAIR may also be executed by placing a LOAD-AND-GO (LGO) tape of REPAIR in the back cassette deck and pressing RESTART key (and RUN key simultaneously on the 5500).

In either case, the pack to be checked must be placed in a

drive connected to the computer and brought on line.

In the following examples, a pictogram of the state of the CRT display will be given followed by a brief explanation and instructions for the operator.

Note that a pound sign (#) in one of the bottom two lines of the pictogram represents the cursor position. The cursor will be flashing when the operator is required to respond to the information on the screen.

36.4.2 Sign-on and drive number specification

```
                    DATAPOINT DOS.  REPAIR




DRIVE NUMBER: #
```

The screen appears as above when REPAIR has been loaded and execution has begun.

The operator must enter the logical number of the drive holding the disk pack that is to be REPAIRed.

36.4.3 Cylinder Lockout

```
                    DATAPOINT DOS.  REPAIR




DRIVE NUMBER: 0
LOCKOUT CAT: FORMAT LOOKS OK.
DO YOU WANT TO LOCK OUT ADDITIONAL CYLINDERS ?  #
```

The screen may appear as above when REPAIR is ready to accept cylinder lockout. Cylinder lockout is a way of reserving disk space from DOS use. If cylinders are to be locked out, there will generally be a sticker or label on the case of the disk pack with the numbers of the cylinders to be reserved. If there are cylinders to be locked out refer to the Cylinder Lockout with REPAIR Section.

If no cylinders are to be locked out, enter "N".

36.4.4 Directory check monitor

```
                                                                  0 0
                                                                  0 0
                                                                  0 0

DRIVE NUMBER 0
LOCKOUT CAT: FORMAT LOOKS OK.
DO YOU WANT TO LOCK OUT ADDITIONAL CYLINDERS ? N
WORKING CAT: FORMAT LOOKS OK.
```

The screen appears as above when the cylinder lockout option has not been taken and the CAT and Directory check phase has begun. Specifically, note the vertical numbers at the right center of the screen: these numbers monitor the cycling of the directory check. If something besides these numbers appears on the screen after the cylinder lockout is completed, refer to the CAT errors and Directory read/write errors Section.

No operator response is required.

36.4.5 Directory Errors

```
DIRECTORY ENTRY COPY: DELETE INCOMPLETE


          M A S T E R :                          C O P Y :

0 3 3 0                          3    3 3 3 3                          3    0 0
0 0 5 0                          7    7 7 7 7                          7    0 2
2 6 4 1 C A T          C M D 7   7    7 7 7 7 C A T          C M D 7   6    0 0

ENTER: 1=MASTER->COPY, 2=DELETE BOTH, 3=NO CHANGE: #
```

The screen will appear as above (in general -- specific words will vary) if REPAIR finds an error in the directory.

For explanation of the messages refer to the Directory Errors section of Medial Interface below.
To delete the erroneous file enter the number corresponding to DELETE BOTH (in this example, 2=DELETE BOTH).

36.4.6 Retrieval Information Blocks check

```
          RIB MASTER:              (PFN 000)       RIB COPY:
                                       *
                                       *
                                       *
                                       *
                                       *
                                       *                        0 0
                                                                0 0
                                                                0 0
```

The screen appears as above during the Retrieval Information Blocks check. The vertical numbers at the right of the screen monitor the cycling of the RIB check. The column of asterisks is displayed only while a RIB is actually being checked. If a pack does not have many files on it the asterisks will not appear during most of the RIB check.

No operator response is required.

36.4.7 Retrieval Information Blocks Errors

```
PFN ERROR                LRN ERROR     * PFN ERROR                LRN ERROR
4TH BYTE NOT 0377                      * 4TH BYTE NOT 0377
1ST SEG.DES. ERROR                     * 1ST SEG.DES. ERROR
MULTIPLE ALLOCATION 00001              * MULTIPLE ALLOCATION 00001
CYL.ADR.OVERFLOW         CYL.ERROR     * CYL.ADR.OVERFLOW         CYL.ERROR
RIB TERMINATOR ERROR                   * RIB TERMINATOR ERROR
                      0 3 2 0                          3
                      0 0 1 2                          7
                      1 0 0 3 S Y S T E M 0   S Y S 7

DELETE THE FILE ? #
```

The screen will appear as above (in general - specific words will vary) if REPAIR finds an error in a RIB.

For explanation of the messages refer to the RIB Errors section of Medial Interface below.

To delete the erroneous file enter "Y".

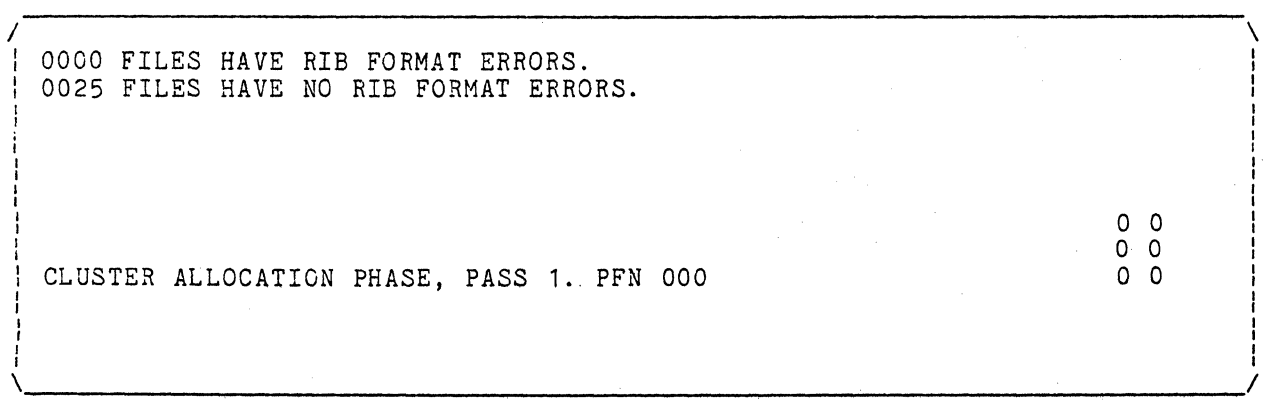
36.4.8 End of RIB check

```
0000 FILES HAVE RIB FORMAT ERRORS.
0025 FILES HAVE NO RIB FORMAT ERRORS.




CLUSTER ALLOCATION PHASE, PASS 1. PFN #
```

The screen appears as above when the RIB check phase is finished. The messages at the top of the screen are a summary of the information accumulated during the RIB check phase. The message near the bottom of the screen is notification to the operator that REPAIR is ready to begin the cluster allocation phase.

To proceed, depress the <ENTER> key. If no RIB format errors remain on the disk, pressing <ENTER> is not required.

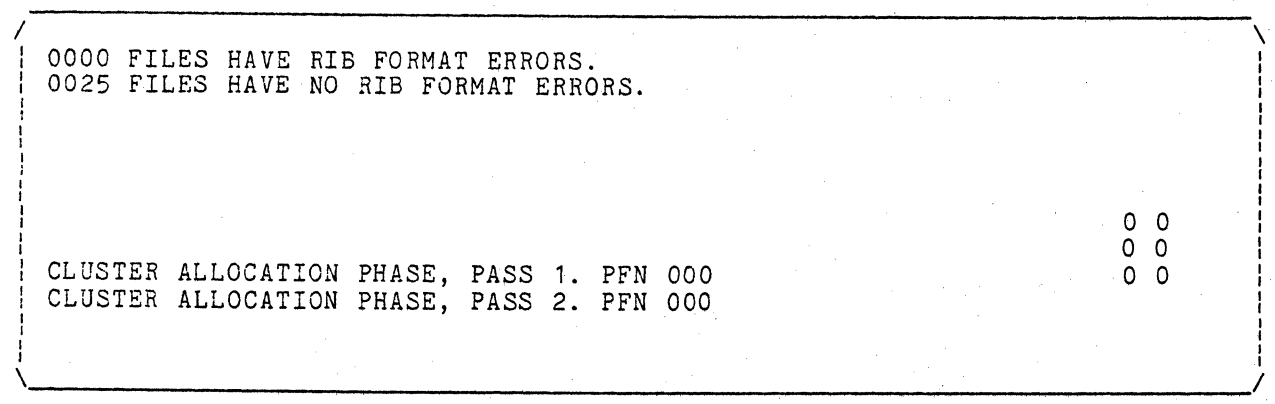
36.4.9 Cluster allocation phase, Pass 1



The screen appears as above during the first pass of the cluster allocation phase. The vertical numbers at the right of the screen are the pass cycle monitor.

No operator response is required.

36.4.10 Cluster allocation phase, Pass 2



The screen appears as above during the cluster allocation phase, pass 2. The bottom message is displayed and the cycle monitor numbers at the right of the screen are restarted when pass 2 begins.

No operator response is required.

36.4.11 Cluster allocation phase, pass 3

```
 0000 FILES WITH ALLOCATION CONFLICTS.
00000 CLUSTERS IN THOSE FILES.




                                                          0 0
                                                          0 0
CLUSTER ALLOCATION PHASE, PASS 1. PFN 000                 0 0
CLUSTER ALLOCATION PHASE, PASS 2. PFN 000
CLUSTER ALLOCATION PHASE, PASS 3. #
```

The screen appears as above at the end of the cluster allocation phase, pass 2. The messages at the top of the screen are a summary of the information gathered during cluster allocation phase pass 1 and 2. The message at the bottom of the screen indicates that REPAIR is ready to begin the cluster allocation phase pass 3.

To proceed, depress the <ENTER> key. If no allocation conflicts are present, it is not necessary to press <ENTER>.

36.4.12 Cluster Allocation Conflicts

```
PFN 200                                   PFN 220
 0 0 3 0                          3        0 0 3 0                          3
 0 0 6 0                          7        0 0 6 0                          7
 3 0 4 1 S I N           C M D 7           3 6 4 1 S O U T         C M D 7
# OF CLUSTERS IN FILE: 00001               # OF CLUSTERS IN FILE: 00002
# OF CONFLICTING FILES: 002                CONFLICTING FILE # 001
# OF CONFLICTING CLUSTERS: 00001           # OF CONFLICTING CLUSTERS: 00001
# OF CORRECT PFN/LRN: 00004 OF 00006       # OF CORRECT PFN/LRN: 00000 OF 00006


ENTER: DELETE FILE: 1=LEFT, 2=RIGHT, 3=BOTH; 4=NO CHANGE: #
```

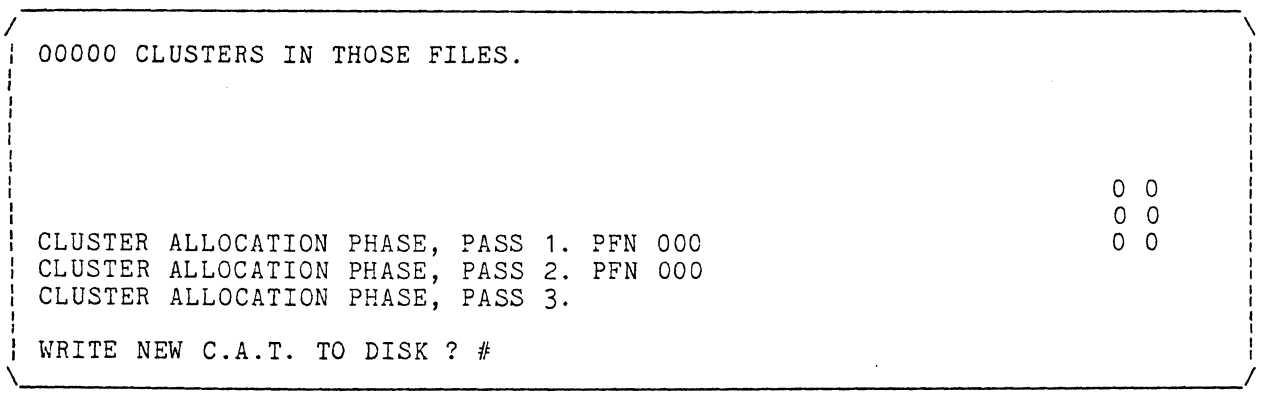
The screen will appear as above (in general, specific words will vary) if REPAIR finds that two or more files are trying to use the same space on disk.

For explanation of the messages refer to Cluster Allocation

Conflicts in Medial Interface below.

To delete the files in error enter "3".

36.4.13 System Table Replacement



REPAIR will compare its generated CAT with the one on disk. If they match, the message:

COMPUTED C.A.T. MATCHES DISK

will appear. Otherwise, the message on the last line of the screen above will appear.

To overwrite the CAT on disk enter "Y". To prevent overwrite of the CAT on disk enter "N". If no errors have been discovered by REPAIR, the operator should enter "Y".

REPAIR will then compare its Lockout CAT with the one on disk. If they match, the message:

COMPUTED LOCKOUT C.A.T. MATCHES DISK

will appear. Otherwise a message will appear asking if the Lockout CAT is to be written back to the disk.

To overwrite the Lockout CAT on disk (making any additional cylinders locked out during the primary stages of REPAIR permanent) enter "Y". To prevent overwrite of the Lockout CAT on disk enter "N". If no errors in the Lockout CAT have been discovered by REPAIR and no additional cylinders were locked out, the operator should enter "Y".

REPAIR will finally generate a Hashed Directory Index and

compare it with the one on disk. If they match, the message:

COMPUTED H.D.I. MATCHES DISK

will appear. Otherwise a message will appear asking if the Hashed Directory Index is to be written back to disk; Enter a "Y" if so, on "N" if not. The H.D.I. check is not performed on a diskette system.

```
                                                                  0 0
                                                                  0 0
CLUSTER ALLOCATION PHASE, PASS 1. PFN 000                         0 0
CLUSTER ALLOCATION PHASE, PASS 2. PFN 000
CLUSTER ALLOCATION PHASE, PASS 3.

WRITE NEW C.A.T. TO DISK ? N
WRITE NEW LOCKOUT CAT TO DISK ? N
COMPUTED H.D.I. MATCHES DISK
DISK REPAIR DONE.
```

The message on the last line of the screen above will appear when the REPAIR program is finished checking the disk. The REPAIR program will attempt to re-load the DOS when it is finished if it was loaded from DOS.

No operator response is required.

36.5 Medial Operator Interface

This section is a rather comprehensive discussion of the various messages and options provided by REPAIR, and is for those who wish to be able to take advantage of the file-saving options available with REPAIR. To use this section will require that the operator gain an understanding of whatever error(s) REPAIR finds that the he wishes to repair. For example, if the only errors on the user's disk are in the directory, it is not necessary to study Retrieval Information Blocks or Cluster Allocation.

This section follows the section numbering scheme of the previous section, Minimal Operator Interface.

When a facet of REPAIR operation is discussed more

appropriately elsewhere, the discussion is not repeated in this section, but the reader is referred to the section containing the discussion. When the section referenced is the corresponding section under Minimal Operator Interface, this section will simply say "See Minimal Interface".

To use this section requires that the user have a copy of and understand the use of either of the DOS commands, DUMP or DUMP93X0. The ability to use the Assembler may be mandatory in some cases.

This section assumes that the REPAIR program is used as an error-finding tool, and that the user, with the aid of one of the DUMP programs and special programs he may create, can fix errors that develop on the disk. A specific example is the case of a file with bad RIBs. REPAIR can tell the operator that the file's RIBs contain errors. Either DUMP program can be used to determine the magnitude of the damage to the RIBs, and, if necessary, where the file's records actually are on disk. If necessary, the user can create a simple Assembly language program to re-create the file's RIBs on disk. Sometimes it will be less effort to re-create a file's RIBs than to re-create the file itself.

36.5.1 Executing REPAIR

If REPAIR is cataloged on the disk (as REPAIR/CMD), and if the DOS is capable of loading and executing it, the fastest and easiest way to get REPAIR started is by simply keying the command at the system console:

```
REPAIR
```

REPAIR may also be executed by placing a LOAD-AND-GO (LGO) tape of REPAIR in the rear cassette deck and pressing RESTART.

In either case, the pack to be checked must be placed in a drive connected to the computer and brought on line.

Note that a pound sign (#) in one of the bottom two lines of the pictogram represents the cursor position. The cursor will be flashing when the operator is required to respond to the information on the screen.

36.5.2 Sign-on and drive number specification

See Minimal Interface for illustration.

After the operator has entered the number of the drive holding the pack to be REPAIRed, REPAIR will wait for that drive to come ready before proceeding to do cylinder lockout.

36.5.3 Cylinder lockout

The Cylinder Lockout with REPAIR Section is a discussion with examples of the cylinder lockout process.

Cylinders are locked out because they give read/write errors or because by system design they are to be reserved for some special use.

If the user is not sure whether cylinders have been locked out on a disk pack (and the Lockout CAT and backup have both been destroyed), either of the DUMP programs can be used to look at the cylinders on disk.

Cylinders that have been reserved for special use can generally be recognized by the formatting of their sectors. Sectors that have not been used by the normal DOS routines will not have the special DOS header information in the first three bytes. The first byte is the PFN (Physical File Number) of the file, and the second and third bytes give the LRN (Logical Record Number) of the record in the file. For records that have been written by the normal DOS, each cluster will have the same first byte, and the second and third bytes will be incremented by one (LSP, MSP).

36.5.4 Directory check monitor

See Minimal Interface for illustration.

The directory check monitor is the means by which REPAIR indicates its progress to the operator. Specifically, the directory check monitor constantly displays the disk address of the current directory entry being checked. This display is in the form of two vertically displayed octal numbers at the right of the screen. The first number is a directory sector number indicator, and the second number is the buffer page address of the directory entry being checked.

If the directory check monitor stops and no other messages are displayed, then the REPAIR program was loaded to memory improperly or something is wrong with the hardware.

If a page in the directory has been accidentally overwritten by a record from a file, then REPAIR will find many errors in that directory page. If while executing REPAIR the operator notices that there are quite a few errors in the directory, he can note the directory page address as shown by the directory check monitor. (The left number of the directory check monitor is the physical sector number of the directory page). Using either of the disk dump programs the operator can look at the bad directory page(s).

If the damage is only to one copy of the directory (the usual case) then REPAIR can recover the directory. However, the operator may wish to use either DUMP command to look at the directory to see if, by examining the data there, he can determine if an error in a user program has caused the directory to be overwritten. Clues to such events can be gleaned by noting the first byte of the record (which would be a file PFN), for example.

36.5.5 Directory errors

The directory is a table of entries for files on the pack. There are two copies of the directory, the MASTER and the COPY. There are 16 pages to each copy of the directory, each page holds entries for up to 16 files. (One disk physical sector is one directory logical page). Thus, the directory has a MASTER and a COPY entry for up to 256 files.

The REPAIR program checks the directory one file at a time. That is, the MASTER and the COPY of a directory entry are checked at the same time.

If an error in the MASTER or the COPY entry or both is detected, REPAIR will display:

1. A brief error description at the top of the screen,

2. The MASTER and COPY entry across the lower center of the screen,

3. An option message near the bottom of the screen.

The error description will indicate whether the error is in the MASTER or the COPY entry or both, and will define the type of

error.

Note that although directory entries for a file may have several types of errors at the same time, REPAIR will only deal with one error type at a time.

The directory entries are displayed under their respective headings- MASTER: and COPY: . The first four bytes and the last byte of each entry are always displayed in vertical octal. The 5th thru 15th bytes (being the file name and extension) of each entry are displayed in ASCII except for bytes in those fields which cannot be displayed in ASCII on the CRT display; those bytes will be converted to vertical octal.

The option message at the bottom of the screen will enable the operator, by selecting and entering a digit, to correct the MASTER entry with information from the COPY entry, to correct the COPY entry with information from the MASTER entry, to delete both entries (and thus the file), or to make specific changes to one or both entries, or to make no change at all to either entry.

Below are examples of the various directory errors that may occur and discussions of the respective messages. The first example is the most complete; the other directory error routines work basically the same way but their examples are not as expanded.

Note that for the examples concerning the directory MASTER, the same messages (transposing the words COPY and MASTER) apply to the directory COPY.

36.5.5.1 Delete errors

Delete errors include those where the directory entry master is deleted and the copy is not deleted, or the directory entry master is partially deleted.

36.5.5.1.1 One entry deleted

```
DIRECTORY ENTRY MASTER: DELETED


          M A S T E R :                          C O P Y :

3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3    0 3 3 0                              3    0 0
7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7    0 0 6 0                              7    0 2
7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7    2 0 4 1 C A T              C M D 7    6 0

ENTER: 1=COPY->MASTER, 2=DELETE BOTH, 3=NO CHANGE: #
```

The screen will appear as above if REPAIR finds a file for which the directory MASTER entry is deleted (filled with 0377's) but the directory COPY is not.

The operator has three options:

1. Enter "1" to copy the COPY entry to the MASTER entry, thus saving the file's name in the directory;

2. Enter "2" to delete both entries, and thus the file;

3. Enter "3" to take no action on the file's entries and continue the directory check.

```
DIRECTORY ENTRY MASTER: DELETED


          M A S T E R :                          C O P Y :

3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3    0 3 3 0                              3    0 0
7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7    0 0 6 0                              7    0 2
7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7    2 0 4 1 C A T              C M D 7    6 0

ENTER: 1=COPY->MASTER, 2=DELETE BOTH, 3=NO CHANGE: 1
*** ARE YOU SURE ? *** #
```

The message on the last line of the screen above will appear when the operator has selected and entered one of the digits given in the option message. REPAIR will always make sure the operator entered what he intended to before proceeding to carry out the function.

To carry out the function selected enter "Y". If "N" is entered the option message will be re-displayed.

```
/                                                                                  \
|                                                                                  |
|                                                                                  |
|                                                                                  |
|          M A S T E R :                          C O P Y :                        |
|                                                                                  |
| 0 3 3 0                        3    0 3 3 0                        3             |
| 0 0 6 0                        7    0 0 6 0                        7    0 0      |
| 2 0 4 1 C A T            C M D 7    2 0 4 1 C A T            C M D 7    0 2      |
|                                                                          6 0      |
| ENTER: 1=COPY->MASTER, 2=DELETE BOTH, 3=NO CHANGE: 1                             |
| *** ARE YOU SURE ? *** Y                                                         |
| DONE.                                                                            |
\                                                                                  /
```

The screen will appear as above if the operator has replied "1" to the message above and replied "Y" to the message "*** ARE YOU SURE ? ***". When REPAIR does an entry to entry copy, both entries are re-displayed to show the operator the results of the copy, and the message: "DONE." is displayed at the bottom of the screen.

No further operator response is required.

```
/                                                                                  \
| DIRECTORY ENTRY MASTER: DELETED                                                  |
|                                                                                  |
|                                                                                  |
|                                                                                  |
|          M A S T E R :                          C O P Y :                        |
|                                                                                  |
| 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3    3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3          |
| 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7    7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7    0 0      |
| 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7    7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7    0 2      |
|                                                                          6 0      |
| ENTER: 1=COPY->MASTER, 2=DELETE BOTH, 3=NO CHANGE: 2                             |
| *** ARE YOU SURE ? *** Y                                                         |
| DONE.                                                                            |
\                                                                                  /
```

The screen will appear as above if the operator has replied "2" to the message above and replied "Y" to the message "*** ARE YOU SURE ? ***". When REPAIR deletes the entries from the directory, the entries are re-displayed to show the operator that the delete has been accomplished, and the message: "DONE." is displayed at the bottom of the screen.

No further operator response is required.

```
                M A S T E R :                          C O P Y :

 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3    0 3 3 0                              3   0 0
 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7    0 0 6 0                              7   0 2
 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7    2 0 4 1 C A T                C M D 7   6 0

 ENTER: 1=COPY->MASTER, 2=DELETE BOTH, 3=NO CHANGE: 3
 *** ARE YOU SURE ? *** Y
```

The screen will appear as above if the operator has replied "3" to the message above and replied "Y" to the message "*** ARE YOU SURE ? ***". REPAIR will make no change to the entries and will resume the directory check.

No further operator response is required.

36.5.5.1.2 Delete Incomplete

```
 DIRECTORY ENTRY MASTER: DELETE INCOMPLETE

                M A S T E R :                          C O P Y :

 3 3 3 3 3 3                        3    0 3 3 0                              3   0 0
 7 7 7 7 7 7                        7    0 0 6 0                              7   0 2
 7 7 7 7 7 7 T            C M D 7    2 0 4 1 C A T                C M D 7   6 0

 ENTER: 1=COPY->MASTER, 2=DELETE BOTH, 3=NO CHANGE: #
```

The screen will appear as above if REPAIR finds a file for which the directory MASTER entry is partially deleted (partially filled with 0377's) but the directory COPY is not.

The operator options and REPAIR actions are the same as for one entry deleted, see the preceding section.

36.5.5.2 RIB Address Errors

RIB Address errors include invalid RIB addresses or unequal RIB addresses between the directory MASTER and COPY.

36.5.5.2.1 RIB Address Invalid

```
DIRECTORY ENTRY MASTER: R.I.B. ADDRESS INVALID


          M A S T E R :                          C O P Y :

0 3 3 0                          3    0 3 3 0                          3    0 0
0 0 6 0                          7    0 0 6 0                          7    0 2
0 0 4 1 C A T          C M D 7   4    2 0 4 1 C A T          C M D 7   6    0

ENTER: 1=COPY->MASTER, 2=DELETE BOTH, 3=NO CHANGE: #
```

The screen will appear as above if REPAIR finds a directory MASTER entry with an invalid RIB address.

In this example, the RIB address of the directory MASTER is invalid because the cylinder address is 000.

The RIB address is the first byte and the top two digits of the second byte of a directory entry. The first byte is the cylinder address and to be valid must be an octal number in the range 001 thru the maximum cylinder number for the DOS in use (DOS.A - 0312, DOS.B - 0312, DOS.C - 0114, DOS.D - 0374, DOS.E - 0312). The top two digits of the second byte define the cluster number and to be valid must be one of the following:

00, 04, 10, 14, 20, 24, 30, 34

For diskette systems, the valid two digits are only:

00, 10, 20, 30

The operator has three options:

1. Enter "1" to copy the COPY entry RIB address to the MASTER entry RIB address;

2. Enter "2" to delete both entries, and thus the file;

3. Enter "3" to take no action on the file's entries and resume the directory check.

```
DIRECTORY ENTRY MASTER: R.I.B. ADDRESS INVALID


          M A S T E R :                          C O P Y :

0 3 3 0                      3    0 3 3 0                      3    0 0
0 0 6 0                      7    0 0 6 0                      7    0 2
0 0 4 1 C A T          C M D 7    2 0 4 1 C A T          C M D 7    6 0

ENTER: 1=COPY->MASTER, 2=DELETE BOTH, 3=NO CHANGE: 1
*** ARE YOU SURE ? *** #
```

The message on the last line of the screen above will appear if the operator has replied "1" to the message above.

To carry out the function selected enter "Y".

```
DIRECTORY ENTRY MASTER: R.I.B. ADDRESS INVALID


          M A S T E R :                          C O P Y :

0 3 3 0                      3    0 3 3 0                      3    0 0
0 0 6 0                      7    0 0 6 0                      7    0 2
0 0 4 1 C A T          C M D 7    2 0 4 1 C A T          C M D 7    6 0

ENTER: 1=COPY->MASTER, 2=DELETE BOTH, 3=NO CHANGE: 1
MOVE ENTIRE ENTRY ? #
```

The message on the last line of the screen above will appear if the operator replied "Y" to the message "*** ARE YOU SURE ? ***".

Enter "N" to have REPAIR copy the RIB address (only) from the COPY entry to the MASTER entry. Enter "Y" to have REPAIR copy the entire COPY entry to the MASTER entry.

The "MOVE ENTIRE ENTRY ?" option is given to give the operator the ability to correct many types of errors in an erroneous entry at one time, rather than correct each error as it is found. If the operator can recognize a severly destroyed entry the first time he sees it, this option can enable him to repair the directory more quickly.

36.5.5.2.2 RIB Addresses not equal

```
DIRECTORY ENTRY MASTER & COPY: R.I.B. ADDRESSES NOT EQUAL


          M A S T E R :                        C O P Y :

0 3 3 0                      3    0 3 3 0                      3    0 0
0 0 6 0                      7    0 0 6 0                      7    0 2
2 0 4 1 C A T          C M D 7    3 0 4 1 C A T          C M D 7    6 0

ENTER: 1=MASTER->COPY, 2=COPY->MASTER, 3=DELETE BOTH, 4=NO CHANGE: #
```

The screen will appear as above if REPAIR finds a file with directory entries with RIB addresses that are both valid but not equal.

In this example, the RIB address in the MASTER is 002,300 and in the COPY is 003,300.

The operator has four options:

1. Enter "1" to copy the MASTER entry RIB address to the COPY entry RIB address;

2. Enter "2" to copy the COPY entry RIB address to the MASTER entry RIB address;

3. Enter "3" to delete both entries, and thus the file;

4. Enter "4" to take no action on the file's entries and resume the directory check.

If it is not obvious by visual inspection of the directory entries which is in error, the operator should note the RIB address as given by each directory entry, and enter "4". If REPAIR later discovers PFN and LRN errors in the actual RIBs for the file (see Retrieval Information Blocks Errors), then the operator can be fairly sure the directory MASTER entry for the file is in error, since only the directory MASTER entry is used to determine the RIB address of a file for the RIB check phase of REPAIR.

If the operator wants to make very sure which, if either, of

the directory entries is correct, he can use the DUMP or DUMP93X0 commands to look at the file after REPAIR has finished execution.

When it is determined which directory entry for the file has the correct RIB address, the operator can execute REPAIR again, this time entering "1" or "2" as appropriate to correct the erroneous directory entry.

If neither entry is correct, and it would be easier to modify the directory entries for the file than to delete them and re-create the file, refer to the available DOS documentation for details on ways to modify the directory sectors on disk.

36.5.5.3 File protection not same

```
DIRECTORY ENTRY MASTER & COPY: FILE PROTECTION NOT SAME

ENTRY MASTER: WRITE PROTECTION
  ENTRY COPY: NO PROTECTION
          M A S T E R :                          C O P Y :

0 3 3 0                    3    0 3 3 0                    3    0 0
0 0 6 0                    7    0 0 6 0                    7    0 2
2 3 4 1 C A T        C M D 7    2 0 4 1 C A T        C M D 7    6 0

ENTER:1=DELETE ENTRIES,2=NO CHANGE;PROTECTION: 3=NONE,4=DELETE,5=WRITE: #
```

The screen will appear as above if REPAIR finds a file with directory entries with protection not the same.

In this example, the directory MASTER entry has WRITE protection indicated for the file, while the directory COPY entry has no protection indicated for the file. Note: where the bits for both WRITE and DELETE protection are set, WRITE protection has precedence, since WRITE protection implies DELETE protection.

The protection indication is in the bottom two bits (bottom digit) of the second byte of a directory entry. If the upper bit of the two is set on (the digit is 2) then the directory entry indicates that the file is DELETE protected. If the bottom bit is set on (the digit is 1 or 3) then the directory entry indicates that the file is WRITE protected. If neither of the two bits is set on (the digit is 0) then the directory entry indicates NO protection for the file, that is, that the file is unprotected.

The operator has five options:

1. Enter "1" to delete both entries, and thus the file;

2. Enter "2" to take no action on the file's entries and resume the directory check;

3. Enter "3" to set both entries to indicate NO protection;

4. Enter "4" to set both entries to indicate DELETE protection;

5. Enter "5" to set both entries to indicate WRITE protection.

36.5.5.4 Name-Extension not equal

```
DIRECTORY ENTRY MASTER & COPY: NAME-EXTENSION NOT EQUAL


          M A S T E R :                          C O P Y :

0 3 3 0                          3    0 3 3 0                          3    0 0
0 0 6 0                          7    0 0 6 0                          7    0 2
2 0 4 1 C A T X X X X X C M D 7       2 0 4 1 C A T          C M D 7       6 0

ENTER: 1=MASTER->COPY, 2=COPY->MASTER, 3=DELETE BOTH, 4=NO CHANGE: #
```

The screen will appear as above if REPAIR finds a file with directory entries that do not have the same NAME/EXTENSION.

The NAME/EXTENSION of a directory entry is located in bytes 5 through 15 inclusively. The NAME/EXTENSION of a directory entry (and the file) is the normal means by which the file is identified and manipulated, especially from the DOS system console.

Note that REPAIR does not seek or identify as erroneous files with NAME/EXTENSIONS that contain non-ASCII characters, since by DOS rules non-ASCII characters are perfectly legal in the NAME/EXTENSION field.

The operator has four options:

1. Enter "1" to copy the MASTER entry NAME/EXTENSION to the COPY entry NAME/EXTENSION;

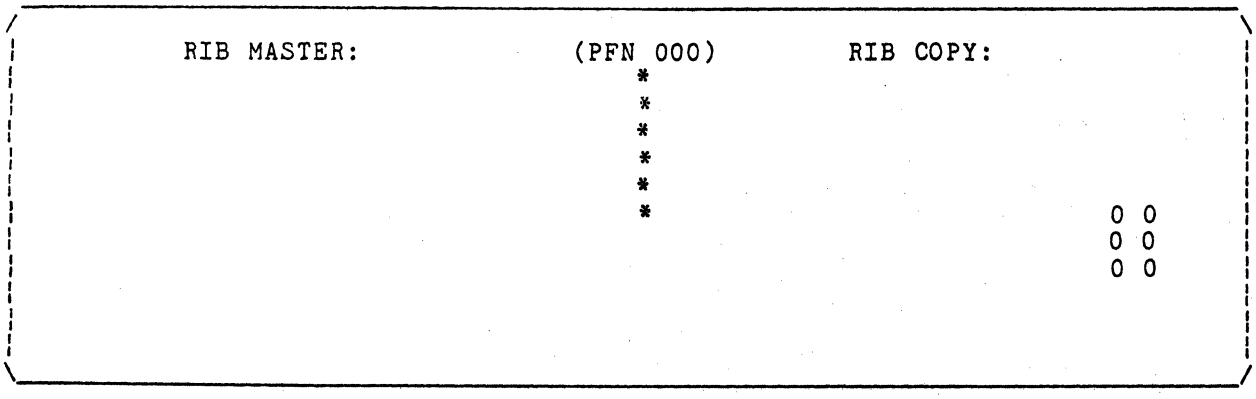
2. Enter "2" to copy the COPY entry NAME/EXTENSION to the

MASTER entry NAME/EXTENSION;

3. Enter "3" to delete both entries, and thus the file;

4. Enter "4" to take no action on the file's entries and resume the directory check.

36.5.6 Retrieval Information Blocks check



The screen appears as above during the Retrieval Information Blocks check.

REPAIR checks Retrieval Information Blocks (RIBs) for all files in the directory with a valid RIB address, in the order of the files' occurrence in the directory. The three-digit octal number after "PFN" in the top line will indicate the actual Physical File Number currently being checked. The two vertically-displayed octal numbers at the right of the screen provide the same information as the directory check monitor described previously.

There are two RIBs for each file, a MASTER and a COPY. The RIB MASTER is the very first record in the file and the RIB COPY is the second record in the file. Each RIB uses one full 256-byte disk sector. Refer to the System Structure Chapter for a description of the structure of the RIBs.

If REPAIR detects any errors in the RIB MASTER a message describing the class of error will be displayed in the portion of the screen under the heading "RIB MASTER:". If REPAIR detects any errors in the RIB COPY a message will be displayed in the portion of the screen under the heading "RIB COPY:".

The PFN indicator and the cycle monitor numbers are

incremented and displayed for each entry in the directory. The column of asterisks is displayed only while the RIBs for a file are actually being checked.

36.5.7 Retrieval Information Blocks errors

```
        RIB MASTER:                (PFN 000)          RIB COPY:
PFN ERROR                LRN ERROR     * PFN ERROR                LRN ERROR
4TH BYTE NOT 0377                      * 4TH BYTE NOT 0377
1ST SEG.DES. ERROR                     * 1ST SEG.DES. ERROR
MULTIPLE ALLOCATION 00001              * MULTIPLE ALLOCATION 00001
CYL.ADR.OVERFLOW         CYL.ERROR     * CYL.ADR.OVERFLOW         CYL.ERROR
RIB TERMINATOR ERROR                   * RIB TERMINATOR ERROR
```

The screen will appear as above if REPAIR finds errors in the RIB MASTER or COPY for a file. Note that all of the messages given in the example above will not necessarily appear. The pictogram above shows the screen as it would appear while the RIB check was in progress. The next pictogram shows the state of the screen when the RIB check has finished and has displayed the file's directory MASTER entry and is ready for operator response.

Below is a discussion of each of the messages in the screen above. In the above pictogram all possible messages are shown in their respective positions for both the RIB MASTER and the RIB COPY. Note that since the RIB MASTER and COPY have the same formats, (indeed, normally they are exact duplicates of each other, except for their Logical Record Number [LRN]) they can have the same errors.

There are two types of errors that a RIB may have: simple and complex. If REPAIR finds only one simple error in only one of the RIBs then the operator will be given the option of having REPAIR correct the error. If multiple simple errors or any complex errors are detected then the errors are too serious for REPAIR to cope with, and will only give the operator a choice between deleting the file or making no change at all. Even with multiple or complex errors the file may be saveable.

PFN ERROR

This message is displayed if the first byte of the RIB is not the file's Physical File Number (PFN). This is a simple error and is correctable under the conditions given above.

LRN ERROR

This message is displayed for the RIB MASTER if the Logical Record Number (LRN) is not zero, and for the RIB COPY if the LRN is not one. This is a simple error and is correctable under the conditions given above.

4TH BYTE NOT 0377

This message is displayed if the 4th byte of the RIB is not 0377. When the DOS object code loader loads a program into memory it skips over disk records with a 0377 in the 4th byte: since the RIBs of a file are not part of the object code of a file their fourth byte should always be 0377 so the loader will not attempt to load them to memory. This is a simple error and is correctable under the conditions given above.

1ST SEG.DES. ERROR

Expanded: First Segment Descriptor Error. This message is displayed if the first segment descriptor of the RIB does not point to itself. Since the RIBs are the first two records in any file, they will always be in the first cluster. The first segment descriptor must point to the beginning of the file, which is the cluster where the RIBs are.

MULTIPLE ALLOCATION 00001

This message is displayed if REPAIR discovers that, according to the RIB's segment descriptors, two or more segments of the file overlap. Specifically, segment descriptors identify clusters on the disk which belong to the given file. If one or more of these clusters is indicated as belonging to more than one segment, then there is mulitple allocation of clusters. The five digit octal number indicates how many clusters are multiply allocated.

CYL.ADR.OVERFLOW

This message is displayed if REPAIR discovers a segment descriptor which indicates that a segment overruns the physical end of the disk. Of course it is not actually possible for a file to extend beyond the upper limit of the disk space, but it is possible for a segment descriptor to erroneously indicate this. For example, a segment descriptor might say, in effect: "This

segment begins at the last cluster on the disk and extends for ten clusters".

CYL.ERROR

This message is displayed if REPAIR discovers a segment descriptor with a cylinder address that is either 0 (always reserved for the Cluster Allocation Table and the Directory) or greater than the maximum cylinder number allowed by the DOS (DOS.A - 0312, DOS.B - 0312, DOS.C - 0114, DOS.D - 0374, DOS.E - 0312).

RIB TERMINATOR ERROR

This message is displayed if REPAIR discovers a RIB that has an incorrect terminator. The logical end of a RIB is indicated by either the actual physical end of the disk record or a pair of 0377's. An 0377 in the first byte of a segment descriptor but a non-0377 in the second byte defines a TERMINATOR ERROR.

```
PFN ERROR                              *
                                       *
                                       *
                                       *
                                       *
                                       *
                      0 2 3 0                          3
                      0 0 6 0                          7
                      2 0 4 1 C H A N G E    C M D 7

DELETE THE FILE ? #
```

The screen will appear as above when REPAIR has completed the RIB check for a file whose RIBs had only one simple error. Note that the screen is rolled up one line so that the heading containing the PFN is no longer displayed. However the directory MASTER entry for the file, containing the NAME/EXTENSION for the file, is displayed under the error message area.

To simply have REPAIR delete the file enter "Y". To attempt to save the file enter "N".

```
                                    *
                                    *
                                    *
                                    *
                                    *
                    0 2 3 0                         3
                    0 0 6 0                         7
                    2 0 4 1 C H A N G E     C M D 7


 DELETE THE FILE ? N
 WRITE CORRECTION TO DISK ? #
```

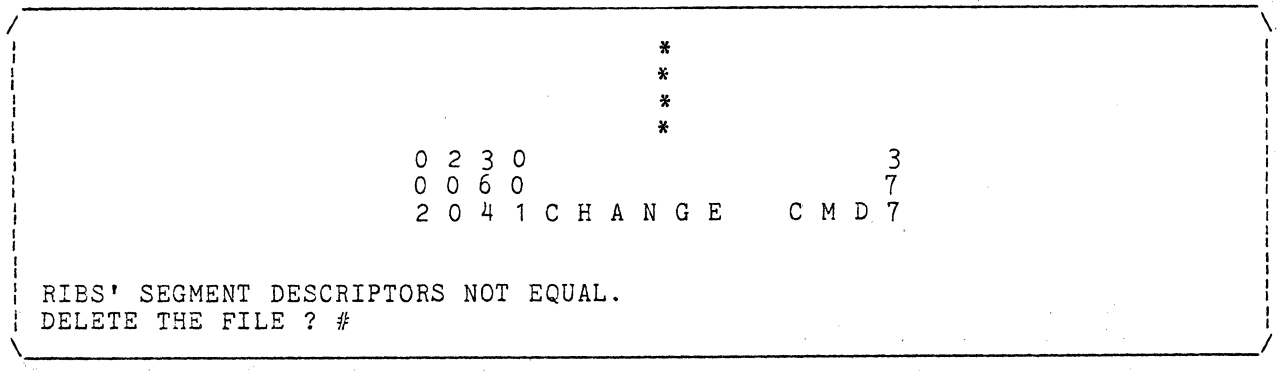
The message on the last line of the screen above will appear if the operator replied "N" to the message above.

Enter "Y" to have REPAIR write the correct RIB information to the RIB in error and resume the RIB check.

```
                                    *
                                    *
                                    *
                                    *
                    0 2 3 0                         3
                    0 0 6 0                         7
                    2 0 4 1 C H A N G E     C M D 7


 DELETE THE FILE ? N
 WRITE CORRECTION TO DISK ? N
 FILE SPACE WILL NOT BE ALLOCATED.#
```

The message on the last line of the screen above will appear if the operator replied "N" to the message above. REPAIR will not allocate space (by setting the appropriate bits in the CAT) for a file if there is any uncorrected/uncorrectable error in either of the RIBs.

REPAIR will wait until the operator depresses the ENTER key before resuming the RIB check.



The messages on the last two lines of the screen above will appear if the RIB MASTER and COPY for a file individually have no format errors but do not describe the same segments for the file.

Enter "Y" to delete the file from the directory. Enter "N" to make no change to the file and resume the RIB check.

NOTE that whether or not the file is deleted REPAIR will not allocate any space on disk for the file (refer to the pictogram and discussion above). The consequence of this is that, although the file will still be accessable, the space it occupies is marked as available for allocation to some other file. As a result, the remains of the file on disk are almost certain to be overwritten by some other file sooner or later.

Complex RIB errors can come in infinitely many kinds and combinations. The REPAIR diagnostics will describe specific errors, but if the user is considering fixing a RIB he must examine the RIB himself and determine what is wrong with it and how to correct it. Sometimes this will involve examining records on disk and determining whether or not the records belong to the file and how they should be organized in the RIBs segment descriptors. Because of the potential complexity of this operation the current version of REPAIR does not attempt the analysis necessary to re-construct a RIB with complex errors.

36.5.7.1 A simple case

A relatively simple-to-fix case might go like this:

1. REPAIR would find a file with simple and complex errors in the RIB MASTER.

2. The user would use either disk dump program to look at the

RIBs and determine that the RIB MASTER had somehow been completely destroyed, but the format of the RIB COPY seemed to be correct.

3. Using the information in the segment descriptors of the RIB COPY, the user would determine that the COPY was correct.

4. The user could then use the DUMP93X0 CASSETTE DUMP command to dump the RIB COPY to cassette, then use the DUMP93X0 CASSETTE LOAD command to load the record to the RIB MASTER.

5. The user would run REPAIR again. This time REPAIR would find that the RIB MASTER for the file had one simple error, namely, that the LRN was incorrect. REPAIR could correct this error.

6. The original error is thus corrected.

36.5.7.2 A Complex Case

The worst case of RIB damage could be corrected in the following manner:

1. REPAIR would find a file with simple and complex errors in both the RIB MASTER and COPY.

2. The user would use either disk dump program to look at the RIBs and determine that the RIBs had somehow been completely destroyed, but that the file following the RIBs was not damaged. (This can happen when a program incautiously uses DOS logical file 0.)

3. Using either disk dump program the user would locate and map all of the file's SEGMENTS on disk.

4. From the information about the file's segments, the user would re-construct the file's RIBs, and write a program to write the RIBs to disk.

5. As a check on the above, REPAIR would be run to insure that the new RIBs for the file did not indicate an allocation conflict with another file.

6. The error is thus corrected.

36.5.8 End of RIB check

See Minimal Interface for illustration.

When all of the RIBs for all of the files on the disk have been checked REPAIR will count the number of files with uncorrected RIB format errors and the number of files with no RIB format errors and will display the counts on the screen. The files that do have RIB format errors will not be allocated space on disk and will not be processed in the cluster allocation phase (below).

36.5.9 Cluster allocation phase, Pass 1

See Minimal Interface for illustration.

The cluster allocation phase of REPAIR re-constructs in memory the CAT from the information in the RIBs of files that have no RIB format errors. The cluster allocation phase consists of three passes. The first pass makes one pass through all the files on the pack with no RIB format errors and builds in memory two CATs: one for all files that have no space (cluster allocation) conflicts with other files and a second for files which do have cluster allocation conflicts.

36.5.10 Cluster allocation phase, Pass 2

See Minimal Interface for illustration.

The second pass of the cluster allocation phase makes another pass through all the files allocated to the first CAT and finds any that may have conflicts with space allocated to the second CAT, and removes those files' space allocation from the first CAT and allocates their space to the second CAT.

36.5.11 Cluster allocation phase, Pass 3

See Minimal Interface for illustration.

The third pass of the cluster allocation phase does an analysis on any two files with disk space (cluster allocation) conflicts with each other and displays the results of the analysis on the screen (see next section). If no files have cluster allocation conflicts REPAIR proceed to System Table Replacement.

36.5.12 Cluster allocation conflicts

```
PFN 200                                    PFN 220
 0 0 3 0                          3         0 0 3 0                          3
 0 0 6 0                          7         0 0 6 0                          7
 3 0 4 1 S I N            C M D 7           3 0 4 1 S O U T          C M D 7
# OF CLUSTERS IN FILE: 00001               # OF CLUSTERS IN FILE: 00002
# OF CONFLICTING FILES: 002                CONFLICTING FILE # 001
# OF CONFLICTING CLUSTERS: 00001           # OF CONFLICTING CLUSTERS: 00001
# OF CORRECT PFN/LRN: 00004 OF 00006       # OF CORRECT PFN/LRN: 00000 OF 00006

ENTER: DELETE FILE: 1=LEFT, 2=RIGHT, 3=BOTH; 4=NO CHANGE: #
```

The screen appears as above (in general, specific words will vary) if REPAIR finds two files with cluster allocation conflicts - that is, if two files have, according to their respective RIBs, been allocated in whole or in part the same clusters on disk.

The possible combinations of file cluster allocation conflicts is myriad. One file may have conflicts with only one other file. One file may have conflicts with many other files. Many files may have conflicts with many files in different combinations of numbers.

REPAIR handles any possible combination of files with cluster allocation conflicts by dealing with only two files at a time. As in the above example, the directory MASTER entry (and some additional information) for a file is displayed on the left of the screen, and the directory MASTER entry (and some additional information) for a file that has cluster allocation conflicts with it is displayed on the right of the screen.

As long as the file on the left of the screen is not deleted, all of the files that have cluster allocation conflicts with it will be displayed in turn on the right of the screen. When all of the cluster allocation conflicts with the file on the left of the screen have been dealt with, then the next file with cluster allocation conflicts will be displayed on the left of the screen, and all files that have cluster allocation conflicts with it will be displayed in turn on the right of the screen, and so on until all files that have cluster allocation conflicts have been dealt with.

The information displayed for the two files having cluster

allocation conflicts is to guide the operator in deciding among the four options given by REPAIR. For explanation of the messages see the next section.

REPAIR corrects a cluster allocation conflict by deleting one of the files involved. If many files are involved in cluster allocation conflicts then the operator will probably want to enter "4" after each display so that he can accumulate the information necessary to decide which files should be deleted and which should be retained (that is, REPAIR will be executed twice, once to gather all the information about cluster allocation conflicts and once to actually delete files).

Specifically, the operator has four options:

1. Enter "1" to delete the file indicated on the left of the screen;

2. Enter "2" to delete the file indicated on the right of the screen;

3. Enter "3" to delete both of the files;

4. Enter "4" to take no action on either of the files and resume the CLUSTER ALLOCATION PHASE, PASS 3.

36.5.12.1 Cluster allocation phase, Pass 3 Messages

The explanations below describe the information given in the respective messages and how the operator can interpret the information.

36.5.12.1.1 Left side of screen

PFN 000

This message gives the PHYSICAL FILE NUMBER of the file whose directory MASTER entry is displayed immediately below it.

The PFN is a means of identifying the file besides the NAME/EXTENSION given in the directory entry. Additionally, the PFN of a file tells the file's relative location in the directory (refer to the System Structure Chapter for a discussion of the directory). This information can be useful, especially with a relatively new disk pack, in indicating which files are older and

which are newer.

DIRECTORY MASTER entry

The directory entry for a file provides the fundamental means of identifying the file on the disk. The directory entry contains information as follows:

The physical disk address of the beginning of the file is given in the first byte and the higher two digits of the second byte. The first byte is the cylinder address and the top three bits of the second byte are the cluster number. Since the RIBs are the first two records in the file, this address points to the file's RIBs.

The protection of the file is given in the bottom digit of the second byte. 1 or 3 = write protection, 2 = delete protection.

The old logical record number limit field is given in the third (LSB) and fourth (MSB) byte of the file, as a 16-bit binary number. This field is currently unused by the "dot-series" DOS, which normally set it to zero when a file is created.

The NAME/EXTENSION of the file is given in the 5th through 12th bytes and the 13th through 15th bytes respectively.

The last byte of the directory entry is the number of the DOS subdirectory on that logical drive containing the file.

OF CLUSTERS IN FILE: 00000

This message gives the number of clusters in the file as a 5-digit octal number.

Besides giving the operator an indication of the size of the file, it can be compared to the number of clusters in the file involved in cluster allocation conflicts (below), to give a relative indication of what percent of the file may be in error.

OF CONFLICTING FILES: 000

This message gives the number of files (in octal) that conflict with the file displayed on the left of the screen.

If the number is very large, and the file not very important to the operator, then the operator may decide to delete the file rather than look at all of the files that have cluster allocation conflicts with it.

OF CONFLICTING CLUSTERS: 00000

This message gives the number (in octal) of clusters that are in conflict for the entire file. If the file has conflicts with many files then this number will almost always be larger than the corresponding number on the right side of the display.

The number of conflicting clusters for a file can give the operator a quantitative indication of possible damage to the file.

OF CORRECT PFN/LRN: 00000 OF 00000

This message gives the number of records in the file that have the correct DOS header information in them (being the PFN in the first byte of the physical record and the LRN in the second and third bytes of the record) for the clusters that are in conflict with other files, and the number of records in the clusters that are in conflict. Both of the numbers are in octal.

If a record in a contested cluster does not have the correct PFN/LRN information, then it has probably been overwritten by a record of a file that also claims the cluster.

This message gives the operator an indication of actual damage to the file. If the number of correct PFN/LRN is high, then there is little damage to the file and the RIB for the file is probably correct. If the number of correct PFN/LRN is very low, then the file has probably been overwritten by another file and/or the file's RIB is incorrect.

36.5.12.1.2 Right side of screen

PFN 000

Same as for left side of screen.

DIRECTORY MASTER entry

Same as for left side of screen.

OF CLUSTERS IN FILE: 00000

Same as for left side of screen.

CONFLICTING FILE # 000

This message provides a counter (in octal) to help the operator keep track of each file among several with which the file on the left of the screen has cluster allocation conflicts. This number can never exceed the # OF CONFLICTING FILES: 000 count.

OF CONFLICTING CLUSTERS: 00000

This message gives the number (in octal) of clusters that are in conflict between the files indicated on the left and right of the screen.

OF CORRECT PFN/LRN: 00000 OF 00000

This message gives the number of records in the file that have the correct DOS header information in them (PFN and LRN) for the clusters that are in conflict with the file indicated on the left of the screen, and the number of records in the clusters that are in conflict. Both of these numbers are in octal.

Refer to the discussion under this message for the left side of the screen. The user will need to be aware of the structure of the files being examined.

The user may wish to use either disk dump program to inspect the actual data on disk before deleting one or both of two files with cluster allocation conflicts.

For a file with cluster allocation conflicts, one of five things may be true:

1. The file may have correct RIBs and all correct records. (That is, the error is in the file(s) having the cluster allocation conflict with this file.)

2. The file may have incorrect RIBs.

3. The file's space has been erroneously allocated to another file, and is occupied by the other file.

4. Another file has erroneously been allocated the file's space, and its space is occupied by this file.

5. Any combinaton of (2), (3) and (4) above.

Either disk dump program can be used to look at the RIBs of files with cluster allocation conflicts. From the information given by the segment descriptors either disk dump program can be used to look at where the file's records should be on disk. If

the records for the file are where they should be according to the RIB, then the file possibly has no errors.

NOTE:

> For files such as Indexed DATASHARE files (physically random access as opposed to ISAM access files), all of the space allocated to a file will not necessarily be used.

The user will need to be aware of the structure of the files being examined.

From the information gathered by examination of the actual data on disk, the user can determine whether a file has errors and if so, whether corrections should be made, and if so, what corrections. In some cases the user may want to change a file's RIB to relocate the file on disk. This, of course, would require careful study of the real allocation of space on the disk and regeneration of the file's RIBs.

36.5.13 System table replacement

See Minimal Interface for illustration.

The CLUSTER ALLOCATION TABLE that will be written to disk is a combination of the CAT for files that had no cluster allocation conflicts and the CAT for files that had cluster allocation conflicts but that the operator did not wish to delete. The allocation for files with cluster allocation conflicts is retained so that if a new file is created it will not take space that is being used by one of the un-deleted but erroneous files, thus compounding cluster allocation conflicts.

Files that will still exist in the directory but will not have space allocated to them will be:

Files with an invalid RIB address in their directory MASTER entry;
Files with any uncorrected error in either RIB

The reason disk space is not allocated to these files is that if REPAIR cannot find the RIB for a file or if the RIB has uncorrected errors then REPAIR has no way of knowing where the file's clusters should be located. Any files of this class are best transferred to cassette (if possible) and KILLed before any new data is loaded to disk.

36.5.14 Termination of REPAIR

See Minimal Interface for illustration.

When the REPAIR program has finished execution and has been run from DOS it will return to DOS. Otherwise it goes into a dead loop; that is, it executes a JUMP to self. This is so that the processor will be "locked up" by the REPAIR program until the operator takes some specific action, such as putting a LGO program or DOS boot cassette in the rear deck and depressing RESTART. If the auto-restart tab were punched out of the cassette in the rear deck and REPAIR executed a HALT instruction upon completion, then the computer would attempt to load and execute the cassette in the rear deck, which the operator may not wish to happen.

36.6 Cylinder Lockout with REPAIR

This section describes the mechanics of locking out cylinders. To accomplish this with the REPAIR program does not require an understanding of the cylinder concept.

Any cylinders that are reserved (locked out) on a disk should be recorded on a sticker or label on the case of the pack. In addition, the cylinders to be locked out are recorded internally on the disk itself in the Lockout CAT and its backup. The list of cylinders to be locked out might look something like:

```
FLAGGED CYLINDERS (or TRACKS):
        40-50
        167
        200-202
```

Obviously, the cylinder numbers locked out cannot exceed the maximum cylinder number allowed on the DOS in use. The following example assumes a cartridge disk system; operation would be identical for any other system. Remember that the numbers used for cylinder lockout are decimal, rather than octal as used in most other portions of REPAIR. The following example shows how a list of cylinders as above would be locked out in the REPAIR program.

```
                    DATAPOINT DOS.  REPAIR



DRIVE NUMBER: 0
LOCKOUT CAT: FORMAT LOOKS OK.
DO YOU WANT TO LOCK OUT ADDITIONAL CYLINDERS ? #
```

The screen appears as above when REPAIR is ready to accept cylinder lockout instructions. The instructions serve as additional cylinders to be locked out. REPAIR will not normally allow cylinders which have been previously locked out to be "unlocked". When REPAIR finishes execution and optionally rewrites the Lockout CATs, the cylinders locked out will be those originally locked out plus those specified by the operator in the following steps.

To lock out cylinders, the operator must enter "Y".

```
                    DATAPOINT DOS.  REPAIR



DRIVE NUMBER: 0
LOCKOUT CAT: FORMAT LOOKS OK.
*** ARE YOU SURE ? *** #
```

REPAIR will make sure the operator wants to lock out cylinders before accepting cylinder numbers to be locked out.

To lock out cylinders, the operator must enter "Y".

```
                    DATAPOINT DOS.  REPAIR



DRIVE NUMBER: 0
LOCKOUT CAT: FORMAT LOOKS OK.
*** ARE YOU SURE ? *** Y
CYLINDER NUMBER<S> <1-202>: #
```

The screen will appear as above when REPAIR is ready to accept the first cylinder(s) to be locked out.

If the operator were locking out the cylinders listed above, he would enter 40-50 and press ENTER.

```
                    DATAPOINT DOS.  REPAIR



DRIVE NUMBER: 0
LOCKOUT CAT: FORMAT LOOKS OK.
*** ARE YOU SURE ? *** Y
CYLINDER NUMBER<S> <1-202>: 40-50
CYLINDER NUMBER<S> <1-202>: #
```

The screen appears as above when REPAIR has accepted the previous cylinder lock-out and is ready for the next cylinder number(s).

According to the above sample list, the operator would now enter 167.

```
DRIVE NUMBER: 0
LOCKOUT CAT: FORMAT LOOKS OK.
*** ARE YOU SURE ? *** Y
CYLINDER NUMBER<S> <1-202>: 40-50
CYLINDER NUMBER<S> <1-202>: 167
CYLINDER NUMBER<S> <1-202>: #
```

The screen appears as above when REPAIR has accepted the previous lock-out and is ready to accept the next cylinder number(s).

According to the above list the operator must now enter 200-202.

```
DRIVE NUMBER: 0
LOCKOUT CAT: FORMAT LOOKS OK.
*** ARE YOU SURE ? *** Y
CYLINDER NUMBER<S> <1-202>: 40-50
CYLINDER NUMBER<S> <1-202>: 167
CYLINDER NUMBER<S> <1-202>: 200-202
CYLINDER NUMBER<S> <1-202>: #
```

The screen appears as above when REPAIR has accepted the previous lock-out and is ready to accept the next cylinder number(s).

According to the above example the operator has no more cylinders to lock out. At this point then, the operator would merely depress the ENTER key to signal REPAIR that no more cylinders are to be locked out. REPAIR would proceed immediately to the cluster allocation table and directory check phase.

36.7 CAT errors and directory read/write errors

This section describes messages displayed by the REPAIR program when it discovers an error (of any kind) in the CLUSTER ALLOCATION TABLEs (CATs) or a read or write error in the directory.

These errors are the first type of error checked for by REPAIR. A format (logic) error in one or more of the CATs is not fatal (will not cause REPAIR to abort), but will be noted to the operator. An uncorrectable read or write error in any of the CATs or the directory is fatal, because the disk pack is in very

serious trouble if hardware errors occur in any of these tables.

REPAIR does not consider a read error in the CAT or DIRECTORY fatal until either an attempt to clear the error by writing back to disk has failed or the operator has instructed REPAIR not to attempt the write. A write error to the CAT or DIRECTORY is always fatal.

There is a working (MASTER) and a backup (COPY) version of the Working CAT, the Lockout CAT, and the directory.

The examples that follow are given in the sequence of their potential occurrence in REPAIR execution.

Important notice: Similar sequences are used for errors in the Lockout CAT and its backup as for the Working CAT and its backup. In this chapter, only the Working CAT sequence is used as an example, since both are directly comparable. The messages for both sequences are largely identical to save space. The user can tell at any time whether the messages refer to the Working or Lockout CATs by looking for the header "LOCKOUT CAT:" or "WORKING CAT:" more or less directly preceding the message.

36.7.1 Cluster allocation table read errors

Note that although this example concerns the CAT MASTER, the same messages (substituting the word COPY for MASTER) apply to the CAT COPY.

```
                    DATAPOINT DOS.  REPAIR




DRIVE NUMBER: 0
LOCKOUT CAT: FORMAT LOOKS OK.
DO YOU WANT TO LOCK OUT ADDITIONAL CYLINDERS ? N
WORKING CAT:
C.A.T. MASTER READ ERROR
WRITE TO DISK TO ATTEMPT TO CLEAR ERROR ? #
```

The messages on the last two lines of the screen above will appear when the REPAIR program has detected a read error in the CAT MASTER. Notice how in this case, the header "WORKING CAT:" implies that the read error has occurred in the Working CAT MASTER as opposed to the Lockout CAT MASTER.

To have REPAIR attempt to clear the read error enter Y; otherwise enter N.

```
DRIVE NUMBER: 0
LOCKOUT CAT: FORMAT LOOKS OK.
DO YOU WANT TO LOCK OUT ADDITIONAL CYLINDERS ? N
WORKING CAT:
C.A.T. MASTER READ ERROR
WRITE TO DISK TO ATTEMPT TO CLEAR ERROR ? Y
READ ERROR CLEARED.
```

The message on the last line of the screen above will appear when the operator has replied "Y" to the message above and the attempt to clear the read error was successful.

No further operator response is required.

```
DRIVE NUMBER: 0
LOCKOUT CAT: FORMAT LOOKS OK.
DO YOU WANT TO LOCK OUT ADDITONAL CYLINDERS ? N
WORKING CAT:
C.A.T. MASTER READ ERROR
WRITE TO DISK TO ATTEMPT TO CLEAR ERROR ? N
READ ERROR UNCORRECTABLE.
THE PACK IS NOT FIXABLE.
```

The messages on the last two lines of the screen above will appear if the operator replies "N" to the message above or if the write to disk did not clear the read error. The REPAIR program will not accept any further commands. To get any other program running on the computer the operator must press the RESTART key.

No operator response is required.

```
                    DATAPOINT DOS.  REPAIR




DRIVE NUMBER: 0
LOCKOUT CAT: FORMAT LOOKS OK.
DO YOU WANT TO LOCK OUT ADDITIONAL CYLINDERS ? N
WORKING CAT: FORMAT LOOKS OK.
THE C.A.T. MASTER HAS DEVELOPED A READ ERROR
THE PACK IS NOT FIXABLE.
```

The messages on the last two lines of the screen above will appear if a read error occurs when REPAIR reads the CAT MASTER for the second time during the CAT check. This read error is automatically considered fatal because it is evidence of a transient hardware error in the CAT.

No operator response is required.

36.7.2 Cluster Allocation Table is destroyed

Note that although this example concerns the CAT MASTER, the same messages (transposing the words COPY and MASTER) apply to the CAT COPY.

```
                    DATAPOINT DOS.  REPAIR




DRIVE NUMBER: 0
LOCKOUT CAT: FORMAT LOOKS OK.
DO YOU WANT TO LOCK OUT ADDITIONAL CYLINDERS ? N
WORKING CAT:
THE C.A.T. MASTER IS DESTROYED
WRITE C.A.T. COPY INTO C.A.T. MASTER ? #
```

The messages on the last two lines of the screen above will appear when the REPAIR program has discovered that the CAT MASTER is destroyed but the CAT COPY appears to be valid.

To have REPAIR copy the CAT COPY into the CAT MASTER, enter "Y". Otherwise, enter "N".

```
DRIVE NUMBER: 0
LOCKOUT CAT: FORMAT LOOKS OK.
DO YOU WANT TO LOCK OUT ADDITIONAL CYLINDERS ? N
WORKING CAT:
THE C.A.T. MASTER IS DESTROYED
WRITE C.A.T. COPY INTO C.A.T. MASTER ? Y
DONE.
```

The message on the last line of the screen above will appear when the operator has replied "Y" to the message above and the write to the CAT MASTER was successful. REPAIR will proceed to check the directory

No operator response required.

```
DRIVE NUMBER: 0
LOCKOUT CAT: FORMAT LOOKS OK.
DO YOU WANT TO LOCK OUT ADDITIONAL CYLINDERS ? N
WORKING CAT:
THE C.A.T. MASTER IS DESTROYED
WRITE C.A.T. COPY INTO C.A.T. MASTER ? Y
DISK WRITE ERROR FOR C.A.T. MASTER.
THE PACK IS NOT FIXABLE.
```

The messages on the last two lines of the screen above will appear if a write error occurs when REPAIR tries to write to the CAT MASTER. The REPAIR program will not accept any further commands. To get any other program running on the computer the operator must press the RESTART key.

No operator response is required.

```
DRIVE NUMBER: 0
LOCKOUT CAT: FORMAT LOOKS OK.
DO YOU WANT TO LOCK OUT ADDITIONAL CYLINDERS ? N
WORKING CAT:
THE C.A.T. MASTER IS DESTROYED
THE C.A.T. MASTER & COPY ARE DESTROYED
THE C.A.T. MASTER & COPY WILL HAVE TO BE RECONSTRUCTED FROM THE R.I.B.'S
```

The messages on the last three lines of the screen above will appear if REPAIR discovers that both copies of the CAT are destroyed. After the messages are displayed REPAIR will proceed to check the directory At the conlusion of REPAIR simply write the new CAT to disk to correct the error.

No operator response is required.

36.7.3 Cluster Allocation Table Copies Do Not Match

```
                    DATAPOINT DOS.  REPAIR



DRIVE NUMBER: 0
LOCKOUT CAT: FORMAT LOOKS OK.
DO YOU WANT TO LOCK OUT ADDITIONAL CYLINDERS ? N
WORKING CAT:
C.A.T. MASTER & COPY DO NOT MATCH
THE C.A.T. MASTER & COPY WILL HAVE TO BE RECONSTRUCTED FROM THE R.I.B.'S
```

The messages on the last two lines of the screen will appear when REPAIR has discovered that the CAT MASTER and COPY versions do not agree with each other. Since it is not possible for REPAIR to choose which version is correct at this point, it will proceed to check the DIRECTORY. At the conclusion of REPAIR, simply write the new CAT to disk to correct the error.

No operator response is required.

36.7.4 Directory Read Errors

Note that although this example concerns the directory MASTER, the same messages (transposing the words COPY and MASTER) apply to the directory COPY.

```
                    DATAPOINT DOS.  REPAIR




 DRIVE NUMBER: 0
 LOCKOUT CAT: FORMAT LOOKS OK.
 DO YOU WANT TO LOCK OUT ADDITIONAL CYLINDERS ? N
 WORKING CAT: FORMAT LOOKS OK.
 DIRECTORY PAGE MASTER READ ERROR
 WRITE TO DISK TO ATTEMPT TO CLEAR ERROR ? #
```

The messages on the last two lines of the screen above will appear when REPAIR has detected a read error in the directory MASTER.

To have REPAIR attempt to clear the read error enter "Y", otherwise enter "N".

```




 DRIVE NUMBER: 0
 LOCKOUT CAT: FORMAT LOOKS OK.
 DO YOU WANT TO LOCK OUT ADDITIONAL CYLINDERS ? N
 WORKING CAT: FORMAT LOOKS OK.
 DIRECTORY PAGE MASTER READ ERROR
 WRITE TO DISK TO ATTEMPT TO CLEAR ERROR ? Y
 WRITE COPY PAGE TO MASTER PAGE ? #
```

The message on the last line of the screen above will appear if the operator has replied "Y" to the message above.

To have REPAIR copy the directory COPY page to the directory MASTER page enter "Y", otherwise enter "N". If "N" is entered the directory check will continue.

```
DRIVE NUMBER: 0
LOCKOUT CAT: FORMAT LOOKS OK.
DO YOU WANT TO LOCK OUT ADDITIONAL CYLINDERS ? N
WORKING CAT: FORMAT LOOKS OK.
DIRECTORY PAGE MASTER READ ERROR
WRITE TO DISK TO ATTEMPT TO CLEAR ERROR ? Y
WRITE COPY PAGE TO MASTER PAGE ? Y
DONE.
```

The message on the last line of the screen above will appear when the write to the directory MASTER has been successful. The directory check will continue.

No further operator response is required.

```
DRIVE NUMBER: 0
LOCKOUT CAT: FORMAT LOOKS OK.
DO YOU WANT TO LOCK OUT ADDITIONAL CYLINDERS ? N
WORKING CAT: FORMAT LOOKS OK.
DIRECTORY PAGE MASTER READ ERROR
WRITE TO DISK TO ATTEMPT TO CLEAR ERROR ? Y
WRITE COPY PAGE TO MASTER PAGE ? Y
DIRECTORY PAGE MASTER WRITE ERROR
THE PACK IS NOT FIXABLE.
```

The messages on the last two lines of the screen above will appear if the operator replied "Y" to the message above and REPAIR detected a write error when it attempted to write to the directory MASTER. The REPAIR program will not accept any further commands. To get another program running on the computer the operator must press the RESTART key.

No operator response is required.

```
DRIVE NUMBER: 0
LOCKOUT CAT: FORMAT LOOKS OK.
DO YOU WANT TO LOCK OUT ADDITIONAL CYLINDERS ? N
WORKING CAT: FORMAT LOOKS OK.
DIRECTORY PAGE MASTER READ ERROR
WRITE TO DISK TO ATTEMPT TO CLEAR ERROR ? Y
WRITE COPY PAGE TO MASTER PAGE ? Y
DIRECTORY PAGE MASTER READ ERROR
THE PACK IS NOT FIXABLE.
```

The messages of the last two lines of the screen above will appear if the operator replied "Y" to the message above and REPAIR detected a read error when it attempted to re-read the directory MASTER page it had just written.

No operator response is required.

```
DRIVE NUMBER: 0
LOCKOUT CAT: FORMAT LOOKS OK.
DO YOU WANT TO LOCK OUT ADDITIONAL CYLINDERS ? N
WORKING CAT: FORMAT LOOKS OK.
DIRECTORY PAGE MASTER READ ERROR
WRITE TO DISK TO ATTEMPT TO CLEAR ERROR ? Y
WRITE COPY PAGE TO MASTER PAGE ? Y
DIRECTORY COPY PAGE HAS DEVELOPED A READ ERROR
THE PACK IS NOT FIXABLE.
```

The messages on the last two lines of the screen above will appear if the operator replied "Y" to the message above and REPAIR detected a read error when it attempted to re-read the directory COPY to compare it against the directory MASTER page just written and re-read.

No operator response is required.

```
DRIVE NUMBER: 0
LOCKOUT CAT: FORMAT LOOKS OK.
DO YOU WANT TO LOCK OUT ADDITIONAL CYLINDERS ? N
WORKING CAT: FORMAT LOOKS OK.
DIRECTORY PAGE MASTER READ ERROR
WRITE TO DISK TO ATTEMPT TO CLEAR ERROR ? Y
WRITE COPY PAGE TO MASTER PAGE ? Y
DIRECTORY PAGE MASTER & COPY DO NOT MATCH
THE PACK IS NOT FIXABLE.
```

The messages on the last two lines of the screen above will appear if the operator replied "Y" to the message above but the directory page MASTER and COPY did not match after the page copy had been made. This error is automatically considered fatal because it is evidence of a hardware error in the directory.

No operator response is required.

```
DRIVE NUMBER: 0
LOCKOUT CAT: FORMAT LOOKS OK.
DO YOU WANT TO LOCK OUT ADDITIONAL CYLINDERS ? N
WORKING CAT: FORMAT LOOKS OK.
DIRECTORY PAGE MASTER READ ERROR
WRITE TO DISK TO ATTEMPT TO CLEAR ERROR ? Y
DIRECTORY PAGE COPY ALSO GIVES READ ERROR
WRITE TO DISK TO ATTEMPT TO CLEAR ERROR ? #
```

The messages on the last two lines of the screen above will appear if the operator replies "Y" to the message above and REPAIR detects a read error in the directory COPY page.

To have REPAIR attempt to clear the read error enter "Y", otherwise enter "N". If the write is successful REPAIR will continue with the directory check.

```
DRIVE NUMBER: 0
LOCKOUT CAT: FORMAT LOOKS OK.
DO YOU WANT TO LOCK OUT ADDITIONAL CYLINDERS ? N
WORKING CAT: FORMAT LOOKS OK.
DIRECTORY PAGE MASTER READ ERROR
WRITE TO DISK TO ATTEMPT TO CLEAR ERROR ? Y
DIRECTORY PAGE COPY READ ERROR
WRITE TO DISK TO ATTEMPT TO CLEAR ERROR ? Y
READ ERROR UNCORRECTABLE.
THE PACK IS NOT FIXABLE.
```

The messages on the last two lines of the screen above will appear if the operator replied "N" to the message above or if the write to disk did not clear the read error.

No operator response is required.

CHAPTER 37. REWIND COMMAND

REWIND - Rewind the cassette tape.

REWIND [REAR or DECK1]

The cassette in the front deck is rewound unless "REAR" or "DECK1" is specified. If no cassette is in place in the deck, the rewind will proceed but only after a cassette is put into place. The cassette can be fully wound onto the clear leader at the very end of the tape, since the rewind command starts by slewing the tape backwards for a few seconds first. This both takes up any slack that may be present in the cassette before the high-speed rewind starts, and also ensures that the tape is not on the clear leader when the actual rewind begins.

CHAPTER 38. SAPP COMMAND

SAPP - Append two source files creating a third

SAPP <file spec>,[<file spec>],<file spec>

The SAPP command appends the second source file after the first and puts the result into the third file. If extensions are not supplied, TXT is assumed. The first two files must exist. If the third file does not already exist, a new file will be created. The first file's end of file record is discarded and the copy is terminated by the end of file mark in the second file.

Omitting the second file specification causes the first file to be copied into the third file. Note that neither the first or second file is changed.

The first and third file specifications are required. If either is omitted the message

NAME REQUIRED

will be displayed.

The second and third file specifications must not be the same.

CHAPTER 39. SORT COMMAND

39.1 Introduction

The Disk Operating System SORT enables any Datapoint Disk user to initiate file sorts directly from the keyboard.

Using a multi-train radix sort technique, the Datapoint processor achieves speeds comparable with much larger systems. The list of options also compares favorably with much more extensive systems. Nevertheless, since it uses the full dynamic nature of the Disk Operating System, it is extremely easy to operate. (Users who have spent several hours figuring out how to set up the myriad of SORT work datasets required, even for the simplest sorts, by other sort packages know what we're talking about.)

For more sophisticated uses, SORT may be called from other programs through CHAIN. Using CHAIN also enables complicated sort options to be reduced to a single file name then callable either from the keyboard or another program. CHAIN also extends the SORT package to operate as a merge, as well.

39.2 General Information

SORT attempts to optimize its speed by placing its work files on a drive separate from the input or output files. Unless otherwise directed, SORT opens its work files on the highest-numered disk on-line, excluding the disk containing the input file. Both SORT work files are always placed on the same drive. If SORT selects a drive with insufficient space, it will abort. It may than be necessary to drive-direct its work files.

39.3 Fundamental SORT Concepts

39.3.1 File Formats

All Datapoint systems use a universal text file structure recognized by Databus, Datashare, RPG II, Basic, Scribe, Editor, Assembler, Terminal emulators, etc. Therefore, any text file generated by or for any of the above, may be sorted. The file to be sorted must be on disk, however.

There are two sub-formats a Datapoint file can have: Blocked or Sequential. Blocked files are required to have a single 'string' or 'record' of data per physical disk record. The maximum record size for blocked records is 249 bytes (plus end-of-record and end-of-sector control bytes for a total of 251 bytes). Sequential records have no fixed relationship to physical disk records and are written as densely as possible in the given file space. Nonetheless, blocked files can be read sequentially in the identical way that sequential files are read. In fact, both types of files, when read sequentially, are indistinguishable. Blocked files are used for achieving random access to records. They generally require more disk space than sequential files for the same amount of data.

Space compression implies that the logical position and the physical position of a character in a record may differ. SORT will always expand the spaces to determine the logical position of a character.

When sorting, consider that the result of the sort is not a restructuring of the original file. It is a NEW file which is a restructured COPY of the original file. The original file is never changed.

Therefore, SORT produces a file which is a sorted version of the original. This gives the user the added opportunity of specifying the type of file to be output regardless of the input file format (with one restriction - see the section on Input/Output File Format Options).

39.3.2 The Key Options

The KEY of a sort is the FIELD or that part of the record which is to ORDER the sequence of records. For instance, it can be a person's name, state, employee number, amount in debt or any aspect of the data base identifiable by a fixed position in the record, based upon the column count from the beginning of the record.

Consider the following record (column count scale below for reference only):

```
Mule, Francis A.        242219 123 BARN    SAN ANTONIO     TX
12345678901234567890123456789012345678901234567890123456789012345678901234567890
```

The name begins in column 1 and goes to 22. The employee number spans columns 24-29. The street address is 31-42. The city is 43-58. The State is 59-60

If each person had a record in the file exactly in the above format, SORT could order the sequence of records in the file by any of the above fields. For instance, to get an alphabetical list of the records by name, the key would be 1 to 22 (hereafter referred to as 1-22). The key for sequencing the file in order of employee number would be 24-29. The key for ordering the records by state then city and then employee number would be 59-60,43-58,24-29.

Any portion of the record can be used as a key. Care must be taken when selecting a key to include no more characters than necessary, since each character added to the key slows down the sort.

The key specified for SORT is concatenated to a single string, then sorted character-by-character, with the left-most character being of most significance. It is very important to realize the effect of a right-to-left character sort. To appear in the "right" sequence numeric fields must be right-justified, character fields must be left-justified. If signed numeric fields are sorted, the sign should be moved to the left-most position and the magnitude right-justified; otherwise the resulting sorted sequence will contain positive and negative values in no discernible order, since the "-" and "+" signs are just another character to SORT. A full explanation of character sort concepts is beyond the scope of this manual. Interested users should consult an appropriate information science textbook.

39.3.3 How to Sort a File

Sorting a file is done from the keyboard of the DOS. All the operator must know is the name of the file to be sorted, the name desired for the sorted output file, and the columns containing the key.

For instance, the keyboard issued command for the above example to sort on the name field (1-22), would be:

```
SORT EMPLFILE,SORTFILE;1-22
```

This is assuming that the name of that file was EMPLFILE. It is also the operator's decision as to what the resultant sorted file is called, as the command could have easily been:

```
SORT EMPLFILE,EMPSORT;1-22
```

as well. The second file named is where the resultant sorted output will be placed.

More complicated keys may be stated as well and the command to sort by state and then name would be:

```
SORT EMPLFILE,SORTFILE;59-60,1-22
```

That is all there is to simplified sorting.

Testing SORT for yourself is simple. Most systems have a source code file for a Databus or Assembly language program on the disk. Such programs can be sorted by op-code and provide an interesting analysis of the usage of each instruction type:

```
SORT INFILE,OUTFILE;9-12
```

39.4 The Other Options

39.4.1 Generalized Command Statement Format

The following is the generalized statement format for the Datapoint DOS SORT:

```
SORT IN,OUT[,:DRk][,SEQ][;[[F][O][R][H][GNNNTC][N]][K1]...[,On][,Kn]]
```

Information contained within a pair of square brackets

[] is optional; information within brackets is order-dependent. Commas may be used to delimit parameters. (NOTE that commas MUST be used to delimit sort-key groups.) The first four fields (those ahead of the semi-colon) are considered to be file specification fields. The fields following the semicolon are considered to be sort key parameters. Default conditions are listed below. Typical statements obeying this format are:

```
(1)  SORT INFILE,OUTFILE
(2)  SORT INFILE,OUTFILE;1-3,7-20
(3)  SORT INFILE,OUTFILE;ID1-3
(4)  SORT INFILE,OUTFILE;IDL7-20
(5)  SORT INFILE,OUTFILE;LH11-20
(6)  SORT INFILE,OUTFILE,,SEQFILE
(7)  SORT INFILE,OUTFILE,:DR0,SEQFILE/SEQ:DR1
```

All the above statements will invoke a sort. Each will provide different results. However, notice that in (1) there are no other parameters than the file specifiers. That is because all the specifiable parameters have a default value in case there is no specification for it.

The following list defines the parameters which can be specified:

IN.....This specifies the input file. This file must exist on disk.

OUT.....This specifies the output file. This specification is optional IF AND ONLY IF the 'L' AND 'H' options are used. If an output file is specified AND no disk drive is specified AND the file exists on a drive on-line to the system then the output file will over-write the existing file. If an output file is specified AND no disk drive is specified AND no file of that name exists on a drive on-line to the system THEN a file of the given name will be created on the same drive as the input file.

:DRk.....This specifies the drive for the sort key file. This is only a working scratch file needed during the sort. SORT will attempt to pick the optimum drive on which to put the work file on a multi-drive system.

Experience or special considerations may cause the user to want to specify a work drive.

SEQ.....NON-ASCII COLLATING SEQUENCE FILE
This specifies the file which contains the collating sequence to be used. If omitted, ASCII will be assumed.

F.....FORMAT.
This parameter specifies the output file format: blocked or space compressed (standard editor output format). If the user specifies I (and the input file is also blocked), then the output file will be left blocked.

Without typing the 'I', the output file will be record compressed no matter what the input file. If and only if the input file is a blocked file, you may include the 'I' parameter and cause the output file to be blocked.

O.....ORDER.
This parameter specifies the output file collating sequence: Ascending or Descending. The actual character entered is 'A' or 'D'. The default value is 'A'.

Without typing the 'D', the collating sequence order is considered ASCENDING. Including the 'D' parameter will cause the collating sequence to operate in DESCENDING order. Note that if some keys are to be sorted in ascending order and other keys in descending order, the "On" specification described below should precede each key whose order differs from the order of the key preceeding it. However, if all keys are to be ordered in the same sequence, this parameter need only be specified once.

R.....RECORD FORMAT.
This parameter specifies a special output record format: Limited output file format or Tag file or Keytag file output. The actual character entered is 'L' or 'T' or

'K'. The default value is no special output record format; that is, neither 'L' nor 'T' nor 'K', so that the records in the output file will be exact copies (FULL IMAGE RECORDS) of the records in the input file.

Normally the sort transfers all of the records of the input file to the output file. It is possible, not only to transfer part of each record, but to select only certain records or to include constant literals in each record as well. Including the 'L' parameter in the list of parameters will cause another question to be asked wherein you may specify the limitations and constants. See the section on Limited Output Format Option.

By entering the 'T' character an output file is generated which consists only of binary record number and buffer byte pointers to the input file records. See the section on Tag File Output Format Option.

By entering the 'K' character a standard text format output file is generated which consists of records containing a 5 byte user logical record number, a 3 byte buffer address, and the key. These records are space-compressed and have trailing spaces truncated. See the section on Keytag File Output Format Option.

H.....HARDCOPY OUTPUT.
This parameter specifies that the output of the SORT will be listed on a printer. The actual character entered is 'H'. The default value is no hardcopy output.

Without typing the 'H' no printing will occur and SORT will require that an output file be named. If the 'H' parameter is given and an output file is named then SORT will list the output to a printer and will generate an output file. If the 'H' parameter is given and no output file is named then SORT will list the output to a printer and no disk file output will be

generated.

If the 'H' parameter is given then the 'L' parameter must precede the 'H' parameter.

SORT will print to a local printer or a servo printer. See the section on Hardcopy Output Option.

G.....GROUP INDICATOR

This parameter specifies that the input file consists of Primary and Secondary records and specifies which Group is to be sorted. The actual character entered is 'P' for primary or 'S' for secondary. There is no default value.

If the 'G' option is entered then the NNNTC options must also be entered.

In a file with Primary and Secondary records, a string of records with a Primary record as the first record and Secondary records following it is considered one block, or group, of records.

When the file is sorted on Primary records the output file has the blocks of records re-ordered so that the Primary records are in the sorted sequence; no change is made in the sequence of the Secondary records following each Primary record. When the file is sorted on Secondary records and the first key specified is in ascending sequence, the output file has the blocks of records in the same order as in the input file, but the Secondary records within each block are in the sorted sequences.

When the file is sorted on Secondary records and the first key specified is in descending sequence, the output file has the blocks of records in reversed order as the input file, but the Secondary records within each block are in the sorted sequence.

SORT has no provision for the sorting of Primary and Secondary records in the same

SORT run.

NNN.....NUMERIC position of Primary/Secondary flag.
This parameter specifies the character position for the character (the 'C' parameter) indicating whether the record is a Primary or Secondary record . The number must be specified if the option is taken and must fall in the range 1 to 249.

T.....TYPE of evaluation.
This parameter specifies equivalence or inequivalence of the group indicator character; that is, whether the character in the record will be equal to or not equal to the character specified. The actual character entered is '=' for equal or '#' for not equal. There is no default character, '=' or '#' must be given if the option is taken.

If '=' is given then if the character in the NNNth position of an input file record is EQUAL to the group indicator character -- indicated by 'C' below -- then the record is a member of the specified sort group -- indicated by 'G' above. Otherwise, it is not a member of the specified group.

C.....CHARACTER, group indicator
This parameter specifies the actual test character for determination of a record's membership in the sort group. The actual character entered is any member of the available character set -- this means any combination of eight bits -- except 015. There is no default character: the character immediately following the 'T' parameter is taken to be the 'C' parameter -- except a 015.

N.....This parameter specifies no space compression on output. This applies to Full Image and Limited Output files. It does not apply for blocked or Tag files. If the input file is space-compressed, the 'N' parameter will cause the output file to be non-compressed. If the input file is not

space-compressed, the output file will not be compressed, regardless of the N parameter.

K1.....SSS-EEE
This is the first sort key specification. If no key is specified, the SORT will assume 1-10,i.e. the first ten characters of the record.
SSS is the starting key position.
EEE is the ending key position. The key is limited to 118 characters and must be contained within the first 249 characters of the record.

On.....This specifies the order for the nth key (ascending and descending are indicated by 'A' or 'D'). If omitted the order used on the previous key is assumed.

Kn.....SSS-EEE
The nth sort key specification. The maximum number of keys is that which can be typed without exceeding the input line.

39.4.2 Keys-overlapping and in Backwards Order

The key specification need not be only forward. A specification of 17-12 will cause the 6 delimited characters to be a key but in the order of 17,16,15,14,13,12. This is extremely valuable, clearly, in data which has the most significant digit or character last.

Key specifications may also be overlapping: 1-20,30-15 overlaps 15 to 20. When this occurs, the system will optimize the sort and save time over re-sorting on those columns again.

39.4.3 Collating Sequence File

By specifying a sequence file, the user may substitute any collating sequence for the standard ASCII character set. The sequence file may have any name, but the extension must /SEQ (SEQ is the default extension). If the disk drive number on which the file resides is omitted, SORT defaults to the same drive from which the SORT itself was loaded. This table may be supplied by the user but must meet certain requirements to be loaded:

1. It must be an absolute object file.
2. It must begin loading at location 027400.
3. The first eleven bytes must contain the file name and the extension must be SEQ. (Full 8 - character file name with trailing blanks, then extension.)
4. The table itself must begin loading at location 027400 and occupy 256 bytes (overstoring the file name described in 3). For instance, the source for the EBCDIC sequence file begins:

```
SET     027400
DC      'EBCDIC  SEQ'
SET     027400
DC      0,1,2,3,4,5,6,7,
.
.
.
```

5. If the file is not found on the specified disk drive the following message is displayed:

   SEQUENCE FILE NOT FOUND

6. If the file is found but is not an absolute object file the following message is displayed:

   SEQUENCE FILE FORMAT ERROR A

7. If the file format appears valid, the file will be loaded using DOS routine LOADX$. LOADX$ will return an error code if the load is unsuccessful. The following display will notify the user of the error:

   SEQUENCE FILE FORMAT ERROR n

   where n=0 if file does not exist
   1 if disk drive is off-line
   2 if directory parity fault
   3 if RIB parity fault
   4 if file parity fault
   5 if off end of physical file
   6 if record of illegal format

39.4.4 Ascending and Descending sequences

Changing the collating sequence from ascending to descending is the same as 'reversing' the file, or placing the last first, etc. Sorting a telephone directory in ascending sequence on name produces the familiar order. Should it be sorted in descending sequence, then Mr. Zyk would be first and Mr. Aardvark would be last. The order of collation, when alphabetic, numeric, and punctuation characters all can occur in a column together, follows the character set order. The sequence may be specified for each sort key. However, it need not be specified if it is the same as the key which preceeds it. Therefore, it is possible to sort portions of the key in ascending order and portions in descending order.

39.4.5 Input/output File Format Options

SORT accesses each file sequentially. Due to the techniques used in the Datapoint standard file structure, the sequential reading technique will provide SORT with all of the records in the file whether the file was originally blocked or sequential. Therefore, the file format options only allow specification of the output file's format.

If the input file is blocked, that is one logical record or string per physical disk record, then you have a choice of output formats (F option). If 'I' is chosen, that is blocked, then each output disk record will contain an exact copy of the appropriate input file record. If 'I' is not specified, then the input file, reordered, will be reblocked and appear, generally much more compactly, in the output file in record-compressed sequential format.

If the input file is sequential in its original format, then there is only one choice for the output format; the output file format for a sort on an input file which is sequential must be sequential.

39.4.6 Limited output format option

In many cases, especially when making reports, directories etc. from the data base, it isn't necessary to have the entire record transferred from the input file to the output file during a sort. For instance, an entire personnel data base can be sorted by name to produce an internal company telephone directory. However, it is obvious that all that is needed is the name and

telephone number, NOT all the other payroll information. Therefore, SORT permits transferring only that part of the data base desired.

The following is the generalized statement format for the limited output specification which is entered as a second line of parameters:

<(SSS[-EEE]^*^'QQQ')[/(P^NNNTC)]>[,<DUPLICATE OF PRECEEDING>]...

Where different items within parentheses are separated by ^. Only one item within a pair of parentheses may be specified. Items within square brackets [] are optional and items within corner brackets <> may be repeated and must be separated by commas.

The following list defines the parameters which can be specified:

SSS.....STARTING position within input record.
EEE.....ENDING position within input record.
These parameters specify the character positions within the input record to be copied to the output record. The EEE specification is optional; if it is not specified then only one character, the character at SSS, will be copied from the input record to the output record. The SSS and EEE options must fall in the range 1 to 249.

*.....ASCII TAG output.
This parameter specifies that an ASCII pointer to the input record appear in the output record. The ASCII pointer points to the input file logical record number and the byte in that physical disk record containing the first byte of the input file logical record. If the 'I' parameter was specified in the SORT options then, since the byte in the physical disk record containing the first byte of the input file logical record will always be '1', the '1' will not appear. The ASCII pointer is a DATASHARE compatible, leading-zero and space-compressed ASCII number. The number of digits for the logical record number pointer is five; the largest number that can be represented is 65,535.

The number of digits for the byte pointer (if it is generated; that is, the 'I' parameter was not specified) is three; the largest number that can be represented is 250.

QQQ.....QUOTED character string.
This parameter specifies an actual string of quoted characters that is to be copied into the output record. The quoting symbol is the single quote ' mark. The string may include any characters except the ' mark itself and 015, and must be less than 90 characters long.

P.....PRIMARY record to be source.
This parameter specifies that the information specified by the prior set of START/END positions is to be extracted from the primary record for the current record block, rather than the present (secondary) record. This parameter has no effect when an output record is being generated from a primary record.

NNN.....NUMERIC position of evaluation character.
This parameter specifies the character position for the character (the 'C' parameter below) indicating whether the information specified by the prior set of START/END positions is to be copied from the input record to the output record. The number must fall in the range 1 to 249.

T.....TYPE of evaluation.
This parameter specifies the equivalence or inequivalence of the evaluation character; that is, whether the character in the input record should be EQUAL to or NOT EQUAL to the evaluation charater. The actual character entered is '=' for equal or '#' for not equal. If the evaluation is satisfied, then the information specified by the prior set of START/END positions will be copied to the output record.

C.....CHARACTER, record evaluation.
This parameter specifies the actual test character for record evaluation. The actual character entered is any character except

015.

In the same manner that the key of the records is specified by fixed column number, i.e. 1-10 for the first ten characters, the limited output feature specifies that part of the records to be transferred. Should the response 1-10 be given to the limited output format request, only the first ten characters of each record will be transferred to the output file. The limited output format specifier operates in the same manner as the specification of multiple discontiguous sort key fields. For instance, 1-10,50-70 would transfer thirty-one characters from each record of the input file to the output file. The eleventh character in the output record would be the fiftieth character of the input record, etc.

To invoke the limited output format option, the operator includes the 'L' parameter in the specifier list. If and only if the L is specified during the SORT call, will there be a second question asked of the operator on the next line:

LIMITED OUTPUT FILE FORMAT:

This question requires at least one non-trivial field specification or constant (see next paragraph). The number of field and constant specifications is only limited by that which can fit on the keyed in line.

To permit even more utility in report generation, SORT allows inclusion of constants in the output record that didn't occur in the input record. For instance, assume that the personnel data base was a full record of about 240 characters and that the employee's name appears in columns 80 to 110 and his telephone number was in columns 171 to 180. To make a telephone directory in alphabetical order, one could answer the following to the limited file output format request:

80-110,' - ',171-180

Note that this would put out the name followed by one space, a hyphen, one more space and the number. Any number of input file fields and constants can be placed in the output file up to the limit of the line on which the specification is typed.

Often not every record of the input file is needed in

the output file. Limited output allows selection of records from the input file, based on character evaluation on one character position. For example, if a primary/secondary file is being sorted and only the primary records are desired in the output file, the command could appear as:

```
SORT INFILE,OUTFILE;LP1=*,2-10
LIMITED OUTPUT FILE FORMAT:
1-85/1=*
```

Columns 1-85 of the input record will be written to the output file if column 1 is an *.

Limited output can be used to make more complex selections. If it is desired to output records containing a 0 in column 5 OR a 1 in column 6, the command would be:

```
SORT INFILE,OUTFILE;L5-8,12-15
LIMITED OUTPUT FILE FORMAT:
1-85/5=0,1-85/6=1
```

To output records containing a 0 in column 5 AND a 1 in column 6 would require two SORTs, the first using a limited output to test column 5 and using only a 1-character key, to make the SORT as fast as possible. The second SORT would use a limited output testing column 6 and would be given a sort key to correctly order the output file.

There is no relationship between the primary / secondary specification on the command line and the conditional output specification on the limited output format line.

Also note that the output file requires proportionally less room than the input file when limited. Often this fact can be put to use when the disk file space is nearly exhausted and a sort is required.

39.4.7 TAG file output format option

For some applications it is useful to have a data file sorted into several different sequences. However, to have several copies of a file on disk merely to have it in different sequences consumes a lot of disk space, and indeed if the file is a very large file many copies of it may not fit onto one or even four disk packs.

This problem could be avoided if there were a way to index into the one main file in any of several different sequences. The index pointers could exist as a file, and the index entry for each record in the main file would only have to be three bytes long -- two bytes for the LRN (Logical Record Number) and one byte for the BUFPTR (Buffer Pointer -- a pointer to the beginning of the actual desired record within the disk physical buffer).

SORT provides for the generation of such an indexing file, a TAG file, by the 'T' variation of the 'R' option. A TAG file may be generated for either a sequential or blocked file, and will have the same format for either file. The format of a TAG file is simple:

1. For each record in the input file, the TAG file will have a three byte binary pointer to the first byte of the record.

2. The format of the pointer is:
   Byte 1: MSPLRN (Most Significant Portion of LRN),
   Byte 2: LSPLRN (Least Significant Portion of LRN),
   Byte 3: BUFPTR (Buffer Pointer).

3. The three-byte binary pointers are blocked 83 to a physical disk record.

4. The Physical-End-Of-Record mark is an 003 and the rest 000's.

5. The End-Of-File mark is: beginning at the first byte in the physical record, six 000's, one 003, and the rest 000's.

TAG files may be used by assembly language programs or by RPG II (as Record Address files).

For users writing their own Assembly language code to use a TAG file, it is important to know that the MSPLRN and LSPLRN are together a 16-bit binary pointer to the DOS LOGICAL RECORD NUMBER of the input file, as opposed to the USER LOGICAL RECORD NUMBER. The difference is this: The DOS LOGICAL RECORD NUMBER of a file points to the actual Nth record (starting with zero, the primary RIB) in the file, whereas the USER LOGICAL RECORD NUMBER of a file points to the Nth DATA RECORD (starting with the zeroth data record) in the file. Thus a DOS LRN of zero points to the very first record of the file, which is the master copy of the RIB, a DOS LRN of one points to the second record of the file which is the RIB copy, a DOS LRN of two points to the third record of the file (which is the FIRST DATA RECORD of the file and the USER LOGICAL RECORD NUMBER zero), and so on. The LRN given in the TAG file can NOT be used with the POSIT$ routine unless it is biased

by -2. It is much easier to simply place the LRN from the TAG file directly into the LOGICAL FILE TABLE ENTRY for the file that is indexed.

The case with the BUFFER POINTER byte is similar to the LRN pointer bytes. The BUFFER POINTER byte from the tag file is the DOS BUFFER POINTER as opposed to the USER BUFFER POINTER. The difference is this: the DOS BUFFER POINTER points to the actual Nth byte of a disk buffer (starting with zero), whereas the USER BUFFER POINTER points to the Nth DATA BYTE in the disk buffer; the beginning (zeroth) DATA BYTE in the buffer is the fourth byte in the buffer; the first three bytes are reserved for the DOS. Thus, a DOS BUFPTR of zero points to the very first byte in the buffer, which is the PFN (Physical File Number) of the file, a DOS BUFPTR of one points to the second byte in the buffer, which is the DOS LSPLRN, a DOS BUFPTR of two points to the third byte in the buffer, which is the DOS MSPLRN, a DOS BUFPTR of three points to the fourth byte of the buffer (which is the very first DATA BYTE in the buffer), and so on. The BUFPTR given in the TAG file can NOT be used with the GETR$ or PUTR$ routines unless it is biased by -3. It is much easier to simply place the BUFPTR from the TAG file directly into the LOGICAL FILE TABLE ENTRY for the file that is indexed.

If the TAG file option is specified then the LIMITED OUTPUT FILE FORMAT or the HARDCOPY OUTPUT can NOT be specified.

If a TAG file is generated when the 'P' (PRIMARY SORT) option is specified then TAG file pointers will be generated only to the PRIMARY records in the input file.

If a TAG file is generated when the 'S' (SECONDARY SORT) option is specified then TAG file pointers will be generated that point to each PRIMARY record of the input file (in their original sequence) each primary tag being followed by pointers to the SECONDARY records in the record block <u>in their sorted sequence</u>.

When a TAG file is generated for 'P' or 'S' sorts, no indication is given in the TAG file pointer as to whether the pointer points to a primary or a secondary record; it is up to the user's program to check the records in the indexed file to determine when a record block begins or ends.

39.4.8 KEYTAG File Output Format Option

Requesting a Keytag file output will cause a file (default extension "TXT") to be created. This GEDIT- compatible text file contains the record pointers and the key. The record pointers (first 8 bytes of the rcord) consist of a 5 byte logical record number (range 0 to 65,535) and a 3 byte buffer address. The record number is the user logical record number, that is, zero points to the first data sector. Therefore, the user logical record number, converted to binary, may be used with the POSIT$ routine. The buffer address is the buffer pointer, that is, one points to the first data byte in a sector. It may be biased by 2 and placed directly into the Logical File Table, or if biased by -1, used by the GETR$ routine. This Keytag file output is the Keytag file used by INDEX.

If a sequence file (e.g., EBCDIC/SEQ) is used, the key produced by this option will be translated to that sequence. If the un-translated key is desired, a Keytag file may be created (slower) by requesting ASCII TAG output from the Limited Output Format Option.

39.4.9 HARDCOPY output option

Many times it is desired to have a hardcopy (printed) output from a SORT instead of or in addition to the creation of a disk output file. This can be easily accomplished with SORT by specifying the 'H' (HARDCOPY) option along with the 'L' (LIMITED OUTPUT STRING) option. The 'H' option is essentially an expansion of the 'L' option because disk data files are almost never suitable for full image output to a printer; decimal points need to be inserted into dollar and cents amounts, dashes need to be inserted into part numbers, and spaces need to be placed between dollar amounts and part numbers to columnate the data, and so on. If it is desired to list output records in full image format, it is only neccessary to give:

```
1 - n
```

(where n is the maximum printable character on printer) as the limited output string specification.

Sort will not send a line of over 132 characters to a printer. If the limited output specification designates a longer output record, then the full specified formatting will be applied to the disk output file (if any), but only the first 132 characters of the record will be printed.

If the following special characters are imbedded in the output record, they will be interpreted as indicated:

015 = End-Of-Record and Carriage-Return/Line Feed.
012 = Line Feed.
014 = Form Feed.

SORT will support either a local printer (address 0303) or a servo printer (address 0132). If a servo printer is on-line at the beginning of the FINAL MERGE then it is used as the output printer device; else a local printer will be used. If both printers are available on a system, selection between one or the other cannot be forced by parameterization; if output is desired to the local printer then the servo printer must be turned off.

39.4.10 Primary/Secondary sorting considerations

If the 'P' (PRIMARY) or 'S' (SECONDARY) SORT option is used then the input file must have a PSPSPS.... format in order for SORT to work as expected, where P is one primary record and S is one or more secondary records. The first record in the file should always be a primary record, and the last record should be a secondary record. There should always be at least one secondary record following each primary record. Tertiary and further level records cannot be accommodated by SORT.

In some cases it may be possible to successfully sort a file using the 'P' or 'S' options even if the file does not faithfully follow the above rules. However, the user must exercise great caution if he is to successfully "fudge" a system as complex as SORT. Pitfalls will be many. For example, if a file has the format PPPPSPSPS..., and a sort is done using the 'S' option, the output file will probably not contain the first three primary records at all. This case occurs because when sorting using the 'S' option, pointers are generated for only the secondary records, prefixed by a pointer to the record preceeding the first secondary record of a record block. Since no secondary pointers were ever generated for the first three primary records, they are simply lost. It should be easy for the user to imagine what would happen to a file if a tertiary sort were attempted.

39.4.11 Key File Drive Number

There are three file systems associated with a sort. The first is, of course, the input file. The second is the output file. The third is the keyfile system. (The user only uses the output file - the keyfile system is a scratch file used by the system during sorting). There are actually two files which get opened during the sort for the keyfile system. They are *SORTKEY/SYS and *SORTMRG/SYS. These two files can grow to considerable sizes during the sorting procedure since they are proportional to the number of records and the size of the key field.

There are two considerations for the location of the keyfile system. The first is the problem of room. The keyfile must be on a drive with sufficient room to hold it. The second is speed. The greatest increase in speed occurs in removing the keyfile system from the same drive as the input file. Greater speeds can occur if it is, as well, not on the same drive as the output file. Normally the SORT does a pretty good job of determining the best location of the two keyfile files and it shouldn't be necessary to specify anything for this. However, under complex circumstances, it may be desirable for the operator to specify the drive number for the keyfile. Should this be the case, the user should type in the <:DRk> specification as indicated in the general command format in the Generalized Command Statement Format section.

39.4.12 Disk space requirements

A formula for determining the room in physical disk records that will be required for the SORT work files is:

$$R = \frac{2N(L+P+3)}{S} + 4T$$

where: R = Room in physical disk records (sectors) required on disk.
N = Number of logical records in input file for which keys will be generated:
= number of records in file if not sorting on 'P' or 'S'.
= number of primary records in file if sorting on 'P'.
= number of secondary records in file if sorting on 'S'.
L = Length of the sort key in bytes.
P = 3 if sorting on secondary records,

0 if not sorting on secondary records.
T = number of sort key trains.
S = bytes per block of physical space available to the user (nominally 253 bytes)

The value of T can be computed approximately, as:

$$T = \frac{N(L+P+3)}{5700}$$

39.4.13 LINK into SORT from programs

There are three ways in which a SORT can be initiated:

1. From the keyboard via the DOS COMMAND HANDLER;
2. By using the DOS CHAIN command;
3. By loading and linking to SORT/CMD from an assembly language program.

Datashare users can invoke SORT by using the rollout facility to start or continue a chain (see CHAIN and the DATASHARE User's Guide for more details).

The following detailed information is provided for users writing system-level programs in assembler language, since Datapoint does not release a source listing of the SORT program. Normal usage of SORT requires no knowledge of the following information.

Sort reserves for the user a nominal amount of storage normally occupied by the DOS DEBUG$ routine. The specific memory locations saved are 06144 through 06377. This permits the user to partially overlay his program with the SORT utility and regain control at the completion of the sort. Additionally, the next page of storage, 06400-06777, is available to the user if full image output records are to be generated. The DOS interrupt handler is disabled during the sort but is re-enabled upon completion of the sort. Of course, if the user has a foreground process running before and after the sort, the process must be controlled from within the memory not used by SORT, or when foreground is re-enabled it will vector to whatever SORT left in memory.

The steps to call SORT from an assembler program are as follows:

1. Close files 1, 2, and 3 if open.

2. Set MCR$ (01400-01543) with the command string terminated by a 015.
3. Load the SORT utility.
4. PUSH the stack.
5. Point HL to a parameter table with the format:

```
    PTABLE  DA    LIMSTG
            DA    HEDING
            DA    EXITAD
```

6. RETURN

Where:

LIMSTG = the limited output specification string, terminated by a 015. If there is to be no limitation output specification, put 0. If there is a LIMSTG, it must exist entirely within the range 06144-06377. The LIMSTG must be exactly the characters as they would be entered from the keyboard. Examples follow.

HEDING = the hardcopy heading string, terminated by a 015. If there is to be no hardcopy output, put 0. If there is a hardcopy heading string, it must exist entirely within the range 06144-06377. The HEDING must be exactly the characters as they would be entered from the keyboard. Examples follow.

EXITAD = the first memory location to be executed upon successful completion of the sort. If the sort is to return to the DOS upon completion, put 0. If there is a specific exit address, it must exist within the range 06144-06377. Normally, the instructions at the exit address will load and run the program to be run after the sort, or will re-load a control program of the user's own control system.

A simple example of loading and running sort from an assembler program would be:

```
            .
            .
1.SRTCMD    DC    'SORT INFILE,OUTFILE',015 SORT CMD STRING
2.SRTNAM    DC    'SORT   CMD'      NAME OF SORT UTILITY ON DISK
3.PTABLE    DA    0                 NO LIMITATION STRING
4.          DA    0                 NO HARDCOPY HEADING
5.          DA    0                 NO SPECIAL EXIT ADDRESS
            .
            .
            .
6.RUNSRT    LC    SRTNAM-SRTCMD     MOVE THE SORT COMMAND STRING
```

```
 7.          DE     MCR$              TO MCR$
 8.          HL     SRTCMD
 9.          CALL   BLKTFR
10.          LC     -1                LOAD THE SORT UTILITY
11.          DE     SRTNAM
12.          CALL   LOAD$
13.          PUSH                     PUSH THE SORT STARTING
                                      ADDRESS
14.          HL     PTABLE            POINT TO THE PARAMETER TABLE
15.          RET                      RUN SORT
             .
             .
```

The above sequence of instructions could be located anywhere in memory, except lines 13 thru 15 must obviously reside in a portion of memory from 06144 thru 06377 to avoid being overlayed when the SORT utility is loaded from disk. The above instructions exemplify the simplest possible case of linking to SORT, in that only the SORT command and an input file and an output file are specified, all other options are defaulted. The above instructions have the same effect as calling SORT by entering the line:

```
SORT INFILE,OUTFILE
```

to the DOS COMMAND HANDLER.

Here is a line-by-line explanation of the instructions:

Line 1 defines the SORT command string. This is accomplished by a simple DC statement of a quoted ASCII string followed by a 015. The quoted ASCII characters are exactly the same that would be keyed in to the DOS Command Handler if the sort were being initiated from the keyboard. The 015 is the string delimiter and is the same character that is placed after a string by the KEYIN$ routine when the "ENTER" key is depressed. The SORT command string can be up to 100 characters long including the 015 because the MCR$ area is 100 bytes long. Note that this is nineteen characters more than can be specified from the keyboard.

Line 2 defines the name of the SORT utility main overlay. Notice that the complete name of the SORT given here must be exactly the name as listed in the DOS directory of files. The eleven ASCII characters in a file name specification include an eight character filename and a three character extension. Since the filename of SORT is only four characters, it must be followed by four spaces before the extension of "CMD" can be given.

Line 3 defines the beginning of the six-byte parameter table. The first two bytes of the parameter table specify the address of the beginning of the Limited Output Specification string. In this example there is to be no limited output specification string, so an address of 0 is given.

Line 4 defines the address of the beginning of the HARDCOPY HEADING string. In this example there is to be no hardcopy output, so an address of 0 is given.

Line 5 defines the address of the Exit Address, or the address to which the SORT is to exit when it is successfully completed. (If something goes wrong during the sort, exit is to the DOS.) In this example there is to be no special exit address, so an addess of 0 is given.

Line 6 begins the actual process of calling SORT from the program. Lines 6 thru 9 move the SRTCMD string from wherever it is in memory to the MCR$ area.

Line 10 specifies that SORT is to be loaded from wherever it is found in the disk drives that are on-line to the system. Refer to the chapter on System Routines if you are not familiar with the DOS LOAD$ routine.

Line 11 points to the name of the SORT utility main overlay in memory, given in SRTNAM, line 2.

Line 12 calls the DOS LOAD$ routine which finds the SORT main overlay program on disk and loads it into memory, leaving the starting address in HL.

Line 13 puts the starting address of SORT on the P-counter Stack.

Line 14 points to the Parameter Table, lines 3, 4, and 5. The way that SORT knows that it is being run by the DOS Command Handler or by a user program is by comparing the values of the HL contents with the entry point of SORT. If the values are equal, as they are immediately following a LOAD$, then SORT asks for a Limited Output Specification string and a Hardcopy Heading string if they are specified in the SORT COMMAND string. If the values are not equal, then SORT checks the memory pointed by HL for the location of the Limited Output Specification string, the Hardcopy Heading string, and an Exit Address.

Line 15 effects the actual transfer of execution to the SORT utility. Since the starting address of the SORT was PUSHed onto

the P-counter stack, a RETurn instruction JuMPs to the SORT starting address.

39.5 The use of CHAIN with SORT

The reader should first familiarize himself with CHAIN by thoroughly reading the CHAIN Section.

CHAIN is a system whereby the operator of a Datapoint DOS may pre-define a procedure sequence of his own programs, system commands and utilities (including keyboard answers to questions requested by these programs) and have them called and sequentially executed by a single name. This feature is especially powerful when using SORT since there may be a repetitive sequence of routines with complex parameterizations which could make good use of simplification.

A Datashare program can link to SORT by executing a ROLLOUT instruction to a user-built CHAIN file which includes the SORT command line and, if specified, the Limited Output specification line and a Hardcopy Heading line, followed by the DSBACK program to re-load the Datashare.

39.5.1 How to Set up a chain file for SORT

The author of a chain file only needs to remember that ALL questions that the system requests INCLUDING those initiated by the executing programs MUST BE ANSWERED from the chain file just as though they would be typed in from the keyboard.

For instance, the initiation of a sort

"SORT INFILE,OUTFILE;I3-42"

could be done through chain. To do this, use EDIT or BUILD to type in that exact sequence of characters into a file. Note that the file will, in this case, consist of a single line as typed above. The file can be any name, but for purposes of simplifying the explanation, it shall be referred to as "CHAINFIL". If "CHAINFIL" consists of that single line, and if the operator types the command "CHAIN CHAINFIL" to the DOS, the SORT specified above would be initiated. If the 'L' specification were included in the statement above, then SORT would ask for another line of information. In this case, the file "CHAINFIL" would have to have two lines in it with the first being the SORT command and the

second being the limited output file format specification.

39.5.2 **Naming a repetitive SORT procedure**

Frequently there are sorts and printouts and other procedures which occur together and for which a name invoking the procedure would be a great simplification.

For instance, in the telephone directory example above, the process of sorting the file into a limited output file and then listing it on a local printer could be procedurized as follows:

```
SORT EMPFILE,TELFILE;L80-110
80-110,'  - ',171-180
LIST TELFILE;XL
TELEPHONE DIRECTORY FOR XXXXXXXXXX CORPORATION
```

Note that there are four statements. The first is the SORT command. The second is the answer to the limited format initiated by the 'L' in the SORT command. The third is the DOS LIST command with the specifiers of 'X' which says 'without line numbers' and the 'L' which means local printer. Then there is a forth line which the LIST command requests - the heading. This question must also be answered in the chain file. If the above four statements were placed in a file by the Editor (or by any other means) and then CHAIN were invoked with that file specified, the result would be a printed telephone directory from the personnel files.

39.5.3 **Using CHAIN to cause a merge**

Consider a situation wherein a system has a master file called 'MASTER' and a file of records to be added, in sequence, to the master file called 'ADDFILE'. To merge these two files in sorted sequence at the end of each day would normally require a sequence of keyed in operations which are somewhat complicated and error prone. CHAIN can cause an effective MERGE and assign it a single name as follows:

```
SAPP MASTER,ADDFILE,MASTER
SORT MASTER,SCRATCH;1-20
KILL MASTER/TXT
Y
NAME SCRATCH/TXT,MASTER/TXT
```

Note that the procedure:
1) appends the ADDFILE to the MASTER file.
2) Sorts the extended MASTER file into a SCRATCH file.
3-5) Renames the SCRATCH file as the new MASTER file. Thus, it is apparent that a merge can be effectively achieved using SORT and by using chain to pre-define the procedure.

39.6 SORT Execution-Time Messages

This section describes the operator messages that SORT may display on the CRT screen during execution. Some of the messages are monitor messages to keep the operator informed of the progress of the program, while other messages are error messages.

DOS. VER. n.n SORT COMMAND - date

This message is the SORT sign-on.

SORT OVERLAY MISSING.

This message is displayed if the SORT/OV1 file is not on the same drive as the SORT/CMD file.

INPUT FILE REQUIRED.

This message is displayed if no filename was specified for the first file specification. This would happen if a command line such as:

SORT ,OUTFILE or SORT /TXT,OUTFILE

were entered.

OUTPUT FILE REQUIRED.

This message is displayed if no filename was specified for the second file specification AND if the 'L' and 'H' options were not specified.

BAD DEVICE SPECIFICATION.

This message is displayed if a drive specification in a file specification was not entered in a valid format.

OUTPUT FILE SAME AS INPUT.

This message is displayed if the FILENAME and EXTENSION of the INPUT file and the OUTPUT file are the same, and the DRIVE NUMBER for each file is the same or not specified for EACH file.

INPUT FILE NOT FOUND.

This message is displayed if the INPUT file could not be found on any drive on-line to the system if no drive was specified, or on the drive given if a drive was specified. If no extension is supplied in the file specification an extension of TXT will be assumed; in this case if a file FILENAME/TXT is not on-line or on the drive specified then the INPUT file will not be found.

INPUT FILE RIB ERROR.

This message is displayed if a read parity error occurs when the INPUT file's RIB is checked to determine the INPUT file's length.

KEY FILE SPECIFICATION ERROR.

This message is displayed if a FILENAME or EXTENSION is given for the KEY DRIVE specification.

KEY FILE DEVICE SPECIFICATION ERROR.

This message is displayed if the drive specification for the KEY file is not a valid drive spec.

SORT KEY FILE PLACED ON DRIVE #

This message is displayed if the KEY DRIVE was not specified on a multi-drive system. The message is to notify the operator of the location of the KEY file. The # stands for a valid drive number.

OPTION FIELD ERROR.

This message is displayed if a semicolon (;) is entered at the end of the SORT command line but is not followed by any option specifications.

OPTION SPECIFICATION DUPLICATION.

This message is displayed if a command line such as:

SORT INFILE,OUTFILE;DLA

were entered. The 'D' and 'A' options are both variations of the ORDER option, and obviously both cannot occur simultaneously.

HARDCOPY ONLY IF LIMITED OUTPUT SPECIFIED.

This message is displayed if the 'H' option is specified but the 'L' option was not given previously.

ILLEGAL HEADER SPECIFICATION.

This message is displayed if the 'P' or 'S' option is given but is immediately followed by the 015 byte -- the "ENTER" key.

ILLEGAL HEADER KEY EVALUATION.

This message is displayed if the character immediately following the 'PNNN' or 'SNNN' option is not '=' or '#'.

ILLEGAL SORT KEY SPECIFICATION.

This message is displayed if a key position of 0 or greater than 249 was specified, or if a key position was not terminated by , or - or 015, or if a two-position key was not terminated by , or 015.

SORT KEY TOO LONG.

This message is displayed if the total sort key is longer than 118 characters long.

OVERLAPPING SORT KEY SPECIFICATIONS---SORT OPTIMIZED.

This message is displayed if the same record positions were specified for more than one sort key group. SORT does not

repeat duplicate positions in sort key generation and thus saves processing and disk read/write time.

OVERLAPPING SORT AND HEADER KEYS---SORT OPTIMIZED.

This message is displayed if the same record position is specified as a sort key position and a header indication position. The position is removed as a sort key position and the key is thus shortened. The effect is as for the previous message.

LIMITED OUTPUT FILE FORMAT:

This message is displayed if SORT has accepted the SORT command line including all option specifications and if the 'L' option has been given. The operator must enter the limited output specification line.

NULL LIMITATION SPECIFICATION.

This message is displayed if the 'L' option was given but the limitation specification was only 015 -- the "ENTER" key. If the 'L' option is given then a non-empty limited output specification string must also be given.

INVALID LIMITATION SPECIFICATION.

This message is displayed if the limited output specification does not fit the syntax given in the section on Limited Output Format Option. Usually the fault is that a comma was not placed between option specification groups, or double quotes " were used instead of single quotes '

ENTER THE HARDCOPY HEADING:

This message is displayed when the limited output specification has been accepted and if the 'H' option was given. The operator must enter from 0 to 79 characters of information which will be printed at the top of each page printed during SORT output generation.

SEQUENCE FILE NAME REQUIRED

This message is displayed when the sequence file field is blank and the file specification fields have not been terminated with a semi-colon or an end of line designator.

SEQUENCE FILE NOT FOUND

This message is displayed when SORT requests the sequence file be OPENed and DOS cannot locate the file on the disk drive indicated. Note that if the drive is not specified, the drive on which the SORT/CMD resides is implied.

SEQUENCE FILE FORMAT ERROR A

This message is displayed when SORT determines that the sequence file specified is not an absolute object file.

SEQUENCE FILE FORMAT ERROR n

This message is displayed when SORT receives an error return from LOADX$ when an attempt is made to load the sequence file. The value of n may be 0-6 and is defined as follows:

0 If file does not exist
1 If disk drive is off-line
2 If directory parity error
3 If RIB parity fault
4 If file parity fault
5 If off end of physical file
6 If record of illegal format

LIMITATION SPECIFICATION OVERFLOW

This message indicates that limited output parameters entered require more memory (256 bytes) than allocated by SORT.

INTERNAL ERROR -- GET SYSTEM HELP !!!

This message indicates a probable hardware error occurred during a limited output string sort. SORT cannot continue executing.

THE FOLLOWING MESSAGES MAY BE DISPLAYED DURING SORT INITIALIZATION IF SORT WERE LINKED TO BY AN ASSEMBLY LANGUAGE PROGRAM:

INVALID LIMITATION STRING ADDRESS.

INVALID HARDCOPY HEADING STRING ADDRESS.

INVALID USER EXIT ADDRESS.

One of these messages is displayed if the corresponding entry in the parameter table linkage data was not either 0 or in the range 06144-06377 inclusive.

LFT ENTRIES 1->3 NOT CLOSED WHEN SORT ENTERED.

This message is displayed if the user left one of the logical files 1, 2, or 3 open upon linking to the SORT utility.

LIMITATION STRING MISSING.

This message is displayed if the 'L' option was given in the SORT command string but the pointer to the limited output format string in the parameter table linkage data was 0, indicating no limited output format string specified.

HARDCOPY HEADING STRING MISSING.

This message is displayed if the 'H' option was given in the SORT command string but the pointer to the hardcopy heading string in the parameter table linkage data was 0, indicating no hardcopy heading string specified.

THE FOLLOWING MESSAGES ARE DISPLAYED AFTER THE SORT INITIALIZATION IS COMPLETED:

BUILDING SORT KEY TRAIN n.

This message is displayed when all parameter specifications have been accepted and SORT has started the extraction of the sort keys from records of the INPUT file and is writing them to

the *SORTKEY/SYS file.

SORT KEY FILE OVERFLOW.

This message is displayed if there was not adequate room on the KEY DRIVE to hold the *SORTKEY/SYS file. If *SORTKEY/SYS file overflow occurs the file is deleted from the disk before the message is displayed.

NULL OUTPUT FILE.

This message is displayed if no sort key records were generated. A null output file (first record EOF) is prepared before SORT ends.

INTERMEDIATE SORT PASS n.

This message is generated during sorting of the sort key trains on the *SORTKEY/SYS file. The only actual sorting done during a sort is that which can be done on the initial sort key trains, which are made short enough that they will fit in memory. After the sorting of the keys within each initial train, the trains are merged sixteen abreast into larger trains, repeatedly until only one train remains.

INTERMEDIATE MERGE PASS n, TRAIN n.

This message is displayed if more than sixteen sort key trains exist during a merge pass. The intermediate merge pass number is the Nth iteration of the merge process. The train number is the number of the train being output by the merge pass. If more than one train is output by an intermediate merge pass then at least one more intermediate merge pass will be required. If more than sixteen trains are output by an intermediate merge pass then at least two more intermediate merge passes will be required, and so on.

FINAL MERGE: SORT TRAIN n.

This message is displayed during the generation of the output file from the data in the now fully sorted and merged sort key file and from the records in the INPUT file. The sort train number corresponds to the current state of progress as measured

against the number of trains generated by the next to the last intermediate merge pass.

MERGE FILE OVERFLOW

This message indicates not enough disk space is available for the merge file.

OUTPUT FILE OVERFLOW

This message indicates not enough disk space is available for the output file.

CHAPTER 40. SUR COMMAND

40.1 Purpose

When a specific disk is used for more than one purpose, some inconveniences occasionally turn up. Assume for a moment that a user has a disk which he is using for program generation on each of two more or less unrelated projects. When he uses the CAT command, for instance, he will normally see a whole range of files, some of which are not related to the project he may be currently interested in. Or, he may begin editing a new file on the disk, only to find that another user of the same disk may have already had a file of that name. At times like this, it would be convenient to logically partition the directory so that a user would only have a portion of it, the portion he is currently interested in, available to him at one time.

A more concrete example is the DOS itself and its various commands. Obviously Datapoint's DOS.A, DOS.B, and DOS.C bear a strong resemblance to each other. The DOS and most of the command files are configured at assembly time through conditional assembly and equates to support a given disk controller and specific file structure. The result is several different object code files, all with a /ABS extension, for each single source file with a /TXT extension. Yet it is desirable for a number of reasons to keep all of the object code files for all the DOS and commands on a single drive.

Without the DOS subdirectory facility, it is not permitted to have two files on a given logical drive with the same name.

40.2 About Subdirectories

The use of the SUR (Subdirectory Utility Routine) command allows the user to logically partition the directory on a given disk into several smaller subdirectories. Each such subdirectory can then contain zero or more files, up to the combined maximum of 256 files per logical drive. Each subdirectory on a disk has a unique name. Two subdirectories always exist on all drives; these are called SYSTEM and MAIN. The names for the other subdirectories are assigned by the user as he establishes them, and follow the same rules as for any standard DOS file name. As a

subdirectory is created, the name specified by the user is related to a unique number which is referred to as the subdirectory number. The relationship between subdirectory names and subdirectory numbers is not unlike the relationship between DOS file names and physical file numbers. A given subdirectory may have different numbers on different drives, even though the subdirectory name is the same.

It is important to realize that subdirectories are not a way of getting more than 256 files on a drive. This they cannot do. The thing that subdirectories are good for is partitioning the directory and restricting the scope of a file name. This allows several files of the same name to exist on one disk at the same time, without causing the DOS to become confused as to which is the one to be referenced at any time. The way the DOS achieves this is that each of the files is in a "different subdirectory", and hence is uniquely identified even though the name and extension may be identical.

40.2.1 Creation of Subdirectories

Subdirectories are created with the SUR command. All that is required is to specify a name for the proposed subdirectory and request its creation. Creation of a subdirectory does not actually result in any real change to the directory on disk at all; all it does is to cause the specified name to be entered into a table in SYSTEM7/SYS (yes, that's why SYSTEM7 isn't write protected), kept on disk, which relates each subdirectory name with its subdirectory number. The user is allowed to specify which drive he wishes to create the subdirectory on; if he does not indicate a specific drive, the named subdirectory is placed onto all on-line drives if possible.

40.2.2 Deletion of Subdirectories

Subdirectories are deleted with the SUR command. The user specifies the name of the subdirectory he wishes to remove and requests its deletion. Deletion of a subdirectory does not result in KILLing the files within the range of that subdirectory. If a subdirectory to be deleted contains one or more files, the files are first moved from that subdirectory to the one called MAIN before the named subdirectory is deleted. The user is allowed to specify from which drives the subdirectory is to be deleted; if he does not indicate a specific drive, the named subdirectory is deleted from all on-line drives on which it appears.
NOTE: Subdirectories may not be deleted while PS is running.

40.2.3 Being "in a Subdirectory"

The user can define at any time which of the subdirectories on each of his disks contain the current files he is interested in. This is done with the SUR command by specifying the name of the subdirectory containing the files of current interest. This action causes him to be placed "into" the named subdirectory on the drive specified. (If no specific drive is mentioned, he will be placed "into" the subdirectory specified on all on-line drives containing a subdirectory with the given name). It is appropriate to point out that the current subdirectory on each drive need not have the same name; for example, the user could easily be in subdirectory PROGRAMS on drive zero and in subdirectory DATABASE on drive one at the same time.

Once in a specific subdirectory on a drive, that state does not normally change until the user requests being placed into a different subdirectory (again via the SUR command) or re-boots the DOS. Rebooting the DOS causes the user to be placed into the subdirectory named SYSTEM on all drives.

40.2.4 Scope of a File Name

When a program accesses a file under DOS, it tells DOS the name and extension of the file it is looking for and either indicates one specific drive which the DOS is to search for the file, or requests that the DOS look on all on-line drives. In order for the DOS to "find" the given file, the DOS must find a file whose name and extension exactly match the ones specified by the requesting program. If no such file can be found, the DOS returns indicating that the specified file cannot be found and therefore probably does not exist.

When subdirectories are in use, this matching of name and extension is expanded so that in addition to a file's name and extension matching those specified by the requesting program, the file must also be within either the current subdirectory (for that drive) or the one called SYSTEM in order to be "found".

Therefore the scope of a file name can be more or less defined via the following: when a user is in subdirectory X on drive Y, files can be "seen" by his program only if they are in either subdirectory X or subdirectory SYSTEM. Files in any other subdirectory will not appear to exist.

40.2.5 About Subdirectory SYSTEM

It has been shown that files in the subdirectory named SYSTEM are special in that they can be accessed regardless of which subdirectory the user is "in" on a specific drive. Likewise, a special situation also occurs when the user is "in" the subdirectory named SYSTEM. When the subdirectory named SYSTEM is the current subdirectory on a given drive, all files on that drive are accessible regardless of which subdirectory they themselves are actually in.

A little caution must be used when a user is in subdirectory SYSTEM on a disk with multiple files of the same name and extension. The caution is that, although each of the files is still associated with one and only one subdirectory, all of the files on a disk are available when the user is "in" the SYSTEM subdirectory. The result is that in this situation, one of the files of the desired name and extension will be referenced; which one is referenced is, however, undefined. Therefore, good practice dictates that if a user has more than one file with the same name and extension on some drive, that he make a point of always knowing which subdirectory he is in (and that it is <u>not</u> SYSTEM) if it matters to him which of his files he references.

40.2.6 Files vs. the User Being "in a Subdirectory"

It is important not to confuse the two distinct concepts of a <u>file</u> being in a subdirectory as opposed to that of [a user] "being in a subdirectory".

A <u>file</u> being in a specific subdirectory is a way of saying that the file cannot be accessed when the current subdirectory is neither that specific subdirectory nor SYSTEM. This relationship, that of a file being in a specific subdirectory, is retained more or less permanently; if a file is placed in subdirectory SUBDIR1 today on a disk, the disk can be removed and stored on a shelf; if tomorrow the disk is taken down from the shelf and re-mounted, that file will still be in subdirectory SUBDIR1.

A <u>user</u> being in a specific subdirectory is a way of saying that the subdirectory in question is "the current subdirectory" on one or more logical drives. The "current subdirectory" on a drive is less permanent and reflects the use of the SUR command since the previous time the DOS was bootstrapped.

40.2.7 Getting a File into a Subdirectory

In general, there are three ways to get a file into a given subdirectory. The easiest and probably most common of these is automatic. Whenever a file is created, it is always placed into the current subdirectory on the drive on which it is created.

Once a file has been thus created, it can be moved between subdirectories with the NAME command. The NAME command can take a file within the scope of the current subdirectory and put it into the current subdirectory if it is not already (which is useful if either the source or destination subdirectory is SYSTEM) or can place it into any other subdirectory the user might wish to put it into.

40.3 Usage

The SUR command is parameterized as follows:

SUR [<name>][/<function>][:DR<n>][,<new name>]

The function performed by SUR is determined by the absence or value of the <function> field and the name field, as described below.

40.3.1 Establishing a "Current Subdirectory"

If the function field is not given, SUR establishes the named subdirectory as the current subdirectory on all drives on which the named subdirectory exists. If the named subdirectory does not exist on one or more drives, the current subdirectory on any such drives is unaffected. If a specific drive is mentioned, then only the current subdirectory on the specified drive is subject to change.

40.3.2 Creating a Subdirectory

If the function field is /NEW, SUR creates the named subdirectory on all drives on which the named subdirectory does not exist. The current subdirectory is not affected by the operation. If a specific drive is mentioned, then the named subdirectory is only created on the specified drive.

40.3.3 Deleting a Subdirectory

If the function field is /DEL, SUR deletes the named subdirectory on any drives on which the named subdirectory exists. If any files are in the named subdirectory, they are moved to subdirectory MAIN before the named subdirectory is deleted. If the subdirectory being deleted is the current subdirectory on that drive, the current subdirectory is also changed to MAIN. Subdirectories SYSTEM and MAIN cannot be deleted. If a specific drive is mentioned, then the named subdirectory is only deleted from the specified drive.

40.3.4 Renaming a Subdirectory

If the function field is /REN, SUR renames the named subdirectory on any drives on which the named subdirectory exists, to the name specified in the new subdirectory name field. If any files are in the named subdirectory, they will be in the subdirectory specified by the new subdirectory name field upon completion of the operation. Subdirectories SYSTEM and MAIN cannot be renamed. If a specific drive is mentioned, then the name of the named subdirectory is changed only on that specified drive.

40.3.5 Displaying Subdirectories

If the subdirectory name field is not given, SUR displays the names of all subdirectories on all on-line drives. The format of the listing is similar to that provided for file names by the CAT command. The number in parentheses to the right of each subdirectory name is the subdirectory number associated with that name (in octal); an asterisk indicates the current subdirectory on each drive. If a specific drive is mentioned, then only the subdirectories present on the specified drive are displayed.

CHAPTER 41. UBOOT COMMAND

The UBOOT command writes a DOS bootblock onto the cassette tape in the front tape deck.

The UBOOT command then rereads the bootblock to insure that the cassette is good. In addition, the bootblock checks its own parity immediately upon loading and halts if it finds it has not been loaded properly.

If the machine halts upon booting repeatedly and other boot tapes work on the same machine, then the boot tape which causes the boot operation to halt is not a good tape and should be replaced.

The boot tape created by UBOOT reads an IPL (Initial Program Loader) block from disk. The IPL block then reads and executes the DOS bootblock (from disk). The IPL and bootblock are put on disk by DOSGEN and PUTIPL.

The UBOOT tape is capable of loading any version 2.3 DOS from any type of disk. If there are multiple types of disks on your system, they will be scanned in the following order:

1. Mass storage disks
2. Cartridge disks
3. Floppy disks

Logical drive zero will be tested on each of the disks. If drive zero is off-line, depressing the "DISPLAY" key will cause a scan of ALL on-line drives. This means that if drive zero is "down", you can generally continue running. When a disk is found that contains a good IPL, it will be selected as the "BOOT DRIVE"; henceforth overlays will be loaded off it. Commands will also be loaded from the booted drive first (default).

CHAPTER 42. UTILITY/SYS

Most of the DOS commands have been put in an absolute library named "UTILITY/SYS". This has the following advantages:

1. Free up some directory and data space.
2. Makes most of the utility programs available on any disk, i.e., UTILITY/SYS can be on any drive on-line.
3. Assures the user that the most current DOS commands will be used.

Using the librarian utility program (LIBSYS 1.1), many user programs could also be added to UTILITY/SYS. A few guidelines for programs that can be members of "UTILITY/SYS":

1. Programs should start at 017000 or higher.
2. Programs that use overlays should use DOS function 13 and 14 to access the library.

If you have placed your own programs into UTILITY/SYS, do not overwrite UTILITY/SYS on a partial gen. Instead, MIN the new UTILITY/SYS using a different file name, then use LIBSYS as follows:

```
MIN
(filename UTILITY/NEW)
LIBSYS UTILITY/SYS
REPLACE UTILITY/NEW
END
KILL UTILITY/NEW
YES
```

To display the members in UTILITY/SYS, enter:

```
CAT *
```

Note: the CAT command also displays the directory of any library (see CAT command).

When keyboard commands are entered, the specified command will automatically be located as either a separate disk file or a member of UTILITY/SYS. Normally a separate file name is first checked, then the library member. To reverse the normal precedence put a leading * or : in front of the command name. For example:

```
*CHANGE SCRATCH/TXT;X
     or
:CHANGE SCRATCH/TXT;X
```

See the chapter on the Command Interpreter for details on selection of a command from the disk directory or from UTILITY/SYS.

CHAPTER 43. SYSTEM DESCRIPTION

43.1 System Philosophy

The objective of DOS is to allow maximum use of the capabilities of a Datapoint disk system with a minimum of effort. The DOS disk structure provides dynamic space allocation and fully random file access capability on all supported disk types. Also provided are an extensive set of utility programs to perform many basic data processing functions. In all system utilities the operator commands are as simple as possible while providing a versatile program capability. Error codes and program messages are mostly presented in English, avoiding complex, incomprehensible messages.

Datapoint DOS is a facilities oriented system. It provides utility programs for general use, and extensive system routines for use in assembler coding. DOS is not a supervisory system; it imposes practically no overhead. The DOS facilities provide a base for Datashare, BASIC, and most other Datapoint languages and systems.

43.2 System Structure

DOS occupies only the lower 8K of memory in the processor. Of this 8K, only the lower 2.8K is necessary for the support of the disks. The first 768 bytes of memory (0 - 01377) contain the object code loader, entry point table, and interrupt handler. Object code may be loaded from 01400 upwards, overlaying much of DOS. If object code is loaded below 01400, the code overstores the loader or entry points and results are unpredictable.

The operating system debug facility and the keyboard and display routines reside between 2.8K and 4K, the cassette driver routines from 4K to 5.4K, and the command interpreter from 5.4K to 8K. It is recommended that user programs start at 017000 (octal).

To achieve its small size in memory, DOS uses disk-resident overlays for the disk file opening, closing, and allocation routines. Most of the system error messages also reside in an overlay, allowing fully descriptive messages without using a prohibitive amount of memory. A set of short utility routines

(DOS Functions) uses a separate overlay area.

The operating system uses a single disk controller with at least one physical disk drive attached. Each "on-line" drive -- a drive containing a disk ready to read -- is assumed to contain a valid DOS disk, which will have all necessary system tables and files present and in correct format. This assumption on the part of the system requires caution on the part of the operator if a disk not fitting this description is mounted. If, for instance, a disk has been mounted to be DOSGENed, the operator must not run any programs that will attempt to use the disk before it has been genned, or an abort will occur indicating system data failure.

DOS is designed to be run interactively by an operator at the processor console. The operator generally enters commands from the keyboard, which the operating system interprets and executes. During execution, status information needed by the executing program is requested from the operator via CRT messages expecting a keyed response.

A DOS utility program (CHAIN) allows execution of predefined processes automatically in a non-interactive fashion, so no operator attention is required. Other utility programs extend this automatic capability such that the system can be made almost completely operator independent if desired.

CHAPTER 44. SYSTEM STRUCTURE

44.1 Disk Structure

44.1.1 Introduction

Any disk used with DOS is a self-contained information structure. A disk contains up to 256 files, each of which is described in system tables on the disk and which resides completely on the one disk. No system information on a disk references any other disk.

The basic structure of disk storage is the file. Files on Datapoint DOS consist of up to 38,400 sectors, or as many sectors as fit on a logical disk, whichever is smaller. The space occupied by a file is mapped in its Retrieval Information Block (RIB), which is the first sector of the file. The Directory stores the name of each file and provides a pointer to locate the RIB, thus completely defining a file.

Space for files is allocated in clusters, a cluster being the smallest allocatable unit of disk space. In general, each cylinder of a disk is divided into 8 equal clusters. On diskette systems a cylinder has only 4 clusters. Thus a cluster consists of 3, 6, or 24 sectors on diskette, cartridge, and mass storage systems respectively. The sectors constituting a cluster are always contiguous and never cross the boundary of a cylinder or head. The Cluster Allocation Table (CAT) and the Lockout CAT maintain a record of clusters in use or unavailable for use and clusters free for use.

The RIB maps the file space in segments; a segment is a set of contiguous clusters. A file then consists of a set of segments located randomly on the disk, each segment being a small block of clusters. Within this space, the file is logically continuous, there being no logical discontinuity at the boundary of a segment.

Each sector within a file carries its own identification. The first byte of a sector contains the Physical File Number (PFN) of the file to which it belongs. The PFN uniquely identifies a file. The second and third bytes contain the Logical Record

Number (LRN) of the sector. The LRN is a count of sectors in the file, starting with 0 at the first sector, and incrementing by one for each successive sector.

All major tables discussed in this section -- the CAT, Lockout CAT, HDI, Directory, and RIB -- are all kept in duplicate. The backup copy of each of the tables helps prevent data loss in event of a read/write error to a system sector.

44.1.2 Disk Space Management: CAT and Lockout CAT

The Lockout CAT indicates locked out cylinders -- cylinders which will not be used by the DOS. Cylinders are automatically locked out at DOS generation if they are found bad by the surface verification. Cylinders may be manually specified for lockout during system generation or during REPAIR. Cylinder 0 is always locked out for system use. Each byte of the Lockout CAT represents a cylinder: byte 0=cyl 0, byte 1=cyl 1, byte 2=cyl 2, etc. The byte value is 0377 (017 on diskettes) if the cylinder is locked out, and is 000, otherwise.

The CAT indicates available space for the DOS; CAT updates are performed automatically as space allocation or deallocation is performed. As in the Lockout CAT, each byte of the CAT represents a cylinder. Each bit of a byte represents a cluster of the cylinder: bit 7=cluster 0, bit 6=cluster 1, etc. (For diskettes, bits 7-4 are zero, bit 3=cluster 0, bit 2=cluster 1, bit 1=cluster 2, and bit 0=cluster 3). If a bit is set (1), the cluster it represents is either in use by a file or locked out; if a bit is clear (0), the cluster is free.

The CAT and Lockout CAT observe some fixed format rules:

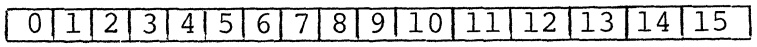
Byte 0 is always 0377
Byte 1 through n may be any value as described above
(n is the number of cylinders on the disk)
Bytes n+1 through 0376 are 0377 (except for directory mapping bytes, if used.)
Byte 0377 is any value. This is the auto-execute PFN and is normally zero.

44.1.3 Files: HDI, Directory Mapping Bytes, Directory, RIB

The Hashed Directory Index (HDI) provides access to, and controls allocation of, the Directory. Each byte of the HDI represents a directory entry, offset from the beginning of the index by PFN. Thus, byte 0=PFN 0, byte 1=PFN 1, byte 2=PFN 2, etc. If the value of the byte is 0377 the directory entry it represents is not in use. When a PFN is in use, a hash code (value 0-0376) generated from the file name is placed in the byte. This value indicates the PFN is in use, and is used to speed directory searching when a file is being loacated by name.

Directory Mapping Bytes are a less sophisticated means of Directory access and control, used in DOS.B version 1 and in diskette operating systems. The mapping bytes are bytes 0357-0376 of the CAT. Each byte represents a directory sector (0-15) and the value in the byte represents the number of entries (0-16) in use in that sector.

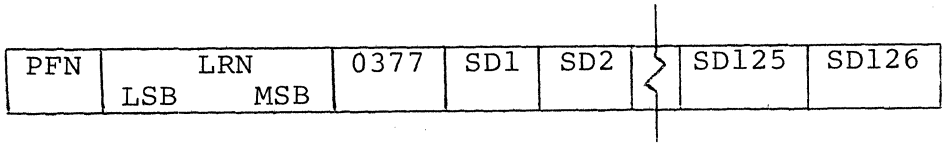
The Directory is 16 sectors (logically referenced as 0-15) containing 256 directory entries, 16 entries per sector. A directory entry contains the name, protection, and subdirectory of a file; it also points to the file's RIB. Directory entry format:



Bytes 0-1 are the RIB address/protection. (See "Addressing Byte Structures".)
Bytes 2-3 are unused (normally zero)
Bytes 4-11 are the file name. A file name is usually ASCII characters as described in the DISK FILES chapter under File Names, padded with blanks to be eight characters long, but may be any values.

Bytes 12-14 are the file extension. Same format rules as file name.
Byte 15 is the subdirectory number, usually 0377, indicating subdirectory SYSTEM.

A Retrieval Information Block (RIB) maps a file's domain on disk. A file is composed of segments, each segment being composed of contiguous clusters. The RIB contains up to 126 segment descriptors which completely describe the clusters allocated to a file.



Each segment descriptor (SD) is two bytes long (see "Addressing Byte Structures"). A segment descriptor of 0377,0377 indicates the end of the RIB. The fourth byte of a RIB is always 0377. The RIB is always the first sector of the file; the RIB copy is the second sector and is identical to the RIB except that its LRN is 1.

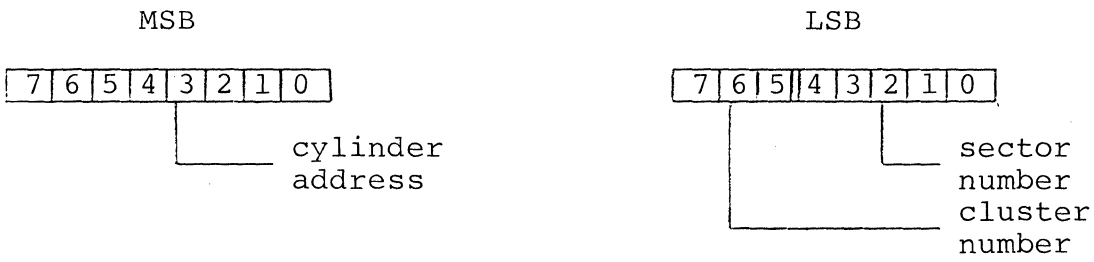
44.1.4 Sector Identification

Every sector of a file contains in its first byte the PFN of the file. The next two bytes are the Logical Record Number (LRN), stored least significant byte first. The PFN and LRN are primarily intended as validation fields when a file record is read. When a file record is written, the PFN and LRN are set correctly; reading a record with a PFN that does not match or an out-of-sequence LRN constitutes a Record Format Error.

Not every sector in the space allocated to a file has this PFN and LRN data. Only sectors that have been used for the file have this informatin set. Unused sectors may have anything in the first three bytes.

44.1.5 Addressing Byte Structures

44.1.5.1 PDA - Physical Disk Address

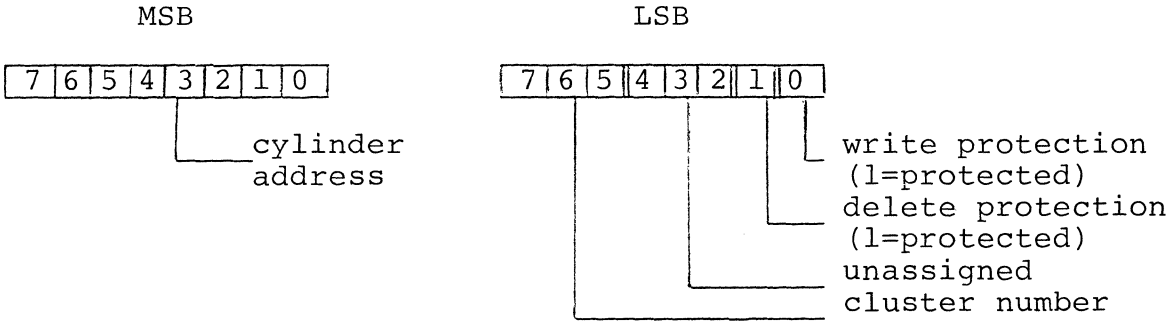


The cluster number references a cluster within a cylinder; values are 0-7 except for diskette systems which use values 0,2,4,6 for clusters 0,1,2,3 respectively. The sector number references a sector within a cluster.

Note: This is the DOS "PDA" and must not be confused with the hardware disk addressing of any particular controller.

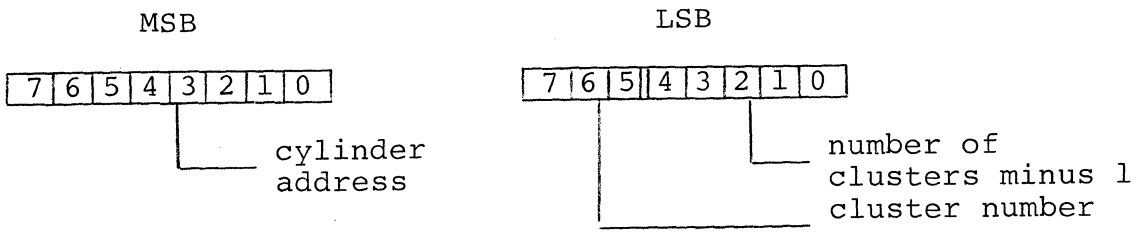
44.1.5.2 RIB Address/Protection

Used in a directory entry to point to beginning of file.



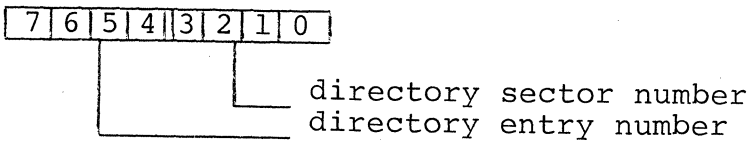
The cylinder address and cluster number, with an assumed sector number of zero, is the PDA of the first sector of the file.

44.1.5.3 Segment Descriptor - used in RIB to define a segment.



The cylinder address and cluster number, with an assumed sector number of zero, is the PDA of the first sector of the segment. The length of the segment in clusters is given by the low-order five bits of the lsb; length can be 1-32 clusters (except DOS.B, 1-10).

44.1.5.4 Physical File Numer - used to access directory and HDI



The directory sector number specifies a sector of the directory (0-15). The directory entry number (0-15) specifies an entry within a sector.

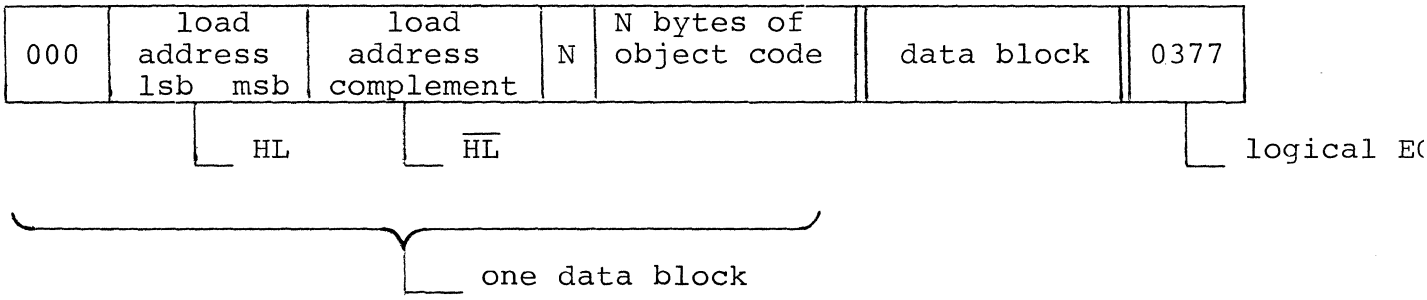
Note: Since directory entries are 020 bytes long, if the low-order four bits of the PFN are set to 0, the resulting value is the byte location of the beginning of the specified directory entry. For example, PFN 0304 references the directory entry beginning at byte 0300 (entry number 014) of sector 4 of the directory.

44.2 Disk Data Formats

The DOS itself does not deal with the user's data below the record level. It only keeps track of where the records are, allowing the user to format the data in any manner he pleases. The user is presented with records that are 253 bytes long, the first three bytes of each sector being reserved for system sector identification as described above. The DOS and its utility programs do make a number of assumptions concerning file structure however, and system operation is much simpler if all files are structured to match these assumptions.

DOS makes assumptions about the structure of text files and about absolute object code files. The structure expected for text files under DOS is described in the chapter on REFORMAT. Any file to be processed by the standard text-handling facilities of DOS must have the standard text format described.

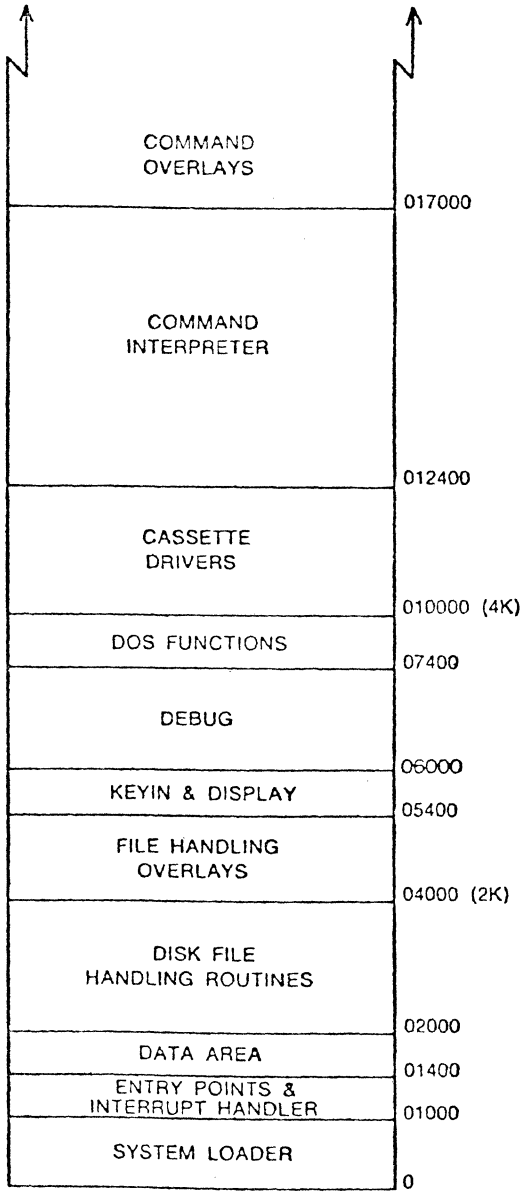
If a file is to be loaded by the system loader, it must be an object code file in the following format:



Note that this is the format of output files from Datapoint assemblers. Any number of data blocks may appear in a record. The leading byte of a data block will always be either 0, indicating a block follows, or 0377, indicating end of record. The special case of N being zero is used to indicate end of file, in which case the HL given is taken to be the starting address of the program loaded.

44.3 Memory Mapping

The DOS occupies memory as shown by the following map:



44.4 Memory Tables

44.4.1 Entry Point Tables

Three entry point tables exist within the DOS. These tables consist of a group of jumps to the various routines made available to the user. These jumps allow the system to be changed without requiring the user to modify his programs. To assure compatability between operating systems and for future versions of DOS, any calls to system routines should use the documented entry points only.

The first entry point table is located between 01000 and 01377. It contains entry points to the routines in the loader (the loader itself, the basic disk read and write drivers, and the interrupt handler) and to the DOS file handling routines. It also contains in-line routines to increment and decrement the HL registers.

The second entry point table is located between 010000 and 010066 and contains entry points to the cassette handling routines.

The third entry point table is located between 013400 and 013452 and contains entry points to routines within the command interpreter. The availability of the command interpreter routines makes small command tasks easy to implement.

See the chapters on System Routines and Routine Entry Points for details on the routine functions and entry point locations.

44.4.2 Logical File Table

The major working table in the system is called the Logical File Table (LFT) and is located from 01544 through 01643. It contains all of the information required by the file handling routines for every file which is currently open (a maximum of three files may be open at any one time - logical files one, two, and three). Once the user has opened a file by its symbolic name, he deals with it by the logical file number under which it was opened. The Logical File Number (LFN) specifies which LFT and which disk buffer memory page are to be used for a file.

The LFT contains for each entry the following information in the order shown (the number in parentheses is the number of bytes used for the item):

PFN	(1)	- Physical File Number, PFN of the file referenced by this LFT
PDN	(1)	- Logical Drive Number (bits 3 - 0) Protection (bit 7 set indicates delete protection, bit 6 set indicates write protection) New Space Allocated flag (bit 5) set if new space has been allocated to this file.
LRN	(2)	- Next Logical Record Number, system LRN of next sequential sector
BLRN	(2)	- Base LRN, first LRN in current segment (system LRN)
CSD	(2)	- Current Segment Descriptor The CSD and BLRN describe the current file segment and allow quick calculation of the PDA to be read/written by treating the LRN as an offset from BLRN. If the desired LRN is not in the current segment, the RIB is re-read and a new current segment established.
RIBCYL	(1)	- Physical Disk Address of RIB, MSB
RIBSEC	(1)	- Physical Disk Address of RIB, LSB Storing the RIB PDA allows quickly locating the RIB when it must be accessed for getting a new segment descriptor, for allocation updates, or for file closing.
MAXLRN	(2)	- Largest system LRN referenced (read, written, or positioned to) since the file was opened. Used for space deallocation at close if new space allocated flag is set.
LRNLIM	(2)	- Largest LRN allowed. Obsolete field, now unused.
BUFADR	(1)	- Current controller buffer byte address, used for byte transfers to or from the disk controller buffer.
XXXXXX	(1)	- Unused
	16	bytes total

There are actually four LFT entries (01544-01563, 01564-01603, 01604-01623, 01624-01643) to correspond to LFNs zero through three. The LFN used for a file specifies which buffer page to use for the disk transfer operation. LFN 0 uses buffer page zero (or 4, 8, or 12), LFN 1 uses buffer page one (or 5, 9, or 13), and so on. The larger buffer page numbers are available

on 4K disk controllers and are specified by the high-order bits of the LFN given to the system routine used. Not all routines recognize the page-select feature of LFN, check the description of each routine in System Routines.

Buffer page zero is a special case and is reserved for system use because the DOS needs a buffer into which it can read the RIB if it is necessary to determine a new current segment when a given access is made. This need is only critical on writes, when the buffer contains the information to be written to the disk. On reads, the user's data will always be the last item to be read and and page zero may be used. Always be careful in use of buffer page zero, however, since an access involving a different logical file may cause logical file zero's disk buffer to be loaded with a RIB. Also, the zeroth disk controller buffer is always used by the system loader in transferring data to memory. Page zero is used so that an overlay may be loaded or another program can be chained to without disturbing any of the standard (one through three) logical files. LFN zero has one final peculiarity, CLOSEs have no effect when issued on LFN zero. Neither space deallocation nor updating of the protection field occur when logical file zero is closed.

The DOS loader uses a set of locations in memory between 4 and 022 to perform the functions of an LFT entry during the loading process. It knows, however, that an object file is always sequential and does not have to have the accessing generalization of the main file handling routines. The file handling routines also use these low memory locations for temporary storage of a specified LFT entry to eliminate having to continually index into the LFT. Also, since the basic disk read and write routines use location 5 to indicate which drive is to be used, having the LFT temporarily stored in the low memory locations automatically selects the correct drive for use.

44.5 Disk Overlays

DOS uses disk overlays to reduce its main memory requirements. The overlays are in disk files SYSTEM1/SYS through SYSTEM7/SYS. The memory-resident DOS is stored in the disk file SYSTEM0/SYS. These eight files must reside in PFN's 0 through 7, the PFN corresponding to the number in the file name.

The system overlay files load into memory between 04000 and 05400 and are loaded by the system as needed. The functions of the overlays are:

```
SYSTEM1/SYS - PREP   - create a new file
SYSTEM2/SYS - CLOSE  - close a file
SYSTEM3/SYS - OPEN   - open an existing file
SYSTEM4/SYS - ALLOC  - allocate more space for a file
SYSTEM5/SYS - ABORT  - display an error message
SYSTEM6/SYS - SCREEN - initialize a RAM display screen
```

SYSTEM7/SYS is the DOS Fuction overlay and is described in the DOS Function section of the chapter on System Routines. The DOS Functions are short overlay routines and load into a separate area of memory. Also, the first sector of SYSTEM7/SYS is used to store subdirectory names (see the SUR command).

When DOS needs an overlay file, it searches for the file on the booted drive.

44.6 The Command Interpreter

The command interpreter resides in locations 013400 through 017777. The command interpreter receives command lines from the keyboard, as described in the chapter on Operator Commands, storing the command line in memory in the Monitor Communication Region (MCR$, location 01400 through 01543). When the line is terminated (ENTER key, 015), the stored command line is scanned and the indicated command program is loaded and executed.

While the command interpreter is waiting for character entry from the keyboard, it runs a test on the disk buffer memory. As soon as a character is ready from the keyboard, the disk buffer memory test is terminated and the normal keyin routine is entered. Even just sriking the CANCEL key will terminate the disk buffer memory test. If an error is detected by the disk buffer memory test, the message "DISK BUFFER FAULT" is displayed and the screen is rolled up one line.

When the command interpreter is intially entered via the entry point DOS$ it will execute the program set for auto-execution if there is one. If the KEYBOARD key is depressed, auto-execution is not performed. The autoed program will also be run any time the system returns to DOS$; exit routines EXIT$ and ERROR$ return via this entry point.

When a command line has been entered, the command interpreter

must attempt to locate and load the specified command program. If the command is obviously bad (a null entry line) the interpreter immediately displays "WHAT?" and waits for a new line. A pound sign (#) for invoking DEBUG is also treated as a special case, causing the interpreter to immediately go to DEBUG. Normally the first field on the command line will be normalized to the form shown below and the file thus specified will be searched for. The sequence of searching for a requested program depends on the format of the command line.

If the operator entered a leading "*" or ":" as part of the command name, a flag called UTILSW (UTILity SWitch) is set, incicating that the specified command is to be located as a member of UTILITY/SYS. If a drive specification was entered as part of the command name, the search goes only to the specified drive, as indicated in the sequence shown below.

The first test the interpreter performs is to check the drive specification entered, if any. If the drive specification is invalid, an error message is displayed and a new command line requested. If the drive specified is valid, or if no drive was specified, the interpreter searches for the command as outlined below.

1. If a drive was specified:
   a. If UTILSW is set:
      (1) Open UTILITY/SYS on the specified drive. If the file is missing or if the specified command is not a member, say "WHAT?", else run the program.
   b. If UTILSW is not set:
      (1) Attempt to open the command file on the specified drive. If successful, run the program. Else:
      (2) If no extension was specified in the command name, open UTILITY/SYS on the specified drive and search for the command as a member of the library. Else:
      (3) "WHAT?" and get another command.
2. If no drive was specified:
   a. If UTILSW is set:
      (1) Open UTILITY/SYS on the booted drive and search for the command as a member of the library. Else:
      (2) Try to open command as a file on booted drive. Else:
      (3) Check for command in UTILITY/SYS on any drive. Else:
      (4) Try to open comand as a file on any drive. Else:
      (5) "WHAT?" and get another command.

b. If UTILSW is not set:
   (1) Try to open command as a file on booted drive. Else:
   (2) If no extension was specified in the command name, open UTILITY/SYS on the booted drive and check for the command as a member of the library. Else:
   (3) Try to open command as a file on any drive. Else:
   (4) If no extension was specified in the command name, open UTILITY/SYS on any drive and check for the command as a member of the library. Else:
   (5) "WHAT?" and get another command.

The command interpreter uses lexical scanning routines to interpret the entered command line. These routines are available for user programs and are described in the chapter on System Routines. The command interpreter scans up to four file specifications from the command line. The file name scan is terminated by a semicolon (;) or end-of-string (015). The file specifications are entered in a normalized symbolic form into the corresponding logical file table entries (0 through 3). The normalized form is not the same as normal LFT information, the LFT area simply provides convenient storage for the file specifications. If desired (and it usually is), the open routine can open a file using the LFT in which the file name used for the open is stored. The format of the normalized form is shown here:

DRCODE	(1)	- Drive select code: logical drive number in binary, no drive spec (0377), invalid drive spec (0376)
0377	(1)	- PDN location of normal LFT, set to 0377 to indicate the LFT is closed.
FILENAME	(8)	- File name specified, padded with trailing spaces to 8 characters. Eight spaces if no name given.
FILEEXT	(3)	- File extension specified. Three spaces if no extension entered.
DRSPEC	(3)	- Logical drive specification (spaces if no spec).

When a program receives control from the command interpreter, LFT's one through three (zero was used to load the program itself) contain normalized entries as indicated above, and MCR$ still contains the command as entered, so the program can retrieve information from its command line. If a program is auto-executed, none of this command line information is available, so any program which tests for information as provided above can not be auto-executed. Conversely, any program intended for

auto-execution must not look for command information. The command AUTOKEY is provided to allow automatic execution of programs requiring command line information.

CHAPTER 45. INTERRUPT HANDLING

45.1 Interrupt Mechanism

Datapoint 1100, 2200, and 5500 processors feature a one-millisecond timed interrupt. Every millisecond, a flip-flop indicating "interrupt pending" is set; the setting of this flip-flop occurs independently of processor instruction cycling. At the beginning of an instruction fetch cycle the status of the interrupt pending flip-flop is checked. If the flip-flop is set, a CALL to the interrupt vector location occurs and the flip-flop is cleared. On 1100 and 2200 processors, the interrupt vector location is address 0. On 5500's the interrupt vector location is address 0167444, which normally immediately performs a jump to location 0. While interrupts are active, location zero is a jump to the interrupt scheduler.

The execution of the CALL ends hardware control of the interrupt. Any interrupt service performed, task scheduling, or prioritizing is under software control.

The machine instruction DI (Disable Interrupts) prohibits recognition of the interrupt pending flip-flop, thereby preventing any interrupt calls until recognition of the flip-flop is reactivated by an EI (Enable Interrupts) instruction.

45.2 Interrupt Scheduler

DOS provides an interrupt scheduler loaded as part of the system boot operation. The scheduler resides between 01201 and 01376 and remains memory-resident. Normal system operation never overstores this scheduler. The basic coding of the scheduler is shown below. The code shown is intended as an example of the structure of the scheduler and is not the exact code used within DOS.

```
INTRPT    DI                    Disable interrupts
          BETA                  Use BETA mode
INT0      CALL   RETURN         Perform each of processes 0
INT1      CALL   RETURN             through
INT2      CALL   RETURN               three
INT3      CALL   RETURN
          MLA    *INTSCN        Rotate to the next
          AD     6                  one of processes
          LMA                       4 through 7
          AD     INT4-6
          LLA                   HL => CALL address
          PUSH                  Jump to the
RETURN    RET                       next CALL
.
INTSCN    DC     0              Rotation counter storage
.
INT4      CALL   RETURN         CALLs for the rotating
          JMP    INTRET             process slots
INT5      CALL   RETURN
          JMP    INTRET
INT6      CALL   RETURN
          JMP    INTRET
INT7      CALL   RETURN
          XRA                   Reset the scan pointer
          MSA    *INTSCN            after calling process 7
INTRET    ALPHA                 back to ALPHA mode
          EI                    Enable interrupts
          RET                   Back to the background
```

All processing performed on an interrupt call is called "foreground", the processing interrupted is referred to as "background". Foreground processing begins with the DI instruction labeled INTRPT above and ends with the RET instruction terminating the scheduler. The above scheduler illustrates the fundamental rules of foreground code on a Datapoint processor:

1. Interrupts must be disabled during foreground. The scheduler disables interrupts initially and does not enable interrupts until immediately before terminating. Any foreground processes must not enable interrupts.

2. Foreground processing is performed in BETA mode, background processing in ALPHA mode. The scheduler sets the machine modes; foreground processes should not change the mode.

3. Foreground processes are CALLed routines and must return

with the stack in the same condition as on entry. Each CALL instruction INT0 through INT7 can be used to call a foreground routine. The scheduler itself uses a simple RET to return to background processing; if any foreground routine modified the stack, the scheduler could exit to the wrong location. Even if the scheduler manages to return properly, the background process uses the same stack (there is only one stack) and any modification performed by foreground routines could be fatal to the background processing.

4. Register contents on entry to foreground processing may be undefined. Normally the BETA mode registers and condition flags will be the same on entry to foreground as they were at the conclusion of foreground processing on the previous interrupt, so contents on entry can be considered as known. Under PS, however, the BETA mode registers and condition flags are not preserved, since they are used by PS and by the other partition. Even when PS is not in use, register contents cannot be predicted if there is a possibility of multiple foreground routines being active. If a single routine is active, registers may be preserved; if another routine or two is made active, they may modify the registers used by the first routine, effectively destroying any expected contents.

45.3 Active Processes

Each of the labeled CALLs in the scheduler, INT0 - INT7, is called a foreground "process" or, sometimes, an interrupt "slot", and is referenced by number 0 through 7.

Normally each foreground process is inactive, since each CALL invokes only a return to the scheduler. A process is made active by overstoring the address RETURN following the process CALL with the entry address of a desired foreground routine. The address so stored is called the "state" of the foreground process. (Two DOS routines, SETI$ and CS$, set the state of a process.) Thus if the address PRINT is stored lsb, msb in INT1+1 and INT1+2 (the address area following the CALL at INT1) the state of foreground process 1 is PRINT. Once a process has been made active -- given a state -- it can again be made inactive by storing the adress RETURN back into the two bytes following the CALL. (Two DOS routines, CLRI$ and TP$, terminate processes in this manner.) While a process is active, the routine it calls will be performed once every interrupt cycle, or every fourth cycle depending on the slot number used.

The scheduler is structured to provide four "one-millisecond" processes and four "four-millisecond" processes. The "one-millisecond" and "four- millisecond" designations refer to the length of time between sequential executions of the process. Interrupt slots 0-3 are one-millisecond processes; each process is executed every time an interrupt occurs. Interrupt slots 4-7 are four-millisecond processes; one of the four is executed every time an interrupt occurs, so any one process is executed only every fourth millisecond.

45.4 Timing Considerations

The greatest constraint on foreground routines is timing, mainly the total length of time required to execute. Since an interrupt occurs every millisecond, the total amount of time spent in foreground must be less than 1000 microseconds. Thus the amount of time spent executing each active foreground routine and the interrupt scheduler itself (130 microseconds on a 2200) must total less than a millisecond. If the time spent in foreground is more than a millisecond, the interrupt pending flag will already be set when the interrupt scheduler executes its final RET, so an interrupt call will immediately occur and no background processing will be performed.

Also, if more than one millisecond is spent in foreground, interrupts can be dropped. The interrupt pending flag has only off and on values. If an interrupt signal occurs and the flag was already set, it simply stays set and the occurrence of the interrupt pulse has no effect -- the interrupt is lost. If, for example, 1200 microseconds is being spent in foreground on each interrupt, only 5 interrupt calls will occur in a 6 millisecond time interval. One interrupt was lost because the flag was already set when its signal occurred. In a similar fashion, interrupts can be lost if interrupts are disabled for too long in background.

Another timing concern is "jitter". Jitter describes the variation in interrupt timing: it is not exactly one millisecond between interrupt calls. The timing variation occurs mainly because of time spent in background with interrupts disabled. If background processing disables interrupts for 200 microseconds (200 microseconds of jitter) it could be 1200 microseconds between interrupt calls if the interrupt pending flag were set immediately after interrupts became disabled. An additional source of jitter is time spent in foreground processes. Any variation in the execution time of process 0 appears as jitter to process 1.

Jitter must be taken into account when designing any program structure. If an external device is being serviced by interrupts and the device presents a character for input every 1.4 milliseconds, jitter must not exceed 400 microseconds. If the jitter were over 400 microseconds, a character could appear ready and then be overstored by the next character before an interrupt occurred to service the device. A good guideline is 200 microseconds maximum for the user's code.

45.5 DOS Interrupt Routines

DOS provides four utility routines for interrupt processing. Use of these routines simplifies interrupt process coding and helps assure DOS compatibility. For full descriptions of parameterization of these routines, see the chapter on System Routines.

45.5.1 SETI$

SETI$ changes the state of a foreground process. SETI$ is usable only from background and is generally used to initiate a previously inactive process. The routine accepts a specified address and stores the address following the CALL instruction of a specified interrupt slot. Even if the process was previously active, the new state is stored over the old state.

45.5.2 CLRI$

CLRI$ terminates a foreground process. The address of RETURN (see sample scheduler above) is stored following the CALL of the specified process number. Any routine active from that interrupt slot is then inactive. CLRI$ is used from background.

45.5.3 CS$

CS$, like SETI$, changes the state of a foreground process, but is used from foreground. A call to CS$ affects only the process performing the call. CS$ changes the state of the process to the address of the instruction following the "CALL CS$" and returns -- not to the invoking routine -- but to the interrupt scheduler. Due to the stack manipulations performed by CS$ it must be called only from the outermost stack level of a foreground routine; it must not be called from a routine called by the main routine. CS$ does not enable interrupts.

45.5.4 TP$

TP$ terminates a foreground process, like CLRI$, but is itself called from foreground. TP$ affects only the foreground process from which it is called, setting the state of that process to RETURN to deactivate the process, and returning to the scheduler. Like CS$, TP$ must be called only from the outermost stack level of a foreground routine. TP$ does not enable interrupts.

45.6 Programming Considerations

45.6.1 Background Code

If interrupt processing is to be used, the mainline program code must be written "interruptable" with the realization it may be interrupted anytime interrupts are not disabled. For most processing, no particular concern is necessary, since if the interrupt processes are coded correctly the stack, registers, and condition flags are unchanged after interrupt processing so the background code will never notice the interruption. Coding for I/O device handling is the most critical part of interruptable code, since during interrupt processing the selected I/O device can change.

Interrupts must be disabled any time the currently selected I/O device is critical: between addressing the device and testing status, between addressing the device and issuing a command, etc. At the same time, interrupts must not be disabled for too long a time, due to introducing excessive jitter or even dropping interrupts. It is especially important to be certain interrupts are enabled for at least one instruction cycle in any wait loop least interrupts be delayed for the duration of the loop.

If the background code uses BETA mode, interrupts must be disabled all the time BETA mode is in use. If an interrupt occurs while in BETA mode, the registers and condition codes will be modified by the scheduler and foreground routines and results on the background program could be disastrous. Background code should not generally use BETA mode.

All DOS utility routines are written completely interruptable and disable interrupts for a maximum of 200 microseconds. DOS routines return with interrupts enabled.

45.6.2 Foreground Code

Duration of foreground routines is of primary concern. If a routine is too long to execute in a single interrupt cycle, split its operation using CS$ or successive four-millisecond processes. Foreground routines should never use a wait loop; they should return, using the delay of background processing to wait for the next interrupt.

Additionally, foreground routines:

1. Must not enable interrupts.
2. Must exit with the stack in the same condition as on entry.
3. Must not use mode instructions.
4. Should not assume register conditions have been preserved.

Be sure to terminate foreground processes when they are no longer needed. A process left active uses up machine time. When a program finishes any active foreground processes remain active. These foreground processes at best slow down the system, and may cause CALLs to locations that have been overstored with other code, causing unpredictable results. Be sure to terminate all foreground processes at program exit.

DOS itself uses foreground processing in only a few instances: the cassette driver routines, the DEBUG P-counter display, and the delay function (DOSFUNC 8).

CHAPTER 46. SYSTEM ROUTINES

46.1 Parameterization

Parameters are passed to the subroutines through the registers. In the discussion of these parameters, the following abbreviations will be used:

LFN - Logical File Number times 16 (16, 32, or 48)
LRN - Logical Record Number (the user's LRN)
PFN - Physical File Number
LFT - Logical File Table

also:

Drive Number - indicates a logical drive number (0 through N, where N is the maximum number of logical drives supported by the DOS in use). In some routines, 0377 is used to indicate that all drives are to be checked.

Name- the address of a field containing exactly eleven bytes. The first eight bytes are the file name and the last three bytes are the file extension by command interpreter convention. The name characters may be any eight bit combinations except the first character must not be a 0377. The command interpreter requires that all characters be letters or digits.

46.2 Exit Conditions

If a routine fails to perform as expected, some indication must be made that the expected action did not occur. This indication is given by the condition flags in the processor being set in a special manner or by control being transferred to a trap location instead of returned via the subroutine mechanism. The 'Exit conditions' section of each subroutine description shows the register contents and condition flags of interest when the routine returns.

46.3 Error Handling

Minor errors are indicated by the Exit Condition of the routine called. Major errors cause a trap -- an automatic grab of program control by the operating system. The trap for each type of error transfers control to a specified location, which will display an appropriate error message.

Minor errors are always non-fatal; the program can test the Exit Conditions and determine what action to take. Major errors can be fatal or non-fatal. When a trap occurs, the system will simply display a message and restore itself, causing a fatal program error. Many major error traps can be intercepted by the program and given special treatment, as described in the section on TRAP$ below.

46.4 Foreground Routines

The chapter on Interrupt Handling contains a complete discussion on the functioning and use of the foreground handling and should be consulted for an understanding of the following routines.

46.4.1 CS$ - Change Process State

CS$ changes a foreground routine's state. It is called by the executing foreground routine and causes its execution address to be changed to the address following the CALL CS$. Execution will not continue at the new address until the next interrupt occurs. CS$ is normally called from the outermost stack level (level 0) of an active foreground process. Calls to CS$ from deeper stack levels of the routine must be very carefully planned and are not recommended.

Entry point: 01033

Parameters: on subroutine stack - see the Chapter on Interupt Handling

Exit conditions: return is made to the scheduler

46.4.2 TP$ - Terminate Process

TP$ deactivates the process called by storing the address of a return instruction in the process call. TP$ is jumped to, not called. TP$ is invoked from the outermost stack level (level 0) of an active foreground process.

Entry point: 01036

Parameters: on the stack - see the Chapter on Interrupt Handling

Exit conditions: no exit, return to Interrupt Scheduler

46.4.3 SETI$ - Initiate Foreground Process

SETI$ activates the interrupt process specified by the number in the C register (0-7) by storing the address given in the D and E register into the CALL instruction for that process and enables the interrupt handler (stores JMP INTRPT in location 0). Interrupt processes zero through three are executed every millisecond while four through seven are executed every fourth millisecond.

Entry point: 01041

Parameters: C = process number (0-7)
DE = address of foreground process

Exit conditions: B,D,E unchanged
H,L = 0

46.4.4 CLRI$ - Terminate Foreground Process

CLRI$ deactivates a foreground process by storing the address of a return instruction into the process call specified by the number in the C register (0-7) and enables the interrupt handler (stores JMP INTRPT in location 0).

Entry point: 01044

Parameters: C = process number (0-7)

Exit conditions: B unchanged
H,L = 0

46.5 Loader Routines

There are two levels of disk handling routines. This section describes the lower level routines which reside in the loader and require numbers physically describing the drive, cylinder, sector, buffer, and file. The section on File Handling Routines describes the upper level routines.

INCHL and DECHL are described in this section only because they are used by the DOS at all levels and because these two routines are loaded as part of the bootblock. In general, the other routines described in this section are not used by typical user programs; most user programs will be better served by the higher level routines described in the section on File Handling Routines.

46.5.1 BOOT$ - Reload the Operating System

BOOT$ loads and executes the operating system (PFN 0 on the booted drive). This action does not affect the interrupt handling facility between 01000 and 01377. Since BOOT$ requires that the operating system always be loaded from specifically the booted drive, BOOT$ should normally only be used in cases where EXIT$ is unusable, for example if the disk handling routines have been overstored. BOOT$ does not close any files before reloading the DOS.

Entry point: 01000

Parameters: none

Exit conditions: does not return

46.5.2 RUNX$ - Load and Run a File by Number

RUNX$ loads the physical file specified and begins its execution. If the file cannot be loaded, a jump to BOOT$ occurs.

Entry point: 01003

Parameters: A = PFN
C = Drive Number

Exit conditions: does not return

46.5.3 LOADX$ - Load a File by Number

LOADX$ loads the physical file specified and returns with the starting address in HL if the load was successful.

Entry point: 01006

Parameters: A = PFN
C = Drive Number

Exit conditions: Carry false: HL = Starting address of file
Carry true: A=0 if file does not exist
1 if drive off line
2 if directory parity fault
3 if RIB parity fault
4 if file parity fault
5 if off end of physical file
6 if record of illegal format

46.5.4 INCHL - Increment the H and L Registers

INCHL increments the sixteen bit value in the HL registers by one. If the routine is entered at INCHL+2, the sixteen bit value in the HL registers will be incremented by the number in the A register.

Entry point: 01011 (01013 for increment by A)

Parameters: HL = number to be incremented
A = increment value if INCHL+2 used

Exit conditions: HL incremented
A equal to the H-register
B,C,D,E unchanged
CARRY condition undefined

46.5.5 DECHL - Decrement the H and L Registers

DECHL decrements the sixteen bit value in the HL registers by one. If the routine is entered at DECHL+2, the sixteen bit value in the HL registers will be decremented by negative the number in the A register (e.g., for decrement by 2, A is set to -2).

Entry point: 01022 (01024 for decrement by -A)

Parameters: HL = number to be decremented

A = decrement value if DECHL+2 used

Exit conditions: HL decremented
A equal to the H-register
B,C,D,E unchanged

46.5.6 GETNCH - Get the Next Disk Buffer Byte

GETNCH gets the character from the physical disk buffer location pointed to by low memory location DOSPTR (location 026) from the disk buffer currently selected, and then increments the contents of the location DOSPTR.

Entry point: 01047

Parameters: DOSPTR = disk buffer address (0-255)

Exit conditions: A = character from disk buffer
(DOSPTR) = (DOSPTR)+1
B,C,D,E,H,L all unchanged

46.5.7 DR$ - Read a Sector into the Disk Buffer

DR$ causes a sector to be transferred from the disk to one of the disk controller buffers. The drive number is given in the least significant bits (the others are ignored) of location TFT+PDN (5). (The number of bits ignored depends upon the particular DOS in use). The physical disk address (LSB) is given in the E register and the physical disk address (MSB) is given in the D register. The disk controller buffer number times sixteen is given in the B register. Interrupts are disabled by this routine a maximum of 100 microseconds.

Compatibility note: Here the user should be reminded that the physical disk address format will vary; the user's program should <u>not</u> make assumptions regarding this format if the program is to be transportable between different DOS. The most significant byte is generally a cylinder number, and the least significant byte is a sector address within a cylinder. This least significant byte especially will vary among DOS. In general, the only safe way to insure a valid, proper physical disk address (PDA) is to get it as a returned item from a system routine (POSIT$ or one of the DOS FUNCTIONs, to be described later). User program generation of or manipulation of physical disk addresses is strongly discouraged.

DR$ tries between four and ten times to read a record (depending upon the disk drive type in use), if parity faults are detected, before giving an abnormal exit status. Note that since this routine is used by all of the higher level routines, all disk reads performed by the disk operating system try to read a record that shows parity problems that same number of times before giving up.

Entry point: 01052

Parameter: B = 16 times buffer number
D = physical disk address (MSB)
E = physical disk address (LSB)
TFT+PDN (at loc 5) = logical drive number

Exit conditions: B,D,E,TFT & PDN all unchanged
Carry false if read successful
Carry true and Zero false if drive off line
Carry true and Zero true if parity fault

46.5.8 DW$ - Write a Sector from the Disk Buffer

DW$ causes the contents of one of the disk controller buffers to be transferred to a sector on the disk. If the write protection on the specified drive is enabled, DW$ will beep continuously until the protection is disabled.

There are two types of write protection in the disk operating system. The first type is a physical protection that is part of the disk drive hardware which will cause DW$ to beep if set. The second type of write protection is a logical protection that is connected with each file on a disk. A bit exists in the directory entry for each file which, if set, will prevent the higher level routines (for example, WRITE$) from calling the DW$ routine. It is important not to confuse these two types of write protection. All references to write protection that follow refer to the logical protection on each file and not to the physical protection on the drive itself.

In DOS, DW$ uses the write/verify mode of the disk controller. This implies that all writes made by these disk operating systems use this mode of writing. As in the DR$ routine, several tries will be made if parity faults occur before abnormal exit will occur. In all other respects, DW$ is similar to DR$.

Entry point: 01055

Parameters: B = 16 times buffer number
D = physical disk address (MSB)
E = physical disk address (LSB)
TFT+PDN (at loc 5) = drive number

Exit conditions: B,D,E,TFT & PDN unchanged
Carry false if write successful
Carry true and Zero false if drive off line
Carry true and Zero true if parity fault

46.5.9 DSKWAT - Wait for Disk Ready

DSKWAT waits for disk ready, controller ready, no disk I/O transfer in progress, and drive online to all be true. If the drive is not online, return is made with the carry flag true, the zero flag false, and interrupts enabled. Otherwise, exit is made with interrupts disabled. This routine is obsolete and is not available under some systems (e.g. PS). Therefore, it should no longer be used.

Entry point: 01060

Parameters: none (drive checked is the selected drive)

Exit conditions: explained above
B,C,D,E,H,L unchanged

46.6 File Handling Routines

A file is dealt with as a logically contiguous and randomly accessible space. The file being used is specified by its symbolic name or by its PFN. The LRN being dealt with within that file is determined by a two-byte number kept within the system (LRN in the LFT). When a file is opened, the LRN is set to two.

Always note the distinction between SYSTEM LRN and USER LRN. The LRN in the LFT is the system LRN. System LRN zero is the primary RIB for the file and system LRN one is the RIB backup. System LRN two is user data sector zero. All logical record numbers supplied to system routines (e.g. POSIT$) are user logical record numbers. These numbers are converted to system logical record numbers before being used by the DOS or placed into the LFT. In the routine descriptions below, "LRN" refers to the user LRN unless otherwise specified.

After each record access (READ$ or WRITE$), LRN is

incremented. Thus, for sequential accesses, the user does not actually specify which record he is dealing with. However, a routine named POSIT$ allows the LRN to be changed to any value between zero and the upper limit on the file, providing a random access facility. (This upper limit depends upon the DOS in use).

All of the logical file handling routines automatically create or verify the PFN and LRN of the file sector being handled (see Disk Structure).

It must be noted that READ$ and WRITE$ provide sequential processing of file sectors, but do not automatically handle the Datapoint sequential text file format. All necessary end-of-record (015) and end-of-sector (003) bytes must be placed in the disk buffer under program control; the system routines do not provide these control bytes. Likewise, the CLOSE$ routine does not provide any end-of-file mark. To provide a valid text EOF, the user program must write an EOF byte by byte. For Datapoint file formats see the chapter on REFORMAT for text file format and the section on Disk Data Formats for object code format.

46.6.1 PREP$ - Open or Create a File

PREP$ searches the directory or directories specified for the given name. If the name is found, the file is simply opened for use as the specified logical file number. Otherwise, a new file having the name specified will be created. If a new file is created, an end of file by GEDIT convention (six zeros followed by an 003) is written in logical record zero. Whether the file is simply opened or is created, the information describing it is stored in the LFT entry specified so that all subsequent references to that file by its LFN will be able to deal with the correct locations on the disk. If the LFT entry specified is already in use when PREP$ is called, the file that the entry specifies will be closed (see the section on CLOSE$) and then the new file opened in its place.

DE is the address of an 11-byte string which is the name of the file being specified (as explained before under the section on PARAMETERIZATION).

Entry point: 01063

Parameters: B = LFN
C = Drive Number or 0377
DE= Points to name string

Exit conditions: B = LFN; other registers indeterminate

Traps:

SPACE	if a new file must be created and no space is left or no more directory entries are available
OFF-LINE	If the DRIVE specified is off-line.

46.6.2 OPEN$ - Open an Existing File

OPEN$ is similar to PREP$ except for the action taken if the file specified does not exist. In this case, return is made with the Carry condition true (return is made with it false if the file exists). In addition, a file may be opened by PFN. To specify the PFN of the desired file, set the D register to zero and load E with the PFN. Action taken to open by PFN is the same as that taken if a name is specified.

Entry point: 01066

Parameters: B = LFN
C = Drive number or 0377
DE = Pointer to file name or
D = 0,E = PFN

Exit conditions: B = LFN; other registers indeterminate
Carry true if the file is non-existent

Traps: none

46.6.3 LOAD$ - Load a File

LOAD$ opens the specified file as logical file zero and then calls the system loader to load it into memory. Exit is made with the Carry condition set if the file is non-existent, or if the drive specified (if any) is off line. If the load is successful, return is made with the starting address in the H and L registers.

Entry point: 01071

Parameters: same as for OPEN$ (except B not required)

Exit conditions: B = LFN (always zero)

HL = starting address if good load
Carry true if file non-existent or drive off-line

Traps:

OFFLIN	drive went off line after loading began
RPARIT	file contains parity fault
RANGE	loader ran off end of file
FORMAT	record of bad loader format

46.6.4 RUN$ - Load and Run a File

RUN$ opens the specified file as logical file zero and then calls the system loader to load it into memory. Return is made to following the call if the name specified cannot be found in the directory or directories specified. If any loading errors occur, the operating system is reloaded. Otherwise, control is transferred to the starting address given by the loader.

Entry point: 01074

Parameters: same as for OPEN$
(except that B is not required)

Exit conditions: returns if name not in directory
operating system reloaded if bad load
otherwise, control is passed to the
starting address of the new file.

Traps: none

46.6.5 CLOSE$ - Close a File

When new space is allocated for a file, a large contiguous piece (up to one full segment) is taken in an effort to keep the file as physically contiguous as possible. When this allocation takes place, a flag in the LFT, called the new space allocated flag, is set. The LFT also contains a number which is the largest LRN referenced while the file was open. When CLOSE$ is called, the file is physically truncated after the largest LRN referenced, if the new space allocated flag is set. Thus, if only a few records of the new space allocated have been used, the rest of the space is freed for use in other files. However, if all of the space is used, the file will consist of a large amount of physically contiguous space. Note that if CHOP$ was called with the D register set to -1 (0377), and the LRN in the LFT has not been changed, a call to CLOSE$ will delete the entire file and remove its entry from the directory.

After the file has been truncated, if necessary, CLOSE$ then writes the copies of the protection bits and old file length limit field that are in LFT entry back into the directory. Therefore, one only needs to change these entries in the LFT and then close the file to have them changed in the directory. This is the basis for the functioning of the CHOP$ and PROTE$ routines. Since the protection bits and old file length limit field are not changed on the disk until the CLOSE$ routine is called, if one changes these numbers and then, for some reason, reloads the system without calling the CLOSE$ routine (by depressing RESTART before the file is closed, for example) the disk will retain the old values.

NOTE: If you want to de-allocate file space (CHOP$) following a protection change (PROTE$), you must CLOSE and re-OPEN the file.

```
          DE      NAME                FILE NAME
          LC      -1                  DRIVE
          CALL    OPEN$               OPEN THE FILE
          LC      1                   CHANGE PROTECTION
          CALL    PROTE$              CHANGE PROTECTION
          CALL    CLOSE$              NOW, SET THE PROTECTION
          DE      NAME                RE-OPEN THE FILE
          LC      -1
          CALL    OPEN$
          DE      -1                  CHOP THE FILE
          CALL    CHOP$
          CALL    CLOSE$              AND DELETE IT.
.
.
.
NAME      DC      'SCRATCH TXT'       FILE NAME TO BE DELETED
.
.
.
```

After the protection and file length limit have been stored in the directory, CLOSE$ then vacates the LFT entry specified by storing an 0377 in the second byte of the entry (this is the drive number and 0377 denotes that the LFT entry is not in use). CLOSE$ simply returns if the LFT entry is not in use.

Entry point: 01077

Parameters: B = LFN (16,32,48; 0 => NOP)

Exit conditions: B = LFN; other registers indeterminate

Traps: none

46.6.6 CHOP$ - Delete Space in a File

CHOP$ sets the LFT entry to deallocate file space following the given LRN. If the CLOSE$ routine is called after the call to CHOP$ without the LRN being changed, the space after the specified LRN will be physically deleted from the file, making it free again for allocation by the system. Note that if the D register is set to -1 (0377) upon entry to CHOP$, calling the CLOSE$ routine will completely delete the file from the system (removing its entry from the directory as well as freeing all of its space). When an entry is deleted from the directory, all sixteen bytes of the directory for that entry are set to 0377 (value set by the system generation program for unused directory entries).

Remember that calling CHOP$ only affects the LFT entry and that no physical change on the file is effected until CLOSE$ is called.

Entry point: 01102

Parameters: B = LFN
DE = LRN if D not 0377
D = -1 (0377) to delete entire file

Exit conditions: B = LFN; other registers indeterminate

Traps:

RANGE	DE not less than MAXLRN referenced
DVIOLA	delete protection is set
WVIOLA	write protection is set

46.6.7 PROTE$ - Change the Protection on a File

PROTE$ changes the file protection bit and/or upper file length limit copies that are kept in the LFT. The protection bits, given in the C register, are changed only if the least significant bit of the C register is a one. The old upper file length limit field is changed only if the sign bit of D is one on entry. Therefore, setting the number to zero prevents the limit field from being changed. Note that the file length field is obsolete and is no longer used by the DOS; it is maintained for future use.

Remember that calling PROTE$ only affects the LFT entry and

that no physical change on the file is effected until CLOSE$ is called.

Entry point: 01105

Parameters:
```
B = LFN
C = new protection:
          C0 = 1 for protection change
          C6 = 1 for write protection
          C7 = 1 for delete protection
DE = new LRN limit field; 0 for no change
```

Exit conditions: B = LFN; other registers indeterminate

Traps: none

46.6.8 POSIT$ - Position to a Record within a File

POSIT$ positions the file logically to the user LRN given. If the user LRN given is -1, the current value in the LFT is used for positioning the head and the LFT entry is not changed. Note that positioning to user LRN zero performs a logical 'rewind' of sequential files.

Entry point: 01110

Parameters:
```
B  = LFN
DE = LRN (use LRN from LFT if DE = -1)
```

Exit conditions:
```
B = LFN
D = Physical Disk Address (MSB)
E = Physical Disk Address (LSB)
ZERO FALSE:  DE are valid, position was valid
ZERO TRUE:  DE are invalid, specified sector not
              in allocated space
other registers indeterminate
```

Traps: none

46.6.9 READ$ - Read a Record into the Buffer

READ$ causes the record pointed to by the LRN in the LFT entry specified by the LFN given, to be transferred from the disk to the disk controller buffer that corresponds to the LFN given. The LRN is incremented by one after the read if it was successful. READ$ perorms four to ten retries, if a parity fault is detected before giving a parity trap. Attempting to read a record that is

not physically allocated will cause the 'RANGE' trap.

Entry point: 01113

Parameters: B = LFN

Exit conditions: B = LFN; other registers indeterminate
LRN = LRN + 1 if successful

Traps:

RANGE	LRN out of range
RPARIT	record unreadable
FORMAT	PFN or LRN in record incorrect
OFFLIN	drive off line

46.6.10 WRITE$ - Write a Record from the Buffer

WRITE$ first takes the PFN and LRN values from the appropriate LFT and stores them into the first three bytes of the disk controller buffer that corresponds to the LFN given. It then transfers that buffer to the disk sector specified by the LRN. The LRN is incremented after the write if it is successful. Note that all system routines use DW$ in writing records and hence try up to ten times to obtain a good write, if a parity fault is detected, before giving the trap.

If WRITE$ tries to write a record beyond the space already allocated to the file, it will automatically try to allocate more space. If the space is available, it is allocated and the write occurs. If there is no more physical space on the disk or if there are no more entries in the RIB available for the new segment descriptor, a 'SPACE' trap is given.

Entry point: 01116

Parameters: B = LFN

Exit conditions: B = LFN; other registers indeterminate
LRN = LRN + 1 if successful

Traps:

WVIOLA	file is write protected
WPARIT	write/verify failure
OFFLIN	drive off line
RANGE	LRN < 0
SPACE	explained above

46.6.11 GET$ - Get the Next Buffer Character

The LFT contains an entry called BUFADR (not to be confused with loc. 026 used by GETNCH) which points to a character in the disk controller buffer that corresponds to the given LFN. Each buffer contains 256 characters but since the system uses the first three bytes in each sector to store the PFN and the LRN of each record, the user has only 253 bytes available.

Whenever READ$, WRITE$, or POSIT$ are executed, they set BUFADR to point to the third byte in the disk controller buffer (by setting the BUFADR field of the LFT entry to a three). Whenever GET$ is called, the byte pointed to by this pointer is fetched from the disk controller buffer and the pointer is incremented. If the byte being returned is not a valid user data byte (i.e. BUFADR was 0,1,or 2 on entry) then carry is true on return, and register A contains the specified byte of the buffer (which will be PFN or one of the LRN bytes.) Note that the next buffer is not read automatically from the disk; the pointer simply ends-around. Upon the first call of GET$ which returns carry true, the PFN will be obtained since it is contained in buffer location zero. A byte may also be accessed by simply setting BUFADR to the desired location.

Entry point: 01121

Parameters: B = LFN

Exit conditions: A = the byte obtained from the buffer
All other registers preserved
Carry true if location 0,1,or 2 accessed

Traps: none

46.6.12 GETR$ - Get an Indexed Buffer Character

GETR$ is similar to GET$ except that it uses the logical buffer address supplied in the C register instead of the physical buffer address in the LFT for the address of the disk buffer byte to return. Calling GETR$ has no effect on the buffer pointer kept in the LFT. The physical buffer location is obtained by adding three to the value given in the C register to skip past the system data in the first three bytes in the disk buffer. Thus the user is presented with a logical space within a record that is addressed from 0 through 252. Normally, GETR$ exits with the value in the C register incremented by one and the carry condition false. However, if the C register is between 253 and 255

(inclusive) upon entry, it will not be incremented and exit will be made with the carry condition true. In either case, the buffer byte located by the C register value plus three is returned in the A register. Therefore, the user may obtain any buffer byte with GETR$ but must remember to supply an address which is the physical buffer address minus three and remember not to assume that the C register will be incremented if he plans to access one of the first three physical bytes.

Entry point: 01124

Parameters: B = LFN
C = buffer location

Exit conditions: A = byte obtained
C = C + 1 if carry false
Carry true if 252 < C < 256
All other registers preserved

Traps: none

46.6.13 PUT$ - Store into the Next Buffer Position

PUT$ is similar to GET$ except that the byte presented in the A register on entry is stored into the buffer. Also, on return register A contains the physical address of the next byte to be accessed in the disk buffer. Carry is true if the byte stored was stored into the last physical location in the buffer. Here a reminder is appropriate: remember that in standard, EDIT-format records, the last two bytes (at least) of the buffer are not used, and an 03 occurring earlier in the sector indicates logical-end-of-sector. (A complete description of the format for DOS text files can be found in the chapter describing the REFORMAT command.)

Entry point: 01127

Parameters: A = the byte to be stored in the buffer
B = LFN

Exit conditions: A as described above (physical address of next byte)
All other registers preserved
Carry true if location 255 was stored into

Traps: none

46.6.14 PUTR$ - Store into an Indexed Buffer Position

PUTR$ is identical to GETR$ except that the byte presented in the A register is stored into the buffer.

Entry point: 01132

Parameters:
A = byte to be written
B = LFN
C = logical buffer location

Exit conditions:
C = C + 1 if carry false
Carry true if 252 < C < 256
All other registers preserved

Traps: none

46.6.15 BSP$ - Backspace One Physical Sector

BSP$ decrements the LRN in the LFT entry specified by the LFN given and then executes POSIT$. No check is made to prevent BSP$ from backing into a RIB. However, if one calls BSP$ and attempts to backspace back beyond system LRN 0 (user LRN -2, which is the master RIB) ZERO TRUE will be returned (as for POSIT$).

Entry point: 01135

Parameters: B = LFN

Exit conditions:
B = LFN; other registers indeterminate
ZERO FALSE: valid backspace
ZERO TRUE: invalid backspace (attempt to backspace past master RIB)

Traps: none

46.6.16 BLKTFR - Transfer a Block of Memory

BLKTFR moves the number of bytes specified in the C register (0 causes transfer of 256 bytes) from the memory location starting where HL points to the memory location starting where DE points. Note that since exit is made with HL and DE pointing after the last byte moved and C equal to zero, transfers of more than 256 bytes may be made by first setting C to zero, calling BLKTFR enough times to make the residual number of bytes to transfer less than 256, setting C to the residual number of bytes to be

transferred, and then calling BLKTFR one last time. For example:

```
HL      SOURCE
DE      DEST
LC      0
CALL    BLKTFR
CALL    BLKTFR
LC      25
CALL    BLKTFR
```

will cause 537 bytes to be transferred from SOURCE to DEST.

Entry point: 01143

Parameters: C = number of bytes to be moved
(0 moves 256 bytes)
HL = source address
DE = destination address

Exit conditions: HL = HL + C (HL + 256 if C = 0)
DE = DE + C (DE + 256 if C = 0)
B = unchanged
C = zero

Traps: none

46.6.17 TRAP$ - Set an Error Condition Trap

There are eight non-fatal error conditions, concerning the disk operating system file handling facilities, that may be trapped by the user. If the trap corresponding to a certain error is not set by this routine, the system displays a pertinent message and reloads the system. If the trap is set, control is transferred to the address specified when the trap was set, with the subroutine return address stack in the state it had before the calling of the file handling routine that caused the error condition.

The only disk errors that cannot be trapped are those associated with the system tables on the disk. The occurrence of these errors causes the message

FAILURE IN SYSTEM DATA

to be displayed. The other errors that cannot be trapped have to do with: the LFT entry not being open when a routine which tried to use data from the entry was called, invalid logical file

numbers, invalid drive numbers, invalid trap numbers, and invalid physical file numbers.

If a trap occurs during a call to READ$ or WRITE$, the logical record number (LRN) in the logical file table (LFT) is NOT incremented; if the user wishes to continue processing records past the one which caused the trap, he must increment the LRN in the LFT himself first.

TRAP$ sets the trap whose number is given in the C register to the address supplied in the D register (MSB) and E register (LSB). The trap is cleared by calling TRAP$ with D and E equal to zero. The trap is also cleared when the error condition occurs, at which time the B register will be loaded with the Logical File Number involved and control transferred to the indicated address.

In the following table, the mnemonic given after the trap number is the one used in the previous routine explanations. The capitalized lines are the messages displayed if the trap is not set.

0 - RPARIT - PARITY FAILURE DURING READ
A parity fault while reading a data record causes this trap.

1 - WPARIT - PARITY FAILURE DURING WRITE
A parity fault while writing a data record causes this trap.

2 - FORMAT - RECORD FORMAT ERROR
The physical file number or logical record number in the record read not matching the ones contained in the logical file table entry causes this trap. The physical position of a record is obtained from information in the retrieval information block and the PFN and LRN in the record are only checked to ensure that the drive is functioning correctly and that the user is not trying to read a record he has not written. This trap has nothing to do with the 253 data bytes provided to the user.

3 - RANGE - RECORD NUMBER OUT OF RANGE
During a read, an access below zero or to a record above the currently allocated space causes this trap. During a write, an access below zero causes this trap.

4 - WVIOLA - WRITE PROTECT VIOLATION
An attempt to write on, delete, or shorten a file with the write protection bit set causes this trap.

5 - DVIOLA - DELETE PROTECT VIOLATION
An attempt to delete or shorten a file with the delete protection bit set causes this trap.

6 - SPACE - FILE SPACE FULL
An attempt to allocate more space when either the disk is full or no more segment descriptor slots in the RIB are available causes this trap.

7 - OFFLIN - DRIVE OFF LINE
An attempt to use a drive that is either physically absent or not online causes this trap.

Note that the causes given for the various traps are the causes for DOS to issue the appropriate messages. Some of the DOS Command programs also cause the issuance of some of these messages for related reasons. For example, several DOS Utilities indicate a RECORD FORMAT ERROR if the sector formatting of a file being processed does not follow GEDIT (or DOS EDITOR) standards. In such cases the above details are sometimes not valid descriptions of the problem; in this example the 253 data bytes encountered may be the cause of the record format error.

Note also that FORMAT and RANGE traps are frequently the result of sequentially reading or otherwise processing a file which has no valid EOF, resulting in the program running off the logical end of the file.

Entry point: 01146

Parameters: DE = trap address
C = trap number

Exit conditions: register contents indeterminate

Traps: none

46.6.18 EXIT$ - Reload the Operating System

EXIT$ closes any logical files (one through three) that are open and then reloads the operating system. EXIT$ is the normal exit for all DOS programs.

If MCR$ (01400) contains exactly two forward arrows ">>" followed by a command line, followed by an 015, the command interpreter will be reloaded, and the command line in MCR$ is

scanned and executed. This technique is useful if your program started at a location lower than 017000, and you want to execute another program after your program completes. For example, after your program completes, you may want to invoke the DOS. CHAIN command to perform additional processing on your data file. Before ending your program, move a character string to MCR$ like this:

```
>>CHAIN PROC1(015)
```

then, JMP EXIT$.

Entry point: 01151

Parameters: none

Exit conditions: no exit

Traps: none

46.6.19 ERROR$ -- Reload the Operating System

ERROR$ is identical to EXIT$ in all respects except for the fact that jumping to ERROR$ will abort an active CHAIN (refer to the CHAIN command in this manual for more details). A user program would exit through ERROR$ if an error of severity suggesting aborting a CHAIN occurred.

Entry point: 01140

Parameters: none

Exit conditions: no exit

Traps: none

46.6.20 WAIT$ -- DOS Wait-a-While "NOP" Routine

This routine, after being called, returns with all registers, condition codes, and the stack preserved; in effect a "NOP". Normally, the return is immediate. This routine should be used in loops which wait for time non-critical conditions to occur (e.g. waiting for the keyboard operator to release the DISPLAY key). I/O status, including in particular the device addressed, is subject to change on return.

Entry point: 01170

Parameters: none

Exit Conditions: Registers and condition codes unchanged

Traps: None

46.7 Keyboard and Display Routines

46.7.1 DEBUG$ - Enter the Debugging Tool

The debugging tool enables the programmer to load files by number, examine and modify memory locations, set break points, and execute sections of his program. This facility greatly simplifies the task of debugging machine language programs.

The debugging tool can be entered from the command interpreter by entering a single pound sign (#) on the command line or from the user's program by jumping to the entry point. When debug is executing, two numbers are displayed vertically in the last column of the screen. The five-digit top number is an address and the three-digit bottom number is the content of that address. After these numbers are displayed, input is requested from the keyboard as indicated by a flashing cursor. Commands to the debugger are given in the form <n>X where <n> is any number of octal digits and X is a command character. The command is executed immediately upon depression of the command character key without waiting for the ENTER key (the ENTER character is a command in itself).

All keys that are not recognized are ignored with a beep signaling the rejection. The BACKSPACE key is ignored but since commands use only the lower eight or sixteen bits of <n>, errors in the entry of numbers can be corrected by striking several zeros and then entering the correct digits. Alternatively, the CANCEL key causes the current input line to be erased without changing the current address. Although display stops if the cursor runs off the screen during input, characters are still accepted.

The debugger maintains a current address that is usually displayed as the five digit number at the right of the screen. There are times, however, when the five digits at the right of the screen do not reflect the current address and caution must be exercised to avoid confusion as to the value of the current

address. The ENTER key is normally used to change the current address, but depressing it without preceding it with any digits will cause the current address to be displayed. Therefore, if there is any doubt about the number being displayed on the screen, simply depressing the ENTER key will ensure that the current address is being displayed.

Whenever the debugger is entered either from a jump to the entry point or from a return from a break point or call command, a beep is given and the state of all of the alpha mode registers and condition flags is saved. The value initially displayed is the top of the stack at entry, unless DEBUG was entered from a DOS DEBUG breakpoint; in this case the address displayed is the address where the breakpoint was set. In all cases, the stack is preserved as at entry and the current address is set to the address displayed at entry. This display enables the user to tell exactly the state of his program when the debugger was entered. Whenever a memory location is called or jumped to, the state of all of the alpha mode registers and condition flags is restored from the values saved at entry. Since these values are saved in memory, the programmer can simply modify these locations to change the values used to initialize the state of the alpha machine before control is transferred.

The major debugging technique is the setting of break points at critical places in the program and the execution of portions of the program while checking the values of the registers and critical memory locations at each break. The debugger sets a break point by storing a jump instruction, to a special entry point in itself, in the current address and the following two locations. (Notice that setting break points less than three bytes apart is therefore not a good idea.) Before the jump is stored, the content of the memory locations to be used is saved in a table in the debugger. When the break point is reached, the memory locations are restored with their original contents. A maximum of four break points may be active at any one time. A command is provided for insuring that all break points have been restored. When a break point is executed, the current address is set to the first byte of the break point jump instruction. Since the J command causes a jump to the current address if no digits precede it, one can continue execution of the routine that was broken by simply depressing the J key. Execution will continue with the first byte that was overstored by the break point jump with the state of the alpha machine exactly like it was before the break occurred. Thus, the programmer can set a break point, start execution, examine the registers when the break occurs (since register viewing does not change the current address) and then depress the J key to continue execution. This technique allows

him to practically single step his program.

ENTRY POINT: 01154

COMMANDS:

B - Set a break point at the location given or, if no number is given, at the current address. Caution should be exercised to insure that the current address is pointing to the desired location if it is used.

C - Execute a call to the number given or, if no number is given, to the current address. The alpha machine state is loaded from the values saved in the debugger before the call is executed. A return to the call causes the debugger to be re-entered and the alpha machine state to be saved.

D - Decrement the current address (any digits given are ignored).

G - Get the physical file specified from the disk. Care must be exercised that a file is not loaded that will overlay the debugger (locations 0-01377 and 06000-07377). If the file does not exist or contains a record of illegal loader format, a beep will be given. The first digit of the last four entered is the logical drive number from which the file is to be loaded. The following three digits are the physical file number. For example, 02003G will load SYSTEM3/SYS from drive two. To load PFN 0115 from drive 0, simply enter 115G.

I - Increment the current address (any digits given are ignored).

J - Execute a jump to the number given or, if no number is given, to the current address. The alpha machine state is loaded from the values saved in the debugger before the jump is executed.

M - Modify the contents of the current address. The least significant eight bits of the octal number given before the command character are used for the new memory value. If no digits are given, a zero is assumed.

P - Turn on the P-counter display (to the left of the current address). This display is a foreground driven routine which takes the value of the P-counter when the interrupt occurred and displays it vertically. This implies that the value shown is the background P-counter at 32 millisecond sample points. When the display is active, simultaneous depression of the KEYBOARD and DISPLAY keys will cause the debugger to be

entered regardless of what is currently being executed in the background. When such entry occurs, the current address points to the location where the background program was interrupted so that execution can be resumed with the J command.

R - Display the saved alpha mode register value. The registers are referenced by number (0-A, 1-B, 2-C, 3-D, 4-E, 5-H, 6-L, and 7-Conditions). The condition code is stored with bits 7=Carry, 6=Sign, bits 5 through 2 always zero, 1=(-Zero and -Sign), and 0=(-Zero and -Parity). (The easiest way to understand this is to realize that the condition code as displayed, added to itself, results in restoring all four conditions to their entry values.) When a register is displayed, the address shown is the memory location used to store the value of that register. This does not, however, affect the current address. The registers may be initialized for a C or J command by simply storing into the memory locations displayed when the registers are displayed.

X - Turn off the P-counter display.

- Clear all break points. The current address will reflect the location of the last point cleared.

. - Perform the M command followed by the I command.

CANCEL - Erase the entered number without changing the current address.

ENTER - Change the current address to the digits entered. If no digits are entered, the current address in effect will be displayed.

46.7.2 KEYIN$ - Obtain a Line from the Keyboard

KEYIN$ obtains a string of characters from the keyboard, displaying them on the screen and storing them in memory as they are entered. When KEYIN$ is called, the cursor is turned on and characters requested. Backspacing off the beginning of the line, entering more than the specified maximum number of characters, or running off the screen is prevented. The routine turns off the cursor and returns when the ENTER key is depressed.

Entry point: 01157

Parameters: C = maximum number of characters accepted (including ENTER)
D = initial horizontal cursor position
E = vertical cursor position
HL= starting location of input buffer

Exit conditions: String terminated by 015
HL= pointing to the 015
D = horizontal position of ENTER
E = unchanged
C = 0
B = undefined

46.7.3 DSPLY$ - Display a Line on the Screen

DSPLY$ displays a string of characters stored in memory on the screen. Certain characters denote control functions according to the following table:

003 - end of string
011 - new horizontal position follows
013 - new vertical position follows
015 - end of string with CR/LF
021 - erase to end of frame
022 - erase to end of line
023 - roll up one line

If the string to be displayed starts with either or both horizontal or vertical cursor controls, then either or both of the corresponding values need not be in D or E at entry. If the cursor is not positioned on the screen with DE or 011 and 013 the results of 021, 022, or 023 are undefined.

Entry point: 01162

Parameters: D = initial horizontal cursor position
E = initial vertical cursor position
HL points to string in memory

Exit conditions: DE = cursor position after the last character displayed
HL = byte after the string terminator
A,B,C undefined

CHAPTER 47. DOS FUNCTION FACILITY (DOSFNC)

The page of memory located between 07400 and 07777 contains a special loader and overlay area. This "loader" can load any one of up to 255 DOS overlays, each up to 124 bytes long. The loader resides in the first half of the page and the overlays all load into the second half of the same page. The overlays reside on disk in physical file 7, called SYSTEM7/SYS. The design of the DOS FUNCTION loader is such that overlays are loaded only if necessary; i.e. if the same overlay is called several times in sequence, it is not reloaded each time. The overlays provide the DOS assembly language programmer with many useful utility functions. Parameterization of DOS FUNCTIONs varies with the individual functions, the only basic requirement being that on entry to the DOS FUNCTION loader, the A register contains the function number (1-255). Use of functions not yet installed will produce indeterminate results, but may result in format traps, range traps, processor halts, and the like. DOS FUNCTIONs are normally loaded from the SYSTEM7/SYS on drive zero.

Upon the first call to DOSFNC (the DOS FUNCTION loader), SYSTEM7/SYS is opened as LF0 and the LFT entry saved within the DOS FUNCTION loader. Upon subsequent calls to DOSFNC, the entry is simply moved back into the LFT, eliminating the need to re-open SYSTEM7/SYS each time a function is loaded. The file is only closed by reloading DOS, either by depressing RESTART or by a program passing control to BOOT$, EXIT$, or ERROR$.

Since new DOS functions will be added as necessary the following descriptions should not be considered exhaustive.

Entry point: 07400

Parameters: A = Function number (1-0377)
Others required by individual functions

Exit conditions: Defined separately for each function.

47.1 FUNC1 - Retrieve Directory and C.A.T. Addresses

Uniform attributes for all subfunctions

```
On entry,     A = function number (1)
              C = subfunction number (0,1,2,3,4,5,6,7)
On exit,      B,C,H,L all unchanged
              CARRY FALSE:  function completed successfully
              CARRY TRUE:   invalid subfunction number
```

All other entry/exit parameters and conditions are described seperately for each individual subfunction.

DOS FUNCTION: 1 SUBFUNCTION: 0

Return the address of a specified directory sector in DE.

```
On entry,     B = directory sector number (0-15) OR
              PFN of entry in the directory sector
On exit,      A indeterminate
              DE = PDA of specified directory sector.
```

DOS FUNCTION: 1 SUBFUNCTION: 1

Return the two byte physical disk address for each of the 16 master directory sectors, into a 32-byte work area provided by the user.

```
On entry,     HL => 32-byte work area to receive the PDA's
On exit,      ALL REGISTERS RESTORED
              user-provided work area contains 16 PDA's,
              one corresponding to each prime directory
              sector, in ascending order. (LSB,MSB)
```

DOS FUNCTION: 1 SUBFUNCTION: 2

Return the two-byte physical disk address of each of the 16 directory sector backups, in ascending order, into a 32-byte user-provided work area.

On entry, HL => 32-byte work area to receive the PDA's
On exit, all registers restored
user-provided work area contains 16 PDA's, one corresponding to each backup directory sector, in ascending order. (LSB,MSB)

DOS FUNCTION: 1 SUBFUNCTION: 3

Return the physical disk address of the cluster allocation table (CAT) in the DE register pair.

On entry, no further conditions
On exit, A indeterminate
DE = PDA of prime CAT

DOS FUNCTION: 1 SUBFUNCTION: 4

Return the physical disk address of the backup cluster allocation table (CAT) in the DE register pair.

On entry, no further conditions
On exit, A indeterminate
DE = PDA of backup CAT

DOS FUNCTION: 1 SUBFUNCTION: 5

Return the physical disk address of the lockout CAT

On entry, no further conditions
On exit, A indeterminate
DE = PDA of lockout CAT

DOS FUNCTION: 1 SUBFUNCTION: 6

Return the physical disk address of the lockout CAT backup

On entry, no further conditions
On exit, A indeterminate
DE = PDA of lockout CAT backup

DOS FUNCTION: 1 SUBFUNCTION: 7

Return the address of a backup directory sector (in DE)

On entry, B = backup directory sector number (0-15)
OR PFN of a file entry contained therein

On exit, A indeterminate
DE = PDA of backup directory sector

47.2 FUNC2 - Retrieve Directory Sector or Filename

Uniform attributes for all subfunctions

```
On entry,     A = function number (2)
              C = subfunction number (0,1,2)
On exit,      ALL REGISTERS RESTORED
              CARRY TRUE: error or invalid subfunction number
```

All other entry/exit parameters and conditions are described separately for each individual subfunction.

DOS FUNCTION: 2 SUBFUNCTION: 0

Read in the directory sector containing the 16-byte directory entry corresponding to the PFN given, on a specified logical drive.

```
On entry,     B = LFN as per DOS standard; (0, 16, 32, 48)
              D = PDN (logical drive number of file)
              E = PFN
On exit,      CARRY FALSE: selected directory sector is in
                           buffer specified, which is the
                           selected buffer upon exit.
              CARRY TRUE:  indicates I/O error.
                           Further defined as follows:
              ZERO FALSE:  specified drive is off-line
              ZERO TRUE:   unable to read sector due to CRCC
                           error during read, or unrecoverable
                           failure to find sector.
```

DOS FUNCTION: 2 SUBFUNCTION: 1

Get the 16-byte directory entry corresponding to a specified PFN on a given logical drive.

```
On entry,     B = LFN as per DOS standard; (0, 16, 32, 48)
              D = PDN (logical drive number of file)
              E = PFN
              HL => 16 byte area to receive the entry
```

On exit,
- CARRY FALSE: entry is in user's area
- CARRY TRUE: indicates I/O error. Further defined as follows:
  - ZERO FALSE: specified drive is off-line
  - ZERO TRUE: unable to read sector due to CRCC error during read, or unrecoverable failure to find sector.

DOS FUNCTION: 2 SUBFUNCTION: 2

Get name/ext (pfn) for a specified numbered file on a specified logical drive. (Same basic format as used by DOS CAT command).

On entry,
- B = LFN as per DOS standard; (0, 16, 32, 48)
- D = PDN (logical drive number of file)
- E = PFN
- HL => 20 byte area to receive the entry

On exit,
- CARRY FALSE: user's 20-byte area contains the name, extension and PFN of the specified file, for example: EDIT/CMD (037) where the right paren is followed by an 003. UNLESS ZERO TRUE: implies that the file number specified does not exist.
- CARRY TRUE: indicates I/O error. Further defined as follows:
  - ZERO FALSE: specified drive is off-line
  - ZERO TRUE: unable to read sector due to CRCC error during read, or unrecoverable failure to find sector.

NOTICE: the use of THIS SUBFUNCTION ONLY (of those in DOS FUNCTION 2) <u>requires</u> that the DOS command interpreter be present (the command interpreter resides from 013400-017000).

47.3 FUNC3 - Retrieve R.I.B. Information

Uniform attributes for all subfunctions:

```
On entry,     A = function number (3)
              C = subfunction number (0,1,2,3)
```
All other entry and exit parameters and conditions are described separately for each individual subfunction.

DOS FUNCTION: 3 SUBFUNCTION: 0

Return the number of sectors allocated to a file on disk.

```
On entry,     B = drive number (like C as provided for OPEN$)
              DE = proper OPEN$ parameters defining the file
                   to be accessed.
On exit,      CARRY FALSE: function completed successfully
                           HL = length of file (MSB,LSB) in sectors
                           RIB for file specified is in LF0
                           disk buffer.
              CARRY TRUE:  indicates an error occurred, any one of:
                           OPEN failed on file specified;
                           unable to read RIB;
                           parity or drive off-line.
```

DOS FUNCTION: 3 SUBFUNCTION: 1

Get the RIB for a specified file into the LF0 disk buffer

```
On entry,     B = drive number (like C as provided for OPEN$)
              DE = proper OPEN$ parameters defining the file
                   to be accessed.
On exit,      ALL REGISTERS RESTORED
              CARRY FALSE: function completed successfully
                           RIB for file specified is in
                           LF0 disk buffer
              CARRY TRUE:  indicates an error occurred, any one of:
                           OPEN failed on file specified;
                           unable to read RIB;
                           parity or drive off-line.
```

DOS FUNCTION: 3 SUBFUNCTION: 2

Read a RIB for a file, given the first two bytes of the directory entry

On entry, B = drive number (like C as provided for OPEN$)
D = RIB pointer, (MSB) from directory, or LFT
E = RIB pointer, (LSB) from directory, or LFT

On exit, ALL REGISTERS RESTORED
CARRY FALSE: function completed successfully
RIB for file specified is in LFO disk buffer
CARRY TRUE, ZERO FALSE: specified drive off-line
CARRY TRUE, ZERO TRUE: parity error during read.

DOS FUNCTION: 3 SUBFUNCTION: 3

Return segment descriptor information from a RIB

On entry, RIB is in LFO disk buffer
BUFADR field in LFO LFT entry points to segment descriptor

On exit, HL, LFO buffer unchanged.
CARRY TRUE: function completed successfully.
A = starting cyl. number for segment
B = starting cluster number for segment
DE = number of sectors in the segment
BUFADR points to next segment descriptor; RIB undisturbed
CARRY FALSE: implies BUFADR pointed after LOGICAL end of RIB;
BUFADR contents undefined.

47.4 FUNC4 - Retrieve DOS Configuration Information

Uniform attributes for all subfunctions:

```
On entry,     A = function number (4)
              C = subfunction number (0,n)
On exit,      A = DOS configuration value
              CARRY FALSE: function completed successfully
              CARRY TRUE:  possibly invalid subfunction number.
```

Different subfunction numbers return different DOS configuration bytes. These values, returned in A, are numeric items which change in value depending upon which DOS is running. The subfunction numbers, along with the significance of the returned value, are:

```
0 - Letter of this DOS (A,B,C,D,etc.)
1 - DOS Version ('2' typ.)
2 - DOS Revision ('3' typ.)
3 - Total number cylinders on disk (203 typ.)
4 - Maximum Logical Drive (3,15 typ)
5 - Year of Compilation (77 typ)
6 - Day of Compilation (49 typ)
7 - Cluster Mask (0340 typ)
8 - Increment Cluster number (040 typ)
9 - Sector Mask (037 typ)
10 - Maximum Sector Number in PDA (23 typ)
11 - Number of Sectors/Cluster (3,6,24 typ)
12 - Number of Clusters/Cylinder (4,8 typ)
13 - Number of Clusters/Track (1,4 typ)
14 - Number of Functions in SYSTEM7 (24 typ)
15-24 (Unused)
25 - Get VOLID address into (DE)
26 - Internal DOS address for DF11
```

47.5 FUNC5 - Request Access to System Tables

This function is used when running under the Partition Supervisor (PS). This function must be called before and after any changes are made to the system tables on any drive.

Uniform attributes for all subfunctions:

```
On entry,     A = function number (5)
              C = subfunction number (0,1)
```

All other entry and exit parameters and conditions are described seperately for each individual subfunction.

DOS FUNCTION: 5 SUBFUNCTION: 0

Request exclusive update permission to system table sectors on disk

```
On entry,     D = physical drive (PDN) of drive
On exit,      CARRY FALSE: function completed successfully
                           exclusive use of specified drive
                           guaranteed.
              CARRY TRUE:  indicates an error occurred.
```

DOS FUNCTION: 5 SUBFUNCTION: 1

Release exclusive update authority for system table sectors on disk.

```
On entry,     D = physical drive (PDN) of drive
On exit,      CARRY FALSE: function completed successfully
                           exclusive use of specified drive
                           released.
              CARRY TRUE:  indicates an error occurred.
```

47.6 FUNC6 - Keyboard / Display Interface Routines Function

Uniform attributes for all subfunctions:

```
On entry,     A = function number (6)
              C = subfunction number (0-11)
On exit,      CARRY TRUE:   illegal subfunction
              All other entry and exit parameters and
              conditions are described sepatately for
              each individual subfunction.
```

DOS FUNCTION: 6 SUBFUNCTION: 0

Check the status of the KEYBOARD and DISPLAY keys

```
On entry,     no further conditions
On exit,      SIGN TRUE:     KEYBOARD key pressed
              PARITY TRUE:   DISPLAY key pressed
              ALL REGISTERS RESTORED
```

DOS FUNCTION: 6 SUBFUNCTION: 1

Check for character ready

```
On entry,     no further conditions
On exit,      ZERO TRUE:     No character present
              ZERO FALSE:    Ready to get character
              ALL REGISTERS RESTORED
```

DOS FUNCTION: 6 SUBFUNCTION: 2

Get a Character from the Keyboard

```
On entry,     (DE) = Horizontal, Vertical Screen Coordinates
On exit,      ZERO TRUE:     No character present
              ALL REGISTERS RESTORED
              ZERO FALSE:    Got the character in (A)
              ALL OTHER REGISTERS RESTORED
```

DOS FUNCTION: 6 SUBFUNCTION: 3

Write the Character in (B) to the Screen

```
On entry,     (DE) = Horizontal, Vertical Screen Coordinates
              (B)  = Character to be written to Screen
On exit,      CARRY TRUE:    (D) or (E) are out of range
```

ALL REGISTERS RESTORED
CARRY FALSE: The Character was written
ALL REGISTERS RESTORED

DOS FUNCTION: 6 SUBFUNCTION: 4

Return the HOME-UP Position in (DE)

On entry, no further conditions
On exit, (DE) = Address of Top Line / Left Column of CRT
ALL OTHER REGISTERS RESTORED

DOS FUNCTION: 6 SUBFUNCTION: 5

Return the HOME-DOWN Position in (DE)

On entry, no further conditions
On exit, (DE) = Address of Bottom Line / Left Column of CRT
ALL OTHER REGISTERS RESTORED

DOS FUNCTION: 6 SUBFUNCTION: 6

Turn on the Cursor

On entry, (DE) = Horizontal, Vertical Screen Coordinates
On exit, ALL REGISTERS RESTORED

DOS FUNCTION: 6 SUBFUNCTION: 7

Rollup the Screen 1 line

On entry, (DE) = Horizontal, Vertical Screen Coordinates
On exit, ALL REGISTERS RESTORED

DOS FUNCTION: 6 SUBFUNCTION: 8

Erase from Cursor Position to End of Frame

On entry, (DE) = Horizontal, Vertical Screen Coordinates
On exit, ALL REGISTERS RESTORED

DOS FUNCTION: 6 SUBFUNCTION: 9

Erase from Cursor Position to End of Line

On entry, (DE) = Horizontal, Vertical Screen Coordinates
On exit, ALL REGISTERS RESTORED

DOS FUNCTION: 6 SUBFUNCTION: 10

Rolldown the Screen 1 line

On entry, (DE) = Horizontal, Vertical Screen Coordinates
On exit, CARRY TRUE: Illegal operation for this device
ALL REGISTERS RESTORED
CARRY FALSE: The screen was rolled down 1 line.
ALL REGISTERS RESTORED

DOS FUNCTION: 6 SUBFUNCTION: 11

Turn off the Cursor

On entry, (DE) = Horizontal, Vertical Screen Coordinates
On exit, ALL REGISTERS RESTORED

47.7 FUNC7 - Test the Disk Buffer Memory

Disk buffer memory test function

This DOS FUNCTION performs a rotating, cycling test of the disk controller buffer memories. It returns upon the keyboard becoming READ READY, or upon encountering a buffer failure, whichever occurs first.

On entry, Doesn't matter.
On exit, ALL REGISTERS UNCHANGED
ZERO TRUE: buffer memory test completed normally
ZERO FALSE: failure indicated in buffer memories

47.8 FUNC8 - Timed Pause

Pause function

This DOS FUNCTION provides the user program with a timed pause. The requested pause may be up to over four hours long.

```
On entry,     B = foreground process number to use (0-7)
              CDE = number of milliseconds to pause
                    (C = most signifigant, E = least signifigant)
On exit,      ALL REGISTERS UNCHANGED
```

Note that if foreground process numbers 4-7 are used, the wait time is effectively multiplied by four, allowing a maximum wait time in excess of eighteen hours. Also note that the time required to start up the DOS FUNCTION is not considered part of the time paused. Since the DOS FUNCTION may or may not be resident when called, this function may wait longer than the quantity in CDE and therefore must not be used for timing really critical, short term intervals.

47.9 FUNC9 - Non-Sharable Resource Status Request

This DOS FUNCTION is used to allocate and de-allocate a system resource. Typically, this function is used when a program is going to run under the Partition Supervisor (PS). The use of this function will prevent conflicting use of I/O devices by the programs running in the two partitions. For example, the DOS utilities that use the printer (LIST, FILES, etc.) all call this function before they use the printer. Then, if a DATASHARE print statement is executed in the other partition, the listings will not be intermixed.

Uniform attributes for all subfunctions:

On entry, A = function number (9)
C = subfunction number (0,1)

All other entry and exit parameters and conditions are described separately for each individual subfunction.

DOS FUNCTION: 9 SUBFUNCTION: 0

Propose to use a non-shareable system resource (printer, tape drive, etc.).

On entry, B = Resource Number
0 - Local Line Printer
1 - Servo Printer
2 - (Un-defined)
3 - (Un-defined)
4 - Cassette Tape Decks
5 - 7 or 9 Track Tape
6 - Multiport Comm Box 1 (all ports)
7 - Multiport Comm Box 2 (all ports)

On Exit, CARRY TRUE, ZERO TRUE: Permission to use granted.
CARRY TRUE, ZERO FALSE: Error
CARRY FALSE, ZERO TRUE: Already allocated, same partition - in this case, go ahead and use the device, but DO NOT deallocate it when finished (subfunction 1).
CARRY FALSE, ZERO FALSE: Already allocated, other partition.

DOS FUNCTION: 9 SUBFUNCTION: 1

Release non-sharable resource for use by next party. This subfunction should be called after a process receiving access to a resource using subfunction zero has received CARRY TRUE, ZERO TRUE return, and finishes using the resource it wanted to use.

The only status returned by subfunction one that is likely to change upon waiting is CARRY FALSE, ZERO FALSE. In this case, the program wishing to release the resource should wait, perhaps five seconds (use function 8), and then retry the request. Any other status is not subject to change.

Note that this subfunction can NOT be used to test for printer busy, since if an invoking program in the same partition had allocated the device, the test would release it, possibly resulting in losing the device to a competing partition. This would be an error and must not be allowed to occur.

INDEFINITE POSTPONEMENT can be prevented by always allocating non-sharable resources in DECENDING numerical sequence (when more than one non-sharable resource is needed at the same time).

On entry, all parameters identical to those for subfunction 0.
On exit, CARRY TRUE: error
CARRY FALSE, ZERO TRUE: normally released
CARRY FALSE, ZERO FALSE: was in use by different partition, therefore not released.

47.10 FUNC10 - Qualify for Execution in Fixed Partition

This function is used to qualify a program to run in a "fixed" partition under the Partition Supervisor (PS), and to provide DOS/PS partition configuration information.

Uniform attributes for all subfunctions:

```
On entry,        A = function number (10)
                 C = subfunction number (0,1)
```

All other entry and exit parameters and conditions are described separately for each individual subfunction.

DOS FUNCTION: 10 SUBFUNCTION: 0

Authorize invoking program to execute in a fixed type partition.

```
On entry,     C = subfunction number (0)
On exit,      No conditions significant
```

DOS FUNCTION: 10 SUBFUNCTION: 1

Provide DOS/PS configuration information.

```
On entry,     C = subfunction number (1)
On exit,      HL => Configuration list (which may not be modified)
                    guaranteed only until the next call to any
                    system routine; list format described below.
              BYTE 0:  Partition ID.
                       Space if not running under PS, otherwise,
                       a unique identifier.
              BYTE 1:  Region Size - in number of K (16, 48, etc)
              BYTE 2:  Number of Disk Buffers (4, 16, etc)
              BYTE 3:  .... ...1 Implies Fixed Partition
              BYTE 4:  Multiport I/O Bus Address for console on port
                       (0 => console on port not active)
              BYTE 5:  Multiport port select code of console port
                       (only if byte 4 is non-zero).
```

47.11 FUNC11 RAM Screen Loader

Uniform attributes for all subfunctions:

On entry, A = function number (11)
C = subfunction number (0,1,2)

All other entry and exit parameters and conditions are described separately for each individual subfunction.

DOS FUNCTION: 11 SUBFUNCTION: 0

Load one or more character combinations into the RAM display character generator.

On entry, B = default first character to be loaded
HL = starting address of character set definition list

The list consists of consecutive entries of either five or six bytes each. The first byte, if present, indicates the 7-bit character combination whose bit pattern definition follows. The presence of the first byte is indicated by its sign bit being set. If the first byte of the first entry is not present, the 7-bit character combination in the B register is used instead. The definition list may contain any mixture of six byte and five byte entries. The end of the list is indicated by an 0200. This implies that the bit combination displayed for a binary zero cannot be imbedded in a list, but can only appear at its beginning; null lists are not allowed. The five data bytes following represent the five columns of bits for each displayed character and can each have values of 0 (a blank column) to 0177 (a vertical line). The 0100 bit is at the top of the character displayed; the 1 bit is on the bottom row of the displayed character.

On exit, CARRY FALSE, ZERO FALSE implies RAM display not present
CARRY FALSE, ZERO TRUE indicates normal completion
CARRY TRUE indicates error (should not occur)

DOS FUNCTION: 11 SUBFUNCTION: 1

Load a single character combination to RAM display

On entry, B = default character to be loaded
HL = address of five or six byte bit pattern definition

The first byte, if present, takes precedence over the character indicated by the B register. Presence of the first byte is indicated by the sign bit being set.

On exit, CARRY FALSE, ZERO FALSE implies RAM display not present
CARRY FALSE, ZERO TRUE indicates normal completion

DOS FUNCTION: 11 SUBFUNCTION: 2

Subfunction two requests reloading of the standard character set on program termination. Calling this subfunction will result in the standard DOS character set being reloaded upon the next entry to DOS$. Entry to the DOS at DOS$ is the result of transfer of control to EXIT$, BOOT$, ERROR$ as well as DOS$. Return to the DOS via NXTCMD, CMDAGN, and CMDINT do not result in the display being immediately reloaded, (but it still will be upon subsequent entry at DOS$ as described).

47.12 FUNC12 - Unassigned DOS Function

This DOS function is unused.

47.13 Overlay Loader (FUNC-13,14,15)

DOS functions 13 and 14 are used to load "overlay libraries". Using these functions, one need only have a single directory entry for a program and its associated overlays (called "members"). The overlay library format is described in detail in the library utility program user's guide (LIBSYS), since that program is responsible for creating and maintaining libraries. Program libraries can be absolute or relocatable code.

Function 13 performs a library lookup by name.

Function 14 actually performs the library load from an absolute library.

Function 15 performs the library load from a relocatable library.

Below is an example of how to use these DOS functions to load an absolute program overlay. In the example, the "root" program has several overlays; the root program was invoked from the keyboard by entering "FLX/ABS".

```
.
.
.
          HL        LFT+LF0             Save opened LFT entry
          DE        SAVELFT             (FLX/ABS)
          LC        16                  16 bytes
          CALL      BLKTFR
.
.
. To lookup the member named "FLXOAW":
NAMLOD    LA        13                  Prepare to lookup by name
          DE        OVLNAM
          HL        SAVELFT
          CALL      DOSFNC              Lookup by name
          JTC       ABORT               True Carry is error.
          JUMP      LOAD                (DE contains LRN if Carry False)
```

```
. To actually load the absolute file (and execute it):
LOAD      LA        14                Prepare to load the file
          HL        SAVELFT
          CALL      DOSFNC            Load starting at LRN (DE)
          JTC       ABORT             True Carry is error.
.                                     (HL is transfer address if Carry False)
          PUSH                        Push entry point onto stack
          RET                         and transfer control there
.
.
.
SAVELFT   SK        16                LFT save area
FLXOAW    DC        'FLXOAW  '        8-byte name
.
.
.
```

Note: all lookups (FUNC13) should be done first, and then all loads (FUNC14 or FUNC15) so functions will not be reloaded as often.

47.14 FUNC-13 Overlay Lookup By Name

Return the LRN of library member <name>, pointed to by (DE) into (DE).

```
On entry,     A = function number (13)
              DE => address of 8-byte file name
              HL => 16-byte save area of user opened LFT
                    (not LF0)
On exit,      CARRY TRUE:  Name not found
              Otherwise:
                A = Library Type-(see LIBSYS user's guide)
                BC = Undefined
                DE = LRN (LSB,MSB) of library member
                HL = Entry value of (DE)+8
```

47.15 FUNC-14 LOAD ABSOLUTE LIBRARY MEMBER

Load the absolute member beginning at LRN given in (DE).

On entry, A = function number (14)
DE = LRN (LSB,MSB) of member to be loaded.
HL => 16-byte save area of user opened LFT (not LF0)

On exit, CARRY TRUE: Unloadable file
Otherwise:
A,B,C,D,E Undefined
HL = Transfer address of member

47.16 FUNC-15 RELOCATABLE LOADER

Uniform attributes for all subfunctions:

```
On entry      A = function number (15)
              B = LFN of Opened Relocatable Library
              C = subfunction number (0 or 1)
              All other entry and exit parameters and
              conditions are described separately for
              each individual subfunction.
```

DOS FUNCTION: 15 SUBFUNCTION: 0

Return the size of relocatable member into (DE)

```
On entry      DE => LRN 0 of library member (from DOSFNC-13 typically)
On exit       CARRY TRUE:  Invalid Library Format
              Otherwise:
              A,C,H,L  Undefined
              B = Entry value
              DE = Program length (LSB,MSB)
```

DOS FUNCTION: 15 SUBFUNCTION: 1

Load a relocatable library member.

```
On entry      HL => Address of Parameter List
              DE => LRN 0 of library member (from DOSFNC-13 typically)
On exit       CARRY TRUE:  Link Error Occurred
              Otherwise:
              A,C = Undefined
              B = Unchanged
              DE => Next available address
              HL => Transfer address
```

Parameter Table for DOS FUNCTION 15 SUBFUNCTION 1:

0-1 Origin Address (LSB,MSB)

2-3 Address of External Reference work area terminated by 000.
0377,0377 => No work area, i.e.,no external references.
Example:

```
RPT     20
DC      'bbbbbbbb',*-1
DC      0
```

4-5 Address of External Definition work area terminated by 000.
0377,0377 => No work area,i.e.,no external definitions.
Example:

```
RPT     20
DC      'bbbbbbbb',*-1
DC      0
```

6-7 LRN (LSB,MSB) of Relocatable Module.

Note: Subfunction 1 over-writes the DOS DEBUG area (06000-07377)

CHAPTER 48. CASSETTE HANDLING ROUTINES

Standard record formats, identifiers, and file marker record conventions on cassettes are established by the Cassette Tape Operating System. Routines capable of dealing with cassettes in a manner compatible with CTOS are provided as part of the Disk Operating System to enhance its overall capability. For detailed information on cassette format and organization, see the Cassette Tape Operating System Manual.

All of the DOS cassette routines are foreground driven and, with the debugging facility and DOS Function 8, are the only routines within the system which make use of the foreground handling facility. Being foreground driven, however, does not alter the manner in which the routines are handled since all interfacing between the background and foreground is handled by the system. It does allow increased speed of operation with the cassettes since the user may be processing one record while the next is being read from or written to the tape. This is evident in the way the DOS slews the tape when transferring information between it and the disk.

Some of the cassette handling routines initiate foreground action and then return immediately to the user while others wait for I/O completion. All of the routines wait for any uncompleted I/O to finish before starting something new. Note that in the cases of reading or writing on the same deck, requesting the next operation before the completion of the first will cause the tape to automatically slew instead of stopping between records. This is only in the case of a read followed by another read or a write followed by another write on the same deck. The only cases where caution must be exercised is in the read and write routines which return immediately after starting the I/O operation. If the user does not wait for the transfer to complete, he could try to use the data before it is read or change the data before it is written. In the second case, records with incorrect parity will usually be generated. Routines are provided, however, which automatically wait for the transfer to complete, relieving the user of having to concern himself with the fact that the routines are foreground driven if he has no need for the advantages.

The various error conditions associated with cassette handling can be trapped by the user. If the trap is not set, an error message similar to the error message generated by CTOS is displayed and the DOS reloaded. If the trap has been set, the

address specified will be jumped to and the trap cleared. The traps are identified in the error message by a letter similar to the CTOS identification. In the relevant cases, the same letter is used in the DOS as is used in the CTOS. In the following routine descriptions the relevant letter will be given in the 'Traps' section.

Most of the cassette routines are parameterized by a deck number given in the B register. This number is a zero for the rear deck and a one for the front deck. The cassette handler routines use interrupt slot 1 for their foreground process.

48.1 TPBOF$ - Position to the Beginning of a File

TPBOF$ positions the cassette in the specified deck to the specified file. The search for the file marker of the desired file is started with backward motion of the tape. If a marker of lower value than the file number requested or the beginning of the tape is encountered, the search will be reversed to forward motion of the tape. If then a marker of larger value than the file number requested, the end of the tape, or a record of unrecognizable format is encountered, an error G will be given. Otherwise, the file is left positioned before the first data record.

Entry point: 010000

Parameters: B = deck number
C = physical file number (0-0177)

Exit conditions: none

Traps:

D	unrecognizable record found
G	file could not be found

48.2 TPEOF$ - Position to the End of a File

TPEOF$ moves the tape forward until the next file mark is found. It then backspaces the tape one record to leave it at the end of the current file.

Entry point: 010005

Parameters: B = deck number

Exit conditions: none

Traps: D unrecognizable record found
E end of tape encountered

48.3 TRW$ - Physically Rewind a Cassette

TRW$ rewinds the cassette on the selected deck by first slewing backwards to ensure that the tape is not on the trailer and then performing a hardware rewind.

Entry point: 010012

Parameters: B = deck number

Exit conditions: none

Traps: none

48.4 TBSP$ - Physically Backspace One

TBSP$ simply executes a hardware backspace function. No checking is performed on the data passed over. However, backspacing onto clear leader causes an end of tape trap.

Entry point: 010017

Parameters: B = deck number

Exit conditions: none

Traps: E beginning of tape encountered

48.5 TWBLK$ - Write an Unformatted Block

TWBLK$ writes the specified number of bytes (0-255; 0 causes 256 to be written) from the memory buffer specified onto the cassette in the deck specified. Only the bytes specified will be written on the tape.

Entry point: 010024

Parameters: B = deck number
C = number of bytes to write (0 for 256)

HL points to start of buffer

Exit conditions: none

Traps:		
	E	end of tape encountered
	Z	premature deck ready status

48.6 TR$ - Read a Numeric CTOS Record

TR$ reads a record of CTOS numeric format into the memory locations specified. The length of the record is stored in the specified memory location and the data bytes are stored in the locations that follow. Return is made from TR$ as soon as the read operation is started but the user cannot use the data until the operation has been completed (see TCHK$). One way to check for operation completion is to call TR$ again with a different buffer as its parameter. Return from the second call will be made as soon as the first operation is completed. This is the mechanism via which multiple buffering is normally achieved. Note that tape motion will not cease if TR$ is called within five milliseconds of the end of the previous record.

If parity problems arise, TR$ tries up to 5 times to read the tape before giving a parity failure trap. Other traps given are end of tape and end of file. If an end of file trap is given, the tape is positioned before the file marker.

Entry point: 010031

Parameters: B = deck number
HL points to data storage location

Exit conditions: none

Traps:		
	D	parity failure
	E	end of tape encountered
	F	end of file encountered

48.7 TREAD$ - TR$ and Wait for the Last Character

TREAD$ performs the TR$ function and then waits for the last character to be read from the tape. This routine should be used when multiple buffering is not being performed since it relieves the user from having to explicitly wait for the last character to be read.

Entry point: 010034

Parameters: same as for TR$

Exit conditions: none

Traps: same as for TR$

48.8 TW$ - Write a Numeric CTOS Record

TW$ writes the specified memory locations in a record of standard CTOS numeric format. It uses (for parity generation) the three locations preceeding the memory location specified which contains the number of bytes to be written and is followed by that number of data bytes.

TW$ returns as soon as the write operation is started. The user must be careful not to change any of the memory locations given as parameters before the last byte has been transferred. This can be achieved by either calling TCHK$ and waiting for completion status or calling TW$ with the next buffer if multiple buffering is being used. Note that tape motion will not cease if TW$ is called before the middle of the IRG is reached from the previous write (140 milliseconds after the last character is written when using a 7.5 ips deck).

Entry point: 010037

Parameters: same as for TR$

Exit conditions: none

Traps:

E	end of tape encountered
Z	premature deck ready status

48.9 TWRIT$ - TW$ and Wait for the Last Character

TWRIT$ executes the TW$ routine and then waits for the last byte to be written on the tape. This routine should be used when multiple buffering is not being performed since it relieves the user from having to explicitly wait for the last byte to be written.

Entry point: 010042

Parameters: same as for TR$

Exit conditions: none

Traps: same as for TW$

48.10 TFMR$ - Read the Next File Marker

TFMR$ reads the tape until a file marker record is found. A trap occurs if a record is encountered that is neither a file marker nor a CTOS numeric data record.

Entry point: 010045

Parameters: B = deck number

Exit conditions: C = PFN of marker found
Tape positioned after marker record

Traps:

D	unrecognizable record found
E	end of tape encountered

48.11 TFMW$ - Write a File Marker Record

TFMW$ writes a file marker record that contains the number specified.

Entry point: 010050

Parameters: B = deck number
C = PFN to be written

Exit conditions: none

Traps: E end of tape encountered
        Z premature deck ready status

48.12 TTRAP$ - Set an Error Condition Trap

TTRAP$ allows the user to trap the various errors associated with cassette I/O. If the trap is not set, an error message of the form

*** ERROR X ON DECK Y ***

will be displayed, where X is one of the letters shown below and Y is a 1 for the rear deck and a 2 for the front deck. The trap is specified by a number according to the following table:

3 - D - parity error
4 - E - end of tape
5 - F - end of file
6 - G - unfindable file

In addition, error Z (cannot be trapped) indicates that the deck ready status bit came true while a record was being written. This status implies that the write routine fell behind in writing characters and most probably indicates that the foreground interrupt handling was disrupted in some fashion (interrupts were disabled too long or an interrupt driven routine was running which imposed too much overhead). It may also be caused by the tape being write protected (left rear tab punch out).

Traps can be cleared by setting their addresses to zero. When the event which causes a trap occurs, that trap is cleared and control passed to the address indicated with the deck number in the B register (0 for rear and 1 for front deck).

Entry point: 010053

Parameters: C = trap number (above)
             DE= trap address (0 clears trap)

Exit conditions: none

Traps: none

48.13 TWAIT$ - Wait for I/O Completion

TWAIT$ waits for any tape operation active to complete. This does not mean that physical motion has stopped since TR$ and TW$ indicate I/O completion when the last character has been transferred. It does mean that all data is free to be processed by the user. TWAIT$ also executes any traps pending upon the completion status being set.

Entry point: 010056

Parameters: none

Exit conditions: B, C, D, and E registers preserved

Traps: any trap pending will be executed

48.14 TCHK$ - Get I/O Status

TCHK$ sets the tape demand flag in the carry condition flag and loads the tape handling status in the A register. The handling status codes are as follows:

000 - PBOF in progress
002 - PEOF in progress
004 - Rewind in progress
006 - Record read in progress
010 - Backspace in progress
012 - File mark read in progress
014 - Record write in progress

377 - Normal completion
206 - Parity error
210 - End of tape
212 - End of file
214 - File not found
262 - Premature deck ready status

Normal use of the cassette routines will not require the user to deal with these status codes or even use the TCHK$ routine. They are provided here to facilitate understanding the listing of the routines.

Entry point: 010061

Parameters: none

Exit conditions: Carry condition = demand flag
A = status code (above)

Traps: none

CHAPTER 49. COMMAND INTERPRETER ROUTINES

This section deals with a series of routines within the command interpreter. Note that these routines are only available for use if the user program does not overlay the command interpreter, which resides in locations 012400-016777.

The first four of these entry points are really more like "exit points", since they are places in the DOS to which users may return in place of EXIT$. The primary advantage to using them in place of EXIT$ is that none of these four entry points result in the DOS being reloaded, a process which takes significant time. Note that since they do not reload the DOS, programs which exit through CMDINT, DOS$, CMDAGN, or NXTCMD must not have overstored any part of the DOS; i.e. they should run completely in locations 017000 upwards. Also, these "exit points" do not clear any traps that the user may have set; therefore the user should clear any traps he has set before exiting in this manner. If this is not done, the system will most likely go astray upon the first subsequent occurrence of a trapped situation.

Most of the other routines documented in this section are routines which are used by one or more of the DOS command programs supplied either on the DOS Generation or DOS Utilities tapes. Since these routines are pointed to by the command interpreter's entry point table and are used by some of the DOS commands, they are documented here primarily for the sake of completeness.

49.1 CMDINT - Return & Scan MCR$ line

CMDINT closes files 1-3 if necessary and processes MCR$ just as it would a command line entered by an operator at the keyboard. (This results in executing the program indicated by the command line.)

Entry point: 01165

Parameters: MCR$ (an 80 byte area of memory starting at 01400) should contain a string resembling a command line terminated with a 015.

Exit conditions: Does not return

49.2 DOS$ - Return & Display Sign On

DOS$ first loads the RAM screen, if there is one, with the character set contained in SYSTEM6/SYS (or CHARSET/SYS if it exists). Once the RAM display has been loaded, it is not loaded until either another bootstrap from cassette, or the appropiate DOS function is invoked by a DOS program. DOS$ then causes a program which has been AUTO'd to be executed. If no programs are set for auto-execution, the DOS sign-on is displayed, files 1-3 are closed if necessary, and the familiar "READY" message displayed. Note again that any traps set by the user program (e.g. via TRAP$) are not cleared unless the DOS is reloaded. This implies that if a user program sets any of the traps and wishes to return via DOS$, NXTCMD, or CMDAGN, it must first clear any traps it has set to prevent the DOS from going astray. DOS$ is the normal starting point of the DOS when a bootstrap operation or a jump to BOOT$, EXIT$, or ERROR$ occurs.

Entry point: 013400

Parameters: none

Exit conditions: Does not return

49.3 NXTCMD - Return & Say "READY"

NXTCMD causes files 1-3 to be closed and displays the familiar DOS "READY" message.

Entry point: 013403

Parameters: none

Exit conditions: Does not return

49.4 CMDAGN - Return & Give Message

CMDAGN causes files 1-3 to be closed and displays a user-supplied message before returning to the command interpreter.

Entry point: 013406

Parameters: HL = address of DSPLY$-format string
DE unused; string should position cursor

Exit conditions: Does not return

DOS CHAIN facility aborts if active

49.5 GETSYM - Get Next Symbol from MCR$

GETSYM causes the next sequential symbol in MCR$ to be scanned off and stored in an 8-byte field called SYMBOL located at 013472. The starting byte scanned in MCR$ is pointed to by INPTR, a byte at location 013455. (INPTR is the LSB of the current byte in MCR$.) The symbol (leading spaces are ignored) must contain only upper case alphabetic or numeric characters. The first illegal character encountered terminates the scan; the illegal, terminating character is stored for the user's inspection (at SYMBOL+8) and SYMBOL is padded on the right with spaces if necessary. If the symbol is longer than eight characters, the first eight only are used; remaining characters, through the terminator, are scanned but not stored. (The terminator is stored at SYMBOL+8 in any case.) On exit, INPTR points after the terminating character unless the terminator is an 015 or a semicolon, in which case INPTR points to the terminator.

Entry point: 013411

Parameters: INPTR => current byte in MCR$, LSB

Exit conditions: SYMBOL = 8-byte symbol as described above
A, SYMBOL+8 = terminator character
INPTR => byte after symbol terminator in MCR$
(except as noted above)
All other registers indeterminate

49.6 GETCH - Get the Next Character from MCR$

GETCH obtains the next character from the Monitor Communication Region (MCR$) and returns it in A. The address of the character to be returned is obtained by using the most significant byte of the address of MCR$ (which is contained within one page) and the contents of INPTR (location 013455) as the LSB. On exit, if zero is true, A = 015 or a semicolon, and INPTR is not incremented (INPTR is never bumped past an 015 or a semicolon); if zero is false, A is not an 015 or a semicolon and INPTR is incremented.

Entry point: 013414

Parameters: INPTR = LSB of address of byte (see above)

Exit conditions: A = character from MCR$
ZERO TRUE/FALSE as described above
B = entry value of INPTR
C,D,E unchanged

49.7 GETAEN - Get Auto-Execute Physical File Number

GETAEN returns the physical file number of the file (on the logical drive specified in C) which is set to be auto-executed by the DOS.

Entry point: 013417

Parameters: C = Logical Drive

Exit conditions: Carry true if I/O error reading the CAT
otherwise, A = auto-execute PFN (0=none)
Zero true if a-e PFN not set
Zero false if A is valid a-e PFN
All other registers indeterminate

49.8 PUTAEN - Set or Clear a File to be Auto-Executed

PUTAEN either sets or clears the auto-execute PFN stored in the CAT on the disk in the logical drive specified in C. The change becomes effective upon the next time DOS is entered at DOS$, either by depressing the RESTART key, the auto-restart tab being punched out of the rear cassette and the processor halted, or jumping to EXIT$, ERROR$, BOOT$, or DOS$.

Entry point: 013422

Parameters: A = PFN to be auto-executed (0 to clear)
C = Logical Drive

Exit conditions: All registers indeterminate
Carry true if I/O error updating CAT

49.9 GETLFB - Open the User-Specified Data File

GETLFB opens logical file specified in B using the file name, extension, and drive select code, stored in the indicated LFT entry, in the normalized form described in the section on the Command Interpreter. The extension, if blank, is assumed to be "ABS". Note: The logical drive specification field is ignored, since the drive select code field is used instead. If an error occurs, carry is true on return and HL points to a DSPLY$ format string complete with cursor positioning bytes and one of the following messages:

```
NAME REQUIRED. (first byte of name field is blank)
INVALID DEVICE. (select code = 0376; :DRn wrong)
NO SUCH NAME. (file not found; the file must exist)
```

Each of the above messages is preceeded by control bytes: 011,0,013,11,023 and followed by an 015. If carry is false upon return, the file named has been successfully opened as the requested logical file number.

```
Entry point:      013425

Parameters:       B = LFN
                  In LFT specified by LFN; see above

Exit conditions:  Carry false if file successfully opened
                  All registers indeterminate
                  Carry true and HL => message if OPEN failed
```

49.10 PUTCHX - Store the Character in "A"

PUTCHX stores the A register at the memory location pointed to by HL, increments HL, and decrements a byte counter maintained in E.

```
Entry point:      013433

Parameters:       HL = address where A is to be stored
                  A  = byte to be stored at HL
                  E  = count to be decremented

Exit conditions:  B,C,D unchanged
                  E  = entry value - 1
                  HL = entry value + 1
```

49.11 PUTCH - Alternate Version of PUTCHX

PUTCH is like PUTCHX except it starts by setting the most significant bit of A to zero and that if A then contains a space (040) it immediately returns zero true; in which case A is not stored, HL not incremented, and E not decremented.

Entry point: 013430

Parameters: same as PUTCHX

Exit conditions: same as PUTCHX except as described above

49.12 PUTNAM - Format a Filename from Directory

PUTNAM is a routine which extracts a name, extension and physical file number for a directory entry and puts them into a place in the command interpreter called "NAME" (located at 013513; the field is 19 bytes long and followed by an 03.) Since this routine is used by the CAT command, the format of the names produced by PUTNAM should be familiar to all DOS users.

Note that on entry, only the <u>most</u> significant 4 bits of C are used, and that CURLOC (location 013463) is to contain the two-byte PDA of the directory sector (LSB,MSB).

Entry point: 013436

Parameters: the directory sector in the disk buffer
B = LFN indicating which buffer
C = PFN of entry being extracted
CURLOC = PDA of directory sector

Exit conditions: CURLOC unchanged
disk buffer unchanged
B unchanged
all other registers indeterminate
ZERO TRUE: file does not exist

49.13 MOVSYM - Obtain the Symbol Scanned by GETSYM

MOVSYM moves the eight-byte SYMBOL described in the section on GETSYM into the eight-byte area pointed to by DE.

Entry point: 013441

Parameters: D,E = address of user's eight-byte area

Exit conditions: B unchanged. All other registers indeterminate.

49.14 GETDBA - Obtain Disk Controller Buffer Address

GETDBA extracts the current disk buffer address in the format acceptable to GETR$ from one of the four LFT entries. It does this by getting the BUFADR from the specified LFT entry and subtracting three from it. On return, H is the address MSB pointing into the command interpreter data area.

Entry point: 013444

Parameters: B = LFN (0,16,32,48)

Exit conditions: A = BUFADR as described above
H as described above
B,C,D,E unchanged

49.15 SCANFS - Scan Off File Specification

SCANFS scans a file specification of the form FILENAME/EXT:Drive (as discussed under FILE names) pointed to by HL into a 16 byte area pointed to by DE. The area pointed to by DE is treated as an LFT entry, that is, the first byte is a drive select code (0376 meaning invalid drive spec, 0377 meaning unspecified drive spec, or the binary drive number), the second byte is 0377 indicating the file is closed, bytes 3 thru 10 are the file name (blank if not given), bytes 11 thru 13 are the extension (blank if not given), and bytes 14 thru 16 are the normalized drive spec (blank if not given). The scanned drive spec may be 2 to 7 characters long; the first character must be "D", the second may be "R", and the remaining must be digits. Therefore ":D0" and ":DR00014" are both legal representations. The normalized represention consists of a "D" followed by "R" and the single digit given or "D" followed by the two digits given; for instance, the above examples in normalized form would be "DR0"

and "D14" respectively. Scanning a VOLID results in the correct drive number being stored in the normalized drive spec field. The scan is terminated by any non-alphanumeric character other than ":" or "/".

Entry Point: 013447

Parameters: DE => "LFT TABLE" entry
HL => string to be scanned

Exit Conditions: DE => byte following "LFT TABLE" entry
HL => byte after terminator (unless 015 or ";" in which case it points to terminator)

49.16 TCWAIT - Test controller memory & wait

TCWAIT is the point in the COMMAND INTERPRETER where it loops testing the disk controller buffer memory while waiting for a command to be keyed in. It is only to be used by the CHAIN command to trap programs returning to DOS.

Entry Point: 013452

Parameters: none

Exit Condition: does not return

CHAPTER 50. USER SUPPORTED INPUT/OUTPUT

When the user desires to use I/O devices other than the keyboard, display, disk, or cassettes, he will use a routine that is not part of the operating system. Many of these devices (for instance, the communications channel) will be serviced by foreground processes which run with interrupts disabled. However, if the user does access an I/O device from a background process, he must realize that as long as interrupts are enabled, some other device can be addressed by a foreground routine. For this reason, the user must disable interrupts between the time he addresses his device and the time he uses it. To reduce the amount of foreground processing real time jitter (discussed earlier) as much as possible, the aim in writing background I/O routines should be to minimize the amount of time that interrupts are disabled. This implies that devices accessed from background programs must be addressed every time they are used. For example:

```
GETBYT  EI              Enable interrupts in case
        LA      DEVADR     looping
        DI              Disable interrupts
        EX      ADR     Address the device
        IN              Get the device status
        ND      2       Check for required bits
        JTZ     GETBYT  Wait if not set
        EX      DATA    Else get the byte
        EI              Enable interrupts after
        IN                 the data input
        RET
```

Note that a little cheating on time was done in the interest of program length. Since the INPUT in DATA mode was done without enabling interrupts, re-disabling them and re-addressing the device was not necessary. One should be judicious in the trade off employed in exercising this freedom.

Note: The user must not do I/O to the disk controller from foreground-driven routines or results can be unpredictable. The DOS disk drivers allow user foreground routines to get control in the midst of a disk I/O operation, under the assumption that the foreground routine will not do anything to the disk controller which would confuse it.

CHAPTER 51. ERROR MESSAGES

PARITY FAILURE DURING READ
: A parity fault occurred while a disk data record was being read.

PARITY FAILURE DURING WRITE
: A parity fault occurred while a disk data record was being written.

RECORD FORMAT ERROR
: The physical file number or logical record number in the record read did not match the values contained in the logical file table.

RECORD NUMBER OUT OF RANGE
: The record accessed had a logical record number less than zero or, during reads, was outside the physical space allocated to the file.

WRITE PROTECT VIOLATION
: An attempt was made to write on a file that had its write protection bit set.

DELETE PROTECT VIOLATION
: An attempt was made to delete a file that had either its write or delete protection bit set.

FILE SPACE FULL
: An attempt was made to allocate space when either the disk was physically full or no more segment descriptor slots were available in the RIB for the given file.

DRIVE OFF LINE
: The drive went off line after the file was opened.

LOGICAL FILE NOT OPEN
: An attempt was made to use an entry in the logical file table that was not opened for use with some file.

INVALID LOGICAL FILE NUMBER
: A routine was called with the logical file number parameter not zero through three.

INVALID DRIVE NUMBER
A routine was called with the drive number not zero through the defined drive number limit (or 0377, if allowed).

INVALID TRAP NUMBER
The TRAP$ routine was called with a trap number not between zero and seven.

FAILURE IN SYSTEM DATA
An unrecoverable parity error occurred while the system was dealing with one of the disk tables or a retrieval information block, or a RIB with incorrect format was accessed.

INVALID PHYSICAL FILE NUMBER
A physical file number reserved for the system was illegally referenced.

INTERNAL SYSTEM ERROR
The error message routine was parameterized with an invalid error message number!

ERROR X ON DECK Y
A cassette routine error has occurred. The X indicates the type of error according to the following table:

D - parity error
E - end of tape
F - end of file
G - unfindable file
Z - write failure

CHAPTER 52. ROUTINE ENTRY POINTS

These entry points are contained in a file called DOS/EPT.

Loader Routines

01000	BOOT$	reload the operating system
01003	RUNX$	load and run a file by number
01006	LOADX$	load a file by number
01047	GETNCH	get the next disk buffer byte
01052	DR$	read a sector into the disk buffer
01055	DW$	write a sector from the disk buffer
01060	DSKWAT	wait for disk ready
01173	DWNV$	DW$ without write verify 2.3 only

Time-critical Scheduling Routines

01033	CS$	change process state
01036	TP$	terminate process
01041	SETI$	initiate foreground process
01044	CLRI$	terminate foreground process

Symbolic File Handling Routines

01063	PREP$	open or create a file
01066	OPEN$	open an existing file
01071	LOAD$	load a file by name
01074	RUN$	load and run a file by name

Logical File Handling Routines

01077	CLOSE$	close a file
01102	CHOP$	delete space in a file
01105	PROTE$	change the protection on a file
01110	POSIT$	position to a record within a file
01113	READ$	read a record into the buffer
01116	WRITE$	write a record from the buffer
01121	GET$	get the next buffer character
01124	GETR$	get an indexed buffer character
01127	PUT$	store into the next buffer position
01132	PUTR$	store into an indexed buffer position
01135	BSP$	backspace one record

Generalized Processing Routines

01011	INCHL	increment HL
01022	DECHL	decrement HL
01140	ERROR$	close all files, exit chain, and reload DOS
01143	BLKTFR	transfer a block of memory
01146	TRAP$	set a disk error condition trap
01151	EXIT$	reload the operating system
01170	WAIT$	DOS wait-a-while "NOP" routine
07400	DOSFNC	DOS function loader

Keyboard and Display Routines

01154	DEBUG$	enter the debugging tool
01157	KEYIN$	obtain a line from the keyboard
01162	DSPLY$	display a line on the screen

Cassette Handling Routines

010000	TPBOF$	position to the beginning of a file
010005	TPEOF$	position to the end of a file
010012	TRW$	physically rewind a cassette
010017	TBSP$	physically backspace one record
010024	TWBLK$	write an unformatted block
010031	TR$	read a numeric CTOS record
010034	TREAD$	TR$ and wait for last character
010037	TW$	write a numeric CTOS record
010042	TWRIT$	TW$ and wait for last character
010045	TFMR$	read the next file marker record
010050	TFMW$	write a file marker record
010053	TTRAP$	set a cassette error trap
010056	TWAIT$	wait for I/O completion
010061	TCHK$	get I/O status

COMMAND INTERPRETER UTILITY ROUTINES

01165	CMDINT	return to command interpreter & scan MCR$ line
013400	DOS$	return to command interpreter & display sign on
013403	NXTCMD	return to command interpreter & say "READY"
013406	CMDAGN	return to command interpreter & give message
013411	GETSYM	get the next symbol from MCR$
013414	GETCH	get the next character from MCR$
013417	GETAEN	get the auto execute PFN
013422	PUTAEN	set the auto execute DFN
013425	GETLFB	open the user-specified file (LFN in B)
013430	PUTCH	store the nonblank character in the A register
013433	PUTCHX	store the character in the A register
013436	PUTNAM	format a filename from a directory block

```
013441    MOVSYM    obtain the symbol scanned off by GETSYM
013444    GETDBA    obtain the disk controller buffer address
013447    SCANFS    scan off a file specification
013452    TCWAIT    test controller memory and wait for command
013455    INPTR     byte pointer for use with GETCH
01200     DOSFLAG2  DOS FLAG byte #2  2.3 only
000053    BOOTDRIV  drive DOS was booted from
01400     MCR$      Monitor Communication Region
01544     LFT       Logical File Table
```

Internal DOS Equivalences

```
00004     DOSPFN    PFN for use by DR$ and DW$
00005     DOSPDN    PDN for use by DR$ and DW$
00026     DOSPTR    BUFPTR used by GETNCH
00027     SDFLAG    Sub-directory existance flag
00030     SDNR      sub-directory numbers (1 per drive)
01377     DOSFLAG   DOS flag byte
4         TFT       temporary file table
0<4       LF0       logical file #0
1<4       LF1       logical file #1
2<4       LF2       logical file #2
3<4       LF3       logical file #3
```

LOGICAL FILE TABLE DESCRIPTION

```
0         PFN       (1) PHYSICAL FILE NUMBER
1         PDN       (1) PHYSICAL DRIVE NUMBER AND PROTECTION
2         LRN       (2) NEXT LRN TO BE DEALT WITH
4         BLRN      (2) FIRST LRN WITHIN CURRENT SEGMENT
6         CSD       (2) CURRENT SEGMENT DESCRIPTOR
8         RIBCYL    (1) PDA (MSB) OF RIB
9         RIBSEC    (1) PDA (LSB) OF RIB
10        MAXLRN    (2) LARGEST LRN REFERENCED
12        LRNLIM    (2) RESERVED FIELD (INITIALLY ZERO)
14        BUFADR    (1) CURRENT CONTROLLER BUFFER ADDRESS
15        XXXXXX    (1) NOT USED
```

DOS MEMORY MAPPING

```
000000    LDRAD$    SYSTEM LOADER
001000    DOSAD$    DOS RESIDENT
004000    OVLAD$    DOS OVERLAYS
005400    DSPAD$    CRT WRITE ROUTINE
005572    KEYAD$    KEYBOARD READ ROUTINE
006000    DEBAD$    DISK DEBUG
07400     FLDAD$    DOS FUNCTIONS PAGE
010000    CASAD$    CASSETTE TAPE DRIVERS
```

```
013400    CMDAD$     COMMAND INTERPRETER
017000    COVAD$     COMMAND INTERPRETER OVERLAYS
000053    BOOTDRIV   DRIVE DOS WAS BOOTED FROM
```

DOS Keyboard/Display Routine Control Byte Equates

```
3         EOS        end of string, no CR/LF
011       H          horizontal position follows
013       V          vertical position follows
015       EOL        end of line, CR/LF
021       ECF        erase cursor to end of frame
022       ECL        erase cursor to end of line
023       R          roll screen up one line
11        BL         number of bottom line on screen
0         TL         number of top line on screen
79        RC         number of rightmost column on screen
0         LC         number of leftmost column on screen
```

DOS FLAG byte #1 (location 01377)

```
1<7       ABTIF      1... .... abnormal program completion
1<6       NETACT     .1.. .... INTERNET facility active
1<5       UBOOT      ..1. .... disk was booted from disk
1<4       CHACT      ...1 .... chaining active
1<3       IS55AVL    .... 1... 5500 instructions available
1<2       PSACT      .... .1.. PS active
1<1       RAMAVL     .... ..1. RAM display available
1<0       ROMBOOT    .... ...1 BOOTSTRAP loaded from ROM
```

CHAPTER 53. DOS QUESTIONS AND ANSWERS

Q. When I write my program, where should I place it in memory?
A. The best address to specify in your SET statement in an assembly language program is 017000. This allows your program full access to the routines in the DOS command interpreter and allows your program to return to the DOS through the NXTCMD and CMDAGN entry points. If the 8.5 K remaining above 017000 is inadequate for your program's needs, you could perhaps start your program at 010000 (assuming your program will not be using the DOS cassette handling routines or command interperter routines.)

Q. Where should I put the data areas used by my program, at the beginning or at the end?
A. Experience in programming the Datapoint computers has found that generally it is best to put program data areas before the program itself. One advantage of this approach stems from the fact that programs can often be made shorter if most or all of the most commonly used data items are contained within one page of memory, eliminating the need to reload the H register as often. Since programs typically start on a page boundary, this automatically means that the first 256 bytes of your data area will be in one common page. Another advantage of this approach is that a person reading a program is frequently aided by seeing the program's data area and error messages, etc., before he plunges into the code itself. This placement also reduces the number of forward references the assembler must contend with. But don't forget to specify the entry point on your "END" statement! The default entry point is to the first byte of code generated. Yours wouldn't be the first program to start executing your data area!

Q. When my program gets control from the DOS, do I need to save the registers so I can restore them before returning to it?
A. No. Under the DOS the saving and restoring of the system's registers by user programs is not necessary.

Q. Talking about returning to the DOS, how should my program do that?
A. When a user program finishes, the normal termination is by jumping to EXIT$.

Q. Does it matter if my program returns to the DOS (to EXIT$, NXTCMD, CMDAGN, or wherever) with the stack at a different level than when my program started? In other words, if my program calls several levels down into subroutines and the subroutine jumps to EXIT$, will that mess things all up?
A. No. Since the stack wraps around, the level is always relative and it makes no difference what is in the stack when the user returns control to the DOS.

Q. What is the best way to pass parameters to my subroutines? Is there any official convention for this?
A. There is no "official convention" for parameter passing. However, experience with programming under the DOS suggests that passing parameters in the registers as typified by the DOS file handling routine parameterization is both efficient and convenient to use. The DOS convention that abnormal returns from subroutines are indicated by carry being true on exit (and further information indicated by the zero condition being true or false) also has proven to be a very handy technique, and one which user programs can probably make profitable example of.

Q. Can I update my data files with EDIT?
A. Most data files cannot be EDITed. EDIT produces a space and record compressed output, regardless of input file format. Also, EDIT will segment records longer than 79 bytes into two or more records. Only if your data file is compressed and has 79 byte (or smaller) records can EDIT be used on it. In general, do not EDIT a data file; write an update program.

Q. What's going on when I run a program and nothing happens; the machine just comes back with READY?
A. This is the system's normal action when it finds an unloadable program. Something - a parity error, a non-object record - made the program unloadable. Try COPYing the program to clear any parity errors. APP the program to test for non-object records. It may be necessary to re-assemble the program or get a new object file from tape or another disk.

Q. I just got a disk that is completely shot; DOSGEN flags every cylinder.

A. Sometimes you can receive the wrong type of disk, due to ordering errors or packaging errors at the manufacturer. If a disk fails to work, pull it out and inspect it.

On mass storage (11-platter) disk packs, look at the bottom platter. The disks Datapoint uses have one index notch on the edge of the platter. Some disks have hard sectoring and will have 24 notches along the edge of the platter. If a Datapoint drive gets one of these disks it thinks every revolution is 24 revolutions! Also check the filter in the center of the pack. The filter must be in place or dirt will get in the drive; it must be clean or there won't be enough air flow to float the r/w heads correctly.

On a cartridge disk, look at the metal spindle in the center of the bottom side of the pack. There should be 25 notches along the edge of the spindle, evenly spaced except for one pair close together. If the spindle doesn't look right, the drive can't handle it.

On diskettes there isn't much inspection possible. Verify that the diskette is a 128-byte sector, IBM compatile diskette and that it wasn't put into the drive upside-down.

CHAPTER 54. 5500 ROMGUIDE

54.1 System ROM Functions

54.1.1 Introduction

The DATAPOINT 5500 DEBUG is a ROM resident program whose immediate accessibility creates a flexible interface between User and machine. This guide is intended to provide the 5500 User with that information essential to the use of the ROM-DEBUG. With this powerful hardware feature the User should quickly develop an aggressive debugging tool.

54.1.2 Startup Procedure

There are four methods of entry to DEBUG:

(1) Forcing entry through manual intervention.
(2) Entry through a BREAKPOINT set by DEBUG.
(3) Entry through a BREAKPOINT imbedded in the User Program.
(4) Entry as the consequence of a RETURN from a DEBUG Call Command.

TO FORCE ENTRY INTO DEBUG, DEPRESS IN SEQUENCE: DISPLAY, RUN, RESTART; keeping each key depressed until all three are down. Then release RUN. This will bring up the DEBUG display and commands may be entered.

Note that depression of the DISPLAY key during the transition from Boot Block read-in to execution during REBOOT will also cause entry into DEBUG.

54.1.3 Saving the Machine State

When DEBUG is entered through console intervention, most of the User's program state is undisturbed. What is not saved is the state of the interrupt enable flip-flop (interrupts are disabled), the state of the base register or sector table (these two are not changed upon entry to DEBUG), the state of the ALPHA/BETA Mode flip-flop (all registers are saved), the state of the I/O system (what device is addressed and the state of its status/data select flip-flop), and the bottom two stack locations.

What gets saved are the ALPHA/BETA Mode registers and condition code flip-flops, the Program Counter (PC) and 016 Stack locations.

Note that there exist default values upon exit from DEBUG for:

(1) ALPHA/BETA Mode flip-flop
(2) Currently addressed device and its Status/Data Mode flip-flop

These can be set using DEBUG commands ('A', 'G' and 'R').

54.1.4 Display Format

The 5500-DEBUG display consists of four lines and occupies the bottom-right corner of the screen.

```
        AAAAAA     : CURADR
        *  NNN     : ASCII,8 BIT OCTAL C[CURADR]
        MMMMMM     : LSB,MSB ADDRESS FORMED AT CURADR.
     nnnnnnn*      : COMMAND INTERPRETER
```

The first (top) line shows the current sixteen bit address.

The second line contains both an ASCII (One character shown as *) and an 8-bit octal (Three characters shown as NNN) representation of the contents of the current address byte.

The third line contains an octal representation of the 16-bit value whose LSB is at CURADR and whose MSB is at CURADR+1. (This is the address format used by JMP, CALL and DA mnemonics).

54.1.5 The Command Interpreter

The bottom line of the display is an interpreter used to edit and input commands to DEBUG. The blinking cursor signifies that the Command Interpreter is awaiting user input.

Data is entered serially into the input display buffer. The cursor is displaced to the right successively as this occurs. The Backspace key erases the character most recently entered, shifting the entry cursor to the left one space. The cancel key deletes the entire entry.

All commands are single characters. Commands which accept input arguments are preceded by the argument, which is entered in octal. Not all commands require an input argument. The last character input to the interpreter must be a legal command. Illegal input is ignored, evoking a BEEP from the 5500. Commands are executed upon their entry into the interpreter (no ENTER key is required), with the current contents of the entry line being cleared. Upon command completion the cursor reappears, awaiting further input.

54.1.6 Command Syntax

This explanation of the command syntax uses the following notation:

nnnn... Indicates an optional sequence of octal digits not to exceed the number of n's given.

If input argument contains more than eight bits of significance special results will occur. In general what will happen is that two bytes of memory will be affected by the command, either a register pair or a memory address in LSB,MSB format.

12345 There exists a set of special commands whose accidental execution is inhibited by the requirement that they contain this unique argument.

54.1.7 Input Command List

nnn A Address the given or current I/O device. No check is made on address format. STATUS is displayed as C[CURADR]. NOTE that the CURRENT Device is readdressed and put into the mode last accessed (Data mode if 'F' or 'G' have been executed subsequent to last 'A' command) prior to resumption of execution through Call, Exit, Jump or User Exit Commands if the last I/O DEBUG command executed is A.

nnnnnn B Set a BREAKPOINT at the given or current address. Upon BPT execution the state of the machine is saved, the memory location at which the BPT was set is restored to its original value and the corresponding BPT table entry is cleared.

The following notes reference the use of the 'B' command.

Overlay BREAKPOINT will not loop. That is: It is not possible to successively set a BREAKPOINT in the same memory location in order to iterate the execution of a program loop. To iterate BREAKPOINT through looping sequence requires 'double Breakpoint'. Twenty BREAKPOINTs can be active at any one time. Note that BPT's DISABLE INTERRUPTS and leave them disabled prior to resumption of execution through Call, Exit, Jump or User Exit commands. This is done to enable testing of Foreground routines with DEBUG. If it becomes necessary to use DEBUG with Interrupts Enabled, the user can place an EI instruction in a main loop of his program. Note that it is impossible for the machine to determine its current register (ALPHA/BETA) mode. Therefore the command mode flip-flop is set to ALPHA when a BP is encountered. If User wishes to test code written in BETA Mode it is necessary that he manually put the Machine in BETA Mode (With 'R' Command) prior to resumption of execution through Call, Exit, Jump or User Exit commands. Similarly, he may have to address the proper I/O device (with A) and perhaps put it into DATA Mode (with G) before continuing executio from a breakpoint. Note that DEBUG will not set a BREAKPOINT over another BREAKPOINT.

nnnnnn C Call the given or current address. The Machine State is restored before execution control is passed to the

Subroutine. A RETURN from the Called Subroutine causes re-entry into DEBUG and hence, causes the Machine State to again be saved.

D Decrement the current address value. Any Input Argument will be ignored.

E Continue execution from a forced or BREAKPOINT entry into DEBUG. Machine State is restored prior to resumption of execution. The interrupts are left disabled. The register mode is set to the last R value (initialized to ALPHA Mode upon BPT or on forced entry), the base register and sector table are not changed, and the I/O device is addressed and optionally set to DATA mode. Note that this command does not depend on any Display Parameters. Prior DATAPOINT Debug software used CURADR as an exit address pointer.

nnn F Fetch next data byte from current or given I/O device. Command will automatically put device in DATA Mode and the device will subsequently be put in data mode when the E command is given.

nnn G Go to data mode in the current or given I/O device when the E command is given.

H * Not Used. *

I Increment the current address value. Any Input Argument will be ignored.

nnnnnn J Jump to the given or current address. Machine State is restored prior to resumption of execution.

12345K Set ASCII keyin mode. Will allow ASCII data to be entered into CURADR in auto-increment mode (i.e. will update CURADR). BACKSPACE moves CURADR back and displays its contents. DELete moves CURADR forward and displays its contents. CANCEL causes a return to normal mode.

L Link to the address pointed to by the Current Address. CURADR is replaced by line 3 (the 16-bit LSB,MSB address formed at CURADR,CURADR+1). The remaining display parameters are updated appropriately. Note that initial display state upon entry into DEBUG can be regenerated by performing the 'S' command, followed immediately by the 'L' command.

(nnn)nnn M Modify the contents of the current address location. If the value of the Input Argument exceeds eight bits of significance, two memory locations will be modified, treating the input argument as an address in LSB,MSB Format for JMP and DA. (A CLICK is sounded to notify the operator of this action.)

N * Not Used. *

O * Not Used. *

nnnnnn P Load the Base Register with the 8-bit value (nnnnnn - 0100000)

12345Q Load the Sector Table. CURADR => Table whose first byte equals the number of entries to be loaded. The following bytes contain arguments to be loaded into the Sector Table.

R Switch Alpha/Beta Mode register display. The ASCII character displayed after command execution tells the current display mode: A=ALPHA, B=BETA.

nn S Display the specified stack item (up to 015 Octal). Note: P, 0 => 014 Octal after RESTART. (Since RESTART PUSHes P onto the top of the STACK.)

12345T Start memory test. Displays Memory Size and Pass Counter in right-bottom corner of screen. Maintains running display of Test Failures.

U User mode execute. Command sets USER Flag then executes 'E' Command.

nnn V EX COM4 DEV must be Addressed with A command.
nnn W EX WRITE STATUS is displayed
nnn X EX COM1 after command issue.
nnn Y EX COM2 'nnn' is the current output byte.
nnn Z EX COM3

nnnnnn Set Current Address to nnnnnn. Command has no efect unless it is preceeded by an Input Argument.

<Cancel> Cancel entry line.

<BSP> Backspace on entry line.

(nnn)nnn . Modify the contents and then increment the current

address. If Input Argument has more than eight bits of significance, two memory locations are modified, treating the argument as an address in LSB,MSB Format. (a CLICK is sounded).

(nnn)nnn ^ Modify the contents and then increment the current address. If input argument is null, the last non-null value given is used. If 'last value' exceeded eight bits of significance, two memory locations will be modified. (a CLICK is sounded).

Clear all active (DEBUG set) breakpoints, restoring values.

(nnn)nnn a Display register and pair (with modify option).
nnn b If Input Argument exceeds eight bits, the command modifies a register
(nnn)nnn c pair. Pairs must be modified as indicated.
nnn d (LSB register specifies a pair e.g. L for HL)
(nnn)nnn e Note that to get these command letters
nnn h (lower case) the shift key must be depressed
(nnn)nnn l when the command key is struck.
nnn x

f Condition flags: 7=>C; 6=>S; 1=>-Z&-S; 0=>-Z&-P. The bit pattern which displays the condition flags will replicate the previous state when added to itself. This is probably the easiest way to determine the actual values for Z and P.

54.1.8 DEBUG Command Summary

```
     nnn A - Address the given or last I/O device
  nnnnnn B - Set a break point at the given or current address
  nnnnnn C - Call the given or current address
         D - Decrement the current address
         E - Continue execution
     nnn F - Fetch the next data byte from current I/O device
     nnn G - Go to data mode in the current I/O device
         I - Increment the current address
  nnnnnn J - Jump to the given or current address
         K - Set ASCII keyin mode (12345K)
         L - Link to the address pointed to by the current address
(nnn)nnn M - Modify the contents of the location ponted to by the
             current address
  nnnnnn P - Load the page basing register
         Q - Load the sector table
         R - Switch from alpha to beta mode or vice versa
      nn S - Display the specified stack item
         T - Start memory test ('12345T')
         U - User mode execute
     nnn V - EX COM4 to last I/O device
     nnn W - EX WRITE to last I/O device
     nnn X - EX COM1 to last I/O device
     nnn Y - EX COM2 to last I/O device
     nnn Z - EX COM3 to last I/O device
(nnn)nnn a - Display or update the contents of the A-register
     nnn b - Display or update the contents of the B-register
(nnn)nnn c - Display or update the contents of the C-register
     nnn d - Display or update the contents of the D-register
(nnn)nnn e - Display or update the contents of the E-register
         f - Display the flags (adding the number to itself will
             restore the flags)
     nnn h - Display or update the contents of the H-register
(nnn)nnn l - Display or update the contents of the L-register
     nnn x - Display or update the contents of the X-register
(nnn)nnn . - The equivalent of an M followed by an I
  nnnnnn <enter> - Change the current address
         # - Clear breakpoints
(nnn)nnn ^ - Modify and Increment using last value
```

ROM DEBUG DISPLAY

AAAAAA - The current address (in octal)

AAAAAAA
X NNN
MMMMMM

X - The contents of location AAAAAA (in ASCII)
- or the contents of the specified register (in ASCII)

NNN - The contents of location AAAAAA (in octal)
- or the contents of the specified register (in octal)

MMMMMM - The contents of locations AAAAAA+1 and AAAAAA respectively, concatenated into one octal number
- or the contents of a register pair concatenated into one octal number (XA, BC, DE, HL)

APPENDIX A. DOS.A AND DOS.E

DOS.A and DOS.E are two Disk Operating Systems supporting Datapoint computers operating in conjunction with up to four 9350 series cartridge disk drives.

A.1 Planning for DOS.A/DOS.E

DOS.A and DOS.E are both alike in many respects. Both use the 9350-series disk cartridge drives, and they are each almost identical to the other operationally. The primary operational difference between DOS.A and DOS.E is that DOS.E will support the Datapoint Partition Supervisor, PS, released separately. Operating under PS, DOS.E permits the concurrent execution of more than one partition.

A.1.1 DOS.A Physical Configuration

DOS.A operates in either Datapoint 2200 or 5500 series processors with at least 16K of memory and one or more 9350-series disk drives. Use of a single 9350-series drive is possible, but a multi-drive system should be available for backup and support purposes. Some consideration must be given to the question of copying files from one disk to another, and most systems incorporating the 9350-series disks will have files large enough to make it impractical to transfer them from one disk cartridge to another one cassetteful at a time.

An option which should be considered during the systems planning phase is the High Speed, or so-called "RAM" Display Option. This option is strongly recommended, as it can substantially increase total system throughput (especially on batch-processing oriented systems) at a very small additional cost. This option is field-installable, and is standard equipment on Datapoint 5500 series computers.

A.1.2 DOS.E Physical Configuration

DOS.E differs form DOS.A in that DOS.E requires a 48K Datapoint 5500 computer and two, three, or four 9350-series disk drives attached to a 9357 disk control unit. This enhanced cartridge disk controller contains four times the amount of high speed cache memory contained in the older 9350-series controller, as well as additional hardware features to facilitate the multiprogrammed environment available under PS/DOS.E. (Older 9350-series disk controllers can be easily field-upgraded to 9357 levels.)

A.2 Disk Drives

DOS.A and DOS.E support a maximum of four 9350-series cartridge disk drive units. These outstandingly reliable disk drives, the standards in their field, have established an enviable record for availability and percentage uptime that few other cartridge disk units even hope to achieve.

A.3 Disk Media

The Datapoint 9350-series disk drives use a single platter disk cartridge, media-compatible with the IBM 2315 disk cartridge. Data is recorded in 203 concentric circles on each of the two recording surfaces. Each such circle is referred to as a track.

The disk itself is enclosed within a plastic cartridge which helps to protect it from bumps, jolts, and contaminants while it is not in place in the disk drive. This cartridge and the care taken in its handling and storage are of prime importance in helping to eliminate disk errors and parity failures that contamination can cause.

A.4 Loading and unloading Disk Cartridges

Loading and unloading cartridges from the 9350-series drives is simplicity itself. At the top of the front side of the drive is the cartridge access door. Pulling out and down on the handle opens this door. The cartridge is inserted into the cavity with the "tongue" of the cartridge on top and entering first. When the cartridge is fully inserted, the cartridge access door is closed and the rocker switch marked "LOAD/RUN" is switched to the "RUN" position. When the switch is moved to "RUN", the following things occur:

1) The cartridge access door is locked closed.

2) The indicator lamp marked "LOAD" on the front panel of the drive is extinguished;

3) The disk pack accelerates to its rated speed of 1500 rpm, at which time the indicator lamp marked "READY" lights up. When this lamp lights up, it indicates that the disk has come on-line to the Datapoint computer.

Removing a cartridge which is no longer needed from a drive is a simple reversal of the above steps. First, the "LOAD/RUN" switch is moved to the "LOAD" position. The drive immediately goes off-line to the computer and is swiftly braked to a smooth stop. When the disk comes to a full and complete stop, the door is unlocked and the "LOAD" indicator lamp comes on. At this time, the cartridge access door can be opened with a gentle tug, after which the cartridge simply slips right out. The cartridge should be stored in a suitable storage rack; it should never be left in a place where it might slip and fall onto a hard surface, such as a floor.

A.5 Switches and Indicators

The uncluttered and modern appearance of the Datapoint computers in conjunction with the 9350-series disks permit their use in a wide range of environments, even front-office use if so desired. Few disk-based systems offer such a versatility of placement.

The current cartridge disk drive, manufactured by Wangco, uses a small cluster of controls in the lower right-hand corner of the disk drive front panel. There is a thumbwheel switch for physical drive number selection, which is set at installation and should not be moved thereafter. The rocker switch marked "RUN" and "LOAD" controls disk loading as described above; the "READY", "LOAD" indicator lamp is immediately below this switch. The leftmost controls are a pair of rocker switches marked "PROT CART" and "PROT FIXED". These switches control the write protection status of the cartridge disk and the fixed disk inside the drive. When the indicator lamp behind one of these switches is lit, the corresponding disk is write-protected. The protection can be changed at any time by changing the switch position.

The older cartridge drive, manufactured by Diablo, has only one single rocker switch, (the LOAD/RUN switch which has been previously described) and four color-coded indicator lamps. The

first of these, a white lamp marked "LOAD" comes on to indicate that the drive is ready to have a disk cartridge inserted or removed. The second lamp, a yellow one marked "READY" indicates that the cartridge in place has come up to speed and is on-line. The third lamp, an orange one marked "CHECK", is an error indication. This lamp is rarely if ever seen illuminated. If it does light up, taking the drive offline and back may help (switching the LOAD/RUN switch to LOAD and back). If that does not work, try powering down the entire system and then turning it back on again, using the main power switches. If the CHECK condition still is not cleared, call the Datapoint Customer Support Center for technical assistance. The fourth red lamp is marked "PROTECT", and when it is illuminated the computer cannot write on the disk in that drive. The disk is protected each time it is brought to RUN status. Depressing the PROTECT button extinguishes the indicator lamp and write-enables the disk. The disk can be re-protected only by switching the LOAD/RUN switch to LOAD and back to RUN.

A.6 Care and Handling of Disk Cartridges

Disk cartridges for the 9350-series disk drives are precision assemblies and must be treated with some care. It is highly important that they not be dropped, mishandled, or contaminated with dust or other pollutants. The cartridges should be stored in an appropriate storage rack, in an area free from dust and in an environment similar to that where the drives are installed (preferably in the same room with the computer). Users should be very careful to never allow anything to contact the oxide surface of the disk itself.

If the cartridges are shipped by common carrier, they should be repackaged in their original, protective shipping carton and marked "FRAGILE". Disk cartridges should <u>never</u> be mailed by Parcel Post. Upon receipt of a disk cartridge, if there is any evidence of damage the cartridge should not be used until it has been inspected and approved for use by a Datapoint customer engineer.

In addition, any cartridge which has been in a non-computer room environment should be allowed to equalize temperatures in the room with the computer for 24 hours before use if at all possible before attempting to read or write data on the cartridge. In an emergency, placing the cartridge onto a drive and letting it spin up and run for about an hour will usually be adequate, but this procedure should be considered an emergency measure only.

A little care in handling disk cartridges will repay itself several times over in reliable and trouble free service with long life from your disk cartridges.

A.7 Care and Maintenance of the 9350 Drives

Although the 9350 series disk drives are tolerant of contamination levels that would take other drives out of service entirely, they can do their best only when taken proper care of. As with the disk cartridges themselves, cleanliness is of great importance. All efforts should be made to keep the room as dust-free as possible. Since the read/write heads fly so very close to the disk surface, just about 100 millionths of an inch away from the oxide surface, even such small particles in the air as those present in cigarette smoke are apt to cause troubles sooner or later. Any dust that may collect around the disk drives should be regularly cleaned away.

In addition to this user maintenance, the user should also ensure that his local Datapoint customer engineer performs the preventive maintenance procedures outlined in the 9350 series disk drive maintenance manual. These preventive maintenance procedures can be compared to changing the oil and oil filter in the family automobile. An automobile will perform all right for a while without regular oil and filter changes, but sooner or later it will extract a heavy penalty for not having been taken better care of. The same characteristic holds true for disk drives as well.

A.8 Head Crashes

Each of the two heads in the 9350-series disk drive is held against the disk oxide surface by a spring which pushes the head toward the surface with a force of approximately 350 grams. The disk, on the other hand, is spinning at approximately 50 miles per hour relative to the head. The head and disk are kept apart by a micro-cushion of air only about 100 millionths of an inch thick. A head crash occurs when this lubricating air film fails. The main causes of head crashes are foreign particles in the lubricating film, contamination buildup on the surfaces of the disk or read/write heads, or a defective disk surface.

When a head crash occurs, the head rubs directly against the oxide surface of the disk, which frequently loosens more oxide, resulting in further and more severe crashes, and things go progressively downhill from there. Due to the severity of a head crash, not just because of the loss of data on a disk but also due

to the degree of damage to the heads on the drive, it is important to recognize the symptoms of a head crash. In this manner a disk experiencing a head crash can usually be discovered and stopped before the crash reaches catastrophic proportions.

A.8.1 Prevention of Head Crashes

There are three main things that a user can do to help minimize the likelihood of a head crash. These include:

1) Preventive maintenance. Establish a preventive maintenance schedule with your Datapoint customer engineer and stick to it. Make sure that this preventive maintenance gets done. Particularly important is attention to the head/arm assemblies, air filtration system and moving parts.

2) Proper handling and storage of disk cartridges. Disks should be carefully stored in an area free from dust, smoke, and other contamination. Any disks whose cartridges are cracked or broken should be replaced immediately. Disk cartridges should be handled carefully to avoid bumping or dropping. Never insert a dropped cartridge into a drive! Give it to a CE for inspection.

3) Keep the cartridge access door closed. Never leave it open. The longer it is open, the greater the susceptibility to contamination.

A.8.2 Recognition of a Head Crash

In spite of all precautions, chances are that most users will experience a head crash sooner or later. Being able to identify it quickly when it happens can help to minimize the damage. A head crash may be indicated by one or more of the following symptoms:

1) Repetitive hard read or write parity errors. Because of the propagation effect of a head crash, do not move any disk with massive hard parity errors to another drive. If errors persist, then the possibility of a head crash exists and must be investigated.

2) Audible tinkling sound. An audible tinkling sound from the disk, which may progress to a screech, probably indicates a head crash.

3) Visible damage to the disk surface. Any scratch on the recording portion of the disk surface where the aluminum substrate is exposed. Concentric adjacent scratches of any length. A single scratch of over approximately three inches in length. Imbedded particles or an accumulation of loose oxide on the surface. Any of these can indicate that a head crash has occurred.

A.8.3 What to Do if You Have a Head Crash

If you suspect that you have had a head crash, call the Datapoint customer support center at once. In the meantime, observe the following precautions:

1) The disk which was mounted on the drive when the crash occurred should be considered suspect and should not be mounted on any other drive until it has been inspected by the Customer Engineer and approved for use.

2) The drive which experienced the crash should not be used until it has been thoroughly checked by the Customer Engineer. Other disks which are probably okay can be damaged by a drive which has had a crash, since the same drive is apt to crash again with any subsequent disk placed in it until it has been properly serviced.

3) Head crashes should be considered to be contagious. A disk which has crashed may have loose oxide or other irregularities on its surface. If the disk is placed into a different drive, these contaminants are apt to very quickly result in a crash occurring on the new drive as well. Since the loose oxide or whatever can build up on the heads of the drive as well as the disk itself, the drive can carry the contaminants of a bad disk over to any number of good disks subsequently used on it, and these can in turn contaminate other drives.

A.9 Preparing Disk Packs for Use

When a disk cartridge is first received from the manufacturer, it is completely demagnetized. However, unlike the other Datapoint Corporation disk drives, on the 9350 series drives the position of the sectors on the disk surface are determined by the sector timing slots around the edge of the disk's hub. Therefore, no special preparation of the disk (other than the DOSGEN process itself which is always required) is necessary

before a new cartridge can be used by the DOS.

A.10 Disk Organization under DOS.A/DOS.E

This section describes the logical organization of the data on the disk when operating under DOS.A/DOS.E and how it relates to the general DOS file concepts as described in the chapter on System Structure. In this chapter it is assumed that the user is familiar with these concepts and has read and is familiar with the basic DOS file structuring.

A.10.1 Logical Drive Mapping

Under DOS.A, each physical disk cartridge corresponds with precisely one logical drive. Since the 9350-series disk controller is only capable of attaching four 9350-series disk drives, that means that only four logical drives (numbered 0, 1, 2, and 3) are legal under DOS.A.

A.10.2 Size of a Logical Drive

Each logical drive is two tracks on each of 203 cylinders of the physical disk cartridge. This results in 406 tracks of 24 sectors each, or a total of 9,744 total sectors on a disk cartridge. Since cylinder zero is reserved for system tables, only 9,696 sectors fall into allocatable file space and therefore only 9,696 sectors are available for storage under the DOS.A file management scheme. Of these, almost 100 sectors are required for the minimum DOS.A system, the eight DOS.A system files (SYSTEM0/SYS through SYSTEM7/SYS). This leaves on the order of 9500 sectors for user data once the DOS.A proper and a few of the basic commands have been loaded.

A.10.3 Cluster Mapping

Because there are eight bits per byte in the cluster allocation table (or CAT for short), and it is desirable to maintain one byte in the CAT per cylinder of available space on the drive, each cylinder on a logical drive (containing 48 sectors, total) is broken into eight groups, each one containing six physically contiguous sectors. Each such group is called a cluster. The first four clusters per cylinder are recorded on track zero of the cylinder, and the second four clusters of that cylinder are recorded on the other side of the disk, which is

track one, of the same cylinder.

Due to the fact that space is always allocated in terms of an integral number of clusters, this implies that the minimum file size under DOS.A is six sectors and that file size will always be a multiple of this number.

A.10.4 Segments under DOS.A

Disk space under Datapoint Corporation's DOS is always allocated in contiguous chunks of clusters called segments. When space is allocated, the largest segment on the disk (up to the maximum possible sized segment) is allocated, to keep the file as free of fragmentation as possible. By limiting the allocation size to the size of a full segment, the problem of allocating all available space on a disk to a first scratch file before a second one is subsequently opened is minimized. If several scratch files are opened and space in them is allocated at regular intervals, the resultant segments will be interleaved, resulting in minimized access time as the heads randomly access throughout the scratch area. The desire to make segment size small (to minimize file space conflicts and help to optimize use of space on the disk) and yet large (to maximize processing speed, maximize file size and minimize the number of RIB accesses) resulted in a segment size of thirty-two clusters. This compromise results in a 192-sector segment (thirty-two clusters of six contiguous sectors each) allowing easy addressability of a maximum size file while still allowing the segment size information to be kept within five bits as required for RIB compatibility with the other versions of DOS.

A.10.5 Maximum File Size

Under DOS.A, the maximum file size available is about 9,600 sectors. This is because there are 9,696 allocatable sectors of which almost 100 are used for the DOS.A system files. In practice, the user should not ever construct a system which pushes against the limits of available file size on a disk, since this fails to allow for future growth and expansion of his system. Another consideration is that if any tracks need to be locked out on the disk cartridge due to surface defects, then there may not be enough space left on the disk for his file.

Files bigger than about 9,000 sectors should be kept on larger disk systems, such as 9370 series disks under DOS.B or other appropriate DOS. If files larger than that size must be kept under DOS.A, then the files should be segmented into two or

more distinct files and logically concatenated at the user program level, the same as would be necessary for files larger than about 800 sectors on the 9380 series diskettes.

A.10.6 Cluster Allocation Table and Directory

Each disk cartridge used under DOS.A has its own, completely self-contained directory and file structure, just as for all Datapoint Corporation DOS. There are sixteen directory sectors on each disk cartridge, located in consecutive sectors starting at sector six on track zero of cylinder zero. Therefore, the sectors go from sector six to sector 025 (octal). The cluster allocation table is at sector zero of track zero, cylinder zero. The lockout cluster allocation table is at sector one of track zero, cylinder zero. The hashed directory index is at sector two of track zero, cylinder zero. The backup copies of each of these are in the corresponding locations of track one of cylinder zero.

The Hashed Directory Index, maintained by the DOS, resides in sector two of track zero, cylinder zero. This table enables directory lookups to go about four times faster than was possible under DOS 1.2. The technique works as follows:

Given an eleven byte file name and extension, an arithmetic/logical operation upon the file name results in an eight-bit quantity referred to as a hash code. This code is essentially a condensation of the 11 bytes of file name and extension information into only one byte. Obviously, the information is not complete; there are only 256 distinct eight-bit hash codes possible, while there are literally billions of legal file names and extensions. However, the condensation of information is such that looking at the hashed directory index allows the DOS to substantially restrict the range of directory sectors it must examine when doing a directory lookup. Each hash code for the file names in the directory is stored into the hashed directory index, offset by the physical file number (PFN) of the file with the corresponding name and extension.

Note that there is a calculated danger in the hashed directory approach. The danger is that if the hashed directory index is overwritten or otherwise destroyed accidentally, files may become inaccessable even though they are clearly shown (by doing a CAT command on the disk, for example) to be present. When this occurs on a disk, the technique to repair the disk is the REPAIR command. When the REPAIR command is almost finished, specify that the Hashed Directory Index is to be rewritten to the disk. This causes the HDI to be regenerated from the actual disk

directory and rewritten. In general, the Hashed Directory Index is rarely if ever destroyed in actual disk usage, and contributes greatly to overall system performance if many directory lookups are being done.

A.11 Internal DOS Parameterization

This section describes the DOS.A-dependent details of the parameterization of DOS.A system routines.

A.11.1 Physical Disk Address Format

Under DOS.A, physical disk addresses are presented (for example, as input to the DR$ and DW$ routines) in a two-byte format quite similar to that used under the other DOS. The most significant byte (which is traditionally placed in the D register) is the cylinder number, just like for DOS.B and DOS.C. The less significant byte (usually placed in the E register) has its most significant three bits representing a cluster number within the cylinder (any combination of these three bits is valid) and the least significant five bits representing a relative sector number within the specified cluster. Only the values zero through five are valid for the least significant five bits, since there are only six sectors per cluster.

A.11.2 Hardware Address Structure

The hardware disk address for 9350 disks also requires two bytes. One byte specifies cylinder number. The other byte specifies a sector number, 0 - 027 on the bottom surface, 040-067 on the top surface. This hardware address is used only for the DUMP9350 program and internally to the DOS routines DR$ and DW$.

APPENDIX B. DOS.B

DOS.B is Datapoint Corporation's Disk Operating System supporting Datapoint 2200 and 5500 series computers operating in conjunction with up to two 9370 series disk drives.

B.1 Planning for DOS.B

The recommended configuration for a DOS.B system includes 16K or more of memory in the 2200 or 5500 series computers. Use of a single 9370-series drive is possible, but the user should at least have access to a double-drive system for backup purposes. Some consideration must be given to the question of copying files from one disk pack to another, and users of the 9370-series "Mass Storage" disk systems will typically have files far too big to consider transferring from one disk to another one cassetteful at a time.

Another option which should be strongly considered is the High Speed, or so-called "RAM" Display Option. This option can substantially increase total system throughput (especially on batch-processing oriented systems) at a very small additional cost. The RAM Display option is field-installable, and is standard equipment on Datapoint 5500 series computers.

B.2 File Storage Capacity under DOS.B

Under DOS.B, each 9370-series disk unit is dealt with as two logical drives. Each of these two logical drives contains 38,976 sectors of 256 bytes each and can store up to 256 files. Of these, about 250 sectors and about ten files are used by the operating system and a few basic commands, leaving about 10 million bytes of usable space per logical drive, or up to roughly 20 million bytes of storage total for each disk storage unit in the configuration.

Other features of DOS.B include a large maximum file size: up to 30,237 data sectors in a single DOS.B file (not including the end-of-file mark and two RIBs).

B.3 Disk Drives

Datapoint DOS.B supports one or two 9370-series disk drives attached to one 9370-series disk controller. These drives are high-performance, random access disk units. They are the equal in every way to drives in constant daily use on the very largest mainframe computer systems, and offer substantially better performance than is available with the disks being used on many medium-scale mainframe computer systems.

B.4 Disk Media

The Datapoint 9370 series comprises two different types of drives. Models 9370-9373 use an 11-platter disk pack, media-compatible with the IBM 2316 disk storage module. On these packs data is recorded in 203 concentric circles on each of the 20 recording surfaces. Each such circle is referred to as a track. Models 9374 and 9375 use a single-platter disk which records data on 408 tracks (DOS uses only 406 of these).

The disk pack is enclosed within a plastic enclosure when it is not in place in the drive. This cover is intended to help keep the disk free from dust, pollen, smoke and other contaminants and is of prime importance in helping to eliminate disk errors and parity failures that contamination can cause.

B.5 Loading and unloading Disk Packs

B.5.1 Models 9370-9373

On the right side of the top of the 9370-series disk drives is the disk access cover. While holding the disk pack by the top center handle, remove the bottom portion of the disk pack enclosure by turning the bottom knob with the other hand. Then raise the disk access cover and carefully lower the disk pack into the cavity, still holding the disk pack by the top handle. When the pack has fully seated onto the spindle, turn the disk pack top center handle fully clockwise, until firm resistance is met. It is important that the pack be solidly in place before removing the top cover. After the pack has been properly mounted, the top cover should be slowly and carefully removed by lifting it straight upwards. Avoid letting the cover tilt and wedge against the edges of the disk platters as it is being drawn upwards as this can affect the precision alignment of the disk pack. The

access cover should be closed as soon as the disk pack cover has been fully removed, and the top and bottom halves of the disk pack protective cover should be immediately put back together to keep out dust and other contaminants.

To remove a disk pack, first place the "START/STOP" switch on the operator control panel of the drive to the STOP position. This immediately takes the drive off line and activates dynamic braking circuits in the drive which will brake the pack to a smooth but rapid stop in about twelve seconds. The disk access cover on top of the drive must not be opened before the pack has come to a full and complete stop. When this has occurred, raise the disk access cover and carefully lower the top portion of the disk pack cover down onto the pack. Be certain not to get it skewed since if the cover wedges against the edges of the platters it is possible to affect the critical surface-to-surface alignment of the pack, which will damage it. When the cover is fully lowered onto the pack, turn the handle in the center of the top of the cover counterclockwise until a distinct click is heard. This click indicates that the pack has been released from the drive spindle and may now be removed. Lift the disk pack and top cover together carefully out of the drive and immediately reattatch the bottom cover to the base of the pack, locking it firmly in place by a twist of the knob in the center of the bottom portion of the canister. The pack should be stored horizontally on a shelf (never on edge!) and in a position where it is not apt to be dropped or pushed accidentally over an edge. If another disk is not to be mounted immediately into the drive the pack was just removed from, the disk access cover should be closed right away to help prevent the entrance of dust, smoke or other contaminants into the drive mechanism and access arm assembly.

B.5.2 Model 9374/9375

At the top of the front panel of the drive is a handle for access. Pull forward and down on this handle to release the drive, then slide the entire drive forward to expose the cavity in which the disk fits. The disk pack itself has a handle on the top of the case. To open the disk pack, place the handle folded flat against the case and slide the lock button to the left, then - holding the lock button on - lift the handle to its full vertical position. This action releases magnetic clamps and allows the bottom of the disk cover to fall off. Lower the disk into the cavity in the drive, being sure it is fully seated. Now lower the handle on the top of the disk container. Invert the bottom of the disk pack cover and place it on top of the disk, inside the drive. It is essential the disk pack bottom cover be placed in the drive,

since the disk will not run if the cover is not present. Finally, slide the drive back into its cabinet, closing the access door.

Removing a disk is the exact reverse of inserting it. To remove the disk from the cavity in the drive, the lock button on the handle must be held to the left, just as for opening the disk cover.

B.6 Switches and indicators

B.6.1 Models 9370-9373

Two types of drives are repesented in these model codes; both use the same controller and the same disk packs. Older drives were manufactured by Memorex; the drives now shipped by Datapoint are "Telex" drives, manufactured by ISS.

B.6.1.1 Memorex Drives

The large physical drive number (just to the right of the READ-WRITE/READ ONLY switch) lights up when the heads are loaded onto the disk surface and typically at this time the drive will be on line.

The smaller numbers to the right serve as an indication of the position of the heads as they perform cylinder seeks to positions nearer or farther from the center of the disk pack. The exact physical cylinder number to which the heads are positioned at any given time can be determined by adding together the numbers which are illuminated, giving a cylinder number in decimal; the cylinder number in octal can be determined by noting which of the eight number positions are illuminated and considering those illuminated to be "1" bits and those not illuminated to be "0" bits. The bits then can be converted easily to a three-digit octal number by grouping them in groups of 2,3,3: a technique familiar to users conversant with octal.

The words "READ ONLY" illuminate to indicate that the drive is in the so-called "Write Protected" mode. In this mode, the computer cannot write anything onto the disk in that drive, but can only read the information already on the pack. This light is the indication of whether a drive is write-protected or not, and does not always immediately reflect the position of the read-only switch. See "Common Features" below.

The words "FILE UNSAFE" light up when the safety circuits in the drive detect one or more of about a dozen different conditions that they consider would endanger the data on the disk pack if continued disk operation were attempted. The FILE UNSAFE condition can be caused by (among other things) unusually severe power surges, and infrequently by a program going completely haywire and giving flagrantly illegal commands to the disk drives. If this light comes on during use of the system, the first remedy to try is to push the switch marked "START/STOP" to the "STOP" position. After the disk has come to a complete, braked stop (which should take about twelve seconds), push the switch back to the "START" position. If the problem which caused the FILE UNSAFE condition to occur was spurious, the drive will power back up normally and come on-line again in about sixty seconds. If the FILE UNSAFE condition occurs again (usually immediately upon completion of the sixty-second power-up delay) and repeatedly, it probably indicates a hardware malfunction and time to call the Datapoint Customer Support Center.

B.6.1.2 "Telex" Drives

The controls and indicators on Telex drives are essentially identical to those on the Memorex drives. When the drive is on-line, a green indicator light comes on indicating "FILE READY". There are no indicator lights for head position; cylinder position of the heads can be read on a vernier scale mounted on the top of the access arm assembly and visible through the top of the loading cover. A white indicator lamp indicates "READ ONLY" when the drive is protected, and, as on the Memorex drives, the read/write status of the drive does not immediately reflect the setting of the READ/WRITE - READ ONLY switch (see "Common Features" below). A red indicator lamp indicating "SELECT LOCK" is equivalent to the "FILE UNSAFE" indicator on the older drives.

B.6.1.3 Common Features

Changing the setting of the READ WRITE/READ ONLY switch only affects the drive if the drive is deselected. Therefore, this switch should be normally kept in the READ-WRITE position except for special purposes, and should usually be returned to the READ-WRITE position as soon as possible after the special purpose is completed. If the DOS command interpreter is active (as indicated by the familiar DOS "READY" message) and the READ ONLY lamp and the READ-WRITE/READ ONLY switch do not concur (indicating that that drive is selected), simply entering a blank from the system console is a simple technique to ensure that the drive

becomes deselected so that the revised setting of the READ ONLY switch will take effect.

The other switch on the operator control panel on the 9370-series drive (marked ENABLE/DISABLE) is for use only by the customer engineer and should always be left in the position marked ENABLE. This switch, when set to DISABLE, takes the drive off line (although the drive ready indicator stays illuminated). The switch is only active when the drive is de-selected, and this is the reason why the switch should not be used casually. If drive zero, for example, is DISABLEd and then the computer is bootstrapped, drive zero will not be de-selected at least until the DOS has completely booted itself up. And until the drive is de-selected, turning the switch to the ENABLE position has no effect. The only solution for this problem if it occurs is to completely power down the entire system and bring it back up again with the switch in the ENABLE position. So in general, it is a good idea to keep this switch set to ENABLE and let it go at that.

B.6.2 Model 9374/9375

The 9374 disk drives are controlled by a small cluster of switches in the lower right-hand corner of the front panel of each drive. A thumbwheel switch sets the physical device number; this switch is set at installation and should not be reset thereafter.

A LOAD/RUN rocker switch controls loading the disk. An indicator light below the rocker switch indicates if the drive is in LOAD, ready for a disk to be removed or inserted, or in RUN, on-line to the processor.

A pair of rocker switches labeled PROT CART and PROT FIXED control write protection of the removable disk pack and the fixed platter within the drive. When one of these switches is illuminated the corresponding disk is write protected. The protection setting of either disk can be changed at any time by using the PROT switch. When a disk is first spun up (first brought to READY status) both disks will be write protected for at least three minutes to ensure thermal stabilization. If the drive is cold, the write-protect delay could be longer. The delay for thermal stabilization is necessary because the 9374 disk is a very high-density storage medium.

B.7 Care and Handling of Disk Packs

Disk packs for the 9370 series disk drives are precision, high-technology assemblies and must be treated as such. It is of extreme importance that they not be mishandled, dropped, or contaminated with dust or other pollutants. The packs should be stored strictly horizontally (not on edge) on a shelf clean from dust and in an environment similar to that where the drives are installed (preferably in the same room with the computer).

On the bottom of each 11-platter disk pack is a fine nylon filter, normally white. This filter should be replaced at least once per year, or more often if indicated by discloration or airborne debris.

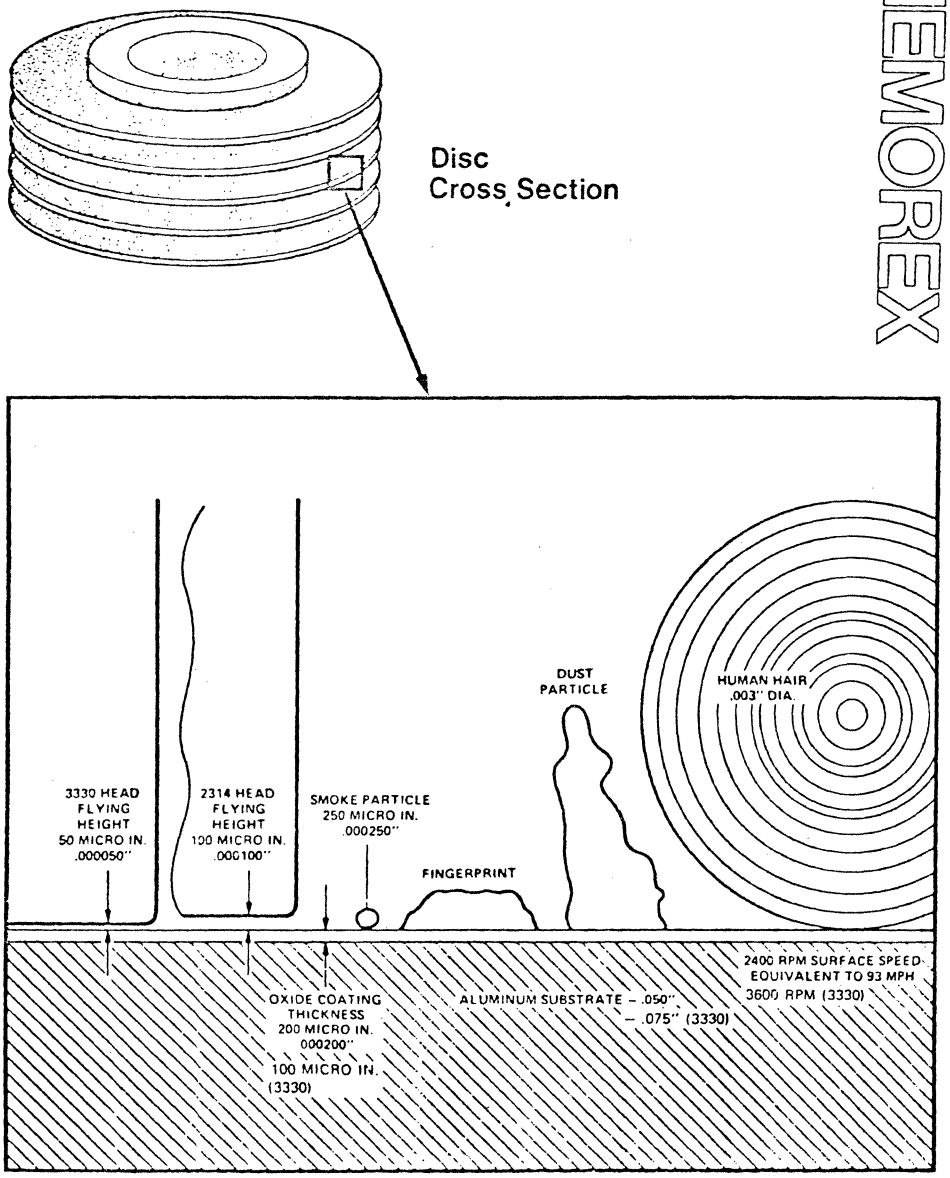
If the packs are shipped by common carrier, they should be repackaged in their original shipping carton and marked "FRAGILE". Disk packs should never by mailed by Parcel Post.

In addition, any pack which has been in a non-computer room environment should be allowed to equalize temperatures in the room with the computer for 24 hours before use if at all possible before attempting to read or write data on the pack. In an emergency, placing the pack onto a drive and letting it spin up and run for about an hour will usually be adequate, but this procedure should be considered an emergency measure only.

A little care in handling disk packs will repay itself several times over in reliable and trouble free service with long life from your disk packs.

B.8 Care and Maintenance of the 9370 Drives

The 9370 series disk drives are full scale mainframe computer peripherals and deserve to be taken care of. As with the disk packs, cleanliness is of paramount importance. All efforts should be made to keep the room as dust-free as possible. Since the read/write heads fly so very close to the disk surface, just 80 millionths of an inch away from the oxide on the 11-platter packs, even such small particles in the air as those present in cigarette smoke may cause troubles eventually. The drawing in this section, reproduced here courtesy of Memorex Corporation, graphically depicts the relative proportions of disk head flying height and common office pollutants, and should help to explain why the need for cleanliness and good housekeeping practices is so important.



Users of the system should be careful to close the disk access cover (or slide the drive back in the cabinet) as soon as the pack loading or unloading is complete and keep disk packs in their protective covers at all times to prevent contamination. If disk pack canisters become soiled, they should be cleaned carefully with a mild detergent solution and carefully wiped dry. Users should be very careful to never allow anything to contact the oxide surface of the disk pack itself.

In addition to this user maintenance, the user should also ensure that the preventive maintenance procedures outlined in the

9370 series disk drive maintenance manual are performed.

B.9 Head Crashes

Each of the heads in the 9370-series drive is held against the disk surface by a spring which pushes the head toward the surface with a force of about 350 grams. The disk on the other hand is spinning at about 80 miles per hour relative to the heads. The thing which keeps the head and disk apart is a micro-cushion of air only eighty millionths of an inch thick. A head crash occurs when this lubricating air film fails. The main causes of head crashes are foreign particles in the lubricating film, contamination buildup on the surfaces of the disk or read/write heads, or a defective disk surface.

When a head crash occurs, the head rubs against the oxide surface of the disk, which frequently loosens more oxide, resulting in further and more severe crashes, and things go progressively downhill from there. Due to the severity of a head crash, not just because of the loss of data on a disk but also due to the degree of damage to the heads on the drive, it is important to recognize the symptoms of a head crash. In this manner a disk experiencing a head crash can usually be discovered and stopped before the crash reaches catastrophic proportions.

For a description of symptoms of a head crash and appropriate prventive and restorative action, see Appendix A under "Head Crashes".

B.10 Preparing Disk Packs for Use

When a disk pack is first received from the manufacturer, it is completely demagnetized and is not usable until it has been formatted. The formatting process writes the entire surface of the disk pack with track and sector addresses which later allow the controller to identify where a given sector is on the surface of the disk.

When this information is later read by the DOS, any errors discovered in the formatting information are treated in the same way as a parity error in the written sector information itself, thus resulting in up to nine or ten re-tries before returning with a parity error indication. Sometimes, if the parity error indicated by the DOS is due to an error developing in the formatting information on the disk, the parity error on the disk can be completely eliminated by using the BACKUP command to save

as much of the information on the pack as possible and then reformatting the pack. (After reformatting the pack, any data that had been on the pack is destroyed and it must be DOSGENed like a new one). Even what at first appear to be "hard" parity errors can occasionally be cleared this way.

Formatting information is written onto the disk using the INIT9370 command, either from a working DOS or from a LOAD & GO cassette. (NOTE: although in general the commands from the DOS cannot be run without the DOS being active, INIT9370 is one of the few exceptions). Following successful completion of the INIT9370 command program, the disk pack it was used on can be DOSGENed (twice, once for each of the two logical drives) and will normally not need to be re-formatted again for the duration of its lifetime.

B.11 Disk Organization under DOS.B

This chapter describes the logical organization of the data on the disk when operating under DOS.B and how it relates to the general DOS file concepts as described in the chapter on System Structure. In this section it is assumed that the user is familiar with these concepts and has read and is familiar with the basic DOS file structuring.

B.11.1 Logical Drive Mapping

Under DOS.B each physical volume is broken into two logical drives. This is done for reasons of addressing. It is simply not possible to address all of the sectors on an entire 9370 disk volume using only two bytes of physical disk address, and the two byte physical disk address is central to all of Datapoint Corporation DOS's operations. It is not practical to change this characteristic without making changes which would result in invalidating many user-written programs and many large systems which run under the DOS. Therefore the disk was broken into two halves, and one bit of the effective physical disk address is taken from the logical drive number.

For the 9374 drives the removable disk is one logical drive, and the fixed disk is a second logical drive.

The first eight recording surfaces on the 11-platter disk (heads are numbered from zero to nineteen starting at the top of the disk drive) correspond to logical heads zero to seven on the even logical drive, and the next eight recording surfaces on the

disk correspond to logical heads zero to seven on the odd logical drive (physical heads eight through fifteen). Physical heads sixteen through nineteen (and the corresponding recording surfaces on the disk pack) are not used by DOS.B.

B.11.2 Size of a Logical Drive

Each logical drive is eight tracks on each of 203 cylinders of the physical drive. This results in 1624 tracks of 24 sectors each, or a total of 38,976 total sectors on a logical drive. Of these, about 38,400 remain when the DOS has been generated, the system tables constructed on the disk, and a few basic commands loaded. The 9374 disks are addressed using the same structure of 203 cylinders, 24 sectors per track. Physically the 9374 disks have 406 cylinders with 48 sectors per track, but the disk controller itself provides an interface between the physical and logical structure, so the processor "sees" a drive structured just like a 9370 drive.

B.11.3 Cluster Mapping

Because there are eight bits per byte in the cluster allocation table (or CAT for short), and it is desirable to maintain one byte in the CAT per cylinder of available space on the drive, each track on a logical drive represents one DOS cluster, and is represented in the CAT by exactly one bit. Since the DOS uses eight tracks per logical cylinder, this results in exactly eight clusters per cylinder of twenty four sectors each.

Due to the fact that space is always allocated in terms of an integral number of clusters, this implies that the minimum file size under DOS.B is twenty-four sectors and that file size will always be a multiple of this number. It turns out that choosing a full track as the smallest allocatable unit of space has other advantages as well from a system standpoint, since it allows some programs (like DOS.B COPY) to make several simplifying assumptions about the data in a file which enables them to copy data and reference information in a file substantially more easily and efficiently than would be otherwise possible.

B.11.4 Segments under DOS.B

Space under Datapoint Corporation's DOS is always allocated in contiguous chunks of clusters called segments. When space is allocated, the largest segment on a drive up to the maximum possible sized segment is allocated, to keep the file as free of fragmentation as possible. By limiting the allocation size to the size of a full segment, the problem of allocating all available space on a disk to a first scratch file before a second one is subsequently opened is minimized. If several scratch files are opened and space in them is allocated at regular intervals, the resultant segments will be interleaved, resulting in minimized access time as the heads randomly access throughout the scratch area. The desire to make segment size small (to minimize file space conflicts and help to optimize use of space on the drive) and yet large (to maximize processing speed, maximize file size and minimize the number of RIB accesses) resulted in a segment size of ten clusters. This compromise results in a 240-sector segment (ten clusters or tracks of 24 sectors each) allowing a maximum file size of over 30,000 sectors while still allowing internal disk address and segment size calculations to be done using faster single precision arithmetic techniques.

B.11.5 Maximum File Size

Under DOS.B, the maximum size file available is 30,238 data sectors. This number is the result of 126 segment descriptors in the RIB, each of which points at one segment of 10 tracks of 24 sectors each:

24 sectors x 10 tracks x 126 segments = 30,240 total sectors

Since the first two sectors of each file under the DOS are used for the RIB and its copy, that leaves 30,238 sectors available to the user for the storage of his data. Files longer than this number will have to be segmented or logically concatenated at the user program level, the same as would be necessary for files larger than about 9600 sectors on the 9350 series disks.

B.11.6 Cluster Allocation Table and Directory

Each logical drive under DOS.B contains its own directory and cluster allocation table, just as for all Datapoint Corporation DOS. There are sixteen directory sectors on each logical drive, located in consecutive sectors starting at sector five on logical track zero of cylinder zero. Therefore, the sectors go from sector five to sector 024 (octal). The cluster allocation table is at sector zero of logical track zero, cylinder zero. The lockout cluster allocation table is at sector one of logical track zero, cylinder zero. The backup sectors for all of these are in the corresponding locations on logical track one of the same cylinder.

The Hashed Directory Index, maintained by the DOS, resides in sector two of track zero, cylinder zero. This table enables directory lookups to go about four times faster (on full disk directories) than was possible under DOS.B Version 1. The technique works as follows:

Given an eleven byte file name and extension, an arithmetic/logical operation upon the file name results in an eight-bit quantity referred to as a hash code. This code is essentially a condensation of the 11 bytes of file name and extension information into only one byte. Obviously, the information is not complete; there are only 256 distinct eight-bit hash codes possible, while there are literally billions of legal file names and extensions. However, the condensation of information is such that looking at the hashed directory index allows the DOS to substantially restrict the range of directory sectors it must examine when doing a directory lookup. Each hash code for the file names in the directory is stored into the hashed directory index, offset by the physical file number (PFN) of the file with the corresponding name and extension.

Note that there is a calculated danger in the hashed directory approach. The danger is that if the hashed directory index is overwritten or otherwise destroyed accidentially, files may become inaccessable even though they are clearly shown (by doing a CAT command on the disk, for example) to be present. When this occurs on a disk the technique to repair the disk is the REPAIR command. When the REPAIR command is almost finished, simply specify that the Hashed Directory Index is to be rewritten to the disk. This causes the HDI to be regenerated from the actual disk directory and rewritten. In general, the Hashed Directory Index is rarely if ever destroyed in actual disk usage, and contributes greatly to overall system performance if many directory lookups are being done.

B.12 Internal DOS Parameterization

This section describes the DOS-dependent details of the parameterization of DOS.B system routines.

B.12.1 Physical Disk Address Format

Under DOS.B, physical disk addresses are represented in a two-byte format in a manner quite similar to that used under the other DOS. The most significant byte (which is traditionally placed in the D register) is the cylinder number. The less significant byte (usually placed in the E register) has its most significant three bits representing a cluster number within the cylinder (or logical track number in the specific case of DOS.B) and the least significant five bits representing the sector number within the specified cluster. Since there are 24 sectors in a cluster, the five bit sector number has a valid range of 0-027.

B.12.2 Hardware Address Structure

The 9370 series drives use three bytes for hardware addressing. The cylinder number is one byte (range 0-0312), the head number is a second byte (range 0-023), and the sector number a third byte (range 0-027).

APPENDIX C. INTRODUCTION TO DOS.C

DOS.C is Datapoint Corporation's Disk Operating System supporting Datapoint 1100, 2200 and 5500 computers operating in conjunction with up to four 9380 series diskette drives.

C.1 Planning for DOS.C

The recommended configuration for a DOS.C system includes 16K of memory in the 1100 or 2200 series computers and 16K or more in 5500 series computers. Use of a single 9380-series drive is possible, but the user should at least have access to a double-drive system for backup purposes. Some consideration must be given to the question of copying files from one diskette to another. Users with 2200 and 5500 series computers (having cassette tape drives) can use cassettes if necessary as a standard exchange medium for file transfers (except for files bigger than about 450 sectors, too large to fit on a single side of a cassette). Those with Diskette 1100 series computers do not have cassette tape drives and hence must use diskettes as their file transfer medium. Since DOS.C software is distributed on diskettes by Datapoint Corporation, the user will need to have at least a two drive system to copy the software from the diskette received from Datapoint to his working diskette(s). Single drive systems should be considered only by those intending to use them as satellite systems (example: data entry stations) and planning on having at least one other system with two or more drives (for program development and file processing applications).

Another option which should be strongly considered is the High Speed, or so-called "RAM" Display option. This option can substantially increase total system throughput and responsiveness (especially in applications displaying a lot of text on the screen, such as data entry) at a very small additional cost. The RAM Display is field-installable (although less expensive when factory-installed), and is standard equipment on Datapoint 5500 and Diskette 1100 series computers.

C.2 Performance of DOS.C

Users who are currently using Datapoint computers in cassette-based systems will find substantial improvements in performance when they upgrade to DOS.C. The 9380 series diskette drives are several times faster than cassettes for ordinary sequential data transfers; random-access type operations (such as sorting and ISAM file access) can easily be two orders of magnitude faster than is attained using tape cassettes.

Users who are currently using competitive diskette-based equipment will generally find that total system performance of Datapoint systems will exceed that which they are accustomed to. This improvement is due to the generally superior data handling techniques and file structuring as used in Datapoint's DOS. These characteristics stem from the fact that instead of employing a lower-performance cassette-style file structure as a base for the operating system, Datapoint chose instead to adapt the same advanced and dynamic disk file access techniques as used in its other DOS to the new diskette media. The result is a degree of software sophistication previously unavailable in business-oriented systems of this size.

The obvious side benefit of this DOS compatibility is that not only is virtually all of the Datapoint DOS software library available to diskette users but that most user programs which were written originally for other Datapoint DOS systems will run unmodified (except for possible file size limitations and timing differences due to the slower access times of the 9380 series disk drives) under DOS.C.

In addition to the increased speed of access of the 9380 series drives as compared to cassettes, another big advantage is that the total amount of storage available on a diskette-based system is about four times the amount usable on cassette systems, even when both cassette drives are in use.

It should be recognized that DOS.C is not expected to eliminate the usefulness of larger capacity, higher performance disks. Many users will have applications which are too involved or too large for the 9380 series diskette drives. Users who need very large data files or very high speed random access to disk storage will find the performance they are looking for in the other versions of Datapoint DOS.

C.3 Disk Drives

Datapoint DOS.C supports up to four 9380-series flexible diskette drives through their integral disk controller unit. The disk controller contains 1024 bytes of high speed, random access memory which buffers four sectors between the disk drives and the Datapoint computer, enabling greater I/O device autonomy and improved overall system performance.

C.4 Disk Media

The Datapoint 9380-series flexible diskette drives use a flexible diskette for data storage. This diskette is media and format-compatible with the IBM 3741-style flexible diskette.

Data is recorded in 77 concentric circles on only one side of the diskette (as per the IBM standards for diskette data interchange). Each such circle is referred to as a track. Although each such track on the diskette actually contains 26 physical records of 128 bytes each, these are paired up by the Datapoint 9380 series diskette controller (an integral part of the diskette system) so that to the Datapoint computer each track appears to consist of 13 records (called sectors) of 256 bytes each. In Datapoint DOS documentation, unless explicitly indicated to the contrary, the term sector always refers to a 256-byte logical sector, and it is strictly incidental that this sector is broken into two physical 128 byte records for transfer to and from the diskette media.

The diskette is permanently enclosed within a durable plastic cover. This cover provides for easy insertion of the diskette into the diskette drives and provides structural rigidity for the media when it is not in use. In addition, the plastic cover provides a degree of environmental protection for the diskette and its oxide surface from damage caused by careless handling.

C.5 Loading and Unloading Diskettes

Upon observation of a diskette, three holes through the plastic diskette cover will be noted. Each of these holes allows one to see a portion of the oxide-coated surface of the diskette itself.

The large, round hole in the center of the cover is used by the diskette drive for the hub which clamps to the diskette and turns the diskette within the cover.

The longer, narrower radial slot towards one edge of the enclosure is the slot through which the read/write head in the diskette drive contacts the diskette's oxide coating for data transfer operations.

The smaller round hole present on the diskette is the hole through which the index hole, a hole in the diskette proper about the diameter of a pencil lead, is sensed by the controller and used for timing and control purposes.

The reason for the description of these holes is that they provide the definitive reference for indicating the proper direction of insertion of the diskette media into the 9380 series drives. When the diskette is properly inserted, the edge of the diskette with the long slot is inserted first. The smaller hole (the one through which the index hole is sensed) will be the last of the three holes in the cover to enter the drive, and it will be positioned toward the tabletop rather than downwards toward the floor.

The diskette loading slot is covered by a long, narrow handle. A rectangular pushbutton to the side of the handle is pushed to open the handle for diskette insertion and removal. When inserting the diskette, it will meet with a spring resistance after being inserted about 3/4 into the drive. Press the diskette gently into place until the spring catches and the diskette stays in place without being held in with the finger. Be careful not to push the diskette too far into the drive, as this can cause the innermost edge of the diskette's plastic cover to be wedged between some metal projections on the diskette drive which could possibly result in damage to the diskette. After the diskette is in place, pull the door/handle to the left until it latches closed. As the door is pulled closed, the hub engages the diskette, bringing it to its rated rotational speed of 360 rpm (and then online) almost immediately.

To remove a diskette, first ensure that all input/output activity on the diskette has completed. (This is necessary since it is possible to open the drive door, which takes the diskette offline, in the middle of a write operation; this can result in improper data being written onto the diskette.) Then press the button to the left of the door/handle. The door will open and the diskette will emerge in much the same way toast pops out of a toaster. Upon removing the diskette from the drive, it should be immediately placed in its protective paper envelope to help protect the surface from abrasive contaminants and other elements which could damage it.

C.6 Drive Numbering and Switches

Diskette drives are normally installed in the cabinet starting from the left. These drives are numbered 0, 1, 2, and 3 respectively from left to right. These numbers constitute the physical drive number. In the case of DOS.C, the same number is also sometimes referred to as the DOS logical drive number, or frequently just drive number.

The main power switch for the diskette unit is located on the underside of the tabletop and to the left side of the diskette drives, positioned toward the front of the cabinet. Sliding this switch towards the rear of the diskette drive cabinet turns the diskette unit on, and sliding the switch towards the user turns the diskette unit off. This switch should normally always be left in the ON position.

There are no other controls or switches intended for use by the user on the 9380-series diskette drives.

C.7 Care and Handling of Diskettes

Diskettes are sturdy media which will give long and trouble-free service if they are handled with reasonable care.

1) Diskettes should always be stored in their protective paper envelopes when not inserted in a drive. These envelopes should then be stored in the protective boxes in which the diskettes are received from the manufacturer.

2) Do not force too many diskettes into one box. They should not be placed under heavy pressure, as this can warp the diskette media, possibly causing read/write errors.

3) Diskettes should not be rolled, folded or otherwise subjected to strains which could crease the media.

4) Never touch the oxide coating of the diskette through the holes in the plastic cover. Human skin has oils on it which will attract and retain dust and other abrasive contaminants if these oils get onto the diskette's surface. In addition, contact between hard surfaces and the diskette oxide can scrape away the information-carrying oxide from the diskette surface; this will usually result in unrecoverable errors on the diskette.

5) Diskettes should not be subjected to strong magnetic fields.

6) When mailing diskettes, they should either be placed between two sheets of corrugated cardboard (for rigidity and protection while going through the mails) or placed in some suitable protective carrier. Many diskette media manufacturers sell mailers specifically designed for use in sending diskettes, either singly or in multiples, through the mail.

7) Diskettes can generally be taken through airport security x-ray and metal detecting equipment without danger of damage to the information recorded on the diskette.

C.8 Preparing Diskettes for Use

When a diskette is first received from the media manufacturer, it contains formatting information recorded across the entire usable surface of the diskette. This information is provided to allow the controller to identify where a given sector is on the surface of the disk, and also allows the controller to verify proper head positioning in the drive mechanism. Normal reading and writing on the diskette does not destroy the formatting information contained thereon.

Only diskettes in 3740 format (128 byte sectors) are usable by DOS.C. Diskettes that have been reformatted with bad tracks flagged and alternate tracks substituted cannot be used. Also, diskettes in System 32 format (256 byte sectors) or I.B.M. 3600 format (512 byte sectors) cannot be used.

Diskettes cannot be used by DOS.C until they have been generated with the DOSGEN program described earlier. Datapoint DOS uses its own unique file structure which is capable of more sophisticated data and file manipulation than the standard IBM file structure which is intended for data entry and not for actual computer data processing. This more sophisticated file structure is what results in the need for DOSGEN before a diskette can be used by DOS.C.

A special note regarding disks which are to be used in drive zero is appropriate. All of the newer releases of DOS commands use DOS FUNCTIONs (as described in the DOS USER'S GUIDE). These functions are resident on the diskette in the file SYSTEM7/SYS. When updated versions of DOS.C and associated utilities are received from Datapoint Corporation, the file SYSTEM7/SYS may also

have one or more new DOS FUNCTIONs resident. Therefore, wholesale copying of DOS commands from newly received diskettes to older diskettes with older versions of SYSTEM7/SYS will frequently result in commands which either work or do not work depending on whether the older or newer version of SYSTEM7/SYS is present on the disk in drive zero. Therefore the user should generally keep his DOS commands disk more or less intact and not use a newly released diskette to supply commands to previously DOSGENed diskettes; instead, he should freshly generate as many system diskettes (including whichever DOS commands he intends to use) as he needs.

C.9 Suggested Disk Organization Techniques

Due to the relatively small capacity of the flexible diskette, careful consideration should be given to which files should be put on which diskettes. Users with single drives for data entry and related applications will have little choice in such matters. However, for users with multi-drive systems being used for program development, the following convention is suggested:

1) DOS system diskettes. These diskettes contain the system files (as do all diskettes for use with DOS.C) and whichever DOS commands the user intends to use. Usually this diskette will be used in drive zero during program generation, debugging and other DOS system-type functions and because of this will contain all of the DOS itself, DOS commands and latest set of DOS FUNCTIONs as released by Datapoint. This diskette will also frequently contain the editor scratch file, SCRATCH/TXT.

2) Source program diskettes. These diskettes can be considered as library file diskettes. These diskettes contain programs, Dataform forms, and other user text files used, for example, during program generation. Once these programs are finalized, they can be copied to DOS System diskettes or User System diskettes as appropriate.

3) User system diskettes. These diskettes are similar in intent to DOS System diskettes but differ in that they are intended more for specific application use rather than for general program development and debugging. These diskettes will usually be used with Databus 1100, Scribe, DOSBASIC, or Dataform and/or have large numbers of user-written application programs on them. These disks will usually not contain the more specialized DOS commands

(and other files) such as DUMP9380, REPAIR, DOS/EPT, MASSACRE, APP, CHANGE, DUMP and the like.

4) Data file diskettes. These diskettes contain primarily user data files. Typical characteristics of files on this type of diskette: non-executable, user information; SCRIBE text; other things which are user-entered (or generated) but not programs as such.

5) Scratch diskettes. These diskettes are diskettes containing no important files. These diskettes are suggested for use in transferring files from one diskette to another, and to provide diskettes containing large unallocated areas for use as scratch files by programs using scratch files (for example, SORT and EDIT commands).

As support for the above five basic types of diskettes, the following color-coding convention is suggested for diskette labels:

red - DOS System diskettes
green - User System diskettes
blue - Text files (source programs and SCRIBE text)
yellow - Data files
grey - Scratch diskettes

For best results, users should use only diskette media provided by those manufacturers recommended by Datapoint Corporation.

C.10 Disk Organization under DOS.C

This chapter describes the logical organization of the data on the disk when operating under DOS.C and how it relates to the general DOS file concepts as described in the chapter on System Structure. In this chapter it is assumed that the user is familiar with the basic DOS file structuring.

C.10.1 Radius Spiraling and Sector Skewing

Under DOS.C, the sectors on the diskette are logically renumbered to allow substantially increased performance over what would be possible otherwise. Program loading, in particular, goes about three times faster than would be possible if this were not done. This renumbering of the sectors on the track is referred to as sector skewing. This sector skewing takes the form of placing

logically sequential sectors about four sectors apart on a track of the diskette. Thus logical sector zero on track zero would appear in physical sector zero; logical sector one would appear in physical sector five; logical sector two would appear in physical sector ten; and so forth.

In addition to rearranging the order of the sectors on a track of the diskette, the starting points (logical sectors zero on each track) do not line up along a straight-line radius as do the physical sectors zero. Instead, the starting point for numbering sectors on a track spirals inwards. Therefore, the logical radius line (sectors zero, for example) forms a spiral on the diskette surface, and hence the term radius spiraling. The intention behind radius spiraling is twofold: one, it allows for head seek time between adjacent tracks while rapidly scanning through a data file (in addition to the processing time lag provided by the normal sector skewing); two, it allows searching the directory (which is along a logical radius of the diskette, as will be described later) about three times faster than would otherwise occur. Together with sector skewing, radius spiraling aids in achieving much higher overall system performance than is obtainable on most competitive diskette based systems.

Use the chart below to convert from the logical to physical sector. First divide the decimal track number by 4. The remainder gives the appropriate column. Run down the left side to the logical sector and across to the appropriate column to get the physical sector number.

LOGICAL SECTOR	REMAINDER OF TRACK/4			
	0	1	2	3
0 (0)	0	05	012	02
1 (01)	05	012	02	07
2 (02)	012	02	07	014
3 (03)	02	07	014	04
4 (04)	07	014	04	011
5 (05)	014	04	011	01
6 (06)	04	011	01	06
7 (07)	011	01	06	013
8 (010)	01	06	013	03
9 (011)	06	013	03	010
10 (012)	013	03	010	0
11 (013)	03	010	0	05
12 (014)	010	0	05	012

Note that physical sector addresses are never used by DOS. Even in DUMP9380 the sector address entered is taken as a logical

sector address except when in EBCDIC mode, when it is considered a physical sector address.

C.10.2 Size of a Diskette

There are 77 tracks on a diskette, each of which contains logically thirteen 256-byte sectors (physically twenty-six 128-byte sectors). This yields a total of 1001 sectors, or a grand total of 256,256 bytes of storage capacity. The first track on the diskette (the one nearest the edge of the diskette) is not used by DOS.C, in order to help provide compatibility with IBM equipment. Additionally, the logical last sector on each track (sector 12 if one counts starting at 0) is not used by DOS.C for data, for reasons which will be described in subsequent sections. Subtracting these two unallocatable areas results in a total allocatable file space of 912 sectors. About ninety sectors of these are used by the DOS for its system files, leaving about 825 sectors for user files, a user file capacity of over 200,000 bytes. This constitutes about twice the capacity of a tape cassette on each diskette. Due to the higher data storage efficiency attained by Datapoint software, most users will find that the total number of records stored on a Datapoint format diskette will be as large as, and in most cases substantially larger than, the number achieved on competitive systems.

C.10.3 Cluster Mapping

Under DOS.C, each track of the diskette consists of 4 clusters of three sectors each. This implies that one cluster or three sectors is the smallest allocatable unit of space on a diskette, and that all files are multiples of three sectors in length.

In the cluster allocation table, the four clusters on each track are represented by the low-order four bits of each byte. As in other Datapoint DOS, a one bit represents that the corresponding cluster is allocated and a zero bit indicates that the corresponding cluster is available for allocation. The high-order four bits of each byte in the CATs are reserved for future use in DOS.C, and are currently always set to zero.

C.10.4 Segments under DOS.C

The use of a three sector cluster has numerous advantages on the diskette. One which should be immediately apparent is that the amount of space wasted due to always allocating an integral number of clusters is reduced to only an average of one and a half sectors per file. Perhaps a less obvious advantage results from the manner in which the Datapoint DOS allocates disk space to files. During space allocation, the DOS will allocate the first contiguous, maximum-size segment it can find as an initial (or secondary) allocation. Since a segment consists of up to 32 clusters (there are five bits of cluster number information in each segment descriptor), this results in files being initially allocated 96 sectors, assuming that the space on the diskette is sufficiently unfragmented to allow such an allocation. Making this initial allocation smaller than the 192 and 240 sectors as used in some of the other Datapoint DOS allows for several scratch files to be opened on a diskette which already has a few files on it, as each newly opened file will take a smaller bite out of that portion of space remaining unallocated. Making the full segment size much smaller than 96 sectors quickly increases the amount of overhead required to index through the file (since the number of RIB accesses required increases) and decreases performance.

C.10.5 Maximum File Size

Under DOS.C, the maximum file size attainable depends upon the amount of space used on the diskette for system files, but using the current size of DOS.C as an example indicates that at least 800 sectors should be available for user file allocation on a normal data file diskette. Users having only a single diskette drive and therefore having several programs on the diskette in addition to the DOS will have a corresponding reduction in the maximum size of data files they may have. Users whose files exceed the capacity of one diskette will need to segment their files at the user program level much as they would do on a cassette system when a file exceeded the capacity of a single cassette.

C.10.6 Cluster Allocation Table and Directory

Under DOS.C, the use of four three-sector clusters per track results in one unused sector per track. This restriction arises from the facts that (1) all clusters under Datapoint DOS must contain the same number of sectors and no cluster may span a track boundary; and (2) a 13-sector cluster is not practical because it results in excessive amounts of wasted space at the end of each file on the diskette. Since these 76 sectors on the diskette (remember that track zero is not used) are not available for allocation as file space, they are partially put to use for storage of DOS system tables: four cluster allocation table sectors and thirty-two directory sectors. These system tables are positioned in the following manner:

```
Track      0 - Unused;  for IBM compatibility
Tracks  1-16 - Directory copy, for backup purposes
Tracks 17-32 - Primary DOS directory
Track     33 - Working Cluster Allocation Table
Track     34 - Working Cluster Allocation Table backup
Track     35 - Lockout Cluster Allocation Table
Track     36 - Lockout Cluster Allocation Table backup
Tracks 37-76 - Reserved for future DOS use
```

Again recall that each of the above sectors is in logical sector 12 of the track indicated.

In the Cluster Allocation Tables, bytes 239-254 are used for the Directory Mapping bytes. These sixteen bytes each contain either an 0377 or the number of files currently allocated in the corresponding one of the sixteen directory sectors. These bytes are updated automatically by the DOS whenever a file is created or deleted, and are updated by the DOS occasionally if they are found to be inaccurate. The purpose of these directory mapping bytes is to provide improved speed of directory lookups and to allow faster creation of files. They are of the greatest benefit to users who have several drives in their system where relatively few files exist on each drive. The intention is to eliminate the need to read in directory sectors while looking for a file if those sectors are known to not contain any active directory entries, and likewise when looking for an empty slot for use by a new file to eliminate having to read sectors known to have all sixteen directory entries in use.

C.11 Internal DOS Parameterization

This section describes the DOS-dependent details of the parameterization of DOS.C system routines.

C.11.1 Physical Disk Address Format

Under DOS.C, physical disk addresses are represented in a two-byte format in a manner quite similar to that used under the other DOS. The most significant byte (which is traditionally placed in the D register) is the cylinder number. The less significant byte (usually placed in the E register) has its most significant two bits representing a cluster number within the track (all combinations of these two bits are valid since there are four clusters per track) and the least significant two bits representing the sector number within the specified cluster. Because there are only three sectors per cluster, only binary values 00, 01 and 10 are valid for these low-order bits. (The only exception to this rule is that a least significant PDA byte of 0303 permits access to the unallocatable sector on each track, that sector used for system table sectors). (For compatibility reasons, the most significant three bits can be considered the cluster number, yielding clusters numbered 0, 2, 4, and 6).

The unused bits of the least significant physical disk address byte (that is, the center four bits) should always be set to zero.

APPENDIX D. DOS.D

DOS.D is Datapoint Corporation's Disk Operating System supporting 48K Datapoint 5500 series computers operating in conjunction with from two to eight 9370 series disk drives. In addition to the interactive/batch operation as provided in all standard Datapoint Corporation DOS, DOS.D additionally supports Datapoint's Partition Supervisor (called PS, released separately) which provides for the simultaneous execution of multiple DOS programs.

D.1 Planning for DOS.D

The minimum configuration for a DOS.D system requires a 48K Datapoint 5500 series computer, a 9370 disk controller and drive and at least one 9371 disk extension unit. If more storage is desired, up to seven 9371 disk extension units may be included, for a total of eight disk storage units.

D.2 File Storage Capacity under DOS.D

Under DOS.D, each 9370-9373 model disk unit is dealt with as two logical drives. Each of these two logical drives contains 48,576 sectors of 256 bytes each and can store up to 256 files. Of these, about 250 sectors and about ten files are used by the operating system and a few basic commands, leaving about 12.4 million bytes of usable space per logical drive, or up to roughly 25 million bytes of storage total for each disk storage unit in the configuration. Using the 9374/9375 disk drives, file storage is somewhat less due to the capacity of the disks used. For these drives, each logical drive contains 38,976 sectors and provides about 10 million bytes of usable space per logical drive, or roughly 20 million bytes of storage per disk unit.

Other features of DOS.D include a large maximum file size: up to 38,397 data sectors in a single DOS.D file (not including the end-of-file mark and two RIBs).

This availability of up to almost 200 million bytes of fast, random access online storage expands the range of applications for Datapoint 5500 dispersed data processing systems into areas heretofore not practical for anything less than a mainframe computing system due to large-scale data base requirements. In

conjunction with Datapoint's highly successful Datashare package and partitioned operation under PS, Datapoint offers a total business data processing solution of a scope and power that could scarcely have been dreamt of when the initial 2200 was introduced only six or seven short years ago.

D.3 Disk Drives

Datapoint DOS.D supports from two to eight 9370-series disk drives attached to one 9370-series disk controller. These drives are high-performance, random access disk units. They are the equal in every way to drives in constant daily use on the very largest mainframe computer systems, and offer substantially better performance than is available with the disks being used on many medium-scale mainframe computer systems.

D.4 Disk Media

See Appendix B for information on the 9370-series disk drives and disk packs.

D.5 Disk Organization under DOS.D

This chapter describes the logical organization of the data on the disk when operating under DOS.D and how it relates to the general DOS file concepts as described in the chapter on System Structure. In this chapter it is assumed that the user is familiar with these concepts and has read and is familiar with the basic DOS file structuring.

D.5.1 Logical Drive Mapping

Under DOS.D each physical volume is broken into two logical drives. This is done for reasons of addressing. It is simply not possible to address all of the sectors on an entire 9370 disk volume using only two bytes of physical disk address, and the two byte physical disk address is central to all of Datapoint Corporation DOS's operations. It is not practical to change this characteristic without making changes which would result in invalidating many user-written programs and many large systems which run under the DOS. Therefore the disk was broken into two halves, and one bit of the effective physical disk address is taken from the logical drive number. For the 9374 drives the removable disk is one logical drive, and the fixed disk is a

second logical drive.

When using the 2316-type disk pack, each logical drive appears to be 253 cylinders (numbered 0-252 decimal) of eight 24-sector tracks each. The first eight recording surfaces on the disk pack (heads on the 9370 drive are numbered from zero to nineteen starting at the top of the disk drive) correspond to the first 203 cylinders on the first logical drive (the even-numbered one). The next eight recording surfaces on the disk pack correspond to the first 203 cylinders on the second logical drive (the odd-numbered one). The first 203 cylinders on each logical drive is referred to as primary addressing space. Mapping of disk space within primary addressing space is done in an algorithm identical to that used under DOS.B.

Physical heads sixteen and seventeen (and the corresponding recording surfaces on the disk pack) correspond to logical cylinders 203-252 on the even logical drive; heads eighteen and nineteen correspond to logical cylinders 203-252 on the odd logical drive. These cylinders of each logical drive are referred to as the extended addressing space. Since DOS.D assumes that each cylinder consists of eight tracks, each of logical cylinders 203 through 252 are mapped across four physical cylinders of two tracks each from the center of the pack outward. In this way, disk space within primary and extended addressing space can be dealt with by DOS.D in a uniform way at all but the very lowest levels of the disk read/write driver.

Using the 9374/9375 disk drive, there is no extended addressing space, only 203 cylinders of 8 24-sector tracks each. The disk platter itself has 406 cylinders of 2 48-sector tracks each, but the disk controller provides address conversion so the physical structure is transparent to the processor.

D.5.2 Size of a Logical Drive

D.5.2.1 Models 9370-9373

Each logical drive is eight tracks on each of 253 cylinders. This results in 2024 tracks of 24 sectors each, or a total of 48,576 total sectors on a logical drive. Of these, about 48,000 remain when the DOS has been generated, the system tables constructed on the disk, and a few basic commands loaded.

D.5.2.2 Models 9374/9375

Each logical drive is eight tracks on each of 203 cylinders, yielding 1624 tracks of 24 sectors each, or 38,976 sectors on a drive. Of these total sectors, about 38,400 remain when the DOS has been generated, the system tables constructed, and a few basic commands loaded. There is one unused cylinder on each platter-logical cylinder 203, physically cylinders 406 and 407. These innermost cylinders are not normally addressable and are not even formatted by INIT9370. A test program for long-term reliability testing is planned which will require exclusive use of these cylinders.

D.5.3 Cluster Mapping

Because there are eight bits per byte in the cluster allocation table (or CAT for short), and it is desirable to maintain one byte in the CAT per cylinder of available space on the drive, each track on a logical drive represents one DOS cluster, and is represented in the CAT by exactly one bit. Since the DOS uses eight tracks per logical cylinder, this results in exactly eight clusters per cylinder of twenty four sectors each.

Due to the fact that space is always allocated in terms of an integral number of clusters, this implies that the minimum file size under DOS.D is twenty-four sectors and that file size will always be a multiple of this number. It turns out that choosing a full track as the smallest allocatable unit of space has other advantages as well from a system standpoint, since it allows some programs (like COPY) to make several simplifying assumptions about the data in a file which enables them to copy data and reference information in a file substantially more readily and efficiently than would be otherwise possible.

D.5.4 Segments under DOS.D

Space under Datapoint Corporation's DOS is always allocated in contiguous chunks of clusters called segments. When space is allocated, the largest segment on a drive up to the maximum possible sized segment is allocated, to keep the file as free of fragmentation as possible. By limiting the allocation size to the size of a full segment, the problem of allocating all available space on a disk to a first scratch file before a second one is subsequently opened is minimized. If several scratch files are opened and space in them is allocated at regular intervals, the resultant segments will be interleaved, resulting in minimized

access time as the heads randomly access throughout the scratch area. The desire to make segment size small (to minimize file space conflicts and help to optimize use of space on the drive) and yet large (to maximize processing speed, maximize file size and minimize the number of RIB accesses) resulted in a segment size of thirty-two clusters. This compromise results in a 768-sector segment (ten clusters of eight tracks of 24 sectors each) allowing a maximum file size of over 38,000 sectors.

D.5.5 Maximum File Size

Under DOS.D, the maximum file size available is 38,397 data sectors. This number is the result of 126 segment descriptors in the RIB, each of which points at one segment of 32 tracks of 24 sectors each:

24 sectors x 32 tracks x 126 segments = 38,400 total sectors

Since the first two sectors of each file under the DOS are used for the RIB and its copy, and the last sector of most files for an end-of-file mark, that leaves 38,397 sectors available to the user for the storage of his data. Files longer than this number will have to be segmented or logically concatenated at the user program level, the same as would be necessary for files larger than about 9600 sectors on the 9350 series disks.

D.5.6 Cluster Allocation Table and Directory

Each logical drive under DOS.D contains its own directory and cluster allocation table, just as for all Datapoint Corporation DOS. There are sixteen directory sectors on each logical drive, located in consecutive sectors starting at sector seven on logical track zero of cylinder zero. Therefore, the sectors go from sector seven to sector 026 (octal). The cluster allocation table is at sector zero of logical track zero, cylinder zero. The lockout cluster allocation table is at sector one of logical track zero, cylinder zero. The hashed directory index is at sector two of track zero, cylinder zero. The backup copies of each of these are in the corresponding locations of logical track one of the same cylinder.

The Hashed Directory Index, maintained by the DOS, resides in sector two of track zero, cylinder zero. This table enables directory lookups to go about four times faster (on full disk directories) than was possible under DOS.B Version 1. The technique works as follows:

Given an eleven byte file name and extension, an arithmetic/logical operation upon the file name results in an eight-bit quantity referred to as a hash code. This code is essentially a condensation of the 11 bytes of file name and extension information into only one byte. Obviously, the information is not complete; there are only 256 distinct eight-bit hash codes possible, while there are literally billions of legal file names and extensions. However, the condensation of information is such that looking at the hashed directory index allows the DOS to substantially restrict the range of directory sectors it must examine when doing a directory lookup. Each hash code for the file names in the directory is stored into the hashed directory index, offset by the physical file number (PFN) of the file with the corresponding name and extension.

Note that there is a calculated danger in the hashed directory approach. The danger is that if the hashed directory index is overwritten or otherwise destroyed accidentally, files may become inaccessable even though they are clearly shown (by doing a CAT command on the disk, for example) to be present. When this occurs on a disk the technique to repair the disk is the REPAIR command. When the REPAIR command is almost finished, simply specify that the Hashed Directory Index is to be rewritten to the disk. This causes the HDI to be regenerated from the actual disk directory and rewritten. In general, the Hashed Directory Index is rarely if ever destroyed in actual disk usage, and contributes greatly to overall system performance if many directory lookups are being done.

D.6 Internal DOS Parameterization

This section describes the DOS-dependent details of the parameterization of DOS.B system routines.

D.6.1 Physical Disk Address Format

Under DOS.D physical disk addresses are represented in a two-byte format in a manner quite similar to that used under the other DOS. The most significant byte (which is traditionally placed in the D register) is the cynlinder number. The less significant byte (usually placed in the E register) has its most significant three bits representing a cluster number within the cylinder (or logical track number in the specific case of DOS.D) and the least significant five bits representing the sector number within the specified cluster.

APPENDIX E. COMPARSION CHART FOR DOS'S

	DOS.A	DOS.B	DOS.C
min.processor req.	16K 2200	16K 2200	16K 1100
min.disk controller buffers required	1K	4K	1K
max drives/system	4	2	4
platters/phy. drv.	1	11 or 2	1
surfaces used	2	8 or 2 [16] or [4] (20)	1
cylinders used	203	203	76 (77)
tracks used	406	1,624 [3,248] (4,060)	76
sectors/track	24	24	13
sectors/drive	9,744	38,976 [77,952] (97,440)	1,001
bytes/sector	256	256	256
bytes/drive	2,494,464	9,977,856 [19,955,712] (24,944,640)	256,256
user sectors	9,600	38,400	800
user bytes	2,457,600	9,830,400	204,800
sectors/cluster	6	24	3
clusters/track	4	1	4
clusters/cylinder	8	8	4
max.clusters/seg.	32	10	32
max.sectors/seg.	192	240	96
max.segments/file	126	126	126
max.sectors/file (including RIB's)	9600	30,240	800
directory search	HDI	HDI	DMB

Except as noted otherwise, each item is a logical value for one logical drive. Any item enclosed in square brackets [] is a logical value for a physical drive if more than one logical drive is present on a physical drive. Any item enclosed in parentheses () is a physical value for a physical drive if it differs significantly from the number actually used under the DOS.

Right-hand values under DOS.B and DOS.D (following the "or") are

values for 9374/9375 drives, when different from the values for the 9370/9372 drives. In all cases, physical values (values in parentheses) refer to the 9370/9372 drives; the physical value for 9374/9375 drives is always the same as the logical value for a full physical drive (value enclosed in square brackets).

	DOS.D	DOS.E
min.processor req.	48K 5500	48K 5500
min.disk controller buffers required	4K	4K
max drives/system	8	4
platters/phys drv.	11 or 2	1
surfaces used	10 or 2 [20] or [4]	2
cylinders used	253 or 203	203
tracks used	2,024 or 1624 [4,048] or [3248]	406
sectors/track	24	24
sectors/drive	48,576 or 38,976 [97,152] or [77,952]	9,744
bytes/sector	256	256
bytes/drive	12,435,456 or 9,977,856 [24,870,912] or [19,955,712]	2,494,464
user sectors	48,000 or 38,400 [96,000] or [76,800]	9,600
user bytes	12,288,000 or 9,830,400 [24,576,000] or [19,660,800]	2,457,600
sectors/cluster	24	6
clusters/track	1	4
clusters/cylinder	8	8
max.clusters/seg.	32	32
max.sectors/seg.	768	192
max.segments/file	126	126
max.sectors/file (including RIB's)	38,400	9600
directory search	HDI	HDI

	DOS.A	DOS.B	DOS.C	DOS.D	DOS.E
Type of Disk Drive	9350 9354	9370 9374	9380	9370 9374	9350 9354
Number of Drives (min-max)	1-4	1-2	1-4	2-8	2-4
Maximum Logical Drives	4	4	4	16	4
Type of Disk	Scotch 92-204 or equivalent	Scotch 911-0 or equivalent for 9370 /9372 Datapoint Model 80428 for 9374	IBM 128-byte soft-sectored diskette, or equivalent	Scotch 911-0 or equivalent for 9370 /9372 Datapoint Model 80428 for 9374	Scotch 92-204 or equivalent

	DOS.A	DOS.B	DOS.C	DOS.D	DOS.E
PDA of CAT	0,0	0,0	041,0303	0,0	0,0
PDA of CAT Backup	0,040	0,040	042,0303	0,040	0,040
PDA of Lockout CAT	0,1	0,1	043,0303	0,1	0,1
PDA of Lockout CAT Backup	0,041	0,041	044,0303	0,041	0,041
PDA of HDI	0,2	0,2	NA	0,2	0,2
PDA of HDI Backup	0,042	0,042	NA	0,042	0,042
Directory Location	cylinder 0 sectors 6 to 025	cylinder 0 sectors 5 to 024	sector 014 of each cylinder from 021 to 040	cylinder 0, sectors 7 to 026	cylinder 0 sectors 6 to 025
Directory Backup Location	cylinder 0 sectors 046 to 065	cylinder 0 sectors 045 to 064	sector 014 of each cylinder from 1 to 020	cylinder 0 sectors 047 to 066	cylinder 0 sectors 04 to 065

APPENDIX F. DISK DATA FORMATS

F.1 Disk Data Formats

The DOS itself does not deal with the user's data below the record level. It only keeps track of where the records are, allowing the user to format the data in any manner he pleases. The user is presented with records that are 253 bytes long. The system keeps the physical file number in the first physical location of each sector and the system logical record number of the given record in the second (LSB) and third (MSB) physical locations of each sector. This is done to insure that the record obtained is the record desired. The last 253 bytes may contain anything the user chooses. There are, however, some assumptions made by the DOS and the programs supplied with it that deal with disk data. These assumptions fall into several classes described below. The two types normally of greatest interest are object records and symbolic data records. Object records include all records that are to be loaded into memory by the DOS loader. Symbolic data records include all records that are to be handled by the standard data handling programs. These programs include the general purpose editor, the assembler, DATASHARE, RPG II, DOSBASIC, and the DATABUS programs (both source lines for the various compilers and data records handled by the resulting programs).

F.2 OBJECT File Format for Disk

Object files contain binary data which can be loaded using the system loader and then executed. Multiple logical records can be grouped into one physical block.

BYTE	CONTENTS
1	0 => object record-or-0377 => end of block
2	H - load address for record
3	L
4	-H - complement of load address
5	-L
6	count of data bytes following

End of file is indicated when the count byte has a value of zero. For the end-of-file record, the value of HL is the entry

point address to jump to. The object file created by the ASSEMBLER has a system loader object format.

Logical Record Number	Byte #	Description
LRN 0 (RIB)	0	Physical File Number
	1	Logical Record Number (LSB)
	2	Logical Record Number (MSB)
	3	0377
	4	Segment Descriptor 1
	5	
	5	Segment Descriptor 2
	.	
	.	
	.	
	2N+2	Segment Descriptor N
	2N+3	
	2N+4	0377
	2N+5	0377
LRN 1 (RIB COPY)		
LRN 2	0	Physical File Number
	1	Logical Record Number (LSB)
	2	Logical Record Number (MSB)
	3	0 - indicating data block
	4	Starting address of block (LSB)
	5	Starting address of block (MSB)
	6	One's complement of LSB of starting address
	7	One's complement of MSB of starting address
	8	Block length (n)
	9	Beginning of data
	.	
	.	
	.	
	n+9	0 - Next data block
	n+10	Starting address of block (LSB)
	n+11	Starting address of block (MSB)
	n+12	One's complement of LSB of starting address
	n+13	One's complement of MSB of starting address
	n+14	Block length (m)
	n+15	Beginning of block data
	.	
	.	

```
                    .
                    n+m+15- Next data block
                    .
                    .
                    .
                    .       0377 - End of Record

LRN 3               0       Physical File Number
                    1       Logical Record Number (LSB)
                    2       Logical Record Number (MSB)
                    3       0 - Next data block
                    .
                    .
                    .

LRN N               0
                    .
                    .
                    .
                    .       0 - Last data block
                    .       Transfer address (LSB)
                    .       Transfer address (MSB)
                    .       One's complement of the LSB of the
                            transfer address
                    .       One's complement of the MSB of the
                            transfer address
                    .       0 - block length equal to zero signifies
                            end-of-file
```

F.3 Relocatable Code Formats

Relocatable object code is initially assumed to be starting at location 010000 until a "select new PAB" or "select new location" code is encountered.

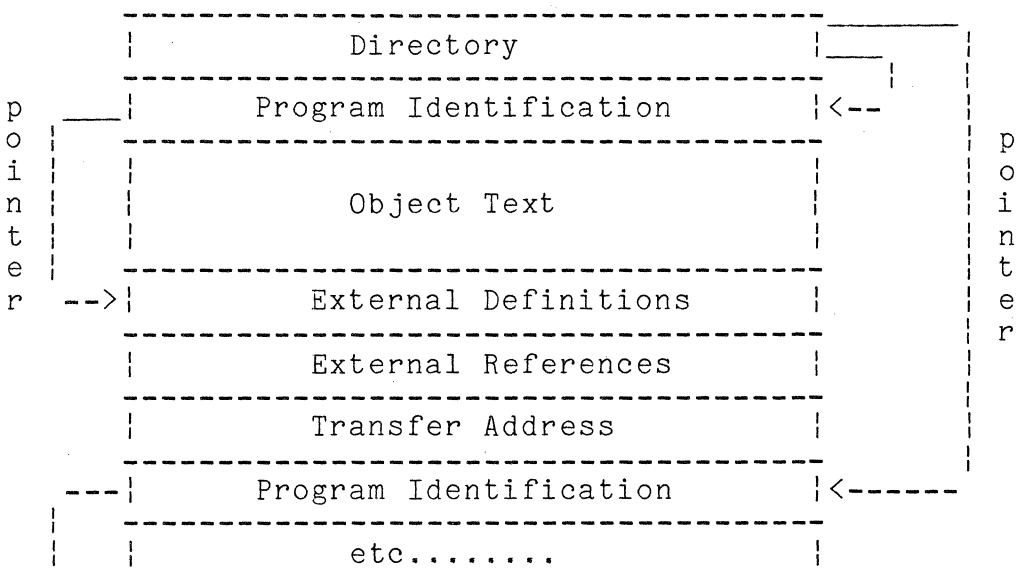
Each sector containing relocatable code starts with a one byte header containing sector contents code. The relocatable code in each sector is followed by a byte containing binary zero.

Sector contents codes are:

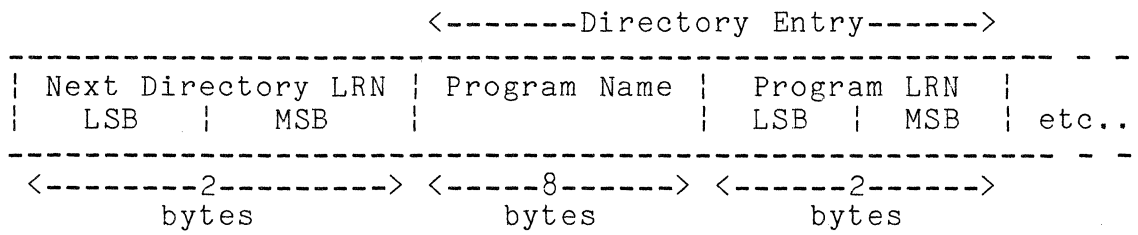
```
0200  Directory
0201  Program Identification
0202  Object Text
0203  External Definitions
0204  External References
```

0205 Transfer Address

Relocatable code files are in library form as follows:

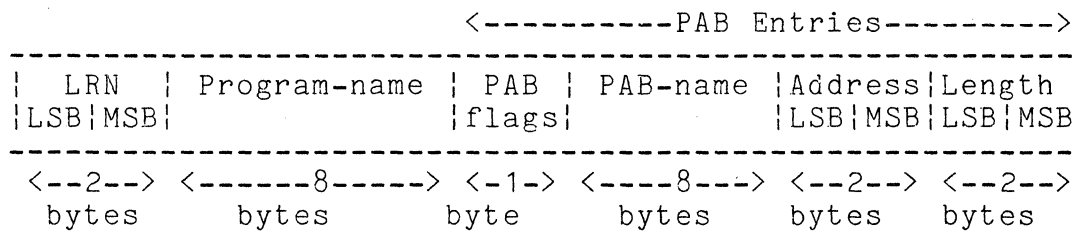


F.3.1 Directory



A directory entry is required for each object program in a library. The first sector of the object code library is reserved as a directory for the first twenty-four programs in the library. If the library contains more than twenty-four programs, a pointer is generated that points to the LRN of the next directory sector (the sector following the twenty-fourth object program). The last directory sector used has a pointer set to 0377, 0377.

F.3.2 Program Identification

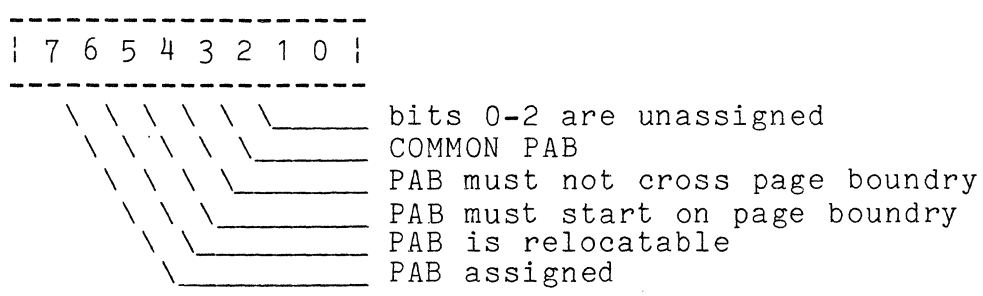


LRN is a pointer to the first sector following object text (the first external definition sector, or the first external reference sector, or the transfer address if there are no definitions or references).

The program name is an eight character name of the program, as reflected in the program id record.

Each PAB (program address block) defines a separate address counter used to assign memory locations. Up to fifteen PAB's can be defined for each program (PAB numbers 1-15). Flag bits are used to indicate relocatability and page sensitivity.

PAB flags:



F.3.3 Object Text

Relocatable object text is interspersed with control bytes used by the linkage editor in creating absolute code.

F.3.3.1 Memory Location

Codes 0160 and 0161 are used to define starting memory locations.

Select New PAB

0160	PAB

PAB defines the number of the Program Address Block to be used for the object code that follows. If the PAB is not in use, the new location will be zero.

Select New Location

0161	LSB	MSB

LSB and MSB define the new location in the current PAB of the next byte of object code.

F.3.3.2 Absolute Text

Codes 0001-0077 precede code and data that does not require relocation.

Absolute Text

1-0077	1-63 absolute text bytes

The code is a count of the number of absolute text bytes that follow.

F.3.3.3 Complex Relocatable References

Codes 0100-0157 are used to define operators and operands of complex expressions that are evaluated by the linkage editor during relocation. Complex expressions are in encoded Polish Postfix notation.

Push Relocatable Location on Stack

0100+PAB	LSB	MSB

PAB, LSB and MSB define the assembled memory location.

Push External Reference on Stack

0120+MSB	LSB

MSB and LSB are an index to an external reference entry.

Push Binary Value on Stack

0140	LSB	MSB

LSB and MSB are a 16 bit binary integer.

Operators:

<	>	.OR.	.XOR.
0141	0142	0143	0144

.AND.	+	-	*
0145	0146	0147	0150

/	Negate	.MOD.
0151	0152	0153

Codes 0141-0153 are expression operators.

Pop Result of Evaluation from Stack:

Pop LSB	Pop MSB	Pop LSB-MSB	Pop MSB-LSB
0154	0155	0156	0157

Codes 0154-0157 terminate evaluation of complex expressions and indicate the form of the absolute code to be generated.

F.3.3.4 Simple Relocatable References

Codes 0200-0377 are used for simple relocatable references consisting of a single relocatable symbol or relocatable symbol plus a non-relocatable displacement. Codes for simple relocation can be decoded as follows:

```
-------------------
| 7 6 5 4 3 2 1 0 |
-------------------
   \ \ \ \ \_ bits 0-3 are part of relocation definition
    \ \ \ \__ external reference
     \ \ \___ inverted address (MSB-LSB)
      \ \____ 16 bit address
       \_____ simple relocatable memory reference
```

LSB Reference

| 0200+PAB | LSB |

LSB defines the relocatable memory location.

MSB Reference

| 0240+PAB | LSB | MSB |

PAB, LSB and MSB define the relocatable memory location. A full sixteen bit address must be given in case a carry occurs between LSB and MSB during relocation.

LSB-MSB Reference

| 0300+PAB | LSB | MSB |

PAB, LSB and MSB define the relocatable memory location.

MSB-LSB Reference

| 0340+PAB | LSB | MSB |

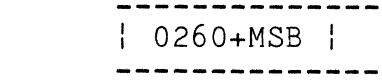
PAB, MSB and LSB define the relocatable memory location.

LSB External Reference

| 0220+MSB | LSB |

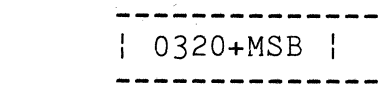
MSB and LSB are an index to an external/forward reference entry table.

MSB External Reference



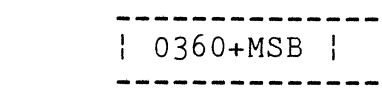
MSB and LSB are an index to an external/forward reference entry table.

LSB-MSB External Reference



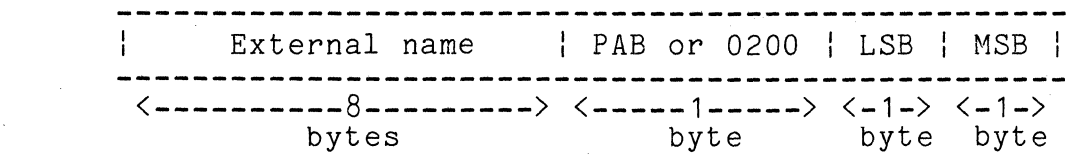
MSB and LSB are an index to an external/forward reference entry table.

MSB-LSB External Reference



MSB and LSB are an index to an external/forward reference entry table.

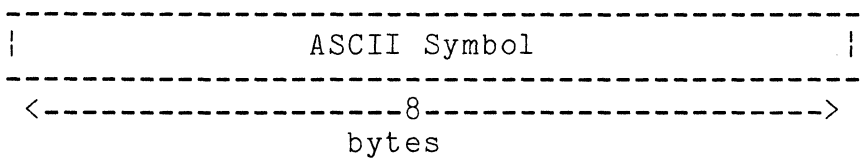
F.3.4 External Definitions



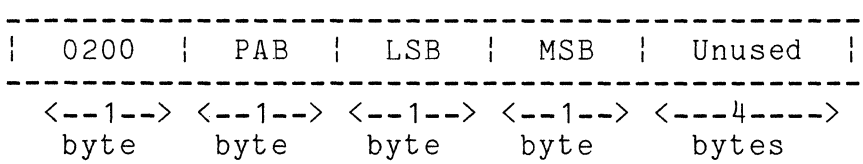
External definitions are external symbols made available to other relocatable modules. External references made by other relocatable modules are linked to external definitions as discussed in Chapter 1. The location of each relocatable external definition is defined by PAB, LSB and MSB. A flag (0200), LSB and MSB define non-relocatable external definition values. Up to twenty-two external definitions can be defined in each external definition sector. All external definition sectors for a given program must be contiguous, and not intermixed with external reference sectors.

F.3.5 External and Forward References (4096 maximum)

External Reference



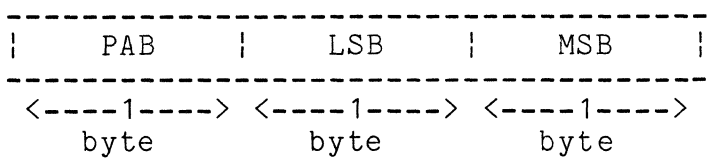
Forward reference



A forward reference is defined as a reference whose value is unknown at some given time in the relocatable module's creation, but whose value is known later, and then is plugged into the forward reference table.

All external reference/forward definition sectors must be contiguous.

F.3.6 Transfer Address



PAB, LSB and MSB define the starting location in the program. If PAB=0377, a starting location was not specified.

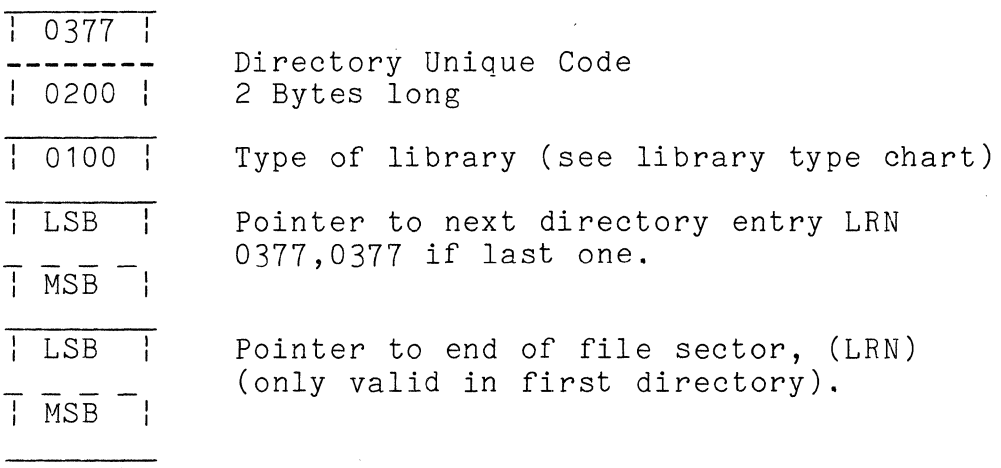
F.4 Format of Library Files

The Library is constructed from two types of entries, Directory Entries and members.

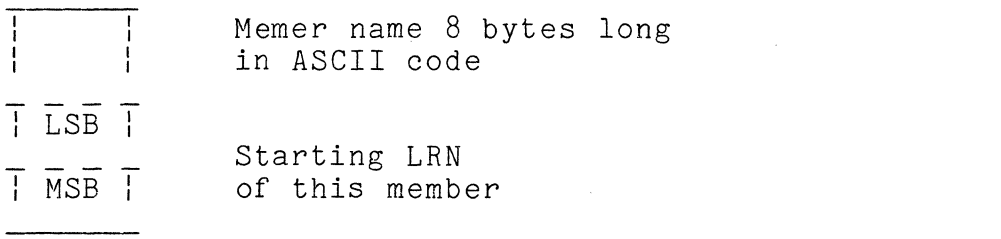
F.4.1 Directory

The first entry of the library file must be the first Directory Entry. Additional directory entries are formatted as required and linked into the directory chain. Each directory has two major parts:

1) The directory header which is 7 bytes. The format is as follows:

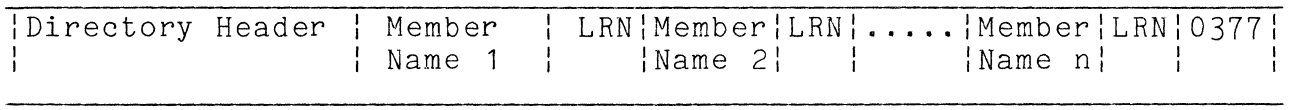


2) Member name entries, each one is 10 bytes long.



One directory entry can contain a maximum of 24 member names. All unused member name entries will be set to 0377's. A deleted member will be set to 0377's.

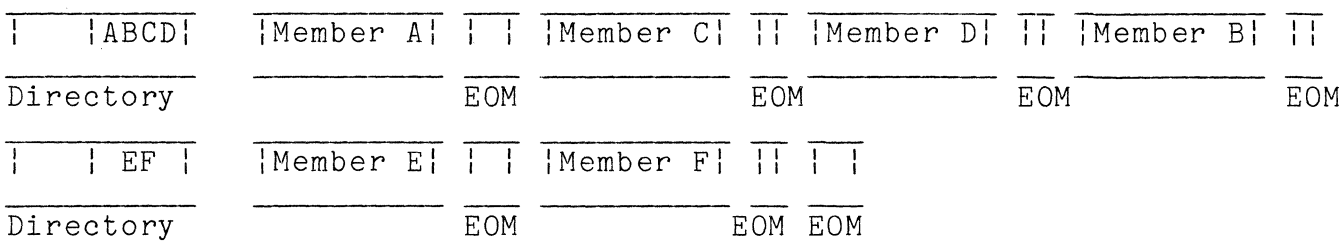
An entire directory entry:



F.4.2 Members

The members are the second type entry of the library. Each member is pointed to by the member name pointer in one of the directory entries. Each member is terminated by an end of member (EOM) code. The EOM is indicated by a sector which contains six bytes of 000 followed by 010.
NOTE: EOM indicates only the end of this member not the end of the library.

A simple library file format



F.4.3 Library Type Chart

If the library contained more than 24 members another directory entry would be placed into the chain of directories.

The following is the bit chart for library types

```
1.  ...   ...   Reserved
.1  ...   ...   Absolute
..  1..   ...   Relocatable
 .  .11   111   Undefined
```

F.5 DATABUS Code File Format

DATABUS files contain code produced by the DATABUS compiler for use by its interpreter. All blocks are 251 bytes long.

BYTE	CONTENTS
1	040 - DATABUS code file indicator
2	H - load address
3	L
4	-H - complement of load address
5	-L

End of file is indicated by bytes 1 through 6 being binary zeros, followed by a binary three.

F.6 DATAFORM Data File Format

Every record created by a DATAFORM form is stored consecutively on the disk terminated with a 015 designated as the end of logical record character. Disk sector boundaries are transversed by placing a 003 to represent the end of physical record. An end of file mark is six zeros followed by a 003 beginning at the start of the next unused sector. This complies with Datapoint's DOS text file structure. However, other characters immediately following the 003 are necessary record descriptors to allow record form linkages and the record backspace feature to be implemented in DATAFORM. The first character following the end of physical record character, 003, represents the form number that created the first logical record starting in that sector, biased by 4. The character immediately following is the absolute address in the sector of the first character of the

logical record. (Note that the first data character of every sector starts at address 003.) There must be a similar pair of characters describing every logical record that starts in that sector. These character pairs must be in that sector and in consecutive order. (i.e. The first pair relates to the first record, the second pair to the second record, etc.) The remainder of the sector, if unused, is filled with zeros. DATAFORM 1100 Version 1 may use the entire 253 bytes available in the sector. However, DATAFORM Version 2 does not use the last two bytes of every sector, only 251 bytes are used.

F.7 MULTIFORM File Format

The first sector of a Multiform file contains information concerning the file name, form library relating to the data, and end of file position. The format of this header sector is described below. The first byte of a sector has a sector address of zero.

SECTOR ADDRESS	DESCRIPTION
0- 2	Reserved for DOS
3	Contains a byte value of 003
4- 11	First 8 characters of the data file name
12- 14	Three characters of the data file extension
15- 16	LRN of the last sector which has data written to it. Must be in LSB,MSB format
17- 24	First 8 characters of the form file name
25- 27	Three characters of the form file extension
28-251	Not care conditions
252	Contains a byte value of 000
253	Contains a byte value of 003
254-255	Reserved for DOS

All records are now written consecutively in a non-space compressed format. Each record is terminated by an 015. The end of the physical record is indicated by an 003. Bytes after the 003 contain special information that Multiform uses. This information is right justified in the sector, which will be described from right to left.

SECTOR ADDRESS	DESCRIPTION
253	Contains a byte value equal to the number of records that start in that sector plus the value 3. Each record that starts in the sector has two bytes that describe its position and the form that created it.
252	Contains a byte value equal to the form number of the form that relates to the last record that starts in the sector.
251	The true sector position of the last record that starts in this sector.

The next preceeding byte pair describes the next to the last record that starts in that sector in the same format as described above. These byte pairs are repeated for every record that starts in that sector. The end of physical record preceeds these record description byte pairs by no more than one character. The exception to this is the last sector in the file which contains data. In this case, immediately preceeding the record description byte pairs will be a byte whose value is the true sector address of the end of physical record character. Note, if no record begins in this sector, sector byte address 253 will contain an 003 and the preceeding byte will have the sector address of the end of physical record character. The next sector in the file will contain the standard DOS end-of-file mark.

F.8 TEXT File Format

TEXT files typically contain data, source statements, or whatever is meaningful to the user. The requirement is that the data contained in the text file must be equal to or greater than 040 (space). The only bytes less than 040 which are allowed are the following:

CONTROL BYTE	SYMBOL	MEANING
000		NULL. The NULL control byte is used in the indication of the end of the file.
003		END-OF-MEDIUM. No more meaningful data is contained

in this block. The EM is NOT a data byte but must be within the block.

011<cnt> SPACE COMPRESSION. The byte following the 011 is a binary count of spaces which have been compressed. <cnt> can be between 2 and 255, inclusive. The 011<cnt> sequence MUST not be split across block (sector) boundaries.

015 END-OF-RECORD. The EOR, also the Enter [ENT] or Carriage Return character, indicates the end of the logical record. It is NOT a data byte.

032 DELETE. The DEL byte indicates the data byte is deleted. The DEL is NOT a data byte. Entire records (including the EOR indicator) can be deleted by over-writing them with DEL bytes.

There is no explicit maximum size for a logical record. A logical record can span as many blocks (sectors) as necessary, within the capacity of the device. A physical block must be less than or equal to 251 bytes, this includes any necessary EOR bytes and the trailing EM byte. Text files can be either compressed or non-compressed. Compressed implies both space and record compressed, using the CMP and EOR control bytes, and filling the block to the maximum of 251 bytes. Non-compressed format has no space or record compression, there is a one for one correlation between logical record and physical block, and the maximum size of the record is 251 bytes, including the EOR and EM control bytes.

End of file is indicated by bytes 1 through 6 being binary zeros <NUL>, followed by a binary three <EM>.

F.9 ISI File Format

The indexed file is a normal GEDIT-compatible text file. The ISAM file is of the following format:

First record - header record

- 0-10 - indexed file name of form filenameext
- 11 - PFN of the ISAM file
- 12 - the sector of the ISAM file RIB
- 13 - the cylinder of the ISAM file RIB
- 14 - PFN of the indexed file
- 15 - the sector of the indexed file RIB
- 16 - the cylinder of the indexed file RIB
- 17-18 - OBSOLETE
- 19 - OBSOLETE

```
      20-22 - last record used in data file (BUFADR, LRN LSB, LRN MSB)
      23-25 - next free entry in ISAM file (BUFADR, LRN LSB, LRN MSB)

Second sector - highest level
  of intermediate level form but contained within a single sector

Third+ sectors - lowest level
   of form:
        KEY/015/NEXBUF/NEXSEC/NEXCYL/RECBUF/RECLSB/RECMSB//KEY....
        as key cannot be split over sector boundary,
        sector is filled with 0377's
  KEY - uncompressed ASCII key with trailing spaces truncated
        0 -> first record
        0377 -> last record
  NEXBUF - buffer address of the next key, 0 implies next sequential
  NEXSEC - sector address of the next key
  NEXCYL - cylinder address of the next key
  RECBUF - buffer address of the indexed record
  RECLSB - logical record number LSB of the indexed record
  RECMSB - logical record number MSB of the indexed record

N+ sectors - intermediate levels
   of form:
        KEY/012/NEXSEC/NEXCYL//KEY...
         as key cannot be split over sector boundary, sector is filled
         0377'
   KEY - uncompressed ASCII key with trailing spaces truncated
         0 -> first record
         0377 -> last record
   NEXSEC - sector address of the next-lower-level key
   NEXCYL - cylinder address of the next-lower-level key

The ASCII keytag file is of the format:
      RECLRN/RECBUF/KEY/015//KEY...
    RECLRN - 5 byte ASCII decimal logical record number of the indexed
             key
    RECBUF - 3 byte ASCII decimal buffer address of the key
                the ASCII decimal numbers have leading blanks
    KEY - compressed ASCII key with trailing spaces truncated
```

Manual Name_____

Manual Number_____

READER'S COMMENTS

Did you find errors in this manual? If so, specify by page.

_____

_____

_____

_____

_____

Did you find this manual understandable, usable, and well-organized? Please make suggestions for improvement.

_____

_____

_____

_____

_____

_____

Name_____ Date_____

Organization_____

Street_____

City_____ State_____ Zip Code_____

All comments and suggestions become the property of Datapoint.

Fold Here

Fold Here and Staple

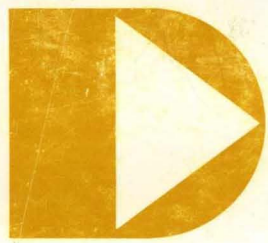
First Class
Permit
5774
San Antonio
Texas

BUSINESS REPLY MAIL
No Postage Necessary if mailed in the United States

Postage will be paid by:

DATAPOINT CORPORATION
Product Marketing
8400 Datapoint Drive
San Antonio, Texas 78284

DATAPOINT CORPORATION



The leader in dispersed data processing ™