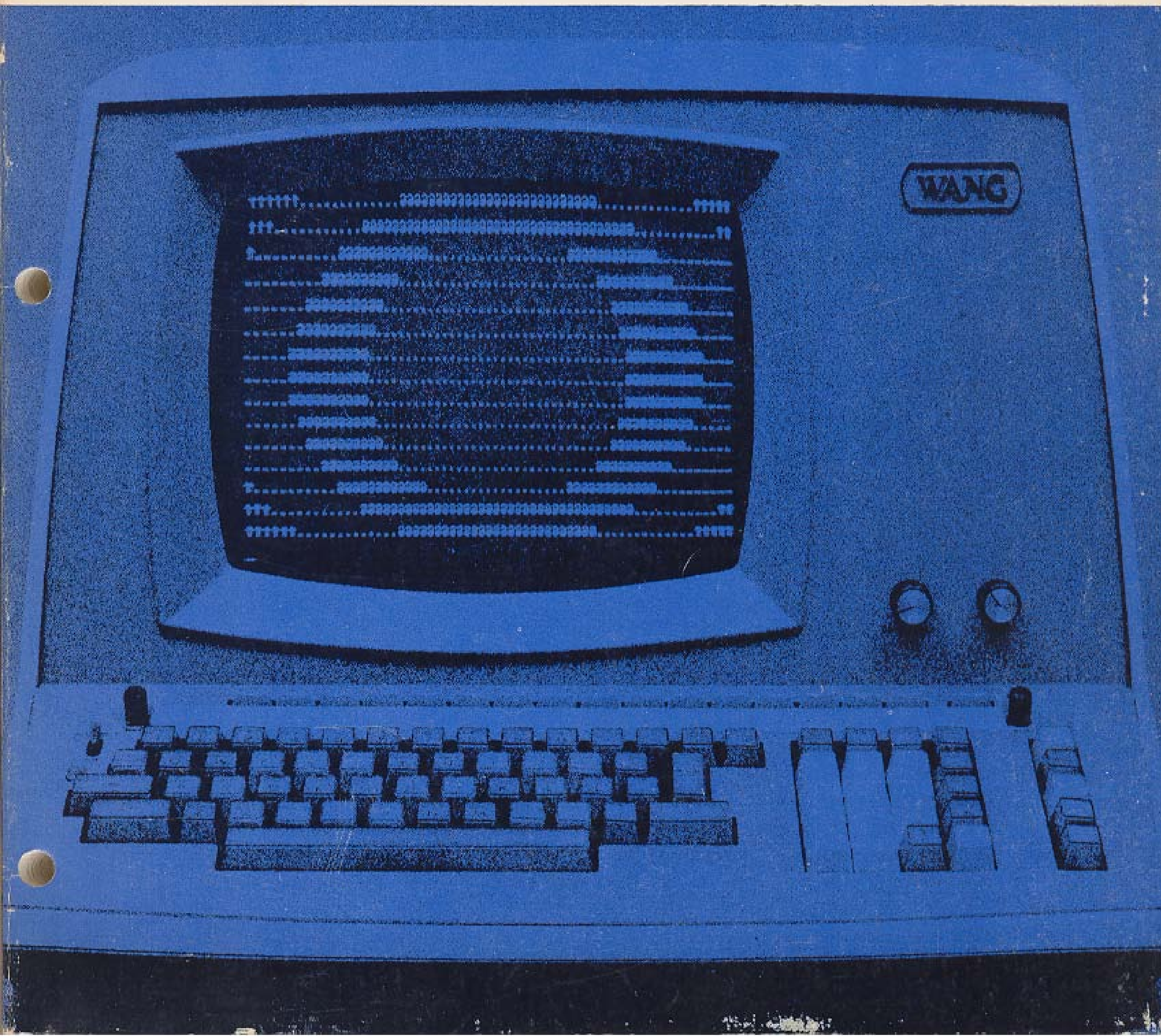
WANG

SYSTEM 2200 DISK MEMORY REFERENCE MANUAL

(Models 2230, 2260, 2270, 2224, and 2230MX)

SYSTEM 2200



SYSTEM 2200

Disk Memory

Reference Manual

(MODELS 2230, 2260, 2270, 2224, and 2230MX)

WANG LABORATORIES, INC.
ONE INDUSTRIAL AVENUE, LOWELL, MASSACHUSETTS 01851, TEL. (617) 851-4111, TWX 710 343-6769, TELEX 94-7421

Disclaimer of Warranties and Limitation of Liabilities

WANG LABORATORIES, INC.
ONE INDUSTRIAL AVENUE, LOWELL, MASSACHUSETTS 01851, TEL. (617) 851-4111, TWX 710 343-6769, TELEX 94-7421

HOW TO USE THIS MANUAL

The System 2200 Disk Reference Manual is designed to serve as an operator's guide to the disk hardware, as a programmer's guide to disk control, and as a reference manual for the BASIC statements which govern disk operations. These three functions are, to the extent possible, carried out in individual chapters or groups of chapters.

Chapter 1 provides an introduction to the concept of information storage and retrieval on the disk, including an overview of the disk hardware and a general discussion of the principles of disk operation. Chapters 2 and 3 offer more specific operational data, including power-on and formatting procedures, etc., on the two types of disk drives marketed by Wang. (The "hard disks," Models 2230 and 2260, are covered in Chapter 2; the "flexible disks," Model 2270 series, are covered in Chapter 3.)

The general procedures for addressing and accessing a disk drive under program control are explained in Chapter 4. These procedures are common to all disk models, and to both modes of disk operation.

Two modes of disk operation are provided by the System 2200, Automatic File Cataloging Mode and Absolute Sector Addressing Mode. The sequence of Chapters 5 through 7 constitute a "programmer's guide" to the control of disk operations in the Automatic File Cataloging Mode. This mode automatically performs many of the complex "housekeeping" tasks associated with disk file maintenance, and is recommended for programmers with limited experience in disk programming. Chapter 5 is particularly designed for the novice, serving as a primer in the fundamental concepts of disk management.

Chapter 8 is a reference chapter for the BASIC statements which comprise the Automatic File Cataloging Mode. Information on each statement is presented in a brief, compact format which makes it quickly accessible to the programmer who is already familiar with the general principles of Automatic File Cataloging.

Chapter 9 serves as a programmer's guide for the second mode of disk operation, Absolute Sector Addressing Mode. This mode enables the programmer to directly access any sector on the disk, providing vast flexibility in the design of custom file access routines. It does not, however, automatically perform the many complex housekeeping operations provided by Automatic File Cataloging Mode, and its use is not recommended for inexperienced programmers. Chapter 10 is a companion reference chapter for the Absolute Sector Addressing statements.

Chapter 11, finally, is a hybrid chapter incorporating both hardware and programming information on the disk multiplexers (Models 2224 and 2230MXA/B), which permit a single disk unit to be accessed by several CPU's. Multiplexer owners should consult this chapter before attempting to install or program the multiplexer.

Assorted information of interest to the disk user has been assembled in the Appendices. Separate Appendices provide performance data on the various disk models and explanations of the disk error codes, as well as a bibliography of disk literature and a glossary of disk terminology.

TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION

1.1 The Development of Electronic Data Processing. 1
1.2 Electronic Data Processing and the Wang System 2200. 2
1.3 Random Access Data Storage 3
1.4 The Disk Unit. 3
Models 2230 and 2260 3
Model 2270 5
1.5 The Disk Platter 7
1.6 Sectors. 8
1.7 Track and Sector Numbering 9
Models 2230 and 2260 9
Model 2270 10
1.8 Disk Access Times. 10
1.9 Staggered Arrangement of Sectors in a Track. 11
1.10 The Model 2230/2260 Cylinder Concept 13

CHAPTER 2: MODEL 2230 SERIES (2230 and 2260)
GENERAL INFORMATION AND FORMATTING INSTRUCTIONS

2.1 Unpacking and Inspection 15
2.2 Installation 15
2.3 Power-On Procedures. 16
2.4 Model 2230/2260 Formatting Instructions. 17
Formatting the Removable Disk Platter (Disk Cartridge) . . . 17
Formatting the Fixed Disk Platter. 17
2.5 Format Errors. 18
2.6 Loading and Unloading the Removable Disk Platter
(Disk Cartridge) 18
Unloading the Disk Cartridge 19
Loading a New Disk Cartridge 21
2.7 Handling and Storage of the Removable Disk Platter
(Disk Cartridge) 23
2.8 Machine Interlocks 24

CHAPTER 3: MODEL 2270
GENERAL INFORMATION AND FORMATTING INSTRUCTIONS

3.1 Unpacking and Inspection 25
3.2 Installation 26
3.3 Power-On Procedure 26
3.4 Loading and Formatting Diskettes 26
3.5 Format Errors. 29
3.6 The Write Protect Feature. 30
3.7 Handling and Storage of Diskettes. 31

TABLE OF CONTENTS (Cont.)

CHAPTER 4: ACCESSING THE DISK PLATTERS (MODELS 2230, 2260, and 2270)

4.1 The 'F' and 'R' Parameters 33
4.2 The Disk Device Address. 34
4.3 Addressing the Third Diskette (Model 2270-3) 35
4.4 Limitations of Certain Disk Models 35

CHAPTER 5: AUTOMATIC FILE CATALOGING PROCEDURES

5.1 Introduction 37
5.2 What Is Automatic File Cataloging? 37
5.3 Sectors and Addresses. 38
5.4 Initializing the Catalog 38
5.5 Saving Programs on Disk. 40
5.6 Retrieving Programs Stored on Disk 41
The LOAD DC Command. 41
The LOAD DC Statement. 42
5.7 Listing the Catalog Index. 44
5.8 Saving Data on Disk. 46
The Hierarchy of Data. 46
Opening a Data File on Disk. 47
Saving Data in a Data File on Disk 48
5.9 The Structure of Data Files. 51
Opening a Second Data File on Disk 54
5.10 Re-Opening a Data File on Disk 54
5.11 Retrieving Data from a Data File on Disk 55
5.12 Skipping and Backspacing Over Logical Records in a Data File . . 58
5.13 Testing for the End of File. 61
5.14 Scratching Unwanted Files. 62
5.15 Moving the Catalog from One Platter to Another 63

CHAPTER 6: DISK DEVICE SELECTION AND MULTIPLE DATA FILES

6.1 Introduction 67
6.2 Disk Device Selection. 67
The Device Table 68
Use of File Numbers in Accessing the #3 Platter (Model 2270-3 Only). 70
Why Use the Device Table?. 70
6.3 Maintaining Multiple Open Data Files on Disk 71
Using a Variable to Store the File Number. 76
6.4 The "Current Sector Address" Parameter 76
6.5 Closing a Data File. 79
6.6 Skipping and Backspacing Over Individual Sectors in a Data File. 81
6.7 The 'T' Parameter. 82
Changing the Default Address 85
6.8 Multiple Disk Units. 86
Models 2230, 2260, 2270-1, 2270-2. 86
Model 2270-3 87
Accessing Multiple Disk Units. 87

TABLE OF CONTENTS (Cont.)

CHAPTER 7: EFFICIENT USE OF THE DISK

7.1 Introduction 89
7.2 Program Files Revisited. 89
7.3 Establishing Temporary Work Files on Disk. 90
7.4 Altering the Catalog Area. 92
7.5 Renaming and Re-Using Scratched Files. 93
7.6 Efficient Use of Disk Storage Space. 95
System 2200 Control Information. 96
Inter-Field Gaps 97
7.7 The LIMITS Statement 99
Form 1 of LIMITS 99
Form 2 of LIMITS 100
7.8 Conclusion 102

CHAPTER 8: AUTOMATIC FILE CATALOGING STATEMENTS AND COMMANDS

8.1 Introduction 103
8.2 System 2200 Statements and Commands. 103
8.3 Basic Rules of Syntax. 104
DATALOAD DC. 106
DATALOAD DC OPEN 107
DATASAVE DC. 109
DATASAVE DC CLOSE. 111
DATASAVE DC OPEN 112
DBACKSPACE 114
DSKIP. 115
LIMITS 116
LIST DC. 118
LOAD DC (Command). 119
LOAD DC (Statement). 120
MOVE 122
MOVE END 124
SAVE DC. 125
SCRATCH. 127
SCRATCH DISK 129
VERIFY 131

CHAPTER 9: ABSOLUTE SECTOR ADDRESSING

9.1 Introduction 133
9.2 Specifying Sector Addresses. 135
9.3 Storing and Retrieving Programs on Disk in Absolute Sector Addressing Mode. 136
Saving Programs on Disk with SAVE DA 137
Retrieving Programs Stored on Disk with LOAD DA. 137
The LOAD DA Command. 138
The LOAD DA Statement. 138
9.4 Storing and Retrieving Data on Disk in Absolute Sector Addressing Mode. 140
Storing Data on the Disk with DATASAVE DA. 140
Retrieving Data from Disk with DATALOAD DA 142
9.5 The 'BA' Statements. 144
9.6 Platter-to-Platter Copy. 146

TABLE OF CONTENTS (Cont.)

9.7 Using Absolute Sector Addressing Statements in Conjunction with Catalog Procedures (Binary Search). 148
9.8 Conclusion. 153

CHAPTER 10: ABSOLUTE SECTOR ADDRESSING STATEMENTS AND COMMANDS

10.1 Introduction 155
10.2 Statement/Command Distinction and General Rules of Syntax. . . . 155
COPY. 156
DATALOAD BA. 158
DATALOAD DA. 160
DATASAVE BA. 162
DATASAVE DA. 164
LOAD DA (Command). 166
LOAD DA (Statement). 168
SAVE DA. 170

CHAPTER 11: THE DISK MULTIPLEXERS (MODELS 2224 AND 2230MXA/B)

11.1 Introduction 173
11.2 The Model 2224 Disk Multiplexer. 173
11.3 Installation of the Model 2224 174
Unpacking and Inspection 174
Installation Procedure 174
Power-On Procedure 176
11.4 The Model 2230MXA/B Disk Multiplexer 177
11.5 Installation of the Model 2230MXA/B. 179
Unpacking and Inspection 179
Installation Procedure 179
Power-On Procedure 180
11.6 Multiplexer Operation. 181
11.7 The Manual Hog Mode Switch (Model 2224 Only) 182
11.8 Programmable Hog Mode (Models 2224 and 2230MXA/B). 182

APPENDICES

APPENDIX A DISK ERROR CODES 187
Explanations of all error codes associated with disk operation.

APPENDIX B MODEL 2230 SPECIFICATIONS. 195
Storage capacity, performance information, and general specifications for the Model 2230.

APPENDIX C MODEL 2270 SPECIFICATIONS. 197
Storage capacity, performance information, and general specifications for the Model 2260.

APPENDIX D MODEL 2270 SPECIFICATIONS. 199
Storage capacity, performance information, and general specifications for the Model 2270.

TABLE OF CONTENTS (Cont.)

APPENDIX E   MODEL 2230/2260 DISK CARTRIDGE MAINTENANCE INFORMATION. 201
Tips on handling, labelling, and storing the Models 2230/2260 Removable Disk Platters (Disk Cartridges).

APPENDIX F   DISK CARTRIDGE COMPATABILITY (MODELS 2230 and 2260). . . 204
Discussion of a technique for accessing a Model 2230 disk cartridge in a Model 2260 disk drive.

APPENDIX G   A GLOSSARY OF DISK TERMINOLOGY 209
Brief definitions of commonly-used disk terms.

APPENDIX H   BIBLIOGRAPHY 219
A list of articles and textbooks which discuss disk file access techniques and disk file design philosophies.

APPENDIX I   EQUIPMENT GUARANTEE AND PREVENTIVE MAINTENANCE INFORMATION. 222
Your Wang guarantee, and an explanation of Wang's Maintenance Agreement.

APPENDIX J   SYSTEM 2200 ERROR CODES. 223
A complete listing of all System 2200 error codes.

INDEX   225

LIST OF EXAMPLES

Example Page

5-1: Initializing the Catalog. 39
5-2: Initializing the Catalog ('LS' Parameter Omitted) 39
5-3: Saving a Program on Disk. 40
5-4: Saving Part of a Program on Disk (One Line Number Specified). . 40
5-5: Saving Part of a Program on Disk (Two Line Numbers Specified) . 41
5-6: Loading a Cataloged Program File from Disk. 42
5-7: Attempting to Load a Non-Cataloged Program from Disk. 42
5-8: Chaining a Program from Disk with the LOAD DC Statement 43
5-9: Loading a Program Overlay from Disk 44
5-10: Listing the Catalog Index 44
5-11: Opening a Data File on Disk 47
5-12: Saving Data in a Data File. 49
5-13: Writing an End-of-File (Trailer) Record to a Data File on Disk. 50
5-14: Writing a Data Trailer Record after a Series of DATASAVE DC Statements. 50
5-15: Reopening a Cataloged Data File 55
5-16: Attempting to Reopen a Non-Cataloged Data File. 55
5-17: Reading Data from a Data File on Disk 56
5-18: Saving and Loading One Logical Record 56
5-19: Loading Portions of a Logical Record. 57
5-20: Skipping Over Logical Records in a Data File. 59
5-21: Backspacing Over Logical Records in a Data File 60
5-22: Backspacing to the Beginning of a File. 60
5-23: Skipping to the End of a File 61
5-24: Testing for the End-Of-File Condition 61
5-25: Scratching Unwanted Files 62
5-26: Copying the Catalog from One Disk Platter to the Other. 64
5-27: Checking the Validity of Files after a Move 64

6-1: Storing Disk Device Addresses in the Device Table 69
6-2: Opening a New Data File with a File Number. 74
6-3: Referencing an Open File by File Number 75
6-4: Referencing an Open File by File Number 76
6-5: Closing a Data File by Reassigning Its File Number. 79
6-6: Closing a Specified File with a DATASAVE DC CLOSE Statement . . 80
6-7: Closing All Currently Open Files with a DATASAVE DC CLOSE Statement 80
6-8: Skipping Over a Number of Sectors in a File 82
6-9: Backspacing Over a Number of Sectors in a File. 82
6-10: Accessing More Than One Disk Platter with the 'T' Parameter . . 84
6-11: Using the 'T' Parameter to Access a User-Selectable Disk Platter 85
6-12: Using the 'T' Parameter with a New Default Address. 86

7-1: Reserving Additional Sectors in a Program File. 90
7-2: Opening a Temporary Work File on Disk 91
7-3: Opening More Than One Temporary Work File 92
7-4: Reopening a Temporary Work File. 92
7-5: Changing the Size of the Catalog Area 93

LIST OF EXAMPLES (Cont.)

7-6: Saving a Program in Space Occupied by a Scratched File. 93
7-7: Opening a Data File in Space Occupied by a Scratched File . . . 94
7-8: Opening a Data File in Space Occupied by a Scratched Program File. 94
7-9: Renaming Scratched Program File with the Same Name. 94
7-10: Renaming Scratched Data File Which Is Still Viable. 95
7-11: Form 1 of the LIMITS Statement ('File Name' specified). 100
7-12: Form 1 of the LIMITS Statement ('File Name' and a File Number Specified). 100
7-13: Form 2 of the LIMITS Statement ('File Name' Not Specified). . . 101
7-14: Form 2 of the LIMITS Statement ('File Name' Not Specified). . . 101

9-1: Saving a Program on Disk with SAVE DA (No Line Number Specified). 137
9-2: Saving a Program on Disk with SAVE DA (Two Line Numbers Specified). 137
9-3: Loading a Program from Disk with the LOAD DA Command. 138
9-4: Loading Program from Disk with a LOAD DA Statement (No Line Numbers Specified) 139
9-5: Loading Program Overlays from Disk with the LOAD DA Statement (Two Line Numbers Specified). 140
9-6: Storing Data on Disk with a DATASAVE DA Statement 141
9-7: Saving a Number of Data Records in Sequential Areas of the Disk with SAVE DA 141
9-8: Writing an End-of-File Record in a Data File with a DATASAVE DA END Statement 142
9-9: Retrieving Data from a Data File on Disk with a DATALOAD DA Statement 143
9-10: Testing for the End-Of-File Condition in a Non-Cataloged Data File 144
9-11: Writing an Unformatted Sector with DATASAVE BA. 145
9-12: Reading a Sector from Disk with DATALOAD BA 145
9-13: Copying a Disk Platter. 146
9-14: Copying a Disk Platter. 146
9-15: Verifying Data Transfer Following a Copy Operation. 147
9-16: Performing a Binary Search on a Cataloged Data File 152

11-1: Activating the Programmable Hog Mode (Models 2224 and 2230MXA/B Disk Multiplexers). 183
11-2: Activating the Programmable Hog Mode (Models 2224 and 2230MXA/B Multiplexers) 183
11-3: Deactivating the Programmable Hog Mode (Models 2224 and 2230MXA/B Disk Multiplexers). 184

LIST OF FIGURES

Figure No.		Page
1-1	Model 2230/2260. Both Disk Drives Utilize the Same Physical Housing	4
1-2	Model 2230/2260 Disk Drive and Disk Platters	4
1-3	Model 2230/2260 Disk Unit with Access Assembly and Read/Write Heads	5
1-4	The Model 2270-2 Dual Removable Diskette Drive	5
1-5	The Model 2270-3 Triple Removable Diskette Drive	6
1-6	Model 2270 Diskette Mounted in a Disk Drive, Showing Access Assembly, Carriage, and Read/Write Head	7
1-7	Concentric Circular Tracks on a Disk Platter	8
1-8	Enlarged View of Sections of Several Tracks Showing How Information Is Stored on a Disk Platter	8
1-9:	One Sector of a Track on a Disk Platter	9
1-10	Cross-Section of a Disk Platter Showing Track Numbering on the Upper and Lower Surfaces (Model 2230-1)	9
1-11	Sector Numbering on a Model 2230-1 Disk Platter	10
1-12	Track and Sector Numbering on a Model 2270 Diskette	10
1-13	Staggered Arrangement of Sectors on Model 2230 Disk Platter	12
1-14	Staggered Arrangement of Sectors on Model 2270 Diskette	12
1-15	Staggered Arrangement of Sectors on Model 2260 Disk Platter	12
1-16	The Cylinder Concept (Models 2230/2260)	13
2-1	Model 2230/2260 System Configuration	15
2-2	Switches on Front Panel of Model 2230	19
2-3	Opening the Cartridge Holding Clamps	19
2-4	Removing the Dust Cover	20
2-5	Releasing the Cartridge Lock	20
2-6	Removing the Cartridge from the Cartridge Bowl	20
2-7	Locking the Cartridge onto the Dust Cover	21
2-8	Carrying a Sealed Cartridge	21
2-9	Separating Cartridge from Dust Cover	22
2-10	Aligning Cartridge in the Cartridge Bowl	22
2-11	Cartridge Loaded in Disk Unit	23
2-12	Cartridge Sealed and Locked in Disk Unit	23
3-1	Model 2270 System Configuration	25
3-2	Model 2270 Diskette, with Tab Covering the Write Protect Hole	27
3-3	Opening the Door of Drive #1	27
3-4	Mounting a Diskette	28
3-5	Formatting a Diskette	29
3-6	Model 2270 Write-Protect Feature	30
5-1	The Catalog Index Listing	45
5-2	Catalog Index Entry for DATFIL 1	48
5-3	Catalog Index Entry for DATFIL 1	49
5-4	Updated Catalog Index Entry for DATFIL 1	50
5-5	Logical Record Consisting of One Sector	52
5-6	Two One-Sector Logical Records	53
5-7	Logical Record Consisting of Three Sectors	54
5-8	Logical Records in TEST 1	58
5-9	Skipping Over Logical Records in a Data File	59
5-10	Backspacing Over Logical Records in a Data File	60
5-11	The Catalog Index Showing Scratched Files	63

LIST OF FIGURES (Cont.)

6-1 The Device Table in Memory. 68
6-2 The Device Table with Disk Addresses Stored Opposite File Numbers #3 and #5. 69
6-3 The Device Table with One File Open (DATFIL 1). 72
6-4 The Device Table with One File Open (DATFIL 2). 73
6-5 The Device Table with Disk Device Addresses Stored Opposite File Numbers #3 and #5 and One Open File (DATFIL 2). 74
6-6 The Device Table with Two Open Files. 74
6-7 The Device Table in Memory with Three Open Files. 75
6-8 Device Table Slot for DATFIL 2 77
6-9 Updated Device Table Slot for DATFIL 2. 77
6-10 Updated Device Table Slot for DATFIL 2. 78
6-11 Updated Device Table Slot for DATFIL 2 Following Execution of a DBACKSPACE BEG Statement 79
6-12 The Device Table Default Slot Following Execution of a SELECT DISK B10 Statement 85

7-1 The Program File PROG 1 90
7-2 Layout of the Platter Surface Showing Catalog Index, Catalog Area, and Non-Catalog Area (Used for Storage of Temporary Files) 91
7-3 One Logical Record, Showing Sector Control Bytes and Start-of-Value Control Bytes for Each Field 97
7-4 Inter-Field Gap in a Multi-Sector Record. 98
7-5 A Multi-Sector Record with No Gaps. 99

9-1 Typical Entry in Customer Credit File 148
9-2 Typical Customer Credit File (Sorted in Ascending Order). . . . 149
9-3 Binary Search Technique 150

11-1 The Model 2224 Disk Multiplexer 173
11-2 Typical System Configuration for Model 2224 Disk Multiplexer and Four Stations 175
11-3 Model 2230MXA Master Board and 2230MXB Slave Boards 177
11-4 Connecting Extension Cable with Standard 12-Foot Cable (2230MXA/B) 178
11-5 Typical System Configuration: Model 2230MXA/B Multiplexer, Disk Unit, and Four Attached CPU's. 179

LIST OF TABLES

Table Page

4-1 Disk Platters and the 'F' and 'R' Parameters. 33
6-1 Disk Addresses for Models 2230, 2260, 2270-1, and 2270-2. . . . 87
6-2 Disk Addresses for Model 2270-3 87

Chapter 1
Introduction

1.1 THE DEVELOPMENT OF ELECTRONIC DATA PROCESSING

The efficient maintenance of accurate, up-to-date records is a major problem in all areas of business and science, and has been at least since the time of the ancient merchant who first stumbled upon the idea of record-keeping. The antediluvian merchant's system was simplicity itself. Every time he bought a cow, he made a mark on his stone; every time he sold one, he scratched a mark out. If he were exceptionally clever and industrious, he might even keep track of his profits and losses in a separate record. The system turned out, however, to have one major drawback: if the merchant were even halfway successful, his records quickly got so heavy that he required a brace of oxen just to move them from one location to another.

The shift from rocks to paper alleviated the weight problem - at least for a while - but other problems quickly arose in its place. As scientists' knowledge of the world increased, as businesses became increasingly more complex and their interests more far-flung, the increasing quantity of information which had to be dealt with dictated that the earlier, rudimentary bookkeeping procedures be supplanted with more sophisticated and efficient techniques. The development of new techniques was helped along first by the invention of the abacus and much later by the discovery of algebra. By the 17th century, the steadily rising tide of information prompted a move by many scientists and mathematicians to develop a "calculating machine" which might aid in bookkeeping computations. A number of such machines were, in fact, invented and their development continued throughout the succeeding two centuries. Despite their usefulness in computational work, however, such devices were of negligible value in the tedious process of recording, sorting, and keeping track of voluminous quantities of data. The concept of a completely mechanized system for handling and storing data remained an unrealizable dream for businessmen and scientists until the end of the 19th century, when Herman Hollerith developed a method for storing data on punched cards. Hollerith's technique, which involved coding data in a pattern of punches, was used successfully in the compiling of the United States census in 1890. Because the punched cards could be handled, read, and stored mechanically, the first truly mechanical data processing systems became possible.

Today, of course, Hollerith's great invention, though less than a century old, seems to us very much like ancient history. The inexorable increase in the quantity of information has assumed, particularly in the last quarter century, the proportions of an information explosion, due largely to enormous advances in the technology of communications. In the face of this information explosion, it became clear to businessmen and scientists alike that the traditional methods of manual bookkeeping - the company clerk with his quill pen and massive ledger - simply were no longer adequate in most cases to the task of maintaining accurate, up-to-date files and records. Even mechanical data processing systems staggered beneath the increasing influx of raw data (the population of the United States when Hollerith introduced his new system in 1890 was, for example, just over 23 million; by 1950, 60 years later, the population had grown to more than 150 Million).

Under the circumstances, the advent of the computer in the 1940's and 50's was providential. The computer's importance for data processing can be directly traced to two important features: its vast data storage capacity, and its extremely rapid computation and data processing capability. Continued improvement and refinement of techniques in both of these areas has given rise to today's high-speed, fully automated data processing systems, which have in turn revolutionized the field of data processing. Because electronic circuitry now performs most of the storage and processing operations previously done mechanically or by hand, modern systems are referred to as electronic data processing (EDP) systems.

Because of its broad range of capabilities, the modern EDP system is ideally suited to perform the basic functions required of a data processing system. It is capable of storing large volumes of data (such as inventory files or test records), of reading in new data (such as daily sales or experimental results) and processing this data against the master files (deleting items sold from the inventory list, adding new test results to established records). It is, finally, capable of outputting the results of any or all of these operations to the user, either in the form of printed copy or in a display.

1.2 ELECTRONIC DATA PROCESSING AND THE WANG SYSTEM 2200

Until the very recent past, an electronic data processing system had to be built around a large, complex, and extravagantly expensive computer. Those who could not afford the staggering cost of such a system were generally forced to purchase time on someone else's system (hence the concept of time-sharing). All too often, however, such an arrangement proved inconvenient to the unhappy time-sharer, who found that the system was most available when he needed it least, and least available when he needed it most.

The Wang System 2200 has been designed to provide a viable alternative to time-sharing for small-scale data processing applications. The System 2200 offers a fast, efficient and relatively inexpensive electronic data processing capability for users whose applications do not require the massive storage capacity and extreme speed of a large computer. Because of its versatile design, the System 2200 can be purchased in a variety of different configurations for data processing applications. A typical data processing configuration would include an input device, a Central Processing Unit (CPU), an output device, and a disk storage device, which provides the access speed and storage capacity needed for large volumes of data.

1.3 RANDOM ACCESS DATA STORAGE

The heart of the System 2200 data processing configuration is the disk drive unit, a random-access external storage device which provides a high-volume data storage capacity, along with extremely fast access speed. A major contributing factor to the disk's high access speed is its random access capability. Each storage location on the disk has a unique identification tag or "address," and can be directly accessed by the system. Thus, unlike sequential-access storage devices (such as magnetic tape drives, punched tape readers, and card readers), the disk does not have to read sequentially through a file in order to locate a desired item of information. Instead, the disk can skip over all intervening records and directly access a specified storage location for data storage or retrieval (hence random-access devices are also referred to as "direct-access" devices).

A random access capability is most valuable when interrogating or updating a large file, since in those cases records are not usually accessed in sequential order. Even in sequential-processing operations which do not make use of its random access capability, however, the disk is significantly faster than most other external storage devices.

Wang offers two basic types of disk drives for use with its systems. The Fixed/Removable Disk Drives (Models 2230 and 2260), also called the "hard disks", offer extensive storage capacity and high performance for large on-line data bases; the Removable Diskette Drives (Models 2270-1, 2270-2, and 2270-3) offer a more modest storage capacity and somewhat slower access speeds, appropriate for applications requiring a smaller on-line data base.

1.4 THE DISK UNIT

Model 2230/2260

The Models 2230 and 2260 Fixed/Removable Disk Units contain a pair of disk platters mounted horizontally on a drive shaft. The platters are mounted one above the other, somewhat as phonograph records are stacked on a record changer (see Figure 1-2). The upper disk platter can be removed from the disk unit and replaced; it is therefore referred to as the Removable Disk Platter. Because it is contained in a sealed, cartridge-like case to protect it from damage when it is removed from the disk unit, the Removable Platter is also sometimes referred to as a disk cartridge. The lower disk platter is positioned about 1 1/2 inches below the Removable Platter on the shaft. It is an integral part of the disk unit which cannot be removed, and is therefore called the Fixed Disk Platter. The shaft itself is coupled to a drive motor which rotates both disk platters at a constant speed. The rotational speed of the Model 2230 is 1500 rpm; the speed of the Model 2260 is 2400 rpm.

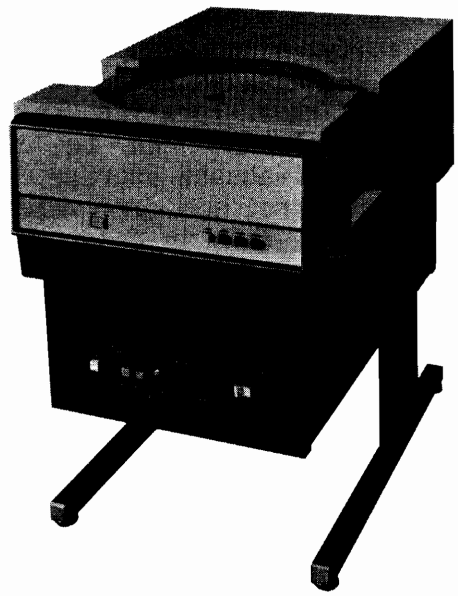


Figure 1-1. Models 2230/2260
Both Disk Drives Utilize the Same Physical Housing

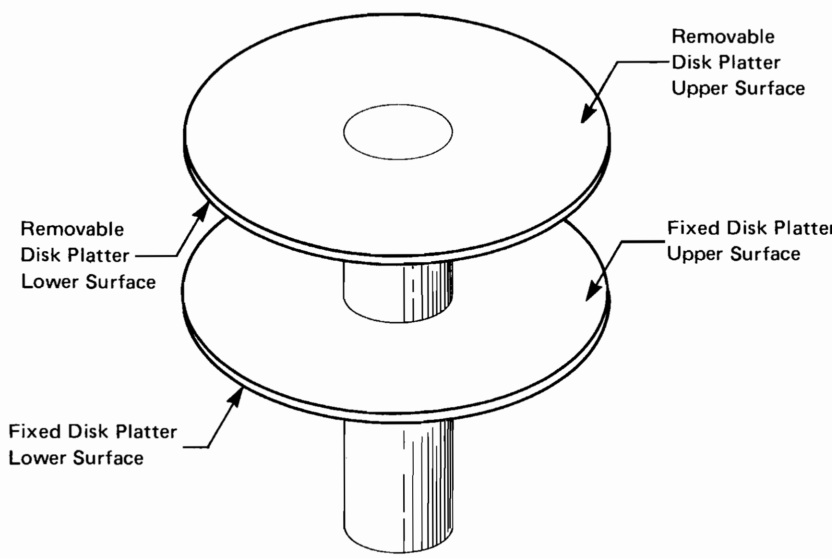


Figure 1-2. The Model 2230/2260 Disk Drive and Disk Platters

The disk unit also contains a comb-type access assembly consisting of four access arms (see Figure 1-3). Attached to the end of each access arm is a read/write head, which is fixed in position and cannot move independently of the access arm. Information is recorded on or read from the surface of a platter via the read/write head.

When a disk statement or command is executed from the System 2200 CPU, the access assembly moves the read/write heads in or out between the disk platters as they rotate. A read/write head can then record information on a surface of a disk platter (write) or retrieve information from a platter (read) as the platter rotates past the head's position.

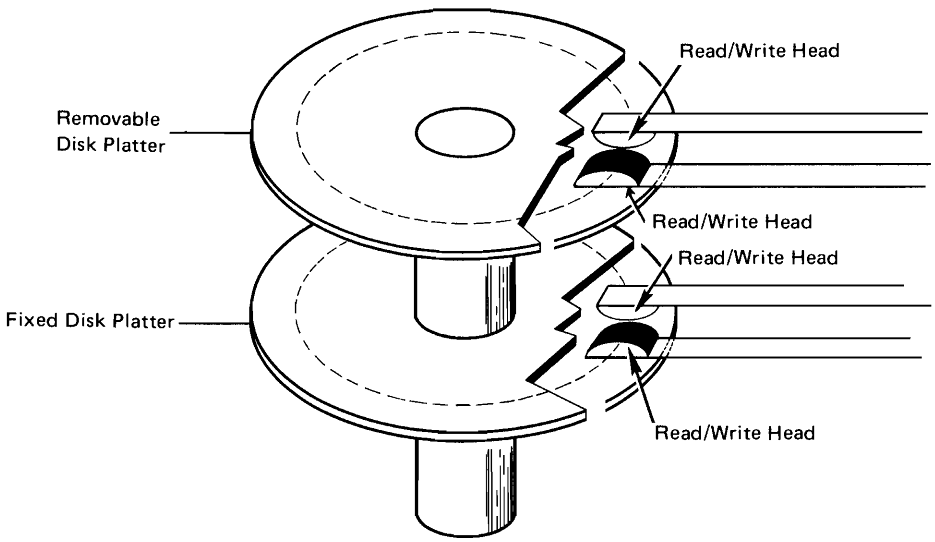


Figure 1-3. Model 2230/2260 Disk Unit with Access Assembly and Read/Write Heads

Model 2270

The Model 2270 Removable Diskette Drive is available in three configurations: the Model 2270-1 (not shown) holds a single diskette drive, the Model 2270-2 (Figure 1-4) holds a pair of diskette drives, and the Model 2270-3 (Figure 1-5) holds three diskette drives. Unlike the Fixed/Removable disks, in which both platters are driven by a single drive, the Model 2270 drives are mechanically independent of each other. Every 2270 diskette unit, irrespective of the Model, has three drive slots. In the single and dual drive configurations (2270-1 and 2270-2), the unused slot(s) are concealed by metal panels.

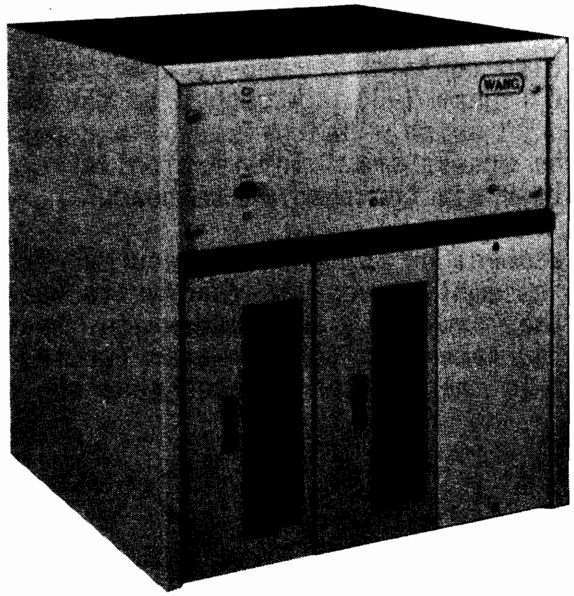


Figure 1-4. Model 2270-2 Dual Removable Diskette Drive
Note That Drive Slot #3, Not Used, is Covered by Metal Panel.

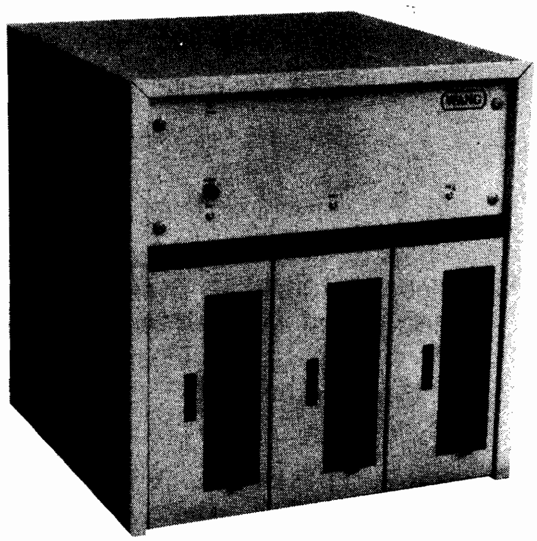


Figure 1-5. Model 2270-3 Triple Removable Diskette Drive

A single diskette is vertically mounted in each drive. Each drive contains a drive shaft and drive motor (which rotates the diskette at a constant speed of 360 rpm), as well as an access assembly with an attached read/write head. The read/write head is affixed to a carriage which moves it back and forth across the recording surface of the diskette in response to commands from the controlling System 2200. The access assembly carriage moves the head in or out over the surface of the spinning disk, until it is positioned at the proper location for reading or recording data. (Although the diskette is sealed in a protective plastic jacket to prevent the accumulation of dust on its recording surface, a window in the jacket permits the read/write head to make electrical contact with the disk surface.) See Figure 1-6.

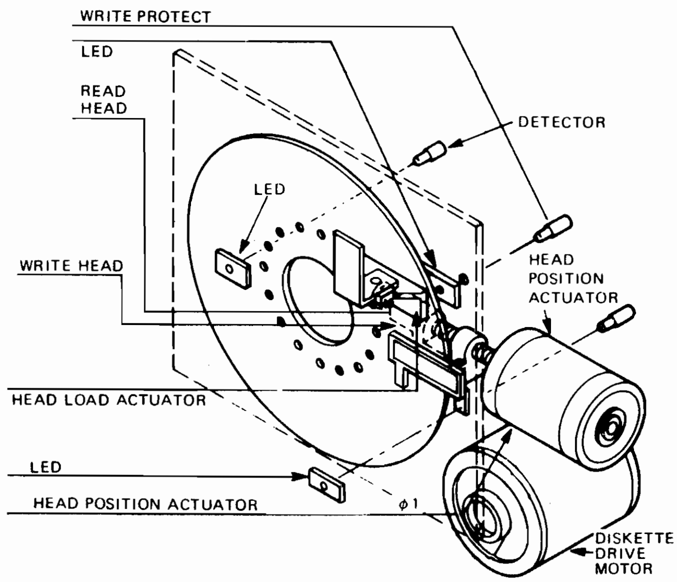


Figure 1-6. Model 2270 Diskette Mounted in a Diskette Drive. Access Assembly, Carriage, and Read/Write Head Are Shown.

1.5 THE DISK PLATTER

The storage medium of the disk unit is the magnetic disk platter, a thin, flat, circular plate coated on one or both sides with a magnetic material, usually iron oxide, and which, except that it has no apparent grooves, closely resembles a phonograph record. The disk platter utilized on the Models 2230 and 2260 is 15 inches in diameter, or about the size of a 33 1/3 rpm photograph record. It is coated with iron oxide on both sides, and both surfaces can be used to record data. The flexible platter, or "diskette," used in the Model 2270 is 7 1/2 inches in diameter, or about the size of a 45 rpm phonograph record. It is coated with magnetic material on only one side; therefore only one surface can be used to record data.

On each recording surface, the magnetic recording material is arranged in concentric circular magnetic tracks (see Figure 1-7). Information recorded on the disk is stored in the form of magnetized spots of iron oxide within a track, much the same way it is stored on magnetic tape (see Figure 1-8).

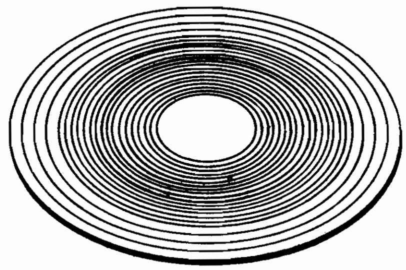


Figure 1-7. Concentric Tracks on a Disk Platter

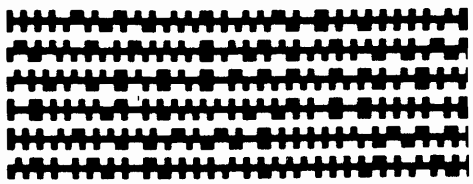


Figure 1-8. Enlarged View of Sections of Several Tracks Showing How Information Is Stored on a Disk Platter

1.6 SECTORS

In general, it is desirable to be able to store and retrieve information in units smaller than an entire track. For this reason, each track is divided into a number of discrete segments called "sectors." A sector is the smallest discrete unit of storage on the disk, with a fixed storage capacity of 256 bytes. Each sector is identified by a unique number called the "sector address," and can be directly accessed by the system. On a Model 2230 or 2260 disk platter, each track is divided into 24 sectors; on a Model 2270 diskette, each track contains 16 sectors.

In addition to the 256 bytes in each sector reserved for data storage, the sector contains several bytes of system control information, written into the sector at the time the platter is formatted. The system control information consists of a two-byte sector address, a two-byte cyclic redundancy check (CRC) total, and, on the 2230 and 2260 only, a one-byte longitudinal redundancy check (LRC) total. (See Figure 1-9.) The sector address is, of course, needed to enable the system to uniquely identify and access each sector. The CRC and LRC totals are the results of checksum tests performed by the system to monitor the integrity of data stored in the sector. (On the 2230 and 2260, the LRC total is stored in a special byte in each sector; on the Model 2270, the LRC totals are recalculated independently following each sector read or write, and are not stored on the disk.) All system control information is created and maintained solely by the disk controller, and is completely transparent and inaccessible to the user.

A different type of control information, called "sector control information" or "format control information," is automatically written by the system along with the user's data in the 256-byte data field of the sector. Because the format control information occupies several bytes of the 256-byte data field, the full 256 bytes are not available for data storage under normal conditions. The format control bytes are discussed in detail in Chapter 7, Section 7.6. A technique for writing data on disk without format control information is described in Chapter 9, Section 9.5. Note that creation and interpretation of the format control information also is an automatic function of the system, and is completely transparent to the user's software.

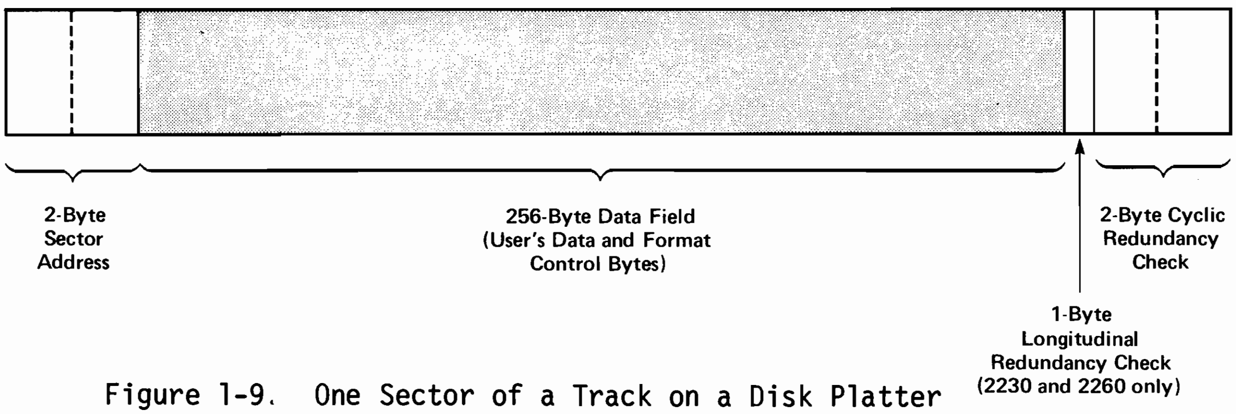


Figure 1-9. One Sector of a Track on a Disk Platter

1.7 TRACK AND SECTOR NUMBERING

Model 2230/2260

The upper and lower surfaces of each disk platter in the Models 2230 and 2260 are isomorphic, that is, they are identical to each other. For every track on the upper surface, there is a corresponding track on the lower surface, and vice versa. Track numbering is sequential on each platter, and alternates from one surface to another, starting with the outermost track on the lower surface (which is designated as track #0), and ending with the innermost track on the upper surface (which may be track #99, #199, #407, or #915, depending upon the disk model; see Figure 1-10). All odd-numbered tracks are located on the upper surface of a platter, and all even-numbered tracks are located on the bottom surface. The tracks are numbered independently on each disk platter.

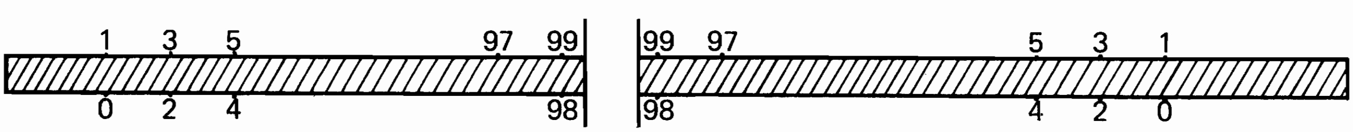


Figure 1-10. Cross-Section of a Disk Platter Showing Track Numbering on the Upper and Lower Surface (Model 2230-1)

Sectors on consecutive tracks are numbered sequentially, beginning at sector #0, which is located at the beginning of track #0, and proceeding from one consecutive track to the next. Sectors also are numbered independently on each platter.

Track #	Sector #
0	0 - 23
1	24 - 47
2	48 - 71
3	72 - 95
.	.
.	.
.	.
99	2376 - 2399

Figure 1-11. Sector Numbering on a Model 2230-1 Disk Platter

Model 2270

Tracks are numbered sequentially on a Model 2270 diskette. Each diskette contains 64 tracks, numbered 0 - 63. Because each platter has only one recording surface, the track numbering is consecutive on that surface, starting with the outermost track (track #0) and proceeding to the innermost track (track #63). A diskette contains a total of 1024 sectors, numbered sequentially from 0 to 1023. Sector #0 is located in track #0 and sector #1023 is located in track #63 (see Figure 1-12). Tracks and sectors are numbered independently on each diskette.

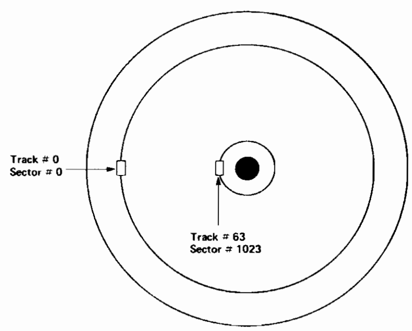


Figure 1-12. Track and Sector Numbering on a Model 2270 Diskette

1.8 DISK ACCESS TIME

Although it is useful, for purposes of general comprehension, to understand how the recording surface is divided up into concentric tracks, the system provides no way of addressing tracks directly. All direct addressing of data stored on the disk is done in terms of sector addresses. When presented with a sector address, the system automatically moves to the track which contains that sector. The exclusive use of sector addresses alleviates needless complexity for the programmer when addressing storage locations on the disk.

In order to retrieve a piece of information from the disk, the system must determine on which disk platter the information is stored and in which sector(s) on that platter the information is contained. Given the platter designation and sector address, the system signals the access assembly to move the read/write head out to the appropriate track, and access the desired sector.

There are, therefore, two distinct operations which must be carried out in order to access any particular sector on a disk platter:

1. The access assembly must move in or out to position the read/write head over the appropriate track on the specified platter. This operation is called the "track access."

2. Once positioned at the correct track, the read/write head must wait until the desired sector rotates beneath it as the platter revolves. This wait is known as the "disk latency period."

The time required to perform the first of these operations is called the track access time; the time required to perform the second operation is called the disk latency time. The duration of the track access time is determined by the speed of the access assembly, and the number of tracks which must be traversed by the access arm in order to locate the target track. The average track access time therefore increases somewhat with the size of the disk configuration. The disk latency time, on the other hand, is determined solely by the rotational speed of the disk unit. The time required for each operation must be included in the total time required to access a sector on a disk platter. (The latency time is normally not significant for sequential access operations; it may, however, be significant for random access operations.) Appendices B and C provide timing information on the Models 2230 and 2260; Appendix D provides timing information on the Model 2270.

1.9 STAGGERED ARRANGEMENT OF SECTORS ON A TRACK

Although sectors are numbered consecutively, starting at zero, on each disk platter, consecutive sectors in a track are not physically contiguous. Instead, consecutively numbered sectors are located several physical sectors apart within a track. On a Model 2230 platter, consecutive sectors are six physical sectors (one-quarter track) apart (Figure 1-13). On a Model 2270 diskette, consecutive sectors are four physical sectors (one-quarter track) apart (Figure 1-14). On the Model 2260, consecutive sectors are located 12 physical sectors (one-half track) apart (Figure 1-15). This "staggered" arrangement of consecutive sectors in a track makes it possible for the disk to access several consecutive sectors in a single revolution of the disk platter during certain multiple-sector read/write operations. In particular, the platter-to-platter MOVE and COPY operations are greatly accelerated by the capability to pick up multiple sectors in a single revolution. On the Models 2260 and 2270, the staggered arrangement of sectors also speeds up the reading and writing of long multi-sector records.

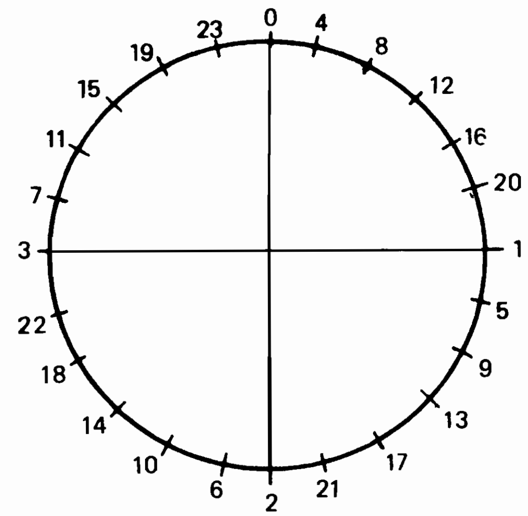


Figure 1-13. Staggered Arrangement of Sectors on Model 2230 Disk Platter

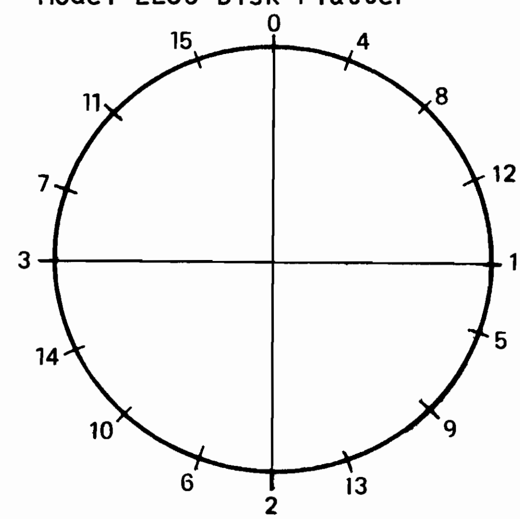


Figure 1-14. Staggered Arrangement of Sectors on Model 2270 Diskette

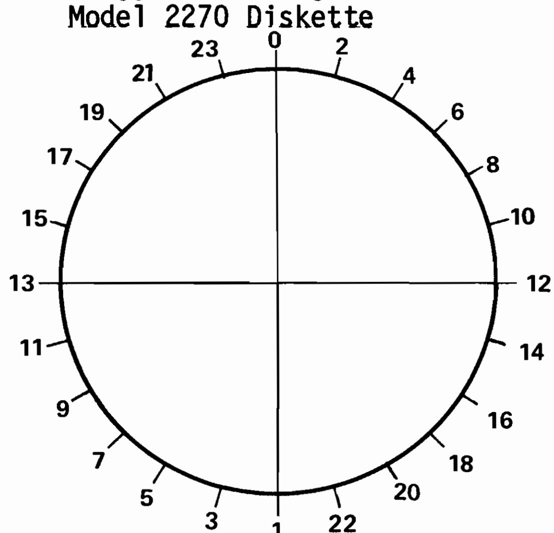


Figure 1-15. Staggered Arrangement of Sectors on Model 2260 Disk Platter

1.10 THE MODEL 2230/2260 CYLINDER CONCEPT

Because the Removable Disk Platter sits directly above the Fixed Disk Platter in the Models 2230 and 2260, tracks 0 and 1 of the Removable Disk Platter are directly above, or in the same vertical plane as, tracks 0 and 1 of the Fixed Disk Platter. If all four tracks in the same plane are connected by imaginary lines, a cylinder is formed (see Figure 1-16).

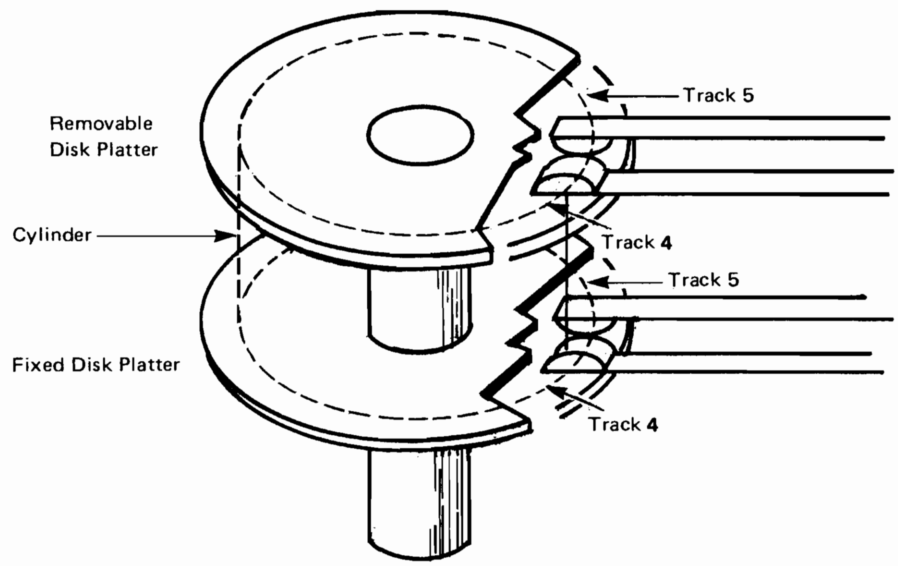


Figure 1-16. The Cylinder Concept (Model 2230/2260)

The access assembly of the Models 2230 and 2260 is designed to access one cylinder (four tracks) with each movement of the access arms. Since each track contains 24 sectors, four tracks contain 96 sectors. Thus a cylinder of information is the amount of information (256 x 96, or 24,576 bytes) which can be accessed with a single movement of the access assembly. Where large quantities of data are to be transferred to or from the disk, judicious use of cylinders can result in a significant savings in total track access time.

Chapter 2
Model 2230 Series (2230 and 2260) General Information and Formatting Instructions

2.1 UNPACKING AND INSPECTION

> NOTE:
>
> Because the disk unit is an extremely sensitive device, it is packed using special techniques to protect it from damage in shipping, and it should be unpacked only by qualified Wang Service Personnel.

2.2 INSTALLATION

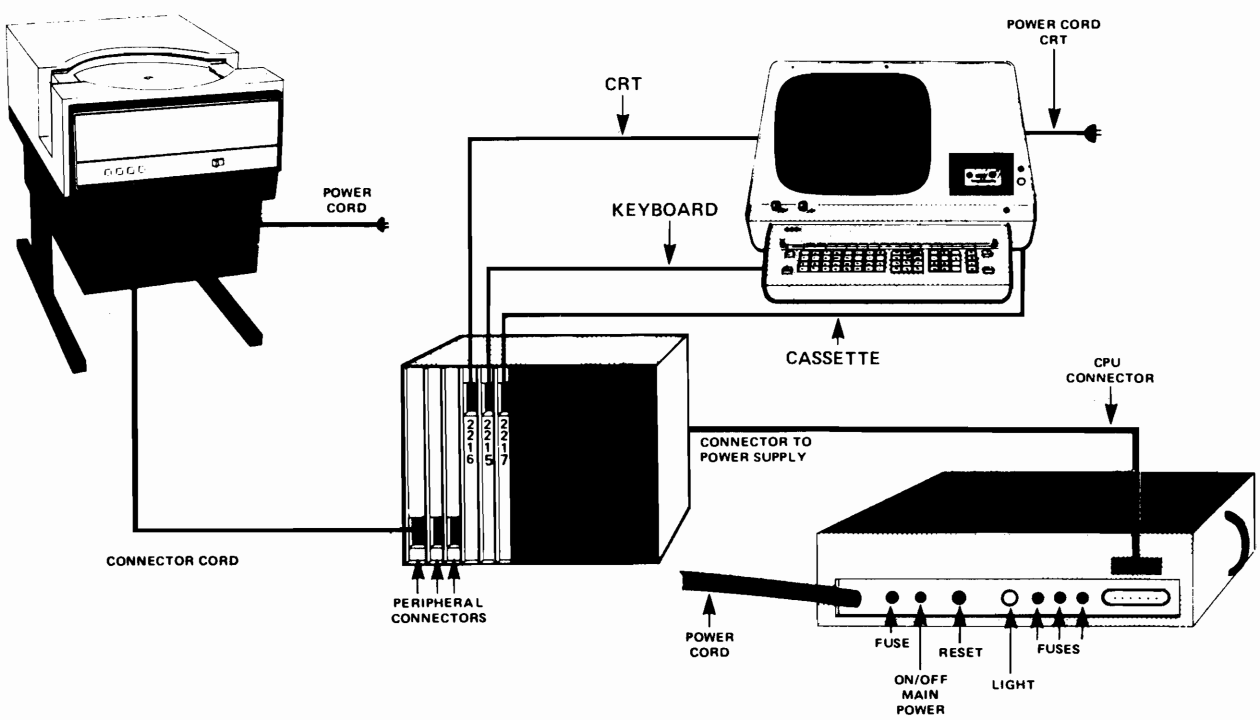


Figure 2-1. Models 2230/2260 System Configuration

The following installation procedure should be observed for the Disk Unit:

1. Plug the disk drive connector cord into the connector receptacle on the CPU chassis. The peripheral connector receptacle on the CPU is labelled for the disk drive. After attaching the cord, make sure that the lock clips are snapped into place at the CPU connection.

2. Plug the disk drive power cord into a wall outlet. Input power requirements for the disk are 115 VAC, 9 amps, 50/60 Hz ± 1 cycle (or 230 VAC, 5 amps, 50/60 Hz ± 1 cycle by special request).

3. Plug the main power cord of the CPU chassis into the Power Supply Unit; plug the Power Supply Unit power cord into a wall outlet.

2.3 POWER-ON PROCEDURES

1. Be sure that the disk LOAD/RUN switch is in the LOAD position. (To be safe, it's a good idea never to power on or off in the RUN mode.)

2. Switch ON the power switches on the CRT and other peripherals, including the disk.

3. Switch ON the main power switch on the Power Supply Unit (the light on the power supply unit glows on). This operation Master Initializes the system.

4. On the disk, the POWER and LOAD lights on the upper panel, as well as the POWER and ERROR lights on the lower panel, illuminate. The CRT display appears as illustrated below:

```
READY
:_
```

> NOTE:
>
> The appearance of an ERROR light at this time does not indicate a disk error; it is simply a reminder that the disk has not yet been formatted and therefore cannot yet be used. When the formatting procedure is begun, the ERROR light goes out automatically.

5. Switch the LOAD/RUN rocker switch (upper left-hand panel) to the RUN position. After about 60 seconds, the yellow READY light (upper right-hand panel) glows.

6. The disk is now ready to be formatted (see Section 2.4).

> **WARNING:**
>
> The Disk Unit should never be left in LOAD mode for a prolonged period of time. Serious and permanent damage to the disk unit may result if the LOAD/RUN switch is left in the LOAD position for more than 30 minutes while the power is ON.

2.4 MODEL 2230/2260 FORMATTING INSTRUCTIONS

Before either disk platter can be accessed and used, it must be formatted. Formatting involves assigning a unique address to each sector on the disk platter, and writing certain control information in each sector which helps the system keep a check on the integrity of information written to and read from the disk. Zeros are then written into the 256-byte data field in each sector. This entire procedure is carried out automatically by the disk controller, which also automatically performs a verification check to be sure that the format is valid.

The Fixed and Removable platters must be formatted separately. When a new Removable Disk Platter (disk cartridge) is initially loaded into the disk unit, the new cartridge must be formatted. Once formatted, a disk platter need never be reformatted unless frequent random read/write errors are encountered.

Formatting the Removable Disk Platter (Disk Cartridge)

1. The disk should now be in RUN mode. If it is not, switch to RUN mode and wait for the READY light to come on (it should take about 1 minute).

2. When the READY light is on:

   a. Set the REMOVABLE DISK/FIXED DISK toggle switch to REMOVABLE DISK.

   b. Turn the FORMAT LOCK key to the ON position.

   c. On the system keyboard, key RESET. If the disk ERROR light has been on, it should now go out.

   d. On the disk front panel, press the FORMAT button. The FORMAT light, located next to the button, illuminates, indicating that the Removable Platter is now being formatted. The formatting procedure takes about 50 seconds. When the FORMAT light goes out, formatting has been completed.

   e. Turn the FORMAT LOCK key to the OFF position. (Note: the disk cannot be accessed while the FORMAT LOCK key is ON.)

Formatting the Fixed Disk Platter

1. If the disk is in LOAD mode, switch to RUN and wait about 1 minute for the READY light to come on.

2. When the READY light comes on, flip the REMOVABLE DISK/FIXED DISK toggle switch to FIXED DISK and follow the procedure described above for the Removable Platter (steps 2b - 2e above).

> NOTE:
>
> After a disk platter is formatted, the FORMAT LOCK key must be turned OFF. Otherwise, the disk cannot be accessed. The formatting procedure causes any data previously stored on the disk platter to be erased.

2.5 FORMAT ERRORS

After formatting a disk platter, the system automatically checks the format for validity. If a format error is encountered, the system rechecks the format three more times. If the error persists following the fourth check, the platter is automatically reformatted and rechecked four more times. This entire procedure (reformat and four format checks) is repeated eight times by the system. If the error stubbornly persists following the eighth attempt, the ERROR light on the lower left-hand panel of the disk is illuminated, indicating that the formatting procedure has aborted. Depending upon where the erroneous sector is located, this entire procedure may take from one to eight minutes, during which time the FORMAT lamp remains lit. Ordinarily, if the FORMAT lamp remains on for more than a minute during the formatting procedure, you should assume that a format error has been discovered and the system is attempting to correct it. If the platter cannot be formatted (i.e., if the ERROR light is illuminated), you have two courses of action:

1. If the Removable Disk Platter cannot be formatted, insert another disk cartridge and repeat the formatting procedure. If the second cartridge cannot be formatted, call your Wang Service Representative.

2. If the Fixed Disk Platter cannot be formatted, call your Wang Service Representative.

If a particular disk platter is at fault, it may be possible to format and use the other disk platter. If the system itself is at fault, neither disk platter can be used. You should never attempt to utilize a disk platter which is not properly formatted.

2.6 LOADING AND UNLOADING THE REMOVABLE DISK PLATTER (DISK CARTRIDGE)

The disk unit contains two disk platters, one Fixed and one Removable. The Fixed Disk Platter comes installed from the factory and cannot be removed from the disk unit. The Removable Disk Platter, as its name indicates, can be taken out and replaced by the operator. Because it is encased in a protective cartridge-like case, the Removable Disk Platter is also referred to as a disk cartridge. The following instructions are for changing the Removable Disk Platter.

Unloading the Disk Cartridge

Step 1 Switch the LOAD/RUN rocker switch on the left front panel of the disk to the LOAD position (POWER switch must be ON). Wait for the LOAD light (white) to come on (about 30 seconds). See Figure 2-2.

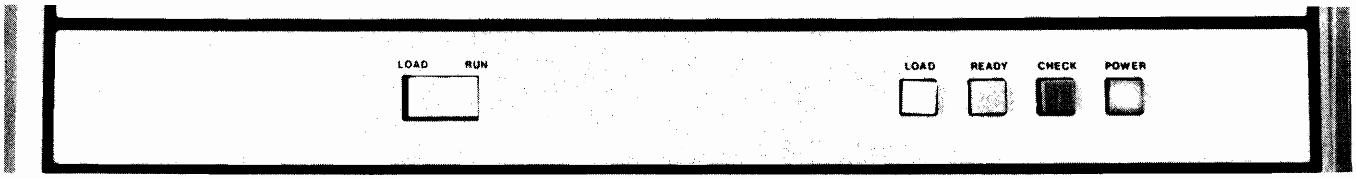


Figure 2-2. Switches on Front Panel of Model 2230/2260

If the LOAD/RUN switch is set to RUN, the disk cartridge cannot be unloaded.

WARNING:
The Disk Drive should never be left in LOAD mode for a prolonged period of time. Serious and permanent damage to the disk unit may result if the LOAD/RUN switch is left in the LOAD position for more than 30 minutes while power is ON.

Step 2 Open the cartridge bowl by pulling back the two cartridge holding clamps (Figure 2-3). If these clamps are locked in the closed position, do not force them. They are locked due to an interlock (see Section 2.8, "Machine Interlocks").

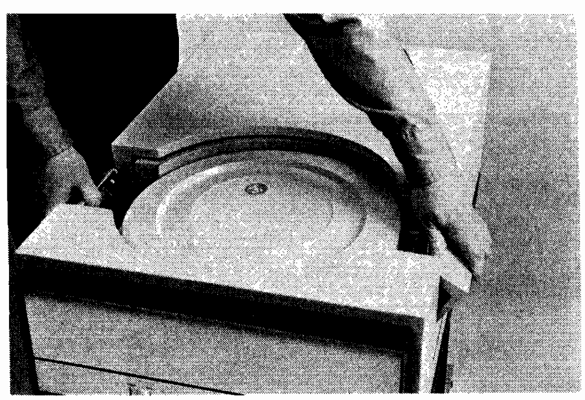


Figure 2-3. Opening the Cartridge Holding Clamps

(To load a disk cartridge, go to Step 7.)

Step 3 Remove the dust cover (Figure 2-4).

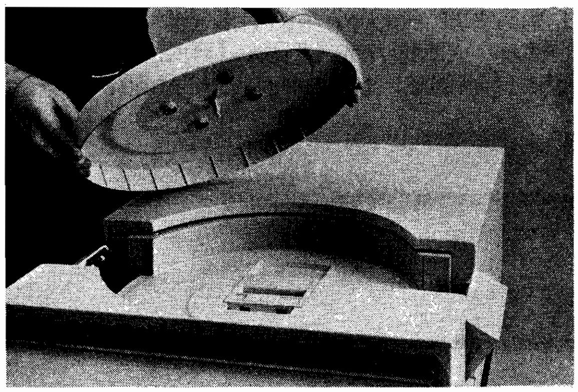


Figure 2-4. Removing the Dust Cover

Step 4 Slide the tab on the cartridge handle to the left, and hold the tab in place while raising the cartridge handle (Figure 2-5). This action separates the disk cartridge from the disk drive, and the cartridge may be lifted out of the disk bowl (Figure 2-6).

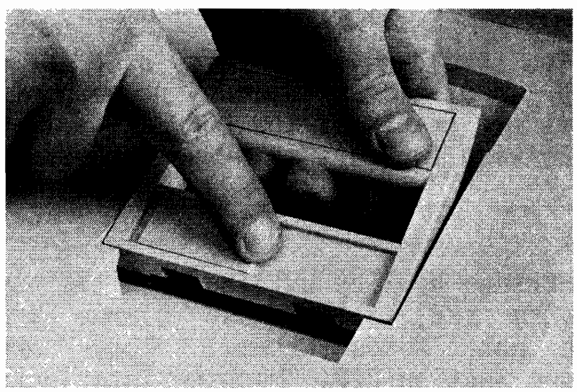


Figure 2-5. Releasing the Cartridge Lock

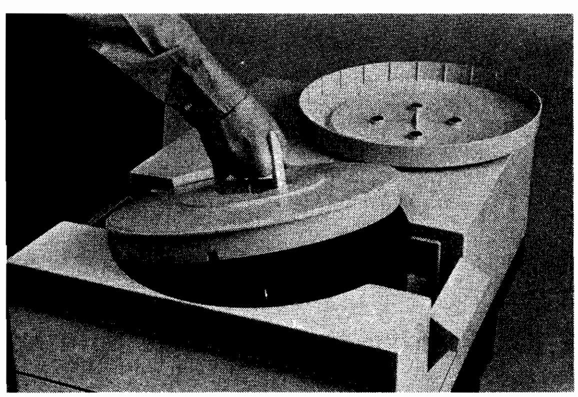


Figure 2-6. Removing the Cartridge from the Cartridge Bowl

Step 5 Turn the dust cover over and set the disk cartridge into it (Figure 2-7). When the cartridge handle is lowered, it locks the dust cover onto the disk cartridge. Both cover and cartridge can be carried as a unit by lifting the handle again without touching the tab (see Figure 2-8).

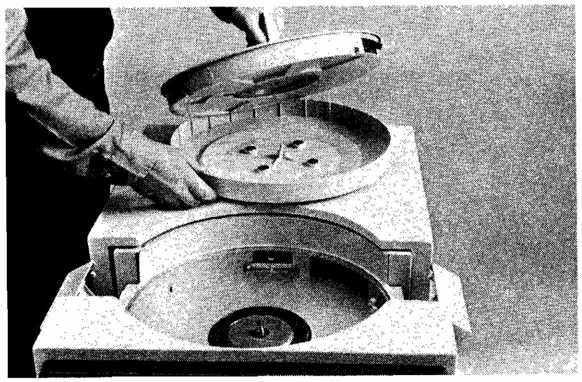


Figure 2-7. Locking the Cartridge onto the Dust Cover

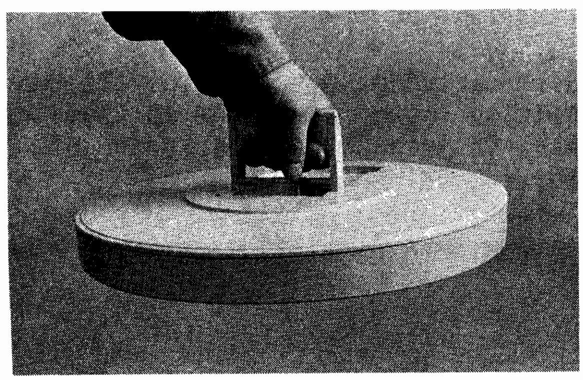


Figure 2-8. Carrying a Sealed Cartridge

Step 6 After a disk cartridge is removed from the disk unit, a replacement cartridge should be loaded into its place immediately. See "Loading A New Disk Cartridge", Step 7.

Loading a New Disk Cartridge

Step 7 Make certain that the LOAD/RUN switch is set to LOAD. A disk cartridge cannot be loaded if the disk is in the RUN mode.

Step 8 Open the cartridge holding clamps on the disk bowl (see Figure 2-3). If these clamps are locked in the closed position, do not force them; they are locked due to an interlock (see Section 2.8, "Machine Interlocks").

Step 9 Remove the new disk cartridge from its dust cover by sliding the tab on the cartridge handle to the left and holding it in position while lifting the handle (Figure 2-5). This action separates the disk cartridge from the dust cover, enabling you to lift the cartridge away from the cover (Figure 2-9).

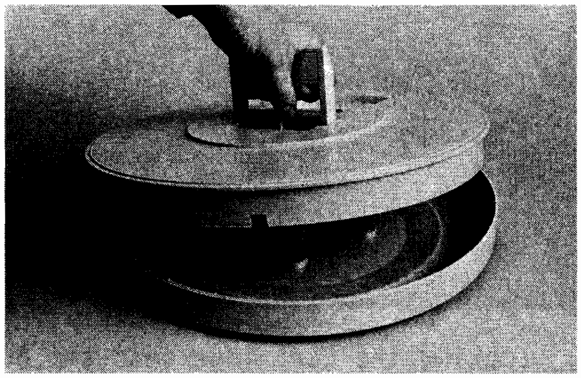


Figure 2-9. Separating Cartridge from Dust Cover

Step 10 Place the disk cartridge over the spindle hub in the cartridge bowl. Position the cartridge so that the cartridge opening for the head entry is located at the rear of the cartridge bowl (Figure 2-10).

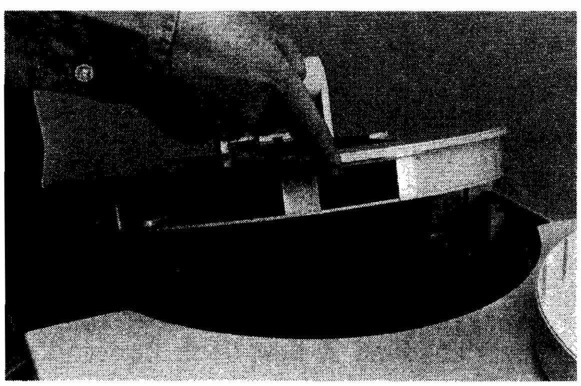


Figure 2-10. Aligning Cartridge in the Cartridge Bowl

When the cartridge is correctly located, it sets squarely in position and does not wobble or rotate. Lower the cartridge handle to lock the cartridge onto the spindle (Figure 2-11).

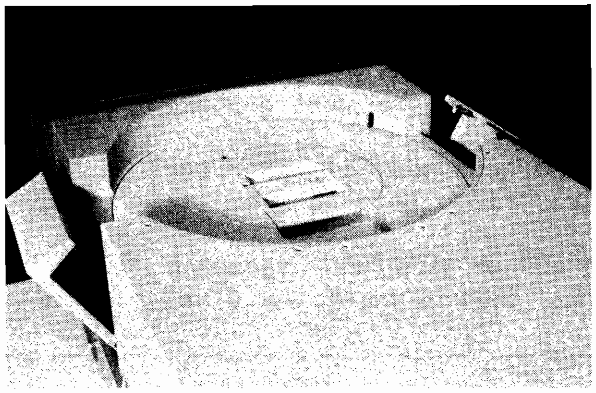


Figure 2-11. Cartridge Loaded in Disk Unit

Step 11 Place the dust cover, open end down, over the disk cartridge, and close the two holding clamps (Figure 2-12).

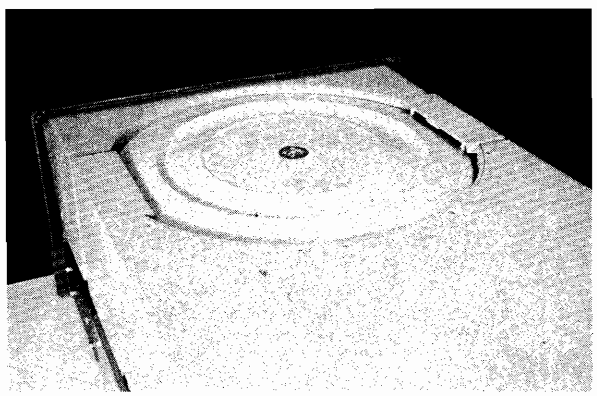


Figure 2-12. Cartridge Sealed and Locked in Disk Unit

2.7 HANDLING AND STORAGE OF THE REMOVABLE DISK PLATTER (DISK CARTRIDGE)

The following practices should be observed when handling and storing a disk cartridge:

1. The cartridge dust cover should be on the cartridge while it is out of the disk unit to insure a positive dust seal and to immobolize the platter.

2. Cartridges can be stored flat or on edge. Cartridges may be stacked on top of one another, but heavy top loading should be avoided.

Refer to Appendix E "Model 2230/2260 Disk Cartridge Maintenance Information", for a more complete discussion of the handling and storage of disk platters.

2.8 MACHINE INTERLOCKS

The cartridge holding clamps cannot be operated while the read/write heads or disk cleaning brushes are positioned over the disk surfaces (i.e., while the disk is in RUN mode), or when equipment power is OFF.

The disk cannot be accessed if the cartridge dust cover is not installed, or if the cartridge holding clamps are open.

An initial power-ON or LOAD/RUN sequence turns off the interlock circuits to allow normal operation of the disk.

> NOTE:
>
> To ensure continuous proper disk operation, the air filter in the disk unit must be checked by a Wang Service Representative every three months. If your disk is under service contract, maintenance checks will be made automatically on a regular basis. If your disk is not under service contract, it is your responsibility to contact a Wang Service Representative to perform a maintenance check at three-month intervals. Disk failures which result from lack of proper preventive maintenance may not be covered under the warranty.

Chapter 3
Model 2270 General Information and Formatting Instructions

3.1 UNPACKING AND INSPECTION

Because the disk unit is an extremely sensitive device, it is packed using special techniques to protect it from damage in shipping, and it should be unpacked and installed only by qualified Wang service personnel.

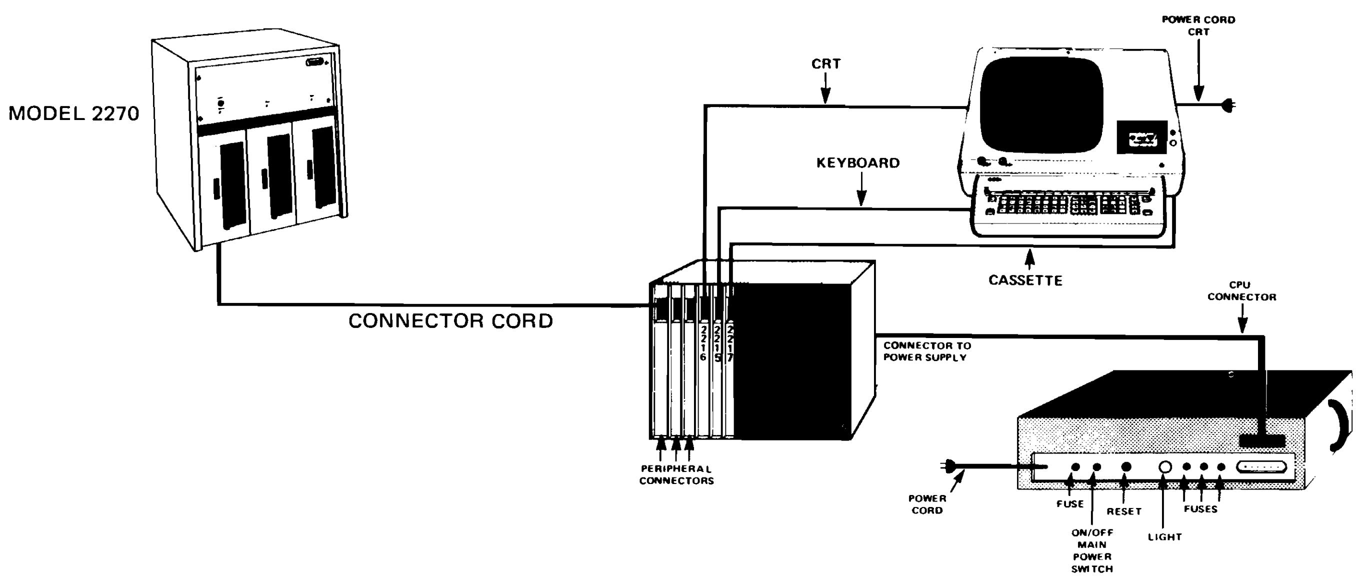


Figure 3-1. Model 2270 System Configuration

3.2 INSTALLATION

The following installation procedure should be observed for the disk unit:

1. Plug the disk drive connector cord into the appropriately labelled disk controller board on the CPU chassis. After attaching the cord, make sure the lock clips are snapped into place at the CPU connection.

2. Plug the disk drive power cord into a grounded (three-hole) wall outlet. Input power requirements for the disk are 115 VAC, 9 amps, 50/60 Hz ± 1 cycle (or 120 VAC, 5 amps, 50/60 Hz ± 1 cycle by special request).

3. Plug the Power Supply Unit power cords and the electrical power cords of all other peripherals into grounded wall sockets.

3.3 POWER-ON PROCEDURE

1. Switch ON the power switches on the disk and all other peripherals. On the disk unit, the POWER lamp illuminates.

2. Switch ON the main power switch on the Power Supply Unit.

3. You can now load and format the diskettes (Sections 3.4, 3.5, 3.6).

3.4 LOADING AND FORMATTING DISKETTES

The procedure for loading and formatting diskettes is the same for all versions of the 2270 (2270-1, -2, and -3). Although formatting must be carried out initially in Drive #1, a formatted diskette may be accessed interchangeably in any drive, and in any 2270 disk unit.

A new, unused diskette must be formatted initially before it can be used to record data. The formatting procedure is a hardware function entirely, and can be initiated by the operator at the touch of a button. Once formatted, a diskette normally should never need to be reformatted. Certain problems which may result from a bad format, such as random read/write errors, can, in some cases, be corrected by reformatting the diskette. It is important to note, however, that the formatting procedure wipes out any data already stored on the diskette.

Each sector on the diskette really consists of 260 bytes (see Chapter 1). Of these, two bytes contain the sector address, and two bytes contain control information required to perform a Cyclic Redundancy Check (CRC). The remaining 256 bytes are available for the user's data. During the formatting procedure, the system writes the sector address and CRC control information in each sector. The remaining 256 bytes of each sector are filled with zeroes.

To load and format a diskette, observe the following steps:

1. Remove a diskette from its envelope, and check to see that the Write Protect hole in the diskette jacket is covered with a tab (Figure 3-2). If the Write Protect hole is uncovered, the diskette cannot be formatted. Refer to Section 3.6 for a discussion of the Write Protect feature.

Figure 3-2. Model 2270 Diskette, with Tab Covering the Write Protect Hole

2. Open the door of Drive #1 by pressing against the door latch, located immediately to the left of the door (Figure 3-3). The door should slide open.

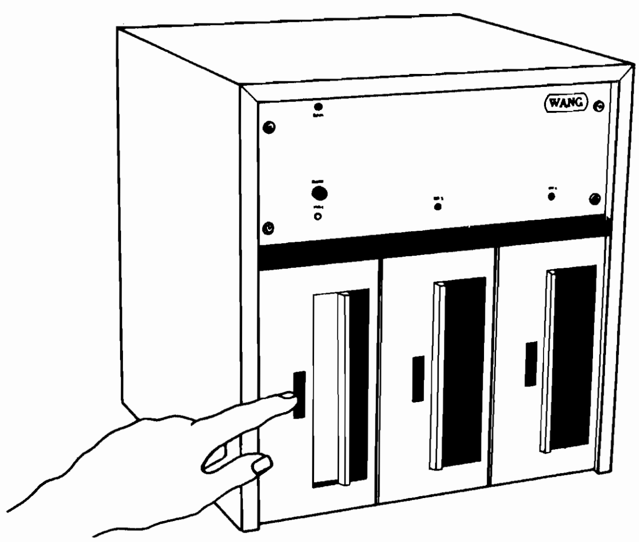


Figure 3-3. Opening the Door of Drive #1

3. Insert a diskette into the open drive slot (Figure 3-4). Push the diskette into the drive slot far enough to catch and hold it in the slot. The plastic jacket in which the diskette platter is sealed is labelled with arrows indicating the proper orientation for mounting. Before mounting, be sure that the Mylar recording disk moves freely with its jacket. Test for freedom of movement by pushing gently against the inner edge of the disk; it should move and turn easily.

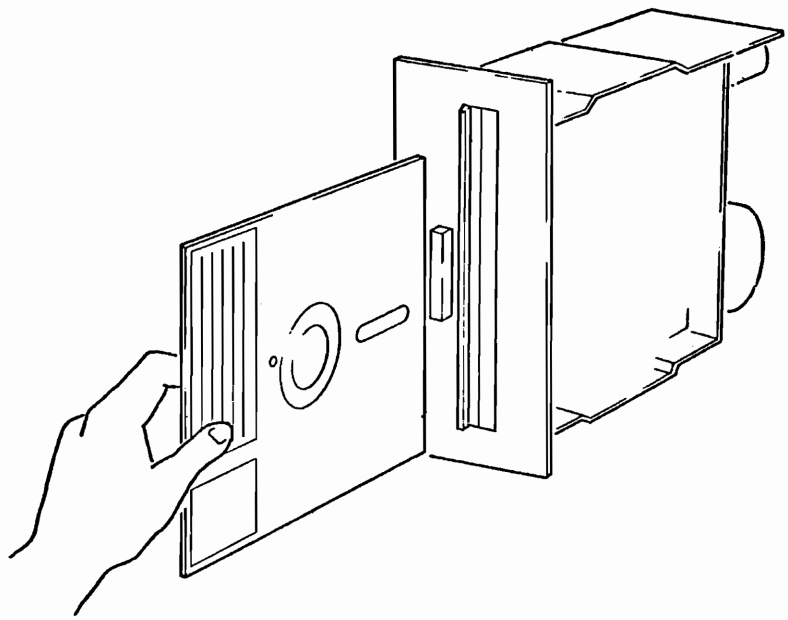


Figure 3-4. Mounting a Diskette

4. Close the drive door by sliding it to the left until it locks into place. Be sure the door is tightly closed.

5. To format the diskette, first key RESET on the keyboard. Next, use a pen or pencil to depress the FORMAT button above Drive #1 on the disk control panel. (The FORMAT button is surrounded by a protective ring to prevent accidental activation of the formatting procedure, a safety feature necessary because the formatting operation automatically erases any data stored on the diskette.) The button must be held in for about one-tenth of a second, or until the format lamp above Drive #1 illuminates.

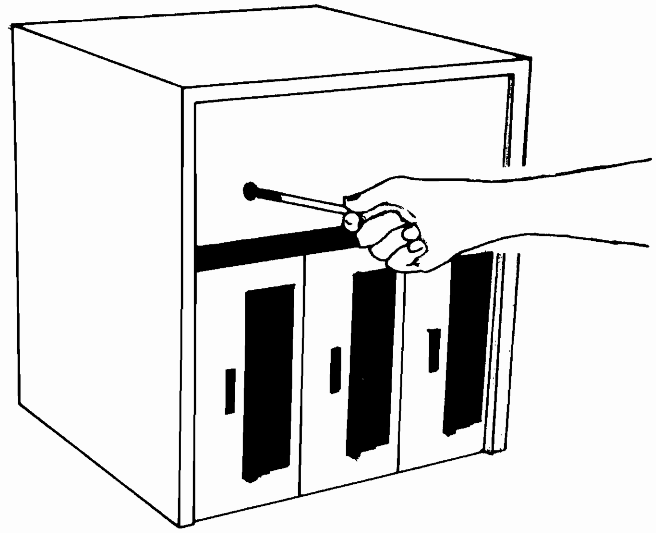


Figure 3-5. Formatting a Diskette

6. The format lamp remains illuminated throughout the formatting operation, which normally requires between 45 and 50 seconds. At the end of this time, the lamp should extinguish, indicating that the formatting process is complete. If the lamp remains lit for longer than 50 seconds, it is an indication that the system has experienced some difficulty with the format, and is attempting to reformat. If for some reason the diskette cannot be properly formatted, the format lamp will start to blink (refer to Section 3.5 on Format Errors).

7. To remove the diskette from the drive, depress the door latch. The drive door is automatically slid open, and a spring-loaded release mechanism ejects the diskette about halfway out of the drive slot. Once formatted, the diskette may be loaded into any drive for recording programs and data.

3.5 FORMAT ERRORS

Immediately after formatting a diskette, the system automatically checks the format to ensure that it is correct. If an error is detected in the format, the diskette is automatically reformatted and rechecked three more times. During this retry process, which may take between four and five minutes, the format lamp above Drive #1 remains lit. Should the error stubbornly persist following the fourth format check, the system signals a format error by causing the format light to blink rapidly on and off. At this point, the disk controller can be reinitialized, and the format light extinguished, by touching RESET on the system keyboard.

A diskette which cannot be properly formatted should not be used for data storage. In most cases, format errors result from one or two simple causes:

1. Drive door not tightly closed. Make sure that the door is closed tightly, and repeat the formatting procedure.

2. Write Protect hole not covered. Remove the diskette and check the Write Protect hole to be sure that it is completely covered. If it is uncovered, the diskette cannot be formatted.

3. Faulty diskette. Insert a new diskette, and repeat the formatting procedure.

If the formatting operation repeatedly aborts with an error on several different diskettes, contact your Wang Service Representative.

3.6 THE WRITE PROTECT FEATURE

Important programs and data recorded on a diskette should be protected against accidental erasure through overwriting or formatting. The Write Protect feature is provided for this purpose.

A small hole punched near the edge of the diskette's plastic jacket controls the Write Protect mechanism. When this hole is uncovered, the diskette is write-protected. No information can be recorded on a protected diskette, nor can it be formatted. Information already stored on the diskette can, however, be read in the normal fashion. Any attempt to write on a protected diskette elicits an ERROR 71. Any attempt to format a protected diskette causes the lamp above Drive #1 to blink frantically.

The Write Protect feature is inhibited by covering both openings of the Write Protect hole with a folded tab (tabs are provided with each diskette for this purpose). The diskette may then be protected again at any time simply by removing the tab.

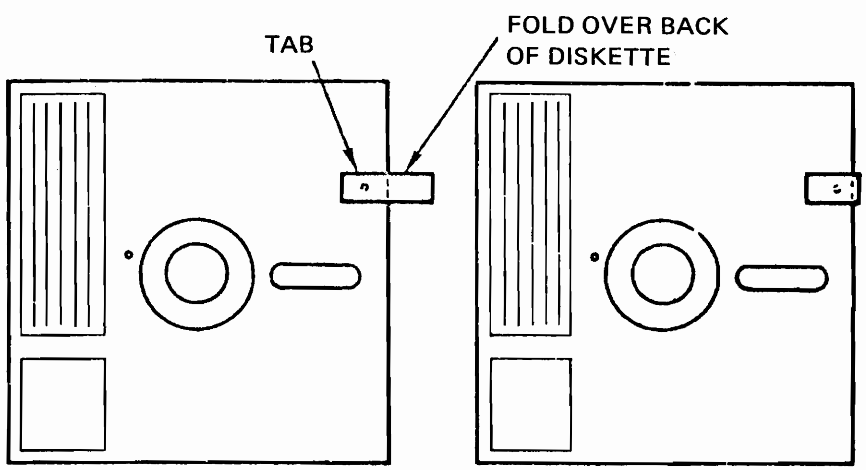


Figure 3-6. Model 2270 Diskette Write-Protect Feature

3.7 HANDLING AND STORAGE OF DISKETTES

Diskettes are light and compact, and may be easily stored in filing cabinets, on shelves, in boxes, etc., either lying flat or standing on edge. A diskette should always be stored in its disk envelope to inhibit the accumulation of dust on the recording surface.

The following suggestions apply to the handling and storage of diskettes:

1. Replace storage envelopes when they become worn, cracked, or distorted. Envelopes are designed to protect the disk.

2. Keep disks away from magnetic fields and from ferromagnetic materials which might become magnetized. Strong magnetic fields (greater than about 50 oersteds) may distort recorded data on a disk.

3. Do not write on the plastic jacket with a ballpoint pen or lead pencil. Use a felt-tip pen.

4. Do not expose a diskette to high temperature or humidity, or to direct sunlight, for prolonged periods. Temperature range for storage of diskettes is 50°F to 125°F (10°C - 52°C), and range of relative humidity is 8% - 80%.

5. Do not smoke when handling diskettes. Heat and contamination from a carelessly dropped ash can damage the diskette.

6. Do not touch or attempt to clean the diskette recording surface. Abrasion to the surface may cause a loss of stored data.

Chapter 4
Accessing the Disk Platters (Models 2230, 2260, and 2270)

4.1 THE 'F' AND 'R' PARAMETERS

The method of accessing individual disk platters is the same for all disk models. In every case, the system uses two parameters - the 'F' parameter and the 'R' parameter - to uniquely identify individual platters. For the fixed/removable disk units (Models 2230/2260), the 'F' parameter uniquely identifies the Fixed Platter, while the 'R' parameter identifies the Removable Platter. For the diskette units, however, no platter has a privileged status (since any platter may be loaded into any drive); in these cases, therefore, the 'F' and 'R' parameters really identify separate drives. The 'F' parameter accesses the platter loaded in Drive #1, while the 'R' parameter accesses Drive #2. In the case of the Model 2270-1, which contains only a single drive, the 'F' parameter alone is used. In the case of the Model 2270-3, which contains three drives, the 'F' parameter identifies both Drive #1 and Drive #3. A special device address must be used in conjunction with the 'F' parameter in order to access Drive #3, however (refer to Section 4.3).

Table 4-1. Disk Platters and the 'F' and 'R' Parameters

PARAMETER	MODELS 2230/2260	MODEL 2270-1	MODEL 2270-2	MODEL 2270-3
F	Fixed Disk Platter	Drive #1	Drive #1	Drive #1 or #3*
R	Removable Disk Platter	Not Legal	Drive #2	Drive #2

*Note - 'F' parameter can be used to access Drive #3 in a Model 2270-3 only when accompanied by a special device address (see Section 4.3).

The 'F' or 'R' parameter must be included in a disk statement or command to reference a particular disk platter. For example, the statement

```
10 LOAD DC F "PROG 1"
```

loads the BASIC program named "PROG 1" from the Fixed Disk Platter of a Model 2230 or 2260, or from the platter currently mounted in Drive #1 of a Model 2270. Alternatively, the same program might be loaded from the Removable Platter of a 2230 or 2260, or from a platter in Drive #2 of a 2270, with the following statement:

```
20 LOAD DC R "PROG 1"
```

The above statement is, of course, illegal for a Model 2270-1, which contains only a single drive, always identified with the 'F' parameter.

4.2 THE DISK DEVICE ADDRESS

Apart from the 'F' and 'R' parameters, which identify individual disk platters within a disk unit, the disk unit itself is identified with a unique three-digit device address. The disk device address enables the system to distinguish a disk from other peripheral devices (such as tape cassette drives, output writers, card readers, etc.), as well as from other disk units on the same system. The device address of the first or primary disk unit in a system is 310. If the system includes two or more disk units, the address is incremented by HEX(10) for each successive unit. (For example, the address of the second disk unit in a system is 320.) Section 6.8 discusses the addressing scheme for multiple disk units.

The disk device address can be included directly in certain disk statements and commands, for example:

```
10 LOAD DC F /310, "PROG 1"
```

Notice that the address is preceded by a "/" when it is directly specified in a disk statement. There are a number of disk statements in which it is not possible to specify a disk address directly. In these cases, the address must be referenced indirectly. The technique for indirectly referencing a disk address is discussed in Chapter 6. It is generally unnecessary to specify the device address either directly or indirectly in a statement, however, because the system automatically uses address 310 if no other address is specified. Address 310 is therefore known as the "default disk address". The only case in which it is necessary to specify an address in a disk statement or command is when an address other than 310 is to be used (e.g., when a second or third, etc. disk unit is being addressed, or when platter #3 in the Model 2270-3 is being accessed, or when the disk is accessed via the Disk Multiplexer). Thus, a statement such as

```
20 LOAD DC F "PROG 2"
```

accesses the disk just as well as statement 10 above.

4.3 ADDRESSING THE THIRD DISKETTE DRIVE (MODEL 2270-3)

The system recognizes only two disk platters, designated by 'F' and 'R', in each disk unit. Thus, drive #3 in a Model 2270-3 Diskette Drive unit is regarded by the system as a separate disk unit, and is assigned a disk device address different from that of the primary address assigned to drives #1 and #2. In general, the address of drive #3 is determined by adding HEX(40) to the primary address assigned to drives #1 and #2. For example, if the primary address is 310, the address of drive #3 is 350; if the primary address is 320, the address of drive #3 is 360, etc. Procedures for accessing the third diskette drive, and the complete addressing scheme for the Model 2270-3, are covered in Chapter 6, Sections 6.2 and 6.8.

Note that the third drive is not assigned a special third parameter, different from 'F' and 'R'. Rather, it is regarded as the 'F' platter of a separate disk unit, and must be accessed by specifying both the 'F' parameter and the special disk address. Attempted use of the 'R' parameter with the special address to access drive #3 will produce an error.

4.4 LIMITATIONS OF CERTAIN DISK MODELS

In general, the discussion of disk statements and commands which follows in Chapters 5 - 9 applies across the board to all disk models, since all 2200 series disks share the same BASIC instruction set. There are, however, two special exceptions to this general rule, regarding the use of the platter-to-platter MOVE and COPY statements (covered in Sections 5.15 and 9.6, respectively). These exceptions apply to the Models 2270-1 and 2270-3:

Model 2270-1

The platter-to-platter MOVE and COPY statements are illegal on the Model 2270-1, for the obvious reason that it holds only a single diskette. The MOVE and COPY operations cannot be carried out between separate disk units (i.e., between one Model 2270-1 and a second disk unit).

Model 2270-3

The platter-to-platter MOVE and COPY operations are legal for drives #1 and #2 in the 2270-3, but illegal for drive #3. Drive #3 is regarded as belonging to a separate disk unit, and the MOVE and COPY operations cannot be carried out between two disk units.

Chapter 5
Automatic File Cataloging Procedures

5.1 INTRODUCTION

Once the disk is unpacked, inspected, installed, turned on, and formatted, you are ready to begin storing information on it. The System 2200 provides two methods of accessing and utilizing the disk, Automatic File Cataloging Mode and Absolute Sector Addressing Mode. Automatic File Cataloging consists of a set of catalog procedures designed to facilitate creating and maintaining files on the disk without concern for where the files are actually located. Absolute Sector Addressing, on the other hand, permits direct access to any sector on the disk; Absolute Sector Addressing statements can be used to design a custom disk operating system, or to write special disk operating procedures such as binary searches, sorting routines, etc.

Chapters 5, 6, and 7 describe and explain the functions and uses of the catalog procedures. The present chapter introduces the concept of cataloging, and discusses the most basic catalog procedures, such as storing and retrieving programs and data on disk, skipping over data records in a data file, listing the contents of the catalog index for each platter, and creating back-up copies of the catalog. Chapters 6 and 7 discuss these and other subjects in greater detail. Chapter 8 provides an alphabetical listing of all catalog statements and commands with a detailed summary of the general format and function of each.

5.2 WHAT IS AUTOMATIC FILE CATALOGING?

Automatic File Cataloging comprises a built-in set of catalog procedures which automatically keep track of the locations of all cataloged files stored on a disk platter. The catalog procedures enable a programmer to create and access program files and data files on disk by name, without knowing where the files are located on the platter. The system itself automatically places each newly created file in an available location, and records this location for future reference.

The catalog procedures provided in Automatic File Cataloging actually consist of 18 BASIC statements which control the storage and retrieval of information on the disk along with a number of auxiliary file maintenance operations. Prior to opening any cataloged files on a disk platter, it is necessary to establish a catalog on the platter. The catalog consists of two parts, the Catalog Index, and the Catalog Area.

All cataloged files (program and data) are stored sequentially in the portion of the platter designated as the Catalog Area. The Catalog Index, which normally occupies a much smaller portion of the platter than the Catalog Area, contains the name and location of each cataloged file. When a file is initially opened, the system automatically stores it in the first available sequential location in the Catalog Area. The system then records the file's name and location in the Catalog Index. Thus, the Catalog Index functions much like the table of contents in a textbook, while the Catalog Area is analogous to the body of text. When the system is subsequently instructed to access a cataloged file, it goes to the Catalog Index, looks up the file's name and location, and moves to the appropriate location in the Catalog Area to access the file. Because the Catalog Index is automatically maintained and consulted by the system itself in Automatic File Cataloging mode, the programmer never needs to know where a cataloged file is actually located on the disk in order to access it.

5.3 SECTORS AND ADDRESSES

The Catalog Index keeps track of the location of each file in the Catalog Area by recording the starting sector address of the file at the time it is stored. You may recall from Chapter 1 that the sectors on each disk platter are numbered sequentially, starting at zero. Each sector has a unique number, or "address".

Each sector has a storage capacity of 256 bytes of information. Thus, for example, a 1000-byte program would occupy four consecutive sectors. Following our analogy above, the sectors can be thought of as pages in a textbook, each with its own number. When a program or data file is stored on the disk with the cataloging procedures, the system automatically records the name of the file and its starting sector address - i.e., the address of the first sector in which information belonging to that file is stored - in the Catalog Index. When information from the file is to be retrieved, the system reads the starting sector address in the Catalog Index, moves to that location, and sequentially reads as many sectors as needed to retrieve the required information.

5.4 INITIALIZING THE CATALOG

Before any information can be recorded on the disk with catalog procedures, the catalog itself must be initialized with a SCRATCH DISK statement. In the SCRATCH DISK statement, you must tell the system three things:

1. The disk platter on which the catalog is to be established. Separate and independent catalogs are established on each disk platter, and each must be initialized independently. The 'F' or 'R' parameter is used to specify the desired disk platter.

2. The number of sectors which are to be reserved for the Catalog Index (any number between 1 and 255 is allowed). The 'LS' parameter is used for this purpose.

3. The address of the last sector to be used for the Catalog Area. (Cataloged files cannot be stored on the disk beyond this sector.) The 'END' parameter is used for this purpose. Obviously, you cannot reserve more sectors for the Catalog Area than there are sectors on the disk platter; thus, the address of the last sector in the Catalog Area must not be higher than the address of the last sector on the disk platter. (This address varies according to the capacity of the disk configuration. Check one of the Appendices B through D to determine the highest legal sector address of your Model.)

Example 5-1: Initializing the Catalog

```
10 SCRATCH DISK F LS=100, END=1000
```

= 5, END = 1023

Statement 10 instructs the system to initialize a catalog on the disk platter designated by 'F' ('F' designates the Fixed Disk Platter on the Models 2230 and 2260, and Drive #1 on the Model 2270). One hundred sectors are reserved for the Catalog Index on this platter (LS = 100), and sector 1000 is specified as the last sector in the Catalog Area (END = 1000). Note that each disk platter must be initialized separately (i.e., with a separate SCRATCH DISK statement).

In deciding how many sectors you should allocate for the Catalog Index, keep in mind the fact that the first sector of the Index (sector 0) can store 15 file names, and each subsequent sector (up to sector 254) can store 16 file names. Thus, if you intend to store 15 or fewer files on a disk platter, one sector will be adequate for the Index. If you intend to store 16 or more files, two or more sectors must be reserved for the Index. It is important to note, however, that the size of the Catalog Index, once fixed, cannot be altered without reorganizing the entire catalog. The Index should therefore generally be allotted enough space to provide for any possible future additional files.

It is not strictly necessary to specify the number of sectors to be reserved for the Catalog Index. If you do not specify the number of sectors to be reserved in your SCRATCH DISK statement (i.e., if the 'LS' parameter is omitted), the system automatically reserves the first 24 sectors on the disk platter for a Catalog Index. In the first sector of the Catalog Index (sector 0), a maximum of 15 file names (and associated file information) can be stored. In each subsequent sector of the Index, a maximum of 16 file names can be stored. Thus, a Catalog Index of 24 sectors provides enough space to store a maximum of 383 file names.

Example 5-2: Initializing the Catalog ('LS' Parameter Omitted)

```
20 SCRATCH DISK R END = 1000
```

Statement 20 instructs the system to establish a Catalog on the disk platter designated by 'R' ('R' designates the

Removable Disk Platter on the Models 2230 and 2260, and Drive #2 on the Models 2270-2 and 2270-3; the 'R' parameter is illegal on a Model 2270-1). Sector 1000 is specified as the last sector to be used in the Catalog Area (END = 1000). Since the 'LS' parameter is omitted, the system automatically reserves the first 24 sectors on the 'R' platter for the Catalog Index.

5.5 SAVING PROGRAMS ON DISK

Once the catalog is initialized, cataloged information can be stored on the disk. Information recorded on the disk must be stored in either of two types if files, program files or data files. A data file may contain a large collection of data. A program file, however, always contains only one BASIC program or program segment.

The SAVE DC command is used to save program files on the disk. One program file is created automatically whenever a SAVE DC command is executed. A program file consists of the BASIC program or program segment being saved, as well as certain control information which is automatically included in the file by the system when the program is stored on disk.

When a program is recorded with a SAVE DC command, the system must be supplied with the following required information:

1. The disk platter on which the program is to be stored ('F' or 'R'). The specified disk platter must have been initialized with a SCRATCH DISK statement.

2. The name of the program. You must name the program so that the system has some way of identifying it when it is stored on the disk. The name can be from one to eight characters in length. It may be specified as a literal string in quotes, or as the value of an alphanumeric variable.

Example 5-3: Saving a Program on Disk

```
SAVE DC R "PROG 1"
```

This command (the term "command" indicates that SAVE DC is not programmable) instructs the system to transfer all program lines currently in memory to the disk platter designated by 'R' and name this program file "PROG 1". The file's name ("PROG 1") and location are automatically listed in the Catalog Index.

It is also possible to save just a portion of a program currently in memory. This is accomplished by including the appropriate line numbers in the SAVE DC command (see Examples 5-4 and 5-5).

Example 5-4: Saving Part of a Program on Disk
(One Line Number Specified)

```
SAVE DC R "PROG 2" 200
```

This command instructs the system to transfer all program lines in memory beginning with line 200 onto the disk platter designated by 'R'. The program is named "PROG 2", and its name and location are automatically entered in the Catalog Index.

Example 5-5: Saving Part of a Program on Disk (Two Line Numbers Specified)

```
A$ = "PROG 3"
SAVE DC R A$ 200, 500
```

This command transfers program lines 200 through 500 from memory to the 'R' disk platter. The program is named "PROG 3", since that is the value of A$, and the program's name ("PROG 3") and location are entered in the Catalog Index.

5.6 RETRIEVING PROGRAMS STORED ON DISK

Cataloged programs are retrieved from the disk with the LOAD DC instruction. The LOAD DC instruction produces a different sequence of events depending upon its mode of execution. When LOAD DC is executed in Immediate Mode, it functions as the LOAD DC command; the LOAD DC command stimulates a specific sequence of operations. When LOAD DC is executed in Program Mode (i.e., on a numbered program line), it functions as the LOAD DC statement; the LOAD DC statement initiates a sequence of operations different from those associated with the LOAD DC command.

The LOAD DC Command

The LOAD DC command is never executable in Program Mode; it is executed in Immediate Mode only. If the LOAD DC instruction appears in a program (i.e., on a numbered program line) it is always interpreted as a LOAD DC statement, and the operations associated with the LOAD DC statement are carried out by the system. The LOAD DC command instructs the system to locate a named program on a specified disk platter, and load the program into memory. The system checks the Catalog Index for the specified program name, determines the program's location in the Catalog Area, and moves to that location to load the program.

Following execution of the LOAD DC command, the newly loaded program is appended to existing program text in memory. New program lines which have the same numbers as program lines already stored in memory replace the currently stored lines in memory. Currently stored program lines which do not have the same line numbers as new program lines are not cleared, however; they remain as lines in the new program. (For example, if the old program has lines numbered 5, 15, 25, etc., and the newly-loaded program lines are numbered 10, 20, 30, etc., the new program in memory has lines numbered 5, 10, 15, 20, etc.) For this reason, it is generally wise to clear memory prior to loading the new program. All of memory can be cleared by executing a CLEAR command prior to executing the LOAD DC command. Alternatively, a CLEAR P command causes only program text to be cleared from memory. After the new program is loaded, it is necessary to key RUN and EXECUTE in order to execute the newly loaded program.

The LOAD DC command must include the following two items of information:

1. The disk platter (either 'F' or 'R') on which the desired program is stored.

2. The name of the program which is to be retrieved (the name may be specified as a literal string in quotes, or as the value of an alphanumeric variable).

Example 5-6: Loading a Cataloged Program File from Disk

```
CLEAR
LOAD DC R "PROG 1"
```

This command instructs the system to load PROG 1 from the disk platter designated by 'R'. When the command is executed, the system accesses the 'R' platter and searches for the program name "PROG 1" in the Catalog Index. Upon locating the name in the Catalog Index, the system checks the starting sector address of the program, and moves to that address in the Catalog Area to begin loading PROG 1 into memory. The new program is appended to existing program text in memory (new program lines which have the same number as program lines already in memory replace the currently stored lines in memory). After the program is loaded, it is necessary to key RUN and EXECUTE in order to begin execution of the new program.

If the program name specified in the LOAD DC command ("PROG 1" in Example 5-6 above) is not located in the Catalog Index on the specified disk platter, an error is indicated. Note that the program name supplied in a LOAD DC command must correspond exactly to the program name listed in the Catalog Index. Any misspelling results in an error.

Example 5-7: Attempting to Load a Non-Cataloged Program from Disk

```
CLEAR
LOAD DC R "PROG 1"
```

This command is meant to retrieve PROG 1 from the 'R' disk platter. Because the program's name is misspelled, however ("PRAG 1" instead of "PROG 1"), the system cannot find a program under this name in the Catalog Index. It therefore signals an error:

```
LOAD DC R "PRAG 1"
                  ↑ERR 80
```

Where Error 80 = "File Not in Catalog".

<u>The LOAD DC Statement</u>

Cataloged programs also can be loaded into memory from disk under program control. The LOAD DC statement is used for this purpose. The LOAD DC statement is executable only in a program (i.e., on a numbered program line). When the LOAD DC instruction is executed in Immediate Mode, it is always interpreted as a LOAD DC command, and the sequence of operations associated with the LOAD DC command (see above) is followed by the system.

The following sequence of operations is associated with the LOAD DC statement:

1. Stop current program execution.

2. Clear all currently stored program text (or a specified portion of currently stored program text) from memory.

3. Clear all noncommon variables from memory.

4. Locate the named program on the specified platter, and load this program into memory (if the specified name cannot be found in the Catalog Index, an error is signalled).

5. Run the newly loaded program.

In a LOAD DC statement, the system must be provided with the following information, in the order indicated:

1. The disk platter (either 'F' or 'R') on which the desired program is stored.

2. The name of the program which is to be loaded (the name may be specified as a literal string in quotes, or as the value of an alphanumeric variable).

Optionally, a third item may be included:

3. One or two line numbers which identify the first and last program lines to be cleared from memory prior to loading the new program.

If no line number is specified in the LOAD DC statement, the system clears all program text from memory prior to loading the new program from disk. As soon as the program is loaded, execution begins automatically at the first (lowest) program line in the newly loaded program. The LOAD DC statement is commonly used to "chain" programs from the disk. Common variables (so specified in a COM statement) are retained in memory for use by each succeeding program in the chain. Noncommon variables are cleared by the LOAD DC statement.

Example 5-8: Chaining a Program from Disk with the LOAD DC Statement

```
100 LOAD DC F "PART 2"
```

When it is executed, statement 100 stops program execution, clears all program text and noncommon variables from memory, and loads in the program PART 2 from the 'F' disk platter. Execution of PART 2 begins automatically at the first (lowest) line in the progra-

If program segments are to be overlayed from disk, it may be desirable to clear out only a specific portion of program text prior to loading the new program segment. In this case, one or two program line numbers can be included in the LOAD DC statement. Inclusion of a single line number in the statement causes all program text beginning at that line to be cleared from memory prior to loading the new program. Two line numbers instruct the system to clear all program text between and including the specified lines prior to loading the new program. In either case, all non-common variables are also cleared. Execution of the newly loaded program begins at the first line number specified in the LOAD DC statement. If this line number does not appear in the newly loaded program, an ERROR 11 (Missing Line Number) is signalled.

Example 5-9: Loading a Program Overlay from Disk

```
200 LOAD DC F "PART 3" 300, 900
```

Statement 200 halts program execution and clears program lines 300 through 900 from memory, along with all non-common variables, prior to loading program overlay PART 3 from the 'F' platter. After PART 3 is loaded, program execution continues automatically at line 300. If PART 3 contains no line number 300, an ERROR 11 (Missing Line Number) is signalled.

5.7 LISTING THE CATALOG INDEX

You can obtain a list of the names and locations of all cataloged files on a disk platter, as well as certain information about the catalog itself, by executing a LIST DC statement. In the LIST DC statement, you must specify the disk platter whose Index is to be listed. When the LIST DC statement is executed, the following information is returned:

1. The number of sectors reserved for the Catalog Index on that disk platter.
2. The address of the last sector reserved for the Catalog Area.
3. The current end of the Catalog Area.
4. The name of each cataloged file.
5. The file type (program or data) of each file.
6. The starting and ending sector addresses of each file.
7. The number of sectors used in each file.

Example 5-10: Listing the Catalog Index

```
50 LIST DC R
```

Statement 50 causes the system to list the contents of the Catalog Index from the disk platter designated by 'R'.

This platter was initialized in Example 5-2 above, and program files were saved in Examples 5-3, 5-4, and 5-5; the listing therefore looks like this:

```
             REMOVABLE CATALOG
      INDEX SECTORS = 00024
      END CAT. AREA = 01000
      CURRENT END   = 00132

      NAME      TYPE  START   END    USED
      PROG 2     P    00051  00112  00062
      PROG 3     P    00113  00132  00020
      PROG 1     P    00024  00050  00017
```

Figure 5-1. The Catalog Index Listing

There are several things which should be noticed about the information in this listing. Notice, first, that all files are stored sequentially. The Catalog Index occupies the first 24 sectors (sectors 0-23). The first file, PROG 1, is stored beginning at the first available sector following the Index (sector 24). PROG 2 begins at the first available sector following PROG 1 (sector 51), and PROG 3 starts with the first sector after PROG 2 (sector 113). Notice also, however, that the Catalog Index entries themselves are not listed in sequential order. That is because entries in the Catalog Index are stored in a "hashed" order, which minimizes the system's search time for finding entries in the Index.

You should observe, finally, that the USED column opposite each program name indicates the number of sectors occupied by that program. In the cases so far discussed, the system automatically uses exactly enough sectors on the disk to store each program. It is also possible to reserve extra sectors in a program file beyond the number needed to store the program; these extra sectors can be used subsequently for additions to the program. The technique for reserving extra sectors in a program file is discussed in Chapter 7.

NOTE TO OWNERS OF THE MODEL 2270:

The Catalog Index listing is always identified as the "Fixed Catalog" or the "Removable Catalog".

On the Model 2270-2, the "Fixed Catalog" refers to the Catalog Index listing for the diskette in drive #1; the "Removable Catalog" identifies the Catalog Index listing for the diskette in drive #2.

On the Model 2270-1, the "Fixed Catalog" identifies the Catalog Index listing for diskette #1. It is not possible to generate a "Removable Catalog" listing.

On the Model 2270-3, the "Fixed Catalog" identifies the Catalog Index listing for Diskette #1 and for Diskette #3; the "Removable Catalog" identifies the Catalog Index listing for Diskette #2.

5.8 SAVING DATA ON DISK

The Hierarchy of Data

Unlike a program file, which always contains only a single program or program segment, a data file normally contains several different items of data. Obviously, it would be unwise simply to dump data on the disk in a random or disorganized fashion, since there would then be no efficient way to retrieve specific items when they were needed. In order to facilitate fast, efficient retrieval of data from the disk, data stored on disk is organized into a well-defined structure or "hierarchy."

The hierarchy of data contains two levels: on the lower level, individual data relating to a single subject (such as a particular customer, or a particular item in the inventory) are organized into a data record (also known as a logical record); at the higher level, a number of related logical records are organized into a data file (say, an inventory file or customer file). An inventory file, for example, typically contains a number of inventory records, each of which in turn contains information about an individual item in the inventory (such as model number, name, price, number in stock, etc.). Whenever a particular piece of information about one of the items in the inventory is needed, the procedure is first to locate the inventory file, then to locate the desired record within the file.

Catalog Mode permits the programmer to open a number of different files on disk or, if it is more convenient, a single large file which occupies the entire Catalog Area. Within each file, the individual records can be as long as necessary (but each record occupies a minimum of one sector on disk, unless special techniques are used to "block" records in a sector). In Catalog Mode, the system automatically keeps track of where each file is located on the disk. It is up to the programmer, however, to locate individual records within the file. A special disk utility, the Wang System 2200 Keyed File Access Method (now available in two new versions, designated KFAM-2 and KFAM-3), provides a file maintenance system which keeps track of the location of each record in a cataloged file, thereby facilitating access to the records. Ask your Sales Representative for further information about KFAM-2 and KFAM-3.

Because the system itself has no way of knowing how many records will be stored in a file, or how many sectors each record will occupy, it is the programmer's responsibility to estimate how many sectors each data file will require. The system must be instructed to reserve adequate space for the file on a designated platter. Thus, two steps are required to save data on the disk:

1. First, a data file must be cataloged, or "opened", with a special statement, DATASAVE DC OPEN. In this statement, the new data file is named, and the number of sectors to be reserved for the file is specified. No data is actually stored in the file at this point.

2. Once the file is opened, data records can be stored in the file with the DATASAVE DC statement.

Opening A Data File On Disk

A data file is opened on the disk with a DATASAVE DC OPEN statement, which requires the following information:

1. The disk platter (either 'F' or 'R') on which the data file is to be opened. This disk platter must have been initialized with a SCRATCH DISK statement. (See Section 5-4.)

2. The number of sectors to be reserved for the data file. Take care that the file does not extend beyond the limits of the Catalog Area (if it does, an error is signalled).

3. The name of the data file. You must name the file so that the system has some way of identifying it. The name can be from one to eight characters in length, and may be specified either as a character string in quotes or as the value of an alphanumeric variable.

When the DATASAVE DC OPEN statement is executed, the specified number of sectors are reserved for the newly-opened file in the Catalog Area. The last sector of the file is used by the system for a special control record which marks the absolute end of the file; no data can be written in the file beyond that point. The file's name and location are also automatically entered in the Catalog Index.

Example 5-11: Opening a Data File on Disk

```
150 DATASAVE DC OPEN F 100, "DATFIL 1"
```

Statement 150 instructs the system to reserve 100 sectors on the disk platter designated by 'F' for a data file, and name this file "DATFIL 1". The file's name ("DATFIL 1") and location are entered automatically in the Catalog Index on the 'F' platter.

> NOTE:
>
> The system automatically allocates the last sector in each data file exclusively for the system control record. The system control record contains control information and pointers used by the system in maintaining the data file, and no data can be stored in this sector. It is also generally desirable to write an end-of-file trailer record in a data file after all data has been stored; the trailer record likewise occupies one sector which cannot be used for data. Thus, it is always good programming practice to reserve at least two more sectors than are actually required for a data file in order to account for the two sectors which cannot be used. For example, if you wish to store 24 sectors of data in a file, you should reserve at least 26 sectors (24 + 2) in the DATASAVE DC OPEN statement.

If a LIST DC statement is executed following line 150 in Example 5-11, the listing should look like this:

```
          FIXED CATALOG
INDEX SECTORS = 00100
END CAT. AREA = 01000
CURRENT END   = 00199

NAME       TYPE  START   END    USED
DATFIL 1    D    00100  00199  00001
```

Figure 5-2. Catalog Index Entry for DATFIL 1

One hundred sectors are reserved for DATFIL 1 (000100-000199), but, despite the fact that no data has yet been saved in the file, the USED column for DATFIL 1 indicates that one sector is already occupied. This is the last sector in the file, automatically set aside for system control information in DATFIL 1. Thus, although 100 sectors were reserved for DATFIL 1, only 99 of those sectors can actually be used for data storage. If 100 sectors are needed for data, at least 101 must be reserved for the data file.

> NOTE:
>
> Wang systems do not distinguish between input files and output files in disk operations. Thus, data can be either written in or read from a file which has been opened with a DATASAVE DC OPEN statement.

Saving Data In A Data File On Disk

Once a data file has been opened on a disk platter, data can be stored in the file with a DATASAVE DC statement. All of the data values (or the variables and arrays containing the data values) which are to be included in one record must be listed in the DATASAVE DC statement. This information is referred to as the DATASAVE DC "argument list." Individual items must be separated by commas. The system automatically groups information from the argument list sequentially in a logical data record, and stores this record in the currently open data file on the disk.

Suppose, for example, that you wish to create a record containing the name, street address, and birth date of an employee, and store this record in the file DATFIL 1. Since DATFIL 1 was recently opened with a DATASAVE DC OPEN statement (Example 5-11), it is the currently open data file on disk. Assuming that the information is stored in several variables, you can transfer the data into DATFIL 1 simply by including the variable names in the argument list of a DATASAVE DC statement, as in Example 5-12:

Example 5-12: Saving Data in a Data File

```
160 A$ = "PETER RABBITT"
170 B$ = "4 OAK DRIVE"
180 N = 032948
190 DATASAVE DC A$,B$,N
```

Statement 190 instructs the system to transfer all values from the variables A$, B$, and N into the currently open data file on disk (DATFIL 1). Collectively, the three items of information "PETER RABBITT", "4 OAK DRIVE", and 032948 constitute one logical record in DATFIL 1.

If, after saving a record in DATFIL 1, you execute a LIST DC F statement, the Index looks like this:

```
          FIXED CATALOG
INDEX SECTORS = 00100
END CAT. AREA = 01000
CURRENT END   = 00199

NAME      TYPE  START   END    USED
DATFIL 1    D   00100  00199  00001
                                ↑
                              USED column
                              not yet updated
```

Figure 5-3. Catalog Index Entry for DATFIL 1.

Notice that the USED column has not yet been updated to reflect the newly stored data in DATFIL 1. Since all the information in this record can be stored in one 256-byte sector, the USED column for DATFIL 1 should read 0002, indicating that one sector in DATFIL 1 has been used for data, in addition to the single sector reserved for system information. Why doesn't it?

The answer is simple: The USED column in a data file is updated only when an end-of-file record has been written in the file. The end-of-file (or trailer) record tells the system, in effect, "no data is stored in this file beyond this point". With this information, the system can determine how many sectors in the file are filled with data, and can update the USED parameter appropriately. A trailer record is not written in the file automatically, however; it must be created by the programmer with a DATASAVE DC END statement. The USED parameter for DATFIL 1 could be updated by following statement 190 in Example 5-12 with a DATASAVE DC END statement, as shown in Example 5-13:

Example 5-13: Writing an End-Of-File (Trailer) Record in a Data File on Disk

```
200 DATASAVE DC END
```

Statement 200 instructs the system to write a trailer record into DATFIL 1.

If you now perform a listing of the Catalog Index, the Index looks like this:

```
              FIXED CATALOG
        INDEX SECTORS = 00100
        END CAT. AREA = 01000
        CURRENT END   = 00199

        NAME      TYPE  START   END    USED
        DATFIL 1   D    00100  00199  00003
                                        ▲
                                     Updated USED
                                     now shows one sector used for
                                     file control, one sector for end-
                                     of-file record, and one sector
                                     for data record.
```

Figure 5-4. Updated Catalog Index Entry for DATFIL 1

As you can see, the USED column is now updated. It is good programming procedure to write a trailer record every time you have finished saving data in a file so that you will always know how much of the file is filled, and how much space remains. However, it is not necessary to write a trailer record after every DATASAVE DC statement; instead, a single DATASAVE DC END statement can be used at the conslusion of a disk write routine.

Example 5-14: Writing a Data Trailer Record after a Series of DATASAVE DC Statements

```
200 DATASAVE DC A()
210 DATASAVE DC B(),N,M(3)
220 DATASAVE DC A$,T$()
230 DATASAVE DC END
```

Lines 200-220 instruct the system to transfer data from the numeric and alphanumeric variables, arrays, and array elements specified in the respective argument lists, and store this data in the currently open data file (DATFIL 1) on disk. Statement 230 instructs the system to write an end-of-file trailer record following the last data record in DATFIL 1, and update the USED parameter for DATFIL 1 in the Catalog Index to indicate how many sectors have been used.

In addition to updating the USED parameter for the file, there are three major advantages to writing an end-of-file trailer record in a data file:

1. The trailer record makes it possible to skip to the end of stored data in a file in order to write new records in the file. (See Section 5-12.)

2. The trailer record makes it possible to test for the end of stored data (last record) in a file when reading through the file sequentially under program control. The IF END THEN statement is used for this purpose. (See Section 5-13.)

3. The trailer record insures against accidentally reading beyond the last valid data record in a file.

> WARNING:
>
> Never use the RESET button to terminate program execution during a disk write routine. RESET causes the disk to immediately terminate any operation and return the read/write head to the home position, even if it is in the middle of writing a sector. Thus, it is possible that a half-written sector may be left in the file following a RESET operation. Any subsequent attempt to read the half-sector results in an error. To avoid this problem, always use the HALT/STEP key if you wish to halt program execution during a disk write routine. HALT/STEP permits the disk to complete the write operation for the current sector before terminating the data transfer.

5.9 THE STRUCTURE OF DATA FILES

Up to now the discussion has focused primarily on the mechanics of saving data on the disk; little attention has been paid to the actual manner in which data is organized and stored by the system. It will be helpful to consider this question briefly now (a more detailed discussion is reserved for the following chapter), prior to discussing the retrieval of data from a cataloged data file on disk.

The major concept to be understood in connection with data files is that of a logical data record. A single logical record (or data record) is created in a file on the disk with each DATASAVE DC statement. The logical record contains all of the data included in the DATASAVE DC argument list, as well as certain control information inserted by the system. Suppose, for example, that the following statements are executed:

```
10 DATASAVE DC OPEN R 200, "DATFIL 1"
20 DATASAVE DC "PETER RABBITT", 01121,B$,N,A()
```

Statement 10, as you know, opens DATFIL 1 on the 'F' platter. Statement 20 creates one logical record in DATFIL 1 containing all the data in the DATASAVE DC argument list. Notice that there are several different types of data in the argument list. The first item is a literal string "PETER RABBITT". Whenever a literal string is specified in a DATASAVE DC argument list, it must be enclosed in quotes. The second item, 01121 is a numeric value which need not be set in quotes. The third item, B$, is an alphanumeric variable, the fourth, N, is a numeric variable, and the fifth, A(), is a numeric array. Empty parentheses are used to indicate that the entire array is to be saved. Thus, if array A() contains four elements, the statement

```
DATASAVE DC A()
```

is equivalent to the statement

```
DATASAVE DC A(1), A(2), A(3), A(4)
```

Array elements are recorded in sequence (two-dimensional arrays are stored row by row). Each individual item in the DATASAVE DC argument list (including each array element) is considered to be a single argument. Thus, if the array A() is dimensioned to contain four elements, it is regarded as a collection of four separate arguments, and the DATASAVE DC argument list in statement 20 consists of a total of eight arguments.

When the DATASAVE DC statement is executed, the arguments are taken in sequence from the argument list and stored in a logical record on the disk (if a two-dimensional array is included in the argument list, the array elements are transferred row by row). Thus, if the following assignments are assumed, the logical record created by statement 20 resembles the figure below.

```
B$   = "10 OAK DRIVE"
N    = 2222
A(1) = 123
A(2) = 456
A(3) = 789
A(4) = 100
```

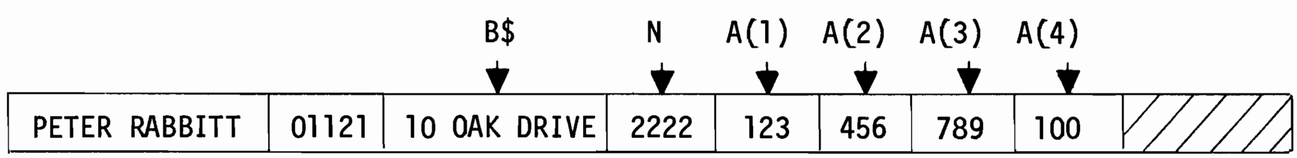


Figure 5-5. Logical Record Consisting of One Sector

The arguments saved in a logical record on disk are commonly referred to as fields within the record. In the record above, for example, "PETER RABBITT" is the first field in the record, while '100' is the last field. It is important to note that when a logical record is read back into memory from disk, each field must be read into a single variable or array element; it is never possible to read two or more fields into a single variable or array element, even if the receiving variable or element is large enough to contain more than one field. Note, too, that alphanumeric fields must be read back into alphanumeric variables or array elements, and numeric fields must be read back into numeric variables or array elements.

In the present example, the logical record occupies somewhat less than one sector. Notice in Figure 5-5 above that the remainder of the sector is unused. The remainder of the sector contains meaningless data, which is ignored by the system (the system provides automatic safeguards against accidentally reading this meaningless data when the record is read back into memory). If another logical record is created (with a second DATASAVE DC statement), the new record begins at the beginning of the next sector. The remaining unused portion of the first sector is not used for the second record. A logical record always begins at the beginning of a sector. This is the case even if the logical record occupies only a very small portion of the sector. For example, consider the statements:

```
30 DATASAVE DC A
40 DATASAVE DC B
```

Each of these statements creates a single logical record containing a single numeric value, and each occupies an entire sector on the disk:

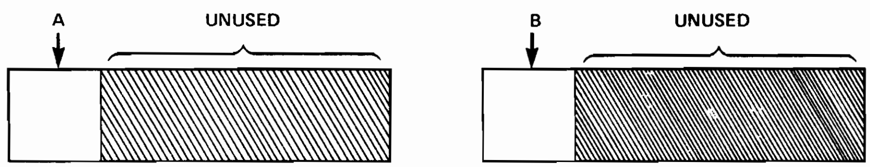


Figure 5-6. Two One-Sector Logical Records

Obviously, this is not a very efficient way to store data. It would surely be more efficient to store both values in a single record, with a single DATASAVE DC statement (e.g., DATASAVE DC A,B). In this case, both values occupy the same sector.

On the opposite end of the spectrum, a single logical record can occupy several sectors - as many sectors, in fact, as are required to store all the data in the DATASAVE DC argument list. Consider, for example, the following routine:

```
60 DIM A(60)
70 DATASAVE DC A()
```

In this case, the DATASAVE DC argument list contains 60 arguments, each consisting of a single numeric array element. Since 28 full-precision numeric values can be stored in a sector, the data in the logical record created by statement 70 occupies two complete sectors and a small portion of a third:

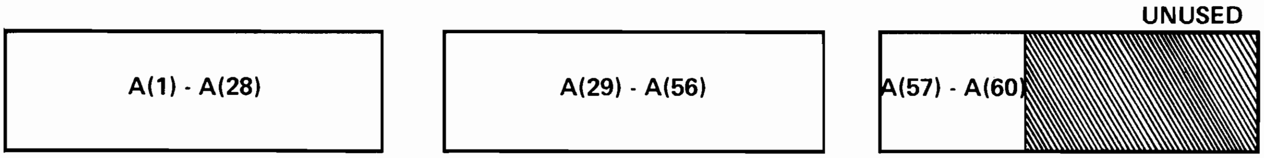


Figure 5-7. Logical Record Consisting of Three Sectors

The logical record created by statement 70 requires three sectors on disk. Remember that the next logical record begins with the next consecutive sector. The remainder of sector number three in this record remains unused.

Opening a Second Data File on Disk

After all the necessary data has been recorded in DATFIL 1, it may be desirable to open a second data file, DATFIL 2, which is to contain a whole new set of records. This is done with a second DATASAVE DC OPEN statement, for example:

```
250 DATASAVE DC OPEN F 500, "DATFIL 2"
```

However, it is important to recognize that the opening of DATFIL 2 in effect "closes" DATFIL 1, since DATFIL 2 now becomes the currently open file on disk, and any DATASAVE DC or DATALOAD DC statement now automatically accesses DATFIL 2 instead of DATFIL 1. Chapter 6 introduces a technique for keeping more than one file open on disk at the same time. For the present, however, it is assumed that only one file can be the currently open file at any given moment.

5.10 REOPENING A DATA FILE ON DISK

After a data file has been opened on disk and subsequently "closed" by opening a second file, the data in the original file can be accessed by reopening the file with a DATALOAD DC OPEN statement. DATALOAD DC OPEN is used to reopen an existing file regardless of whether you intend to store additional data in the file or to read existing data from it. The DATASAVE DC OPEN statement, used to open a file initially, is never used to reopen an existing file; any attempt to use this statement to reopen a file produces an error.

In the DATALOAD DC OPEN statement, you must supply the system with the following information:

1. The disk platter (either 'F' or 'R') on which the file is cataloged.

2. The name of the file.

When a DATALOAD DC OPEN statement is executed, the system searches the Catalog Index on the designated platter for the specified file name.

The file's location is then recorded in memory for any future reference to the file.

Example 5-15. Reopening a Cataloged Data File

```
300 DATALOAD DC OPEN F "DATFIL 1"
```

Statement 300 causes the system to search the Catalog Index on the 'F' platter for the file name "DATFIL 1". When the name is found, the file's location is read and stored in memory for future reference.

Of course, the file name specified in the DATALOAD DC OPEN statement must be the name of a data file currently cataloged on the specified platter. If the system cannot locate the file name in the Catalog Index, an error is signalled.

Example 5-16: Attempting to Reopen a Non-Cataloged Data File

```
300 DATALOAD DC OPEN F "DOTFIL 1"
                                    ↑ERR 80
```

Statement 300 attempts to reopen a data file whose name is not listed in the Catalog Index. Since "DOTFIL 1" is not identical to "DATFIL 1", Error 80 (File Not In Catalog) is indicated.

Once a file has been reopened with a DATALOAD DC OPEN statement, it is possible both to store new data in the file (with a DATASAVE DC statement), and to read existing data from the file (with a DATALOAD DC statement).

5.11 RETRIEVING DATA FROM A DATA FILE ON DISK

Data which is stored on a disk would not have much value if it could not be read back into memory for analysis and processing. In Catalog mode, data is read from a currently open file on disk with a DATALOAD DC statement. When loading data from the disk into memory, you must tell the system which variable(s) and/or array(s) in memory are to receive the data. The list of receiving variables and arrays is specified in a DATALOAD DC statement, and is known as the "argument list" for that statement. As with DATASAVE DC, it is possible to specify an entire array in a DATALOAD DC argument list by following the array name with empty parentheses, e.g., A(), B$(). In this case, each element of the array is regarded as a single receiving argument. The system reads one or more logical records from the currently open file on disk (if no file is currently open, an error is indicated), and stores the data in the variable(s) and array(s) specified in the argument list. The system continues to read data from the file until all arguments in the argument list have been filled, or until there is no more data remaining in the file. If the argument list contains more receiving variables than there are fields in a record, the first fields of the next sequential record are automatically read to satisfy all unfilled variables. The remainder of the second record is then read and ignored, and the system is positioned at the beginning of the third record. If only the first few fields in a record are read (i.e., if the argument list contains fewer receiving arguments than there are fields in the record), the remainder of the record is read but ignored, and the system is positioned at the beginning of the next record.

Example 5-17: Reading Data from a Data File on Disk

```
310 DIM B(60)
320 DATALOAD DC B()
```

Statement 310 dimensions an array B() to hold 60 elements. Statement 320 instructs the system to load enough data from the currently open file on disk to fill array B().

It is in general good programming procedure to read back exactly one logical record with each DATALOAD DC statement. For example, if a record of 60 fields is saved with a DATASAVE DC statement, the argument list in the DATALOAD DC statement should consist of 60 receiving arguments, so that the entire logical record is retrieved.

Example 5-18: Saving and Loading One Logical Record

```
100 DIM A(60)
150 DATASAVE DC OPEN F 100, "DATFIL 1"
160 DATASAVE DC A()
170 DATASAVE DC END
:
:
240 DIM B(10),C(10),D(10),E(10),F(10),G(10)
250 DATALOAD DC OPEN F "DATFIL 1"
260 DATALOAD DC B(),C(),D(),E(),F(),G()
```

Statement 150 opens DATFIL 1 and statement 160 stores data from the array A() (which contains 60 elements) in DATFIL 1. In the intervening program execution, DATFIL 1 is closed. Statement 250 reopens DATFIL 1, and statement 260 loads one logical record (consisting of 60 values) from DATFIL 1 into six receiving arrays, each consisting of 10 elements. Note that it is not necessary for the DATALOAD DC argument list to be identical to the DATASAVE DC argument list, so long as both contain the same number of arguments, of the same types (alpha or numeric).

Example 5-19: Loading Portions of a Logical Record

```
50 DIM B(20),N(30),S(40)
60 DATASAVE DC OPEN F 100 "DATFIL 1"
70 FOR I = 1 TO 30
80 INPUT N(I)
90 NEXT I
100 DATASAVE DC N()
110 GO TO 70
.
.
.
390 DATALOAD DC OPEN F "DATFIL 1"
400 DATALOAD DC B()
410 DATALOAD DC S()
```

Lines 70-90 constitute an input loop used to enter data into array N(), which contains 30 elements. At line 100, this array is recorded in a logical record in the data file DATFIL 1. In subsequent processing, DATFIL 1 is closed, and is reopened at line 390. At line 400, a DATALOAD DC statement is used to read the first logical record from DATFIL 1 into array B(). However, B() contains only 20 elements, while the logical record has 30 fields. The first 20 fields are therefore read into B(), and the remaining 10 fields are read but ignored, since there is no place to store them in memory. At the conclusion of this operation, the system is positioned at the beginning of logical record #2. This record is read into array S() at line 410. However, S() contains more receiving elements (40) than there are fields in the logical record (30). The first 10 fields of the third logical record are automatically read to fill the last 10 elements of S(), and the system is positioned at the beginning of logical record #4.

Other problems can result if a DATALOAD DC argument list does not correspond to the argument list of the DATASAVE DC statement which created the record initially. In particular, you should keep the following two points in mind:

1. Each field in the logical record must be read into a single receiving argument (variable of array element). It is not possible to load two or more fields into one receiving argument. For example, if your record contains two four-character alphanumeric fields, "ABCD" and "EFGH", both fields cannot be read into a single alphanumeric variable, even if the variable can store more than eight characters.

2. Alphanumeric fields must be returned to alphanumeric receiving arguments, and numeric fields must be returned to numeric receiving arguments. Any attempt to read an alphanumeric value into a numeric variable, or vice-versa, results in an ERROR 43 (Wrong Variable Type). For example, if you save a record with the statement

   ```
   50 DATASAVE DC A$,N
   ```

then try to read it back with the statement

```
100 DATALOAD DC N,A$
```

the system generates an ERROR 43 (Wrong Variable Type).

Thus, you should be sure that the size, number, type, and order of the receiving arguments in a DATALOAD DC argument list corresponds to the argument list of the DATASAVE DC statement with which the record was created.

5.12 SKIPPING AND BACKSPACING OVER LOGICAL RECORDS IN A DATA FILE

An existing data file on the disk is generally reopened (with a DATALOAD DC OPEN statement) for one of three reasons:

1. To read data from the file.
2. To store additional data in the file.
3. To change or update existing data in the file.

In any of these three cases, it is usually necessary to access one or more specific logical records within the file. Two catalog statements, DSKIP and DBACKSPACE, enable you to move to a particular record within a file without reading through all intervening records.

The use of DSKIP and DBACKSPACE can be illustrated by considering a file which consists of several logical records:

```
400 DATASAVE DC OPEN F 50, "TEST 1"
410 DATASAVE DC A()
420 DATASAVE DC B()
430 DATASAVE DC C()
440 DATASAVE DC D()
450 DATASAVE DC E()
460 DATASAVE DC END
```

This file, named "TEST 1", occupies 50 sectors on the 'F' platter. Five logical records (statements 410-450) have been stored in TEST 1, and a trailer record has been written following the last logical record. Assuming that each logical record consists of two sectors, the five records occupy ten sectors (see Figure 5-8).

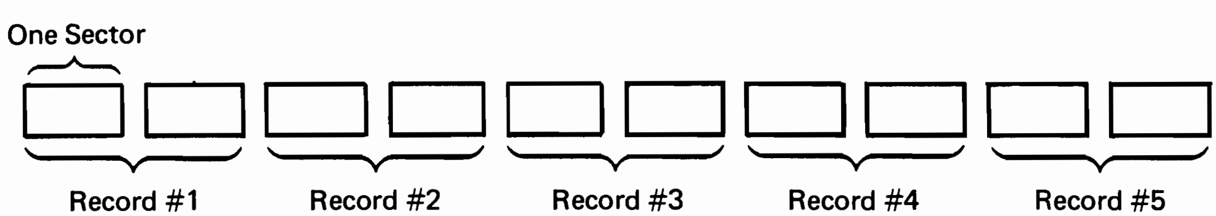


Figure 5-8. Logical Records in TEST 1

Suppose, now, that TEST 1 is closed and subsequently reopened with a DATALOAD DC OPEN statement. When the file is reopened, the system automatically positions itself at the beginning of the file. In order to access any record other than record #1, the system must be instructed to skip ahead through the file to the desired record. Logical records in a data file are skipped with a DSKIP statement. In the DSKIP statement, you must tell the system how many records to skip. Suppose, for example, you wish to read record #3 in the file. Since the system is currently positioned at record #1, it is necessary to skip two records. (See Figure 5-9.)

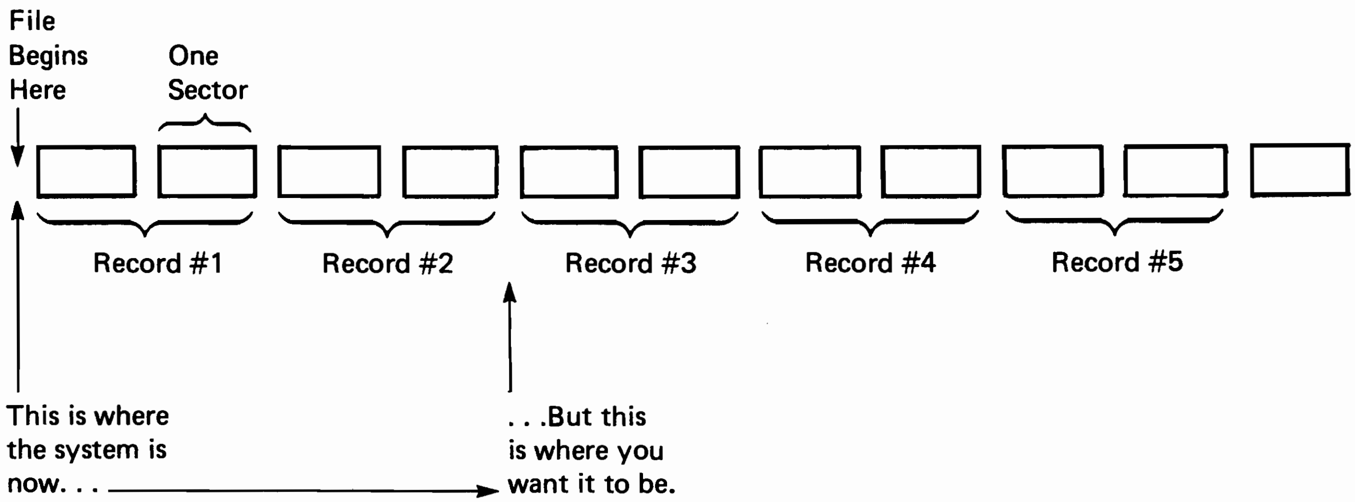


Figure 5-9. Skipping over Logical Records in a Data File

Example 5-20: Skipping over Logical Records in a Data File

```
470 DATALOAD DC OPEN F "TEST 1"
480 DSKIP 2
```

Statement 470 reopens TEST 1. The system is positioned at the beginning of the file. Statement 480 instructs the system to skip two logical records (records #1 and #2), and reposition itself at the beginning of record #3. A DATALOAD DC statement such as

```
490 DATALOAD DC C()
```

now loads record #3 from the file into memory.

Notice that the number supplied in the DSKIP statement specifies how many logical records are to be skipped (remember that each logical record was created by a single DATASAVE DC statement). It does not matter how many sectors are contained in each logical record (record #1 might contain five sectors, for example, while record #2 contains ten, etc.). Be sure, however, that the argument list of the DATALOAD DC statement which is used to load a record corresponds to the argument list of the DATASAVE DC statement which originally created the record.

After a logical record has been loaded, the system is positioned at the beginning of the next logical record. Suppose that you now want to load and check logical record #1 from TEST 1. Since the system is currently positioned at the beginning of record #4 (having just loaded record #3), you must backspace three logical records (see Figure 5-10). You can do so with a DBACKSPACE statement.

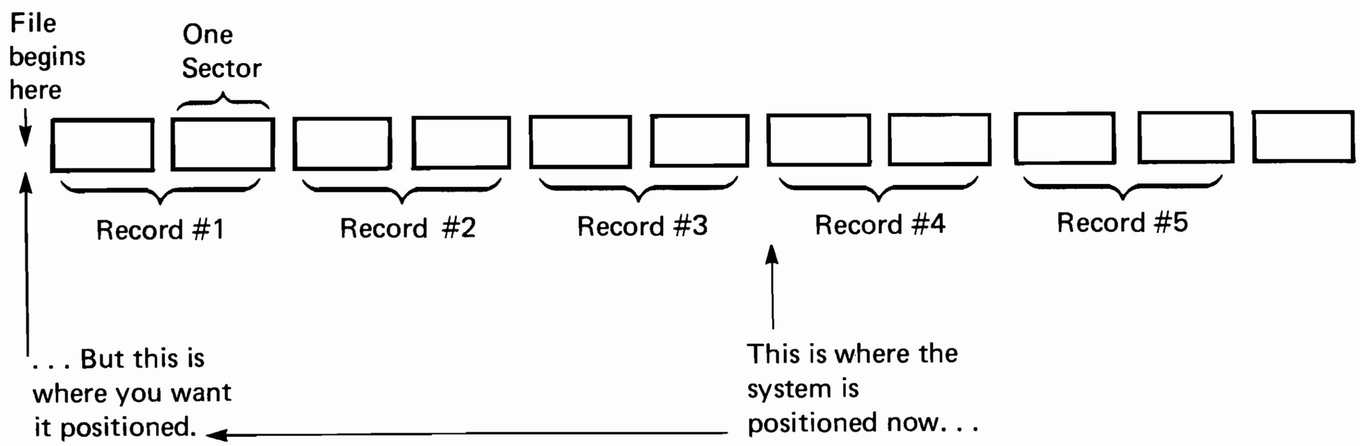


Figure 5-10. Backspacing over Logical Records in a Data File

Example 5-21: Backspacing over Logical Records in a Data File

```
500 DBACKSPACE 3
```

Statement 500 causes the system to backspace over three logical records in the currently open file (TEST 1) on disk. Since the system is currently positioned at the beginning of record #4, it is repositioned to the beginning of record #1 following statement execution. Record #1 can now be loaded with a DATALOAD DC statement such as

```
510 DATALOAD DC A()
```

It is possible to backspace to the beginning of a file from any point in the file with a DBACKSPACE BEG statement. In Example 5-21, for example, it would have been just as easy to access record #1 by backspacing to the beginning of the file and executing statement 510.

Example 5-22: Backspacing to the Beginning of a File

```
500 DBACKSPACE BEG
```

Statement 500 instructs the system to backspace from its current position in the file to the beginning of the file (i.e., the beginning of the first record of the file).

In order to store additional data in a file which has just been reopened, it is necessary to skip to the current end of the file, and begin saving the new data at that point. This can be done with a DSKIP END statement if the current end of file is marked by an end-of-file trailer record. If no end-of-file trailer record has been written in the file, however, an ERROR 82 (No End of File) is returned following execution of the DSKIP END statement. The DSKIP END statement locates the end-of-file trailer record, and repositions the system at the beginning of the trailer record. A new data record can then be saved over the trailer record, and a new trailer record written to mark the new end of the file.

Example 5-23: Skipping to the End of a File

```
520 DSKIP END
```

Statement 520 instructs the system to skip to the current end of the currently open data file on the disk (TEST 1). A trailer record must have been written in the file with a DATASAVE DC END statement (statement 460) following the most recent DATASAVE DC statement (statement 450); otherwise, an ERROR 82 is returned. After the DSKIP statement is executed, the system is positioned at the beginning of the trailer record in the file. A new data record can be saved over the trailer record, and a new trailer record written in the file, with the following statements:

```
530 DATASAVE DC F()
540 DATASAVE DC END
```

5.13 TESTING FOR THE END-OF-FILE

If you have written a data trailer record in your file, you can use it to test for an end-of-file condition when reading the file. For example, suppose that you wish to read all the records from a particular file, but you don't know exactly how many records are stored in the file. You can set up a loop which will continue to load logical records until it encounters a trailer record.

Example 5-24: Testing for the End-Of-File Condition

```
600 DATALOAD DC OPEN F "TEST 1"
610 DATALOAD DC A()
620 IF END THEN 700
:
:
660 GOTO 610
700 STOP
```

Statement 600 opens the file TEST 1, and statement 610 loads a logical record from that file into A(). Statement 620 then tests for the end-of-file record signifying that the last data record in the file has been read. If this is the case, the program jumps to statement 700 and stops. If it is not the case, the data loaded into array A() is processed until, at statement 640, the system is instructed to loop back and load in another record. This example assumes that all records in TEST 1 were written using an argument list identical to A().

> NOTE:
>
> When the end-of-file trailer record is detected by the system with an IF END THEN test, the file's current sector address is set to the address of the trailer record. Thus, the IF END THEN test can be used to cause the system to exit from an input routine after all records have been read, and branch to an output routine which writes additional records in the same file. The first record saved will be written over the trailer record. A new trailer record must, of course, be written following the last new data record.

5.14 SCRATCHING UNWANTED FILES

After the disk has been in use for a while, you may find that a file has outlived its usefulness. Perhaps a program is now hopelessly inefficient and must be replaced, or a data file contains information which is no longer accurate or appropriate. In either case, you may want to be sure that the file cannot accidentally be accessed (this is true especially in the case of a data file whose data is no longer accurate), and you may want to store a new file in the space currently occupied by the unwanted file. You can use the SCRATCH statement (not to be confused with 'SCRATCH DISK') to accomplish both of these tasks.

The SCRATCH statement sets the status of the named file to a scratched condition. A scratched file is not physically removed from the disk. The file's name and location remain listed in the Catalog Index, but the file is flagged as a scratched file. A scratched file has two significant characteristics:

1. A scratched file cannot be accessed by a DATALOAD DC OPEN or LOAD DC statement. That is, no programs or data can be saved in or loaded from a scratched file.

2. A scratched file can, however, be renamed and reopened with a DATASAVE DC OPEN statement or SAVE DC command. In this case, a new file is created in the space previously occupied by the scratched file. (See Chapter 7, Section 7.5.)

Example 5-25: Scratching Unwanted Files

```
750 SCRATCH F "PROG 1", "TEST 1"
```

Statement 750 sets the status of the program file PROG 1 and the data file TEST 1 to a scratched condition; PROG 1 cannot be loaded into memory with a LOAD DC statement, and TEST 1 cannot be opened to load or save data with a DATALOAD DC OPEN statement. New files can be stored in the space occupied by PROG 1 and TEST 1, however. (Refer to Chapter 7 for a discussion of how to reuse the space occupied by scratched files.)

If a LIST DC F statement is executed following statement 750 in Example 5-25 above, the Catalog Index listing looks like this:

```
                        FIXED CATALOG
                    INDEX SECTORS = 00100
                    END CAT. AREA = 01000
                    CURRENT END   = 00269

                    NAME      TYPE  START  END    USED
These files         DATFIL 1   D    00100  00199  00002
are                 TEST 1     SD   00200  00249  00001
scratched           PROG 1     SP   00250  00269  00020
```

Figure 5-11. The Catalog Index Showing Scratched Files

Notice that under "TYPE", PROG 1 reads "SP" and TEST 1 reads "SD". The "S" in this case signifies that each file has been scratched. The renaming and reuse of scratched files is discussed in Chapter 7.

5.15 MOVING THE CATALOG FROM ONE PLATTER TO ANOTHER

Catalog procedures provide a means of copying the contents of the catalog (Catalog Index and Catalog Area) from one disk platter onto another. The MOVE statement is used for this purpose. The MOVE statement is generally used for two reasons:

1. To make a back-up copy of important cataloged files.

2. To eliminate scratched files from the catalog and compress still-active files into the available space, thus making more efficient use of the Catalog Area.

The MOVE statement copies the entire catalog from one disk platter to the other, removing all scratched files from the Catalog Area, and deleting scratched file names from the Catalog Index. After the scratched files are removed, the still-active files are moved up to fill in the vacated sectors. The Catalog Index is then revised to reflect the files' new positions in the Catalog Area. Prior to copying any files or file maintenance information from the first platter to the second platter, MOVE automatically scratches the second platter, setting up a Catalog Index and Catalog Area identical in size to those which are to be moved. The only requirement for the second platter, therefore, is that it be formatted. The user does not need to open a catalog on the second platter with a SCRATCH DISK statement prior to executing the MOVE, since this task is performed automatically by MOVE itself.

Example 5-26: Copying the Catalog from One Disk Platter to the Other

```
450 MOVE FR
```

Statement 450 copies the entire catalog from the 'F' platter to the 'R' platter, squeezing out all scratched files. If the 'RF' parameter is specified instead of 'FR', the copy takes place from the 'R' disk platter to the 'F' disk platter.

After the catalog has been moved from one disk platter to the other, it is good policy perform a test which ensures that all information has been copied accurately. The VERIFY statement can be used to perform such a test. In the VERIFY statement, you must tell the system which platter contains the catalog ('F' or 'R'), as well as the starting and ending sector addresses of the entire catalog. The starting sector of the catalog is always sector 0, since that is the first sector on each platter. The ending sector address varies from one catalog to the next (it was initially specified when the catalog was created with the 'END' parameter in a SCRATCH DISK statement). The ending sector address can be obtained by executing a LIST DC statement for the appropriate platter. The first three items displayed (or printed) by LIST DC are INDEX SECTORS, END CAT. AREA, and CURRENT END. The sector address shown opposite END CAT. AREA is the ending sector address of the catalog. The starting and ending sector addresses in the VERIFY statement must be separated by a comma, and enclosed in parentheses. All sectors between and including the specified sectors are checked by the VERIFY statement.

Example 5-27: Checking the Validity of Files after a Move

```
450 MOVE FR
460 VERIFY R (0,2399)
```

Statement 450 copies all catalog information from the 'F' disk platter to the 'R' disk platter. Statement 460 checks the 'R' disk platter to ensure that all information has been copied correctly. Sectors 0 through 2399 are verified (2399 is the ending sector address of the catalog).

If the test performed by VERIFY turns up no errors, the system returns the CRT cursor and colon to the screen, indicating that the information has been copied accurately. If one or more errors are discovered, the system returns an error message indicating which sector(s) did not copy properly, for example:

```
ERROR IN SECTOR 2027
```

If an error is indicated following a MOVE operation, repeat the MOVE and VERIFY operations. Repeated errors may indicate a faulty platter. Replace the platter and repeat the process; call your Wang Service Representative if the error persists.

> NOTE TO OWNERS OF THE
> MODELS 2270-1 AND 2270-3:
>
> On the Model 2270-1, the MOVE statement is illegal. It is not possible to MOVE the catalog from a disk platter in one Model 2270-1 onto a disk platter in another disk unit.
>
> On the Model 2270-3, it is illegal to attempt a MOVE operation from drive #3 to drives #1 or #2, and vice versa. In order to MOVE the catalog to or from a diskette in drive #3, the diskette must be physically removed from drive #3 and inserted in drive #1 or drive #2.

VERIFY can be used at any time to check the validity of data stored anywhere on the disk. It need not be used exclusively in conjunction with a MOVE operation. It is often wise, for example, to verify existing data on a platter before the platter is used. Many programmers verify important platters regularly at the beginning of daily operation. The CRC and LRC checks performed by VERIFY provide an extra measure of protection against the accidental use of invalid data in important applications.

> WARNING:
>
> It is important that backup copies of important disk-based files be created regularly. Like other storage media, disk platters can be worn out with excessive use, and they are, of course, subject to accidental damage or destruction. To avoid the necessity of recreating your data base following such a potential disaster, you should always maintain one or more backup platters containing duplicates of all important files. Cataloged files can be copied to a backup platter with the MOVE statement. The Model 2270-1, which does not have a MOVE capability, should be backed up on tape cassette.

Chapter 6
Disk Device Selection and Multiple Data Files

6.1 INTRODUCTION

Chapter 5 introduced the most basic catalog procedures, including saving and loading programs and data files, skipping over records within a data file, scratching unwanted files, and moving the contents of the catalog from one platter to the other. In the interests of simplicity and clarity of exposition, however, a number of important but complex disk operations were omitted from Chapter 5. Chapters 6 and 7 are therefore designed to expand and elaborate upon the discussion of catalog procedures begun in Chapter 5. Probably the most significant omission in that discussion was an explanation of how it is possible to keep more than one data file open on a disk at the same time. This subject is especially important because so many data processing problems involve the transfer of data from one file to another, or the storing of data in or reading of data from several different files in the course of processing transactions. Such operations would be time consuming in the extreme if each file had to be reopened every time a record was to be written into it or read from it. Chapter 6 discusses the procedures for maintaining multiple open files on disk simultaneously. The related questions of how the disk is addressed, and how multiple disk units can be operated by a single system, also are examined in this chapter.

6.2 DISK DEVICE SELECTION

Chapter 5 presented you with what was, essentially, a "recipe" for using the disk. You were told that by executing a particular statement which included particular parameters, you could elicit a particular response from the system. The system itself remained a black box, however, whose internal workings were only vaguely hinted at. Although such an approach was appropriate for the purposes of Chapter 5, it cannot safely be followed in the present chapter. Some understanding of the internal operations of the system, particularly those which relate to management of the disk, is a necessary prelude to any discussion of how the system maintains open data files. The first topic to be considered is the mechanism by which the system is able to identify the disk unit and the individual platters within it.

Whenever a disk statement or command is executed, the system has immediate need for at least two items of information: the disk platter which is to be accessed, and the disk unit which contains that platter. The first item is supplied by specifying the 'F' or 'R' parameter in the statement itself. Because several disk units can be attached to the same system, however, the system must also have some way of identifying the disk which contains the specified platter. A three-digit device address is assigned to the disk unit as a means of identifying it.

For certain disk statements and commands, the disk device address can, like the disk platter parameter ('F' or 'R'), be specified directly in the statement or command itself. For example, the statement

```
10 LOAD DC F /350, "PROG 1"
```

causes the system to access the disk unit with device address 350. On the Model 2270-3, this statement accesses Platter #3. In general, however, it is not necessary to specify the device address in a statement or command, since if no address is specified, the system automatically uses the default disk address, 310. The default address is stored by the system in a special section of system memory called the Device Table. Whenever a disk statement or command is executed, the system's first operation is to check the Device Table for a disk device address (unless, of course, the address has been specified in the statement or command itself).

The Device Table

The Device Table in memory consists of seven rows, or "slots", each of which is identified by a unique file number from #0 to #6 (see Figure 6-1 below). The default device address (310) is stored in the Disk Device Address location in the slot opposite #0. For this reason, #0 is referred to as the "default file number," and the slot associated with #0 is called the "default slot."

	FILE NUMBER	DISK DEVICE ADDRESS	STARTING SECTOR ADDRESS	ENDING SECTOR ADDRESS	CURRENT SECTOR ADDRESS
default slot →	#0	310	00000	00000	00000
	#1	000	00000	00000	00000
	#2	000	00000	00000	00000
	#3	000	00000	00000	00000
	#4	000	00000	00000	00000
	#5	000	00000	00000	00000
	#6	000	00000	00000	00000

Figure 6-1. The Device Table in Memory

As you can see, however, each of the remaining six slots (#1 - #6) also has a location for a disk device address (although this location is currently filled with zeroes). Each slot also has locations for three other items of information: a Starting Sector Address, an Ending Sector Address, and a Current Sector Address. The sector address parameters, used by the system to maintain open data files on disk, are discussed in the following section.

The default device address (310) is always stored next to the default file number (#0) by the system itself. Even after the system is Master Initialized (that is, the main power switch is turned OFF and then ON , thus clearing out all of memory), the system automatically returns address 310 to its location opposite #0 in the Device Table (Figure 6-1).

For this reason, it is always possible to execute a disk statement or command without specifying a device address of 310. When, for example, a statement such as

```
10 LOAD DC F "PROG 1"
```

is executed, the system automatically goes to the Device Table and checks for the default address opposite #0.

It is also possible, however, to store a device address in the Disk Device Address location opposite any one of the other file numbers (#1 - #6) in the Device Table. In this case, the device address must be explicitly stored in the table with a SELECT statement.

Example 6-1: Storing Disk Device Addresses in the Device Table

```
50 SELECT #3 310, #5 310
```

Statement 50 instructs the system to store disk device address 310 opposite file numbers #3 and #5 in the Device Table. Following the execution of statement 50, the Device Table looks like this:

	FILE NUMBER	DISK DEVICE ADDRESS	START	END	CURRENT
Default Slot →	#0	310	00000	00000	00000
	#1	000	00000	00000	00000
These slots now available to open new files	#2	000	00000	00000	00000
→	#3	310	00000	00000	00000
	#4	000	00000	00000	00000
→	#5	310	00000	00000	00000
	#6	000	00000	00000	00000

Figure 6-2. The Device Table with Disk Device Addresses Stored Opposite File Numbers #3 and #5

Notice that device address 310 is now stored in the Disk Device Address location opposite file numbers #3 and #5, as well as in the default slot (opposite #0). The file numbers #3 and #5 can now be used in a disk statement or command to reference device address 310 indirectly. For example, if a statement such as

```
60 LOAD DC F #3, "PROG 2"
```

is now executed, the system immediately checks the Device Table for a device address opposite #3. Upon finding address 310, it proceeds to the disk unit and accesses the 'F' platter. If no address were stored opposite #3, or if the address of a device other than the disk (say, a tape cassette drive) were stored there, the system will signal an error when the disk statement is executed.

In summary, then, it is possible to specify a disk device address in two ways: directly (by including the address itself explicitly in the statement), or indirectly (by referencing a file number associated with the appropriate address). Therefore, a statement of the form

```
10 LOAD DC F /310, "PROG 2"
```

is equivalent to the pair of statements

```
10 SELECT #3 310
20 LOAD DC F #3, "PROG 2"
```

Note, however, that the data file manipulation statements (DATASAVE DC OPEN, DATASAVE DC, DATALOAD DC, etc.) do not permit the direct specification of a device address within the statement. In these statements, therefore, the device address must be referenced indirectly via a file number. This restriction is important because file numbers play a most critical role in the manipulation of cataloged data files.

Use of File Numbers in Accessing the #3 Drive (Model 2270-3 Only)

The #3 drive in the Model 2270-3 has a special device address, 350. If this address is stored in a slot opposite one of the file numbers #1 - #6 in the Device Table, subsequent reference to the associated file number will cause the system to access drive #3. For example, the statement

```
50 SELECT #2 350
```

causes device address 350 to be stored opposite #2 in the Device Table. A statement which references file number #2, such as

```
60 LOAD DC F #2, "PROG 1"
```

will now indirectly reference address 350, and access drive #3 to load in PROG 1 from the diskette mounted in that drive.

Why Use The Device Table?

It may appear somewhat inefficient to use a section of memory and a special statement to store device addresses when the address can be supplied in the statement or command itself or when, as in the normal case, no address need be supplied at all. If the Device Table were used exclusively to store device addresses, there would hardly be justification for belaboring the reader with an explanation of its purpose and operation. In fact, however, the Device Table serves a second and far more important function in connection with disk operations. The slots in the Device Table are utilized by the system to hold critical sector address information on currently open data files. Without the Device Table, therefore, it would not be possible to maintain multiple open files on the disk.

> NOTE:
>
> The Device Table slots #1 - #6 are used to store tape file information as well as disk file information. A statement of the form SELECT #1 10A, for example, stores the tape cassette address 10A opposite file number #1 in the Device Table. If you are using disk and tape in conjunction, therefore, be sure to use different file numbers for your tape and disk files. Note, however, that the default slot (opposite #0) is reserved for disk use exclusively; no tape address can be stored in the default slot. The default tape address (assigned in a SELECT TAPE statement) is stored in another portion of memory outside the Device Table.

6.3 MAINTAINING MULTIPLE OPEN DATA FILES ON DISK

The concept of an "open" data file was introduced in Chapter 5 with little exposition. It was pointed out simply that DATASAVE DC OPEN and DATALOAD DC OPEN are used to "open" and "reopen" a data file on disk; the actual procedures followed by the system in opening or reopening a file were left as undefined and faintly magical internal operations.

In fact, of course, there is nothing magical about these operations at all. The system follows a specific and clearly defined procedure in opening a data file. To understand this procedure, however, you should first consider the kinds of information the system requires in order to be able to access a file. Such information includes:

1. The disk platter and disk unit on which the data file is (or is to be) stored.

2. The starting sector address of the file.

3. The ending sector address of the file.

4. The current sector address of the file (i.e., where the system is currently positioned in the file).

Although some of this information (specifically, items #2 and #3) can be found in the Catalog Index, it is efficient for the system to have all of it at hand in one place. As you may already have suspected, that "one place" is the Device Table. The Device Table provides a convenient location in memory for the temporary storage of all information required by the system to access and maintain a cataloged data file. Such information is automatically copied from the Catalog Index on disk into the Device Table whenever a data file is opened initially (with DATASAVE DC OPEN) or reopened (with DATALOAD DC OPEN). In either case, the system first checks the default slot (or one of the other slots, #1 - #6, of a file number has been specified in the statement) for a valid disk address. If the slot contains no address, or an invalid address (for example, a tape address), an error is signalled and execution halts. If a valid address is found, the system proceeds to access the appropriate platter ('F' or 'R') in the specified disk unit.

When an existing file is reopened with a DATALOAD DC OPEN statement, the system merely copies the file's starting and ending sector addresses from the Catalog Index into the default slot (or into one of the other slots, if a file number is used) in the Device Table. The file's current sector address initially is set equal to the starting sector address. When a file is newly opened on disk with DATASAVE DC OPEN, the system first reserves space on the designated platter, and enters the file's name and sector parameters in the Catalog Index. Once this is done, the parameters are copied to a slot in the Device Table. Suppose, for example, that file DATFIL 1 is to be opened on the 'F' platter. Statement 10 below might be used:

```
10 DATASAVE DC OPEN F 100, "DATFIL 1"
```

One hundred sectors are reserved for DATFIL 1 on the 'F' platter. Assuming DATFIL 1 is the first file to be opened on this platter, and assuming that the Catalog Index occupies sectors 0 - 23, the Catalog Index entry for DATFIL 1 looks like this:

NAME	TYPE	START	END	USED
DATFIL 1	D	00024	00123	00001

Once the Catalog Index has been appropriately updated, the sector address parameters for DATFIL 1 are immediately written to the default slot in the Device Table, which therefore looks like this:

	FILE NUMBER	DISK DEVICE ADDRESS	START	END	CURRENT
Default Slot ➤ (DATFIL 1)	#0	310	00024	00123	00024
	#1	000	00000	00000	00000
	#2	000	00000	00000	00000
	#3	000	00000	00000	00000
	#4	000	00000	00000	00000
	#5	000	00000	00000	00000
	#6	000	00000	00000	00000

Figure 6-3. The Device Table with One File Open (DATFIL 1)

The parameters stored opposite #0 are those of DATFIL 1. (Note that the current address of DATFIL 1 is equal to the starting address at this point.) DATFIL 1 is now officially "open", and any DATASAVE DC or DATALOAD DC statement automatically accesses it.

Suppose, however, that a second file is opened:

```
20 DATASAVE DC OPEN F 250, "DATFIL 2"
```

Execution of statement 20 causes the system to run through the same procedure followed in opening DATFIL 1, with the result that DATFIL 1's parameters opposite #0 in the Device Table are replaced by those of DATFIL 2. The Device Table looks like this following execution of statement 20:

	FILE NUMBER	DISK DEVICE ADDRESS	START	END	CURRENT
Default Slot → (DATFIL 2)	#0	310	00124	00373	00124
	#1	000	00000	00000	00000
	#2	000	00000	00000	00000
	#3	000	00000	00000	00000
	#4	000	00000	00000	00000
	#5	000	00000	00000	00000
	#6	000	00000	00000	00000

Figure 6-4. The Device Table with One File Open (DATFIL 2)

DATFIL 2 becomes the currently open file on disk, and any DATASAVE DC or DATALOAD DC statement now accesses it instead of DATFIL 1. The question then arises: if every new file erases information on the previous file from the default slot, how is it possible to have more than one file open at once? The answer to this question is somewhat obvious: different slots in the Device Table can be used to open different data files. Since there are seven slots in the Device Table, a total of seven files can be open at the same time.

You have already seen that the first thing the system does when a disk statement is executed is to check the Device Table for a disk device address. In the two examples just cited, only the default slot (opposite #0) was used for file information. As you know, the system itself automatically keeps the system default address (310) in that slot. Before any of the other slots can be used to open new files, however, the disk device address must be stored in the slot with a SELECT statement, such as the one illustrated in Example 6-1:

```
50 SELECT #3 310, #5 310
```

As you have already seen, this statement instructs the system to store disk device address 310 in the Device Table opposite #3 and #5. The Device Table now looks like this:

	FILE NUMBER	DISK DEVICE ADDRESS	START	END	CURRENT
Default Slot (DATFIL 2) →	#0	310	00124	00373	00124
	#1	000	00000	00000	00000
	#2	000	00000	00000	00000
These slots now available to open new files →	#3	310	00000	00000	00000
	#4	000	00000	00000	00000
→	#5	310	00000	00000	00000
	#6	000	00000	00000	00000

Figure 6-5. The Device Table with Disk Device Addresses Stored Opposite File Numbers #3 and #5, and One Open File (DATFIL 2)

The slots opposite #3 and #5 can now be used, in addition to the default slot, to store the sector address parameters of open files. To use one of these slots, it is necessary only to specify its file number in a DATASAVE DC OPEN or DATALOAD DC OPEN statement. Example 6-2 below uses file #3 to open a second data file on the disk.

Example 6-2: Opening a New Data File with a File Number

```
150 DATASAVE DC OPEN F #3, 50, "DATFIL 3"
```

Statement 150 causes the system to check the slot opposite #3 for a device address. Upon finding address 310, the system goes to the disk unit and accesses the 'F' platter. Fifty sectors are reserved for DATFIL 3, and the file's name and location are entered in the Catalog Index. The file's sector address parameters (starting, ending, and current) are then written in the slot opposite #3 in the Device Table:

	FILE NUMBER	DISK DEVICE ADDRESS	START	END	CURRENT
DATFIL 2	#0	310	00124	00373	00124
	#1	000	00000	00000	00000
	#2	000	00000	00000	00000
DATFIL 3	#3	310	00374	00423	00374
	#4	000	00000	00000	00000
	#5	310	00000	00000	00000
	#6	000	00000	00000	00000

Figure 6-6. The Device Table with Two Open Files

Obviously, the system must have some way of distinguishing DATFIL 2 from DATFIL 3 when data is to be stored in or retrieved from each file. Since the file names are not entered in the Device Table, the system can identify each file only by its associated file number. The file number associated with a file must therefore be used in any subsequent disk statement or command which accesses that file. The default file is, of course, automatically accessed if no file number is specified. Thus, the statement

```
160 DATASAVE DC A$()
```

causes array A$() to be stored in DATFIL 2 (since DATFIL 2's parameters are stored opposite #0 in the default slot), while the statement

```
170 DATASAVE DC #3,A$()
```

causes A$() to be saved in DATFIL 3 (since DATFIL 3's parameters are stored opposite #3).

Example 6-3. Referencing an Open File by File Number

```
10 SELECT #5 310
20 DATASAVE DC OPEN F #5, 50, "FIRST"
30 DATASAVE DC #5, A()
40 DATASAVE DC #5, END
```

Statement 10 writes the disk address (310) in the slot opposite #5 in the Device Table. Statement 20 opens FIRST and assigns its parameters to slot #5 in the Device Table. Statement 30 writes data from array A() into FIRST, and statement 40 writes an end-of-file trailer record to FIRST. Notice that both statements reference FIRST by specifying the file number (#5) to which it is assigned in the Device Table. When statements 30 and 40 are executed, the system immediately checks the slot opposite #5 in the Device Table for a disk address. It then accesses the specified disk and begins storing data at the sector specified in the Current Sector Address parameter of slot #5. Following execution of statement 40, the Device Table looks like this:

	FILE NUMBER	DISK DEVICE ADDRESS	START	END	CURRENT
DATFIL 2 →	#0	310	00124	00373	00129
	#1	000	00000	00000	00000
	#2	000	00000	00000	00000
DATFIL 3 →	#3	310	00374	00423	00379
	#4	000	00000	00000	00000
FIRST →	#5	310	00424	00473	00428
	#6	000	00000	00000	00000

Figure 6-7. The Device Table in Memory with Three Open Files

Existing files reopened with DATALOAD DC OPEN also can be assigned file numbers. It is not required that a file be reassigned its original file number every time it is reopened; the parameters of a file are copied anew into the Device Table each time it is reopened, and it may be assigned to any available slot. The file FIRST, opened initially in Example 6-3 above, might subsequently be reopened and assigned a different file number, as illustrated in Example 6-4 below.

Example 6-4: Referencing an Open File by File Number

```
10 SELECT #4 310
20 DATALOAD DC OPEN F #4, "FIRST"
30 DSKIP #4, END
40 DATASAVE DC #4, B()
50 DATASAVE DC #4, END
```

Statement 10 writes disk address 310 in the slot opposite #4 in the Device Table. Statement 20 opens an existing file, FIRST, and assigns its parameters to slot #4 in the Device Table. Statement 30 skips to the current end-of-file trailer record in the file. Statement 40 saves a new record in the file from array B() over the trailer record, and statement 50 writes a new trailer record in the file. Notice that all reference to FIRST in statements 30, 40, and 50 is in terms of the file number (#4) to which it is assigned in the Device Table. Notice also that #4 is not the file number originally assigned to FIRST when it was initially opened in Example 6-3.

Note that it is possible to reopen the same file repeatedly, using a different file number each time. In this manner, every slot Device Table can be filled with the parameters of a single file. The practical advantage of such an arrangement would, however, be questionable in most cases.

<u>Using A Variable To Store The File Number</u>

If it is convenient, a file number may be referenced in a disk statement as the value of a numeric variable. For example, the statements

```
5 SELECT #3 310
10 A = 3
20 DATALOAD DC OPEN F #A, "DATFIL 1"
```

cause the system to reopen DATFIL 1 on the 'F' platter, and store its parameters opposite #3 in the Device Table (since A=3). (Note that the use of numeric variables to reference file numbers is not legal in the SELECT statement itself. Thus, a statement of the form SELECT #A 310 is not permitted. Similarly, the use of variables is illegal in tape cassette statements.)

6.4 <u>THE "CURRENT SECTOR ADDRESS" PARAMETER</u>

In the discussion of skipping over logical records within data files in Chapter 5, as well as in the recent discussion of storing data in a data file, you have seen why it is important, in fact necessary, for the system to know where it is positioned within a file at all times. If the system does not know, for example, that it has just stored a record ending at sector 86 in a currently open file, then it cannot know that the next record must be saved in that file starting at sector 87. In such a case, the system would obviously be incapable of maintaining data files on disk at all.

The system knows where it is positioned in a file by referring to the Current Sector Address of the file. The Current Sector Address is updated every time a record is saved in or loaded from a file, and every time records are skipped or backspaced in a file. The Current Sector Address always indicates the next sequential sector following the most recent access of a file. For example, suppose that a file DATFIL 2 is to be saved on the 'F' disk platter:

```
300 DATASAVE DC OPEN F #1, 500, "DATFIL 2"
```

The Device Table slot for DATFIL 2 now looks like this:

FILE NUMBER	DISK DEVICE ADDRESS	START	END	CURRENT
#1	310	00060	00559	00060

Figure 6-8. Device Table Slot for DATFIL 2

Notice that the Current Sector Address for DATFIL 2 is identical to the Starting Sector Address. This is the case whenever a file is opened or reopened.

Suppose, now, that you store data from an array, A(), into DATFIL 2:

```
DATASAVE DC #1 A()
```

Assuming that the data from A() occupies one sector on disk, the Device Table slot for DATFIL 2 now reads as follows:

FILE NUMBER	DISK DEVICE ADDRESS	START	END	CURRENT
#1	310	00060	00559	00061

↑ Current Address now updated to show that sector 61 is the next available sector.

Figure 6-9. Updated Device Table Slot for DATFIL 2

Notice that the Current Address is now updated to show that sector 61 is the next available sector in the file, since sector 60 (the first sector in the file) has been filled with data.

You might now save three more arrays of data:

```
310 DATASAVE DC #1, B()
320 DATASAVE DC #1, C()
330 DATASAVE DC #1, D()
340 DATASAVE DC #1, END
```

Following execution of these statements (and assuming each array requires one sector on disk), the Device Table looks like this:

FILE NUMBER	DISK DEVICE ADDRESS	START	END	CURRENT
#1	310	00060	00559	00064

Figure 6-10. Updated Device Table Slot for DATFIL 2

Figure 6-10 illustrates a special case of updating the Current Sector Address. A total of five sectors have been recorded in the file with lines 300-340. Those five sectors are 60, 61, 62, 63, and 64 (sector #64 contains the end-of-file record written at line 340). According to the rule set forth above, the Current Sector Address should equal the address of the next sector at this point (sector #65). Instead, it is set to the address of the end-of-file record (64). The creation of an end-of-file record involves an exception to the rule governing updating of the Current Sector Address: following creation of an end-of-file record with DATASAVE DC END, the Current Sector Address is always set to the address of the EOF record, rather than to the address of the next consecutive sector. In this way, a subsequent DATASAVE DC statement will store the next data record over the EOF record, and the danger of leaving an EOF record in the middle of a file when new data records are saved is avoided.

In order to skip back from the current position to the beginning of the file, a DBACKSPACE BEG statement is used:

```
350 DBACKSPACE #1, BEG
```

This statement instructs the system to set the value of the Current Sector Address equal to the value of the Starting Sector Address. Following execution of Statement 350, the Device Table looks like this:

FILE NUMBER	DISK DEVICE ADDRESS	START	END	CURRENT
#1	310	00060	00559	00060

Current Address now set back to address of first sector in file.

Figure 6-11. Updated Device Table Slot for DATFIL 2 Following Execution of a DBACKSPACE BEG Statement

At this point, the first record can be read from DATFIL 2. DSKIP functions in an analogous manner, causing the Current Sector Address to be updated to reflect the new current location in the file following the skip.

6.5 CLOSING A DATA FILE

You should understand more clearly now the precise meanings of the concepts of "opening" and "closing" a data file. A data file is opened (by a DATASAVE DC OPEN or DATALOAD DC open statement) when its parameters are entered in a slot in the Device Table. A data file is closed when its parameters are removed from the Device Table, either by writing over the parameters with another set of parameters, or by zeroing out the parameters. There are four methods of closing a currently open data file:

1. Assigning the file number currently associated with the file to another file.

2. Executing a CLEAR command with no parameters.

3. Master Initializing the system.

4. Executing a DATASAVE DC CLOSE statement.

Each of these four methods is explained in the following paragraphs:

1. Assigning the file number currently associated with the file to another file causes the parameters of the new file to be written over the parameters of the original file, thus closing the original file.

Example 6-5: Closing a Data File by Reassigning Its File Number

```
110 SELECT #1 310
120 DATASAVE DC OPEN F #1, 110 "DATFIL 1"
150 DATASAVE DC OPEN R #1, 600 "DATFIL 2"
```

Statement 110 selects file number #1 to the disk. Statement 120 opens DATFIL 1, reserves 110 sectors for it on the 'F' disk platter, and causes its parameters to be entered in the Device Table in the slot opposite #1. Statement 150 opens a new data file, DATFIL 2, and stores its parameters in slot #1. These parameters overwrite those of DATFIL 1, effectively closing DATFIL 1.

2. Executing a CLEAR command with no parameters causes all of memory to be cleared, including the Device Table. All the information in the Device Table is zeroed out, thereby closing all files.

3. Master Initializing the system (i.e., throwing the main power switch OFF and then ON) also has the effect of clearing out memory, thus closing all files.

4. Executing a DATASAVE DC CLOSE statement causes all sector address parameters for the specified file or files in the Device Table to be zeroed out, thereby closing the file(s). (DATASAVE DC CLOSE should not be confused with DATASAVE DC END. DATASAVE DC END causes an end-of-file trailer record to be written in the specified file.) The disk device address stored in a slot is not zeroed out by DATASAVE DC CLOSE, however.

Example 6-6: Closing A Specified File With A DATASAVE DC CLOSE Statement

```
200 DATASAVE DC CLOSE
210 DATASAVE DC CLOSE #1
```

Statement 200 causes the sector address parameters associated with the default file #0 (since no file number is specified) to be zeroed out, thus closing the file associated with #0. Statement 210 causes the sector address parameters stored in slot #1 to be zeroed out, thus closing the file associated with #1.

Example 6-7: Closing All Currently Open Files with a DATASAVE DC CLOSE Statement

```
300 DATASAVE DC CLOSE ALL
```

Statement 300 causes all sector address parameters in the Device Table to be zeroed out, thus closing all currently open files.

It is generally good practice to close a data file once precessing of the file is complete. In this way, another operator is prevented from accidentally storing data into the file over currently stored data, and destroying the existing data. It is also good policy to write an end-of-file record in the file prior to closing it, since it will then be possible to skip to the end-of-file and continue storing data in the file when it is subsequently reopened.

When a file is closed (by whatever method) its three sector address parameters are removed from the Device Table. When the file is subsequently reopened with a DATALOAD DC OPEN statement, the Current Sector Address is automatically set equal to the Starting Sector Address.

6.6 SKIPPING AND BACKSPACING OVER INDIVIDUAL SECTORS IN A FILE

In Chapter 5, the discussion of DSKIP and DBACKSPACE was confined to the skipping of logical records within a file. It is also possible, however, to skip individual sectors in a file. This method is a much faster way of moving through a file than skipping records, but its value cannot be fully understood until the process of skipping logical records is examined in greater detail.

Remember that a logical record may consist of any number of sectors. The first logical record in a file might, for example, contain three sectors, while the second contains thirteen. The system has no way of knowing in advance how many sectors are in each record; when the system is instructed to skip or backspace over a prescribed number of records, therefore, it must actually read those records from the file and update the Current Sector Address after the specified number of records have been read. Suppose, for example, that the system is currently positioned at the beginning of DATFIL 1, and that DATFIL 1 is associated with file #1 in the Device Table. If you want to skip three records in DATFIL 1, you would execute a DSKIP #1,3 statement. Such a statement causes the system to run through the following set of operations:

1. Check the Current Sector Address in slot #1 in the Device Table to see where it is currently positioned in the file.

2. Access the disk and read three logical records, beginning at the location specified in the Current Sector Address parameter.

3. After reading the third logical record, check the sector address of the last sector in that record.

4. Set the Current Sector Address in slot #1 equal to one greater than the address of the last sector in logical record #3.

At the end of this procedure, the Current Sector Address in slot #1 is equal to the address of the first sequential sector following record #3.

Suppose, now, that you know there are three sectors in each logical record in DATFIL 1. In this case, if you want to skip three logical records, you can simply instruct the system to skip nine sectors. Since the system knows exactly how many sectors are to be skipped, it need not access the disk and read the records themselves; it simply increments the Current Sector Address in Slot #1 by nine. The process of skipping or backspacing through a file is greatly accelerated, since no disk accesses are required.

The 'S' parameter is used in a DSKIP or DBACKSPACE statement to inform the system that it is to skip a specified number of sectors rather than logical records.

Example 6-8: Skipping over a Number of Sectors in a File

```
400 DSKIP #1, 20S
```

Statement 400 instructs the system to increment the Current Address for the file associated with slot #1 in the Device Table by 20. If the old Current Address was equal to X, the new Current Address is equal to X+20. If each logical record consists of five sectors, this statement has the effect of skipping over four logical records.

Example 6-9: Backspacing over a Number of Sectors in a File

```
410 DBACKSPACE #3, 25S
```

Statement 410 instructs the system to decrement the Current Address for the file associated with #3 in the Device Table by 25. If the original Current Address was equal to Y, the new Current Address is equal to Y - 25. If each logical record consists of five sectors, this statement has the effect of backspacing over five logical records.

When the 'S' parameter is used, it is necessary that every logical record in the file consist of the same number of sectors; otherwise, skipping or backspacing over a number of sectors can lead to serious problems. If the number of sectors skipped does not represent a number of whole records, the system may end up somewhere in the middle of a logical record. In such a case, it will automatically skip to the beginning of the next sequential logical record and begin reading at that point.

6.7 THE 'T' PARAMETER

Until now, only two parameters have been discussed in connection with accessing a disk platter, the 'F' parameter and the 'R' parameter. These parameters are "absolute" in the sense that the reference of each parameter is fixed and cannot be changed (that is, the 'F' parameter can never be used to access the 'R' platter, and vice versa).

Such an arrangement lacks flexibility. It is desirable in certain cases to be able to access both the 'F' and 'R' platters with the same disk statement or command; the 'T' parameter provides such a capability. When the 'T' parameter is specified in a disk statement or command instead of 'F' or 'R', it instructs the system to use the disk device address to determine which platter is to be accessed.

For such a technique to be possible, however, it is evident that each disk platter must have its own device address. This is true only in a very limited sense. The disk device address (e.g., "310") is really a conjunction of two distinct elements, a device type and a unit device address. The first digit of the three-digit disk address is the device type; the remaining two digits constitute the unit device address. It is the device type which can be used to designate a particular disk platter.

3	10
↓	↓
Device Type	Unit Device Address

In all of the examples to this point, a single device type, "3" (e.g., 310, 320, etc.), has been used consistently. However, a second device type, "B" (e.g., B10, B20, etc.), is also permissable in a disk device address. When used in conjunction with the 'T' parameter, a device type of "3" designates the 'F' disk platter, while a device type of "B" designates the 'R' platter:

3	10	B	10
↓	↓	↓	↓
Designates the 'F' platter when 'T' parameter is used	Identifies the disk drive	Designates the 'R' platter when 'T' parameter is used	Identifies the disk drive

For example, the statement

```
10 LOAD DC T/310, "PROG 1"
```

causes the system to access the 'F' platter, while the statement

```
20 LOAD DC T/B10, "PROG 2"
```

causes the system to access the 'R' platter. It should be emphasized that a disk device address is never used by itself to access a disk platter; it is always necessary to specify one of the parameters 'F', 'R', or 'T' in statements which reference a platter parameter.

No mention was made of the "B" device type in previous examples because the device type "B" itself is significant only when the 'T' parameter is used. The 'F' or 'R' parameter, when specified, always overrides the device type. Thus, for example, the command

```
LOAD DC F/310, "PROG 1"
```

accesses the 'F' platter; and so too does the command

```
LOAD DC F/B10, "PROG 1"
```

In this case, the device type ("B") is overridden by the specified platter parameter ('F'), and has no meaning to the system.

SPECIAL NOTE TO MODEL 2270-1 OWNERS:

An address of B10 is illegal when used in conjunction with the 'T' parameter, since the Model 2270-1 contains only a single disk drive, which must be referenced as an 'F' platter (i.e., with address 310). Otherwise, an error results.

SPECIAL NOTE TO MODEL 2270-3 OWNERS:

When the 'T' parameter is used with the Model 2270-3, platter selection is determined by both the device type and the device address. If the device address is the primary address (i.e., X10, X20, X30, X40, etc.) then a device type of "3" designates Platter #1 (i.e., 310, 320, 330, 340, etc.), while a device type of "B" designates Platter #2 (i.e., B10, B20, B30, B40, etc.). If, on the other hand, the device address is not the primary address (e.g., 350, 360, 370, etc.), then Platter #3 is accessed. Note that the addresses B50, B60, and B70 are not legal in this case, since the diskette in drive #3 must be addressed as an 'F' platter.

The 'T' parameter provides maximum flexibility when used in a statement which references a file number specified as the value of a variable. In such a case, the system obtains the specified file number from the value of the variable, then inspects the device type in the device address stored opposite the specified file number to determine which platter to access. This arrangement makes it possible to use the same disk statement to access all platters in the disk unit simply by changing the value of the file number variable.

Example 6-1: Accessing more than One Disk Platter with the 'T' Parameter

```
10  SELECT #3 310, #4, B10
:
:
100 GOSUB'20 (3,"DATFIL 1")
:
290 DEFFN'20 (A,B$)
300 DATALOAD DC OPEN T #A, B$
310 RETURN
```

Here statement 10 stores disk device addresses 310 and B10 in slots #3 and #4 of the Device Table, respectively. Subsequently, the 'GOSUB' statement at line 100 passes the values '3' and "DATFIL 1" to the marked subroutine at line 290. Because address 310 is assigned to file number #3, the DATALOAD DC OPEN statement at line 300 reopens DATFIL 1 on the 'F' platter. The same subroutine could be used to open a different file on a different platter if called from another point in the program and passed a different set of values:

```
200 GOSUB'20 (4,"TEST 2")
:
:
290 DEFFN'20(A,B$)
300 DATALOAD DC OPEN T #A, B$
310 RETURN
```

In this case, data file TEST 2, located on the 'R' platter, is reopened by the subroutine.

The 'T' parameter provides the general capability to write disk statements which can access any disk platter. This feature may prove particularly useful for file update operations, where two versions of the same file may reside on different platters. Users of the Models 2230 and 2260 should find the 'T' parameter helpful in debugging file maintenance programs written for the Fixed Platter by testing them with dummy files stored on the Removable Platter (thus avoiding the danger of erasing legitimate data on the Fixed Platter). Finally, Model 2270-3 owners will find the 'T' parameter helpful because it provides them with a single parameter which can be used to access all three disk platters. For example, a program can be designed which makes a specific platter (and disk unit) selectable by the operator when the program is run:

Example 6-11: Use of the 'T' Parameter to Access a User-Selectable Disk Platter

```
10 INPUT "ENTER PLATTER-NUMBER (1,2, OR 3)", A
20 INPUT "ENTER PROGRAM NAME", N$
30 ON A GOTO 30, 50, 60
40 SELECT #1 310:GOTO 70
50 SELECT #1 B10:GOTO 70
60 SELECT #1 350
70 LOAD DC T #1, N$
```

<u>Changing the Default Address</u>

The system default disk address, 310, is a system-defined parameter which cannot be permanently changed by the programmer. Following Master Initialization, the system automatically returns address 310 to the default slot. It is, however, possible to change the value of the default address temporarily with a SELECT DISK statement. For example, the statement

```
50 SELECT DISK B10
```

causes disk address B10 to be recorded in slot #0 in the Device Table:

FILE NUMBER	DISK DEVICE ADDRESS	START	END	CURRENT
#0	B10	00000	00000	00000

Figure 6-12. The Device Table Following Execution of a SELECT DISK B10 Statement

Once statement 50 is executed, any disk statement or command containing the 'T' parameter with no file number specified causes the system to access the 'R' platter (based on a device type of "B"), rather than the 'F' platter (as would be the case with the system default address, 310). Note that the default address cannot be changed with a statement of the form SELECT #0 B10. This statement is illegal.

Example 6-12: Using the 'T' Parameter with a New Default Address

```
10 SELECT DISK B10
20 DATASAVE DC OPEN T 100, "DATFIL 1"
```

Statement 10 changes the default address (stored in slot #0) from 310 to B10. Statement 20 causes the system to check the default address in the default slot (since no file number is specified) and, since the 'T' parameter is used, to inspect the device type in the address. In this case, the device type is B (B10); the 'R' platter is therefore used to open DATFIL 1.

After it has been changed, the default address can be reset to 310 by:

1. Entering a SELECT DISK 310 statement, or
2. Master Initializing the system (i.e., throwing the main power switch OFF and then ON).

6.8 MULTIPLE DISK UNITS

If only one disk unit is attached to the system, the problem of multiple disk addresses is not a concern, since you will deal exclusively with the primary disk addresses 310 and B10 (and, on the Model 2270-3, 350). Many installations, however, drive two or more disks with a single CPU. (A typical configuration includes one large fixed/removable disk drive for the data base, and a smaller diskette drive for software.) In multiple-disk configurations, the system distinguishes different disk units by means of the last two digits in their device address, called the "unit device address":

```
     3/B              10
      ↓               ↓
    Device        Unit Device Address
    Type          (Identifies disk drive
                  number one)
```

Models 2230, 2260, 2270-1, 2270-2

On the Models 2230, 2260, 2270-1, and 2270-2, the unit device address of each successive disk unit on the same system is computed by adding HEX(10) to the disk device address of the primary disk. The addresses of successive disks are listed in Table 6-1.

Table 6-1. Disk Addresses for Models 2230, 2260, 2270-1, 2270-2

Disk Unit #1 (Primary)	310 or B10
Disk Unit #2	320 or B20
Disk Unit #3	330 or B30
Disk Unit #4	340 or B40
Disk Unit #5	350 or B50
Disk Unit #6	360 or B60
Disk Unit #7	370 or B70
Disk Unit #8	380 or B80

Model 2270-3

On the Model 2270-3, the addressing scheme is somewhat different. The unit device address of drives #1 and #2 in a second and third disk unit on the same system is computed by adding HEX(10) to the primary disk address (310); the addresses for four or more units are computed by adding HEX(01) to the previous address. Similarly, the address of drive #3 is computed for the first three units by adding HEX(10) to the primary address (350), and by adding HEX(01) for each unit beyond the third. The addresses for successive units are listed in Table 6-2.

Table 6-2. Disk Addresses for Model 2243

	Drives #1 and #2	Drive #3
Disk Unit #1 (Primary)	310 or B10	350
Disk Unit #2	320 or B20	360
Disk Unit #3	330 or B30	370
Disk Unit #4	331 or B31	371
Disk Unit #5	332 or B32	372
Disk Unit #6	333 or B33	373
Disk Unit #7	334 or B34	374
Disk Unit #8	335 or B35	375

NOTE:

The device addresses for disk units are set at the factory, or by a Wang Service Representative. The address of each disk unit should be marked on the disk controller board for that unit. If you have questions about addressing multiple disks in a system, contact your Service Representative.

Accessing Multiple Disk Units

The techniques for accessing a disk platter with catalog procedures are the same for additional disk units on a system as for the primary unit. A platter can be accessed in four ways:

1. Specifying the disk device address in a disk statement or command, e.g.:

```
100 LOAD DC R /330, "PROG 1"
```

   Statement 100 loads PROG 1 from the 'R' platter in disk unit number three. Note that there are a number of catalog statements in which the device address cannot be directly specified.

2. By selecting a disk address as the default disk address, and referencing the default address, e.g.:

```
100 SELECT DISK 340
120 DATASAVE DC OPEN F 100, "DATFIL 1"
```

   Statement 110 changes the default address from 310 to 340, and statement 120 opens DATFIL 1 on the 'F' platter of disk unit number four. Note that the default address reverts to the system default address, 310, when the system is Master Initialized.

3. By assigning the disk address to a file number in the Device Table, and referencing the address indirectly, via the file number, e.g.:

```
100 SELECT #3 320
110 DATASAVE DC OPEN F #3, 100, "DATFIL 1"
```

   Statement 100 stores disk address 320 in the #3 slot in the Device Table, and statement 100 opens DATFIL 1 on the 'F' platter of disk unit number two. In this case, the disk unit is determined from the disk address, while the disk platter is specified in the DATASAVE DC OPEN statement ('F'). Alternatively, both the disk unit and the disk platter can be determined from the device address:

```
100 SELECT #3 320
110 DATASAVE DC OPEN T #3, 100, "DATFIL 1"
```

   In this case, both the disk unit (number two) and the disk platter ('F' platter) are determined by inspection of the device address.

4. By assigning the device address to a file number in the Device Table, and referencing the file number indirectly (via a variable), e.g.:

```
100 SELECT #3 B20
105 A = 3: B$ = "DATFIL 1"
110 DATASAVE DC OPEN T #A, 100, B$
```

   Since A = 3, and address B20 is stored in slot #3 in the Device Table, the file DATFIL 1 is opened on the 'R' platter of disk unit number two.

Chapter 7
Efficient Use of the Disk

7.1 INTRODUCTION

This chapter discusses several techniques designed to help you make more efficient use of your disk, both in terms of optimizing the use of disk storage space and speeding up processing time for disk files. The following topics are covered in the chapter:

1. Reserving additional space in program files for program expansion.

2. Establishing temporary work files on the disk.

3. Renaming and reusing scratched files.

4. Efficient use of disk storage space within records.

5. The LIMITS Statement.

7.2 PROGRAM FILES REVISITED

The discussion of saving program files in Chapter 5 restricted itself to cases in which the system used exactly enough disk space to hold the recorded program lines. In many cases, however, it is advantageous to reserve additional sectors within a program file for future expansion of the program. If such additional space is reserved at the outset, the program can subsequently be expanded and written back into its original location in the catalog (the reuse of scratched program file locations is described in Section 7.5). If extra space is not reserved when the file is initially created, the expanded program will not fit into its original space, and must be saved at a new location in the Catalog Area. In this case, the space occupied by the old program is wasted, unless a new file can be found to occupy it. The SAVE DC command provides a means of reserving extra sectors in a program file when the program is initially stored on disk.

In order to reserve extra sectors in a program file, the number of additional sectors to be reserved must be enclosed in parentheses and listed in the SAVE DC command immediately before the program name. The system then automatically adds the specified number of additional sectors at the end of the program file when the program is recorded on disk.

Example 7-1: Reserving Additional Sectors in a Program File

SAVE DC F (10) "PROG 1"

This command instructs the system to record all program lines currently in memory on the 'F' disk platter, and name the file "PROG 1". In addition to the sectors needed to store the program itself, 10 sectors are reserved for future additions to the program (see Figure 7-1).

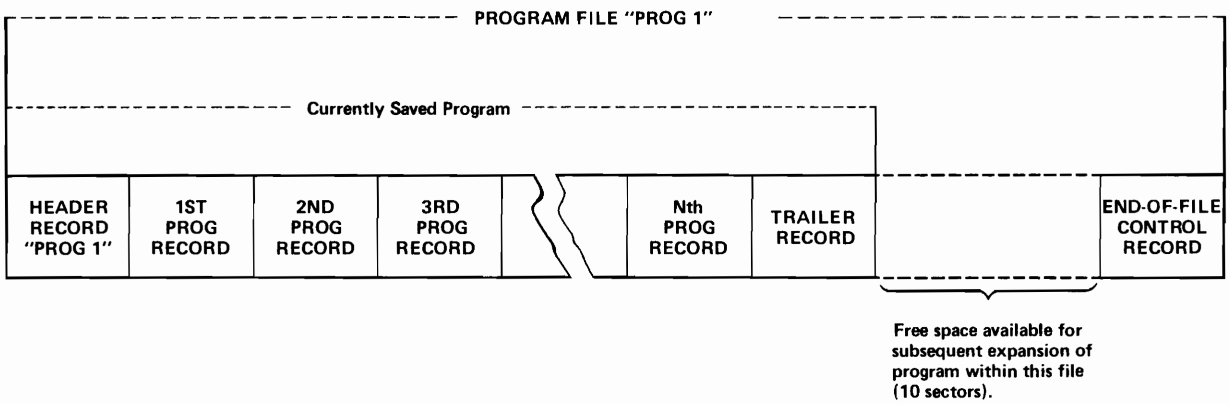


Figure 7-1. The Program File PROG 1 with Ten Extra Sectors Reserved

7.3 ESTABLISHING TEMPORARY WORK FILES ON DISK

Temporary work files can be useful in a variety of data processing operations. A "temporary" work file is opened with a DATASAVE DC OPEN statement, but unlike a regular cataloged file, it is not listed in the Catalog Index, and not stored in the Catalog Area on disk. Its parameters are, however, entered in the Device Table in memory. Temporary files may be used as transaction files, to contain transactions saved over a period of time and processed as a batch, or as scratch files, in which the results of intermediate calculations are stored prior to final processing. They may, in short, be used as a storage area for any type of transient data which is not sufficiently final to warrant storage in a permanent file.

Because they are not cataloged, temporary files must be stored outside the Catalog Area on disk. The end of the Catalog Area (that is, the address of the last sector reserved for the Catalog Area) is specified in the SCRATCH DISK statement when the catalog is established. If temporary files are to be used, the catalog may not occupy the entire platter; a number of sectors must be left outside the Catalog Area for the temporary files. For example, the Model 2230-1 Disk Drive has 2400 sectors on each platter. Since sector numbering starts at zero rather than one, the highest sector address on the Model 2230-1 is 2399. If a number of sectors (say, 100) are to be left available for temporary files, the address of the last sector in the Catalog Area must be 2399 minus 100, or 2299:

```
100 SCRATCH DISK F LS=30, END=2299
```

Sectors 2300 through 2399 are left outside the Catalog Area, and may be used for temporary files.

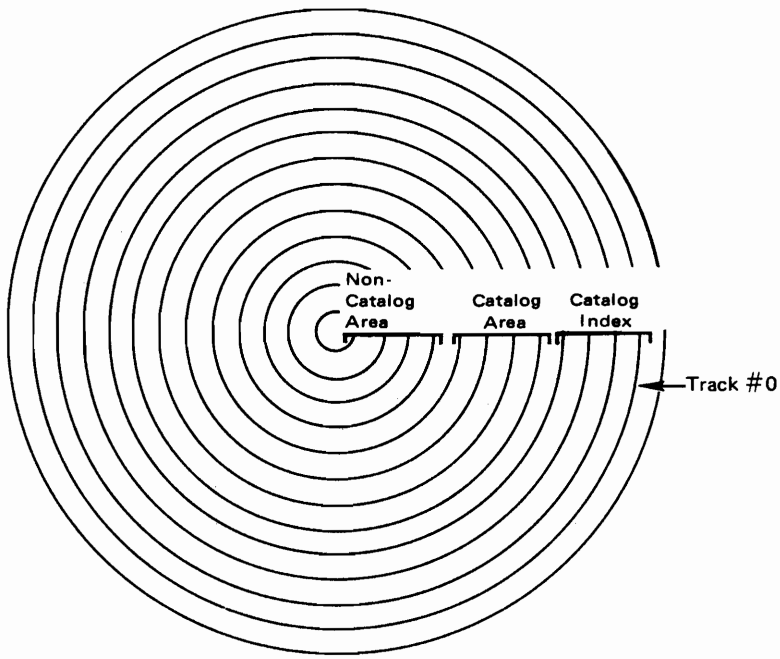


Figure 7-2. Layout of the Platter Surface Showing Catalog Index, Catalog Area, and Non-Catalog Area (Used for Storage of Temporary Files).

Temporary files are opened and accessed with the same BASIC statements used to open and access cataloged files. However, temporary files cannot be named, nor can they be accessed by name. Instead, the special TEMP parameter, along with the beginning and ending sector addresses of the temporary file, must be specified in the DATASAVE DC OPEN statement when the file is opened initially, and again in the DATALOAD DC OPEN statement when the file is reopened.

Example 7-2: Opening a Temporary Work File on Disk

```
300 DATASAVE DC OPEN R TEMP, 2300, 2399
```

Statement 300 opens a temporary work file on the 'R' disk platter. Sectors 2300 through 2399 are reserved for this temporary file (these sectors must be outside the Catalog

Area). No information on the file is entered in the Catalog Index; however, the temporary file's parameters are entered in the default slot (#0) in the Device Table. Following the execution of statement 300, any DATASAVE DC or DATALOAD DC statement which does not specify a file number (i.e., which references the default slot) will read or write data in the temporary file.

Like cataloged files, temporary files can be assigned file numbers. In this way, more than one temporary file can be open at the same time.

Example 7-3: Opening More Than One Temporary Work File

```
300 SELECT #1 310, #3 310
320 DATASAVE DC OPEN F #1, TEMP, 2300, 2349
330 DATASAVE DC OPEN F #3, TEMP, 2350, 2399
```

Statement 300 stores disk address 310 opposite file numbers #1 and #3 in the Device Table. Statement 320 opens a temporary file on the 'F' platter, reserves sectors 2300 through 2349 for that file, and enters the file parameters in slot #1 of the Device Table. Statement 330 opens a second temporary file on the 'F' platter, occupying sectors 2350-2399, and assigns its parameters to slot #3 in the Device Table. Any reference to #1 or #3 in a DATASAVE DC or DATALOAD DC statement accesses these temporary files.

Data is stored in a temporary file just as it is stored in a cataloged file. As with a cataloged file, a data trailer record should always be written in the file at the completion of a data storage operation. As with cataloged data files, the last sector of a temporary data file is used by the system for control information; at least one more sector than the data actually requires should be reserved for the temporary file.

A temporary file is closed in the same way a cataloged file is closed, and is reopened with a DATALOAD DC OPEN statement. The TEMP parameter and the beginning and ending sector addresses of the file must be specified.

Example 7-4: Reopening a Temporary Work File

```
500 DATALOAD DC OPEN F TEMP, 2350, 2399
```

Statement 500 reopens an existing temporary file beginning at sector 2350 on the 'F' disk platter.

7.4 ALTERING THE CATALOG AREA

The upper limit of the Catalog Area is originally set with the END parameter in a SCRATCH DISK statement when the catalog is created. If more room is needed for temporary files, or if more sectors must be devoted to cataloged files, the size of the Catalog Area can be changed with a MOVE END statement. In this statement, it is necessary to specify only the sector address which is to become the new ending sector address of the Catalog Area. Note that MOVE END alters the size of the Catalog Area only; it does not change the size of the Catalog Index.

Example 7-5: Changing the Size of the Catalog Area

```
100 SCRATCH DISK F LS=30, END=2299
:
:
500 MOVE END F = 2199
```

Statement 100 sets the limit of the Catalog Area at sector 2299. Statement 500 moves the limit back 100 sectors, to sector 2199, thereby allowing 100 additional sectors to be used for temporary files (outside the Catalog Area). The Catalog Area may be expanded as well as constricted, but its upper limit must never exceed the highest sector address available on a disk platter. The size of the Catalog Index cannot be changed with MOVE END.

7.5 RENAMING AND REUSING SCRATCHED FILES

Temporary files offer one good way to make the most efficient use of disk storage space. Another way to get maximum use out of available disk storage area is to reuse the space occupied by scratched files. As you saw in Chapter 5, one way to eliminate scratched files is to execute a MOVE operation, since MOVE automatically deletes scratched files when it copies the catalog to a new platter. In many cases, however, it is easier and more efficient to store a new program or new data file directly into space occupied by a scratched file, without moving the whole catalog to a second platter. This is true particularly in the case of revised programs. New files are recorded in the space occupied by scratched files with the SAVE DC command and DATASAVE DC OPEN statement. The file type of the scratched file (program or data) is irrelevant when opening a new file in its space: a program file may be saved in the space occupied by a scratched data file, and a data file may be saved in the space occupied by a scratched program file. The scratched file name must precede the new file name in the SAVE DC command or DATASAVE DC OPEN statement.

Example 7-6: Saving a Program in Space Occupied by a Scratched File

```
SCRATCH R "PROG 1"
SAVE DC R ("PROG 1") "PROG 2" 200, 500
```

The SCRATCH statement causes program file PROG 1 to be set to a scratched status. SAVE DC then stores lines 200 through 500 in the sectors previously reserved for PROG 1, and names the new program "PROG 2". The new file name ("PROG 2") and location are entered in the Catalog Index. The scratched entry for PROG 1 remains in the Catalog Index, although it no longer appears in a listing of the Index.

Notice that the scratched file name must be enclosed in both quotes and parentheses when it is referenced in a SAVE DC command.

Example 7-7: Opening a Data File in Space Occupied by a Scratched File

```
10 SCRATCH F "DATAFIL 1"
20 DATASAVE DC OPEN F "DATFIL 1", "DATFIL 2"
```

Statement 10 scratches DATFIL 1. Statement 20 assigns the sectors previously reserved for DATFIL 1 to DATFIL 2, and updates the Catalog Index accordingly. DATFIL 2's parameters (previously those of DATFIL 1) are entered in the default slot (#0) in the Device Table. The scratched entry for DATFIL 1 remains in the Catalog Index, although it no longer appears in a listing of the Index.

A program file which has been scratched can be reused as a data file, and vice versa.

Example 7-8: Opening a Data File in Space Occupied by a Scratched Program File

```
10 SCRATCH F "PROG 1"
20 DATASAVE DC OPEN F #1, "PROG 1", "DATFIL 3"
```

Statement 10 scratches PROG 1. Statement 20 assigns the sectors on disk previously reserved for PROG 1 to DATFIL 3, and updates the Catalog Index accordingly. DATFIL 3's parameters (previously those of PROG 1) are entered in slot #1 in the Device Table (the disk device address must previously have been stored opposite #1). The scratched entry for PROG 1 is not removed from the Catalog Index, however, although it no longer appears in the Index listing.

It is entirely possible to rename a scratched file with the same name. This feature is useful for revising program files, since the program can be updated and then resaved into the original location with the same name (assuming, of course, that additional space has been reserved in the original file for expansion of the program).

Example 7-9: Renaming a Scratched Program File with the Same Name

```
SCRATCH R "PROG 1"
SAVE DC R ("PROG 1") "PROG 1"
```

The SCRATCH statement scratches PROG 1. The SAVE DC command subsequently resaves an updated version of the program, assigning it the same name ("PROG 1"), and storing it in the same location as the original PROG 1. If there is not enough space in the file for the new program, an error is signalled. In this case, the scratched entry for PROG 1 is removed from the Catalog Index when the program is saved.

Finally, it is also possible to scratch and rename a data file without disturbing the data in the file, if you simply want to give the file a new name.

Example 7-10: Renaming a Scratched Data File Which Is Still Viable

```
10 SCRATCH "DATFIL 1"
20 DATASAVE DC OPEN F "DATFIL 1", "TEST 2"
```

Statement 10 scratches DATFIL 1. Statement 20 renames DATFIL 1 with the name "TEST 2". The data in the file is not disturbed. However, the end-of-file trailer record in the file is lost and the USED column for TEST 2 in the Catalog Index is reset to 1. Thus, you should note the sector address of the trailer record in DATFIL 1 prior to scratching it. After opening TEST 2, you can skip to that location and rewrite the end-of-file record. Throughout this operation, the data is unaltered.

NOTE:

Although the name of a scratched file no longer appears in the catalog listing once the file has been renamed, the scratched file name remains in the Catalog Index. Thus, if a single file is scratched and renamed 16 times, only the final name shows in the catalog listing, despite the fact that all 16 names remain in the Catalog Index itself. Those 16 names would occupy one entire sector of the Catalog Index. Scratched file names can be removed from the Index only by executing a MOVE. The single exception to this rule is the case in which a scratched file is renamed with the same name. In that case, the new name occupies the slot on the Catalog Index occupied by the old name, and no duplication occurs. If it is necessary to scratch and rename files frequently, therefore, provision must be made for the scratched file names when establishing the size of the Catalog Index initially with SCRATCH DISK. Remember that the size of the Index cannot be altered once the catalog has been created.

7.6 EFFICIENT USE OF DISK STORAGE SPACE

The large storage capability of the disk unit may occasionally tempt the programmer to become profligate and inefficient in his use of disk storage space. Specifically, he may be tempted to design his records without due care for packing a maximum amount of data in a minimum number of sectors. Even when the available storage clearly exceeds present needs, however, this temptation should be overcome. Files have a way of outgrowing preliminary estimates at a faster-than-expected rate. Too, a file which is compact can be searched more quickly than one which is loosely layed out and contains large amounts of wasted space. In order to organize data within a record efficiently, it is necessary to understand more precisely how the system stores data in a sector. There are two main points to be considered:

1. Control information: The system automatically records control information along with the data in each sector. The control information occupies space in the data field of a sector, and must be taken into account when calculating how much space is required for a given amount of data.

2. "Gaps" in multisector records: Under certain conditions, gaps may occur between fields in a multisector record. In order to optimize the use of disk storage space, such gaps must be kept to a minimum.

System 2200 Control Information

The System 2200 automatically writes control information in each record created with a DATASAVE DC statement (or DATASAVE DA statement). This information is of two types:

1. Sector control bytes.
2. Start-of-value (SOV) control bytes.

Three sector control bytes are automatically written in each sector of a logical data record. The first two sector control bytes occupy the first two locations in the sector. The third control byte follows the last byte in the last field in the sector, and marks the limit of valid data within that sector. Information in the sector following the last sector control byte (also called the "end-of-block" byte) is regarded as garbage, and is ignored by the system when the sector is read. After taking into account the three sector control bytes, only 253 of the 256 bytes in a sector are initially available for data storage.

In addition to the sector control bytes, a start-of-value (SOV) control byte is prefixed to every field stored in the sector. The SOV byte separates data fields within a sector, marking the beginning of each individual value in the sector.

Consider, for example, the following statements:

```
10 DIM A$(2) 30
20 DATASAVE DC A$(), B$, "ABCD", 123, N
```

The argument list in statement 20 contains six separate arguments, each of which is prefixed with an SOV control byte when saved on disk. (Remember that each element of an array constitutes a single argument. Since A$() has two elements, it must be counted as two arguments.) The logical record created by statement 20 therefore looks like this:

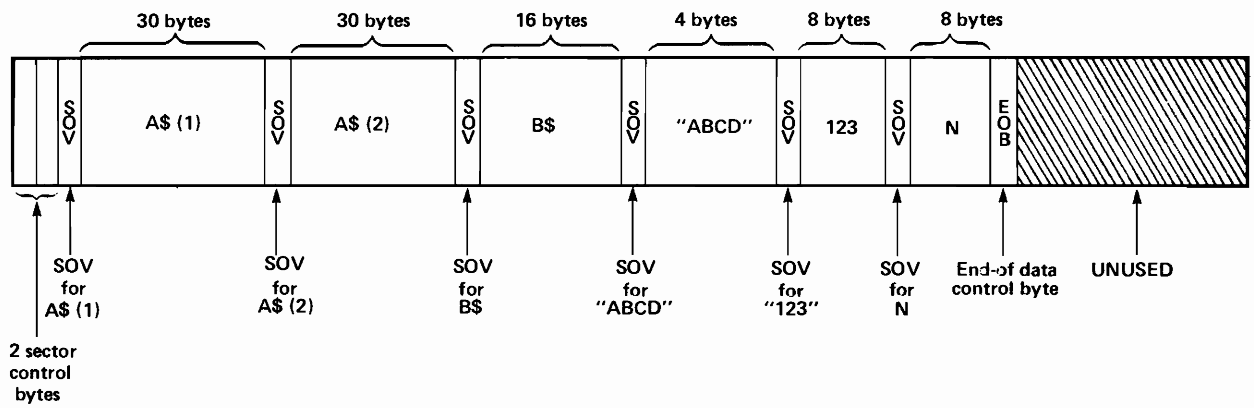


Figure 7-3. One Logical Record, Showing Sector Control Bytes and Start-of-Value Control Bytes for Each Field

From this illustration, it can be inferred that the following disk storage requirements hold:

a) Each numeric value, variable, or array element in the argument list always occupies nine bytes on disk (eight bytes for the numeric value and one byte for the SOV).

b) Each literal string in quotes occupies a number of bytes on disk equal to the number of characters in the literal string, plus one SOV byte.

c) Each alphanumeric variable or array element occupies a number of bytes on disk equal to the dimensioned length of the variable or element, plus one SOV byte.

Note that in the case of an alpha variable or array element, it is the dimensioned size, and not the number of characters actually stored in the variable or element, which must be counted. For example, the routine

```
50 DIM A$ 20
60 A$ = "ABC"
```

produces an alpha variable A$ which occupies 21 bytes on the disk (20 + 1), despite the fact that A$ contains a literal string only three characters in length. The remaining 17 bytes of A$ are blanks (spaces).

Inter-Field Gaps

In no case will the system overlap a single field from one sector to the next. If a field does not fit completely into one sector, it is written in its entirety into the next sequential sector. If record layouts are not carefully designed, this situation often gives rise to gaps between fields in multisector records.

Suppose, for example, that a logical record has been created with the following routine:

```
10 DIM A$(5)50, B$(3)64, C$ 48
:
:
100 DATASAVE DC A$(), B$(), C$
```

You could do some quick calculating and, making sure to add a control byte for each argument, conclude that the total record occupies 499 bytes. Since each sector can hold 253 bytes of data and control information (after the three sector control bytes are subtracted), two sectors can contain a total of 506 bytes. You might assume, therefore, that the record will fit easily into two sectors. Unfortunately, this calculation does not take into account the possibility of an inter-field gap. The argument list from line 100 is saved on disk in the following way:

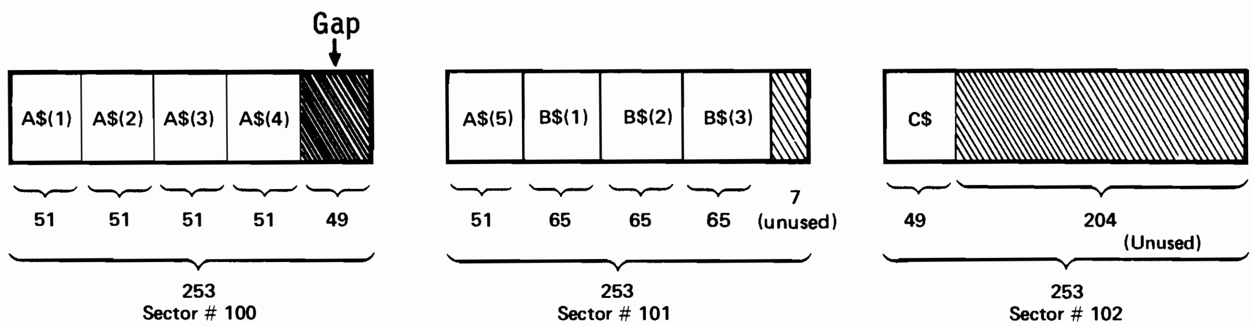


Figure 7-4. Inter-Field Gap in a Multisector Record

Notice that the last field in sector 100 consists of 49 bytes, and is marked "unused". Since A$(5) requires 51 bytes of space, it does not fit into the remaining 49 bytes in sector 100, and the entire field is written into the next sector (sector #101). The unused 49 bytes in sector #100 represent a "gap" of wasted space between A$(4) and A$(5). As a result of this gap, C$ must be written in a third sector. Instead of requiring two sectors, as the figures indicated, this record occupies three sectors. If the file contains, say, 100 such records, it will require 100 more sectors than were initially estimated.

The waste resulting from inter-field gaps can, in many cases, be decreased or eliminated by careful attention to the design of the record. In this case, for example, the record can be made to fit into two sectors simply by rearranging the order of the arguments in the DATASAVE DC argument list:

```
100 DATASAVE DC C$, A$(), B$()
```

The resulting logical record now looks like this:

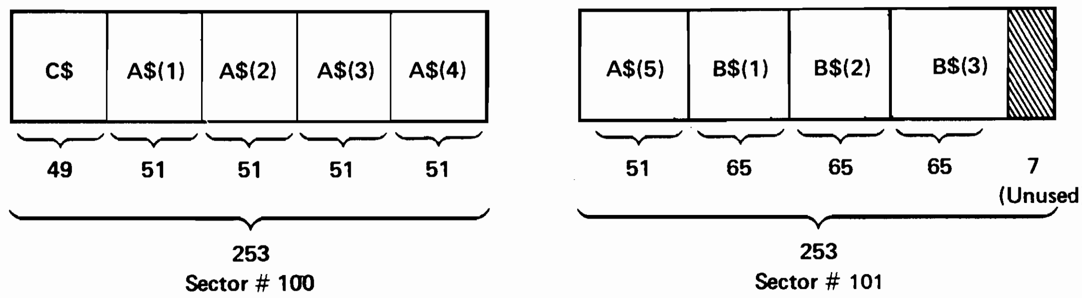


Figure 7-5. A Multi-Sector Record with No Gaps

By moving C$ from the end of the argument list to the beginning, the 49-byte gap in sector 100 is filled, thereby eliminating the need for a third sector in the record.

7.7 THE 'LIMITS' STATEMENT

A special catalog statement, LIMITS, enables the programmer to obtain the sector address parameters of a cataloged file under program control. For catalog operations alone, LIMITS is useful in such ways as, for example, keeping track of the amount of free space remaining in a file during an input routine. When the catalog procedures are supplemented with Absolute Sector Addressing operations (discussed in Chapter 9), which provide direct access to individual sectors, LIMITS becomes a truly powerful programming tool. One important use of LIMITS in conjunction with Absolute Sector Addressing statements is in the binary sea·:h technique described in Chapter 9.

The LIMITS statement has two forms. In Form 1, the name of a cataloged disk file is specified in the LIMITS statement. In this case, LIMITS goes directly to the disk and retrieves the starting sector address, ending sector address, and number of sectors used for the named file from the Catalog Index. In Form 2, the file name is omitted from the LIMITS statement. When this form is used, LIMITS reads the sector address parameters from a specified slot in the Device Table (the default slot if no file number is specified), and retrieves the starting, ending, and current sector address parameters from that slot. In this case, the disk is never accessed.

Form 1 of LIMITS

In Form 1 of the LIMITS statement, the following information must be specified:

1. The disk platter on which the named file resides ('F' or 'R').
2. Optionally, a file number (#0-#6).
3. The name of the file whose parameters are to be obtained.
4. Three numeric return variables designated to receive the file parameters. Variable #1 is set equal to the starting sector address of the file, variable #2 is set equal to the ending

sector address, and variable #3 is set equal to the number of sectors used in the file.

Form 1 of the LIMITS statement reads the Catalog Index entry for the named file and extracts the starting and ending addresses, and number of sectors used. These values are written in the specified slot in the Device Table (if a file number is included) or in the default slot (if no file number is included), and from there are copied to the three designated return variables.

Example 7-11: Form 1 of the LIMITS Statement
('File Name' Specified)

```
60 LIMITS F "TEST", A,B,C
```

Line 60 instructs the system to search the Catalog Index on the 'F' platter for the file "TEST", and retrieve the beginning and ending sector addresses of TEST, as well as the number of sectors used. These values are transferred to the default slot in the Device Table (since no file number is specified in the statement), and are then stored in the variables A,B,C, according to the following scheme:

A = Starting sector address.
B = Ending sector address.
C = Number of sectors used.

Example 7-12: Form 1 of the LIMITS Statement
('File Name' and a File Number Specified)

```
100 LIMITS R #2, "FILE 1", N,O,P
```

Line 100 instructs the system to retrieve the file parameters of FILE 1 from the 'R' platter. The parameters are first read into the slot opposite #2 in the Device Table, and are then stored in the designated return variables N,O,P.

Note that because the Device Table is used as an intermediate step in the retrieval of file parameters by Form 1 of LIMITS, the programmer must take care to specify an unused file number in the LIMITS statement. If the file number of a currently open file is specified, LIMITS will erase the sector address parameters of that file in the process of retrieving the requested file parameters from disk.

<u>Form 2 of Limits</u>

In Form 2 of the LIMITS statement, the following information must be specified:

1. The 'T' parameter.

2. The file number (#0-#6) of a currently open file (if no file number is specified, the default file number, #0, is used).

3. Three numeric return variables designated to receive the file parameters. Variable #1 is set equal to the starting sector address of the file, variable #2 is set equal to the ending sector address, and variable #3 is set equal to the current sector address.

Form 2 of the LIMITS statement reads the sector address parameters from a specified slot in the Device Table, and stores them in the designated return variables. Unlike Form 1, Form 2 does not access the disk to read the Catalog Index, nor does it alter in any way the sector address parameters stored in the Device Table.

Example 7-13: Form 2 of the LIMITS Statement
('File Name' Not Specified)

```
150 LIMITS T A,B,C
```

Line 150 reads the sector address parameters (starting, ending, current) from the default slot in the Device Table (since no file number is specified), and stores them in variables A,B,C in the following order:

A = Starting sector address.
B = Ending sector address.
C = Current sector address.

Example 7-14: Form 2 of the LIMITS Statement
('File Name' Not Specified)

```
200 LIMITS T #3, N,O,P
```

Line 200 retrieves the sector address parameters from the Device Table slot opposite file number #3, and stores those parameters in variables N,O,P.

Note that Form 2 of the LIMITS statement makes no check on the validity of the information read from the Device Table. If a slot contains meaningless parameters (as it might, for example, if its file number had recently been used in an Absolute Sector Addressing statement), this information is returned by LIMITS without an error. It is the programmer's responsibility to ensure that the specified file number is associated with a currently open cataloged file. Because Absolute Sector Addressing operations do not store meaningful file parameter information in the Device Table, LIMITS should not be used with files maintained in Absolute Sector Addressing Mode. (LIMITS may be used in conjunction with Absolute Sector Addressing procedures to process a cataloged file, however; see Section 9.7).

7.8 CONCLUSION

The discussion of catalog procedures proper is now concluded. All of the characteristics of the several catalog statements and commands and their applications have, in greater or lesser detail, been touched upon. The programmer who wishes to make the most efficient use of the catalog procedures should press on, however, and read Chapter 9, which deals with the Absolute Sector Addressing Mode. Absolute Sector Addressing statements and procedures can be used in conjunction with cataloging procedures to produce a more versatile and efficient disk management system. In particular, Chapter 9 discusses the "binary search" technique for directly accessing records in a cataloged file.

Chapter 8
Automatic File Cataloging Statements and Commands

8.1 INTRODUCTION

This chapter contains capsule descriptions and general forms for the following Automatic File Cataloging statements and commands, listed alphabetically for ease of reference:

DATALOAD DC	LOAD DC (Command)
DATALOAD DC OPEN	LOAD DC (Statement)
DATASAVE DC	MOVE
DATASAVE DC CLOSE	MOVE END
DATASAVE DC OPEN	SAVE DC
DBACKSPACE	SCRATCH
DSKIP	SCRATCH DISK
LIMITS	VERIFY
LIST DC	

8.2 SYSTEM 2200 STATEMENTS AND COMMANDS

The distinction between a statement and a command requires some explanation. In general, the term "statement" is a generic term which denotes all BASIC instructions in the System 2200 BASIC language set. There are two categories of BASIC statements:

a. Programmable statements (also referred to simply as "statements").

b. Non-programmable statements (also referred to as "commands").

In its narrower sense, therefore, the term 'statement' denotes BASIC instructions which can be executed within a program (i.e., on a numbered program line), while the term "command" denotes those BASIC instructions which can never be executed in a program (commands are executable in Immediate Mode only). The set of BASIC instructions governing disk operations contains only two commands: SAVE DC and SAVE DA (SAVE DA is discussed in Chapters 9 and 10). These commands cannot be executed in a program. All other disk instructions are programmable statements, and may be executed either in Program Mode (i.e., on a numbered program line) or in Immediate Mode.

A single exception to the command/statement distinction must be noted. Nearly all System 2200 programmable statements can be executed either in Program Mode or in Immediate Mode (as noted above, this is true of all disk statements). In general, the sequence of operations associated with a disk statement when it is executed within a program is identical to the sequence of operations associated with the statement when it is executed in Immediate Mode. LOAD DC (and LOAD DA) represent exceptions to this rule, however. The sequence of operations initiated by a LOAD DC (or LOAD DA) instruction when it is executed in Immediate Mode is significantly different from the sequence of operations initiated by the same instruction when executed on a numbered program line. For this reason, the LOAD DC instruction is treated as two separate and distinct entities, distinguished by their mode of execution: the LOAD DC statement (executed in a program), and the LOAD DC command (executed in Immediate Mode). The LOAD DA instruction is treated similarly in Chapter 10.

8.3 BASIC RULES OF SYNTAX

The notation and rules of syntax employed in the General Forms of disk statements follow the conventions used in the System 2200 Reference Manual. The conventions are summarized below:

1. The following symbols must be included in an actual BASIC statement exactly as they appear in the General Form of the statement:

   a. Uppercase letters — A through Z
   b. Comma — ,
   c. Double Quotation Marks — "
   d. Parentheses — ()
   e. Pound Sign — #
   f. Slash — /

2. Lowercase letters and words in the General Form of a statement represent items whose values must be assigned by the programmer. For example, if the lowercase word "name" appears in a General Form, the programmer must substitute a specific file name (such as "PROG 1"), or an alphanumeric variable containing the name, in the actual statement. Similarly, where the lowercase letter n appears, the programmer must substitute an actual file number (from 0 to 6) or a variable containing a file number.

3. Three special symbols are used in the General Forms as mnemonics, providing the programmer with required information. These symbols are never included in an actual BASIC statement:

   a. Brackets — [ ]
   b. Braces — { }
   c. Ellipses — ...

4. Square brackets, [ ], indicate that the enclosed information is optional, and may be included or not in the actual BASIC statement, at the programmer's discretion.

5. Vertically stacked items represent alternatives, only one of which should be included in an actual BASIC statement:

    a. Square brackets, [ ], enclosing vertically stacked items indicate that all of the items are optional.

    b. Braces, { }, enclosing vertically stacked items indicate that one of the items must be included in an actual statement.

6. Ellipses, ..., indicate that the preceding item(s) may be repeated once or several times in succession.

7. Blanks (spaces), used to improve the readability of the General Forms, are meaningless to the system (unless enclosed in double quotation marks), and may be omitted or included in an actual statement, at the option of the programmer.

8. The sequence in which terms are listed in the General Form of a statement must be followed exactly in an actual statement.

DATALOAD DC

General Form:

DATALOAD DC [#n,] argument list

where:

DC = A parameter specifying Disk Catalog Mode.

#n = A file number to which the disk is currently assigned ('n' is an integer or numeric variable whose value is from 0 to 6).

argument list = { alphanumeric variable / numeric variable / alpha or numeric array designator } [, { ...]

array designator = An array name followed by closed parentheses, e.g., A( ), B$( ).

Purpose:

The DATALOAD DC statement is used to read logical data records from a cataloged disk file and sequentially assign the values read to the variables and/or arrays in the argument list. Before data can be read from a cataloged file, the file must be opened by a DATALOAD DC OPEN or DATASAVE DC OPEN statement. Thereafter, each time a DATALOAD DC statement is executed, the system begins reading data from the file at the next sequential logical record in the file. Arrays are filled row by row. If the DATALOAD DC receiving variable list is not filled by one logical record, the next logical record, or a portion of the next logical record, is read. If the logical record being read contains more data than is required to fill all receiving variables in the argument list, data not used is read but ignored. Each time the DATALOAD DC statement is executed, the Current Sector Address associated with the file in the Device Table is updated to the Starting Sector Address of the next consecutive logical record. If an end-of-file trailer record is read, an end-of-file condition is set, the Current Sector Address is set to the address of the trailer record, and no data is transferred. The end-of-file condition can be tested by a subsequent IF END THEN statement. If the user attempts to read beyond the final sector address for the file, an error is signalled.

Examples:

```
100 DATALOAD DC S(), Y, Z
100 DATALOAD DC #2, A$(), B()
100 DATALOAD DC #B2, B(), C, D$
```

General Form:

$$\text{DATALOAD DC OPEN} \begin{Bmatrix} F \\ R \\ T \end{Bmatrix} [\#n,] \begin{Bmatrix} \text{TEMP, expression 1, expression 2} \\ \text{name} \end{Bmatrix}$$

where:

DC	=	A parameter specifying Disk Catalog Mode.
F	=	Fixed platter, Drive #1, Drive #3.
R	=	Removable platter, Drive #2.
T	=	'F' or 'R' disk platter, depending on device type specified in device address.
#n	=	A file number to which the disk is currently assigned ('n' is an integer or numeric variable whose value is from 0 to 6).
name	=	The name of the cataloged data file to be located and opened. The name is from 1 to 8 characters in length, and is expressed as an alphanumeric variable or literal string in quotes.
TEMP	=	A temporary work file is to be re-opened.
expression 1	=	Truncated value is starting sector address of temporary work file.
expression 2	=	Truncated value is ending sector address of temporary work file.

Purpose:

The DATALOAD DC OPEN statement is used to open data files that have previously been cataloged on the disk. When the statement is executed, it locates the named file on the specified disk platter, and sets up the Starting, Ending, and Current Sector Addresses of the file in the Device Table (the current address is set equal to the starting address). Any subsequent use of the same file number in other catalog (DC) statements accesses this file. If no file number is included, the file is assumed to be associated with the default file number (#0) and can be accessed by subsequent DC statements with the file number omitted, or by specifying #n = #0.

An error will result if the file name cannot be located in the Catalog Index of the specified disk, or if the file has been scratched.

The TEMP parameter is used to reopen a temporary work file; the starting and ending addresses must not be located in the cataloged area. Temporary file areas can be accessed with catalog statements and commands (e.g., DATASAVE DC, DATALOAD DC, etc.).

The DATALOAD DC OPEN statement must be used when reopening an existing cataloged data file; use of the DATASAVE DC OPEN statement results in an error if the named file is already in the catalog and has not been scratched. Therefore, DATALOAD DC OPEN is used to reopen a cataloged file irrespective of whether data is to be written in the file with a DATASAVE DC statement or read from the file with a DATALOAD DC statement.

Examples:

```
100 DATALOAD DC OPEN F "HEADING"
100 DATALOAD DC OPEN R #2, A$
100 DATALOAD DC OPEN T #A, TEMP, 8000, 9000
```

General Form:

DATASAVE DC [$] [#n,] { END | argument list }

where:

DC = A parameter specifying Disk Catalog Mode.

$ = Read after write.

#n = A file number to which the file is currently assigned ('n' is an integer or numeric variable whose value is from 0 to 6).

argument list = { literal string | alphanumeric variable | expression | numeric variable | alpha or numeric array designator } [, ... { }]

array designator = An array name followed by closed parentheses, e.g., A( ), B$( ).

END = Write a data trailer (end-of-file) record.

Purpose:

The DATASAVE DC statement causes one logical record, consisting of all the data in the DATASAVE DC argument list, to be written onto the disk, starting at the current sector address associated with the specified file number (#n) in the Device Table. If no file number is specified in the DATASAVE DC statement, the data is written into the file currently associated with the default file number (#0) in the Device Table. The file must previously have been opened with a DATASAVE DC OPEN or DATALOAD DC OPEN statement. No data can be saved into an unopened file; if the DATASAVE DC statement specifies a file number not associated with a currently open file, an error results.

The DATASAVE DC argument list may include literal strings (e.g., "JOHN JONES") and expressions (e.g., B*C), as well as alphanumeric and numeric variables and arrays.

The 'DC' parameter implies that the data in the argument list is to be written as one logical record in standard System 2200 format, including the necessary control information. The values in the argument list are stored sequentially on the specified disk. Arrays are written row by row. Each single logical record may consist of one or more sectors on the disk.

NOTE:

Each numeric value in the argument list requires 9 bytes of storage on disk. Each alphanumeric variable requires the maximum length to which the variable is dimensioned plus 1 byte; e.g., if the length of A$ is set to 24 characters in a DIM A$24 statement, then A$ requires 25 (24 + 1) bytes of storage on disk. Each 256 byte sector also requires 3 bytes of sector control information (refer to Section 7.6).

The '$' parameter specifies that a 'read-after-write' verification test be made on all data written to the disk. This test provides an extra safeguard against disk write errors, but it also effectively doubles the time required for the DATASAVE DC operation.

If the special END parameter is specified, a data trailer record is written in the file, and the Catalog Index entry for the file is updated so that the number of sectors used by the file includes all sectors up to the trailer record just written. A cataloged file always should be ended by a trailer record. A new data record can be stored in the file by writing over the trailer record, and subsequently creating a new trailer record. (A DSKIP END statement positions the system to the beginning of the trailer record; a DATASAVE DC statement can be executed at that point to store the new record over the trailer record, and a subsequent DATASAVE DC END statement executed to create a new trailer record.)

Examples:

```
100 DATASAVE DC A,X, "CODE#4"
100 DATASAVE DC $ #2, M$, P2(), F1$()
100 DATASAVE DC $ #1, "ADDRESS", (3*1)/100, J$()
100 DATASAVE DC #3, END
100 DATASAVE DC #A, A$()
```

General Form:

DATASAVE DC CLOSE [#n / ALL]

where:

DC = A parameter specifying Disk Catalog Mode.

#n = The file number associated with a currently open file which is to be closed ('n' is an integer or numeric variable whose value is from 0 to 6).

ALL = All currently open files are to be closed.

Purpose:

The DATASAVE DC CLOSE statement is used to close an individual data file or all data files which are currently open, if they are no longer needed in the current or subsequent programs. The DATASAVE DC CLOSE statement closes a file by setting the starting, ending, and current sector addresses associated with its file number in the Device Table equal to zero. When the file is closed, a disk statement referencing that file causes an ERROR 86 (File Not Open) to be displayed.

If the #n parameter is used, the single file associated with that file number is closed. If the ALL parameter is used, every open file is closed. If neither parameter is used, the currently open file associated with the default file number (#0) is closed.

The DATASAVE DC CLOSE statement should not be confused with DATASAVE DC END. The latter writes an end-of-file record at the end of a newly written file. The end-of-file record should always be written prior to executing DATASAVE DC CLOSE.

It is good programming practice to close a file with DATASAVE DC CLOSE upon completion of processing, since it insures that subsequent disk users will not erroneously access the file and possibly destroy data. Likewise, DATASAVE DC CLOSE can be used at the beginning of a program to initialize file parameters to zero before they are set by DATASAVE DC OPEN or DATALOAD DC OPEN. DATASAVE DC CLOSE does not remove disk device addresses from the Device Table.

Examples:

```
900 DATASAVE DC CLOSE
900 DATASAVE DC CLOSE #3
900 DATASAVE DC CLOSE ALL
900 DATASAVE DC CLOSE #A
```

DATASAVE DC OPEN

General Form:

```
                       ⎧F⎫                  ⎧ ⎧old file name,⎫ new file name          ⎫
DATASAVE DC OPEN       ⎨R⎬  [$] [#n,]       ⎨ ⎩expression,   ⎭                        ⎬
                       ⎩T⎭                  ⎩ TEMP, expression 1 , expression 2      ⎭
```

where:

DC	=	A parameter specifying Disk Catalog Mode.
F	=	Fixed platter, Drive #1, Drive #3.
R	=	Removable platter, Drive #2.
T	=	'F' platter or 'R' platter, depending on device type specified in the device address.
$	=	Read after write.
#n	=	A file number to which the disk is currently assigned ('n' is an integer or numeric variable whose value is from 0 to 6).
old file name	=	The name of an existing scratched program or data file which is cataloged on the specified disk platter. The name can be from one to eight characters in length, expressed as an alphanumeric variable or literal string in quotes.
expression	=	The number of sectors to be reserved for the new file.
new file name	=	The name of the data file being opened, expressed as an alpha-numeric variable or a literal string in quotes from 1 to 8 characters in length.
TEMP	=	A temporary work file is to be established.
expression 1	=	Truncated value is the starting sector address of a temporary work file.
expression 2	=	Truncated value is the ending sector address of a temporary work file.

Purpose:

The DATASAVE DC OPEN statement is used to reserve space for cataloged files in the Catalog Area, and to enter appropriate system information in the Catalog Index. It is also used to reserve space for temporary work files outside the Catalog Area, and to reuse space in the Catalog Area occupied by scratched files.

Data files can be opened on any disk platter by including the proper parameter ('F' or 'R') in the DATASAVE DC OPEN statement. Each data file must be opened initially with a separate DATASAVE DC OPEN statement; if multiple files are to be open simultaneously, each file must be assigned a different file number. Since there are seven file numbers available (0-6), a total of seven data files can be open simultaneously.

The '$' parameter specifies that a 'read-after-write' verification test be performed to ensure that all file control information is written correctly in the Catalog Index. This test helps to protect against disk write errors, but also doubles the time required for the DATASAVE DC OPEN operation.

The '#n' parameter is the file number which identifies the newly-opened file in the Device Table. The disk on which the file is stored, along with the file's starting, ending, and current sector addresses, are entered in the Device Table in System 2200 memory. The information in the Device Table is identified only by the file number assigned to the file in the DATASAVE DC OPEN statement. A file number must be included in the DATASAVE DC OPEN statement if more than one file is to be open at one time. If no file number is specified, or if #n = #0, the system automatically assigns the newly opened file to the default slot, #0, in the Device Table. Subsequent reference to a file number in a disk catalog statement or command automatically provides access to the current sector address of the associated file. (For a detailed discussion of the Device Table and the use of file numbers, see Chapter 6.)

The 'old file name' parameter specifies the name of a previously scratched cataloged file (either program or data) which is to be renamed and reused. If the 'old file name' parameter is used in place of the 'expression' parameter, the new file is given the space previously occupied by the scratched file.

If the 'expression' parameter is used instead of 'old file name', the new file is appended at the current end of the Catalog Area, and given a total number of sectors equal to the truncated value of the 'expression'.

> NOTE:
>
> The last sector of each cataloged data file is reserved for systems information. Therefore, the number of sectors available for data storage is always at least one less than the number of sectors reserved for the file.

The 'new file name' parameter is the name of the new data file being opened. If 'new file name' is being stored in space previously occupied by a scratched cataloged file ('old file name'), then 'new file name' can be identical to 'old file name'. Otherwise, 'new file name' must be unique.

The TEMP parameter is used to specify a temporary work file. Temporary files are not cataloged and cannot be located in the Catalog Area. If temporary files are to be used, sufficient space must be left outside the Catalog Area to accommodate them (see SCRATCH DISK).

The 'expression 1' and 'expression 2' parameters identify the starting and ending sectors of the area reserved for a temporary file. An error results if the value of 'expression 1' is less than or equal to the last (highest) sector of the Catalog Area.

Examples:

```
100 DATASAVE DC OPEN R 100, "DATFIL 1"
100 DATASAVE DC OPEN R #1, A*2, "I/O DATA"
100 DATASAVE DC OPEN F #2, "DATFIL 1", "DATFIL 2"
100 DATASAVE DC OPEN F TEMP 1000, 2000
100 DATASAVE DC OPEN T #4, 200, A$
```

DBACKSPACE

General Form:

DBACKSPACE [#n,] { BEG | expression [S] }

where:

#n = A file number to which the data file is currently assigned ('n' is an integer or numeric variable whose value is from 0 to 6).

BEG = Backspace to beginning of file.

expression = Truncated value equals the number of logical records or sectors to be backspaced.

S = Backspace absolute number of sectors.

Purpose:

The DBACKSPACE statement is used to backspace over logical records or sectors within a cataloged disk file. If 'expression' is used alone, the system backspaces over a number of logical records equal to the truncated value of the 'expression', and the Current Sector Address of the file in the Device Table is updated to the starting sector of the new logical record. For example, if 'expression' = 1, the Current Sector Address is set equal to the starting address of the previous logical record. If the BEG parameter is used, the Current Sector Address is set equal to the Starting Sector Address of the file (that is, the starting address of the first logical record in the file).

If the 'S' parameter is used, the truncated value of the expression equals the total number of sectors to backspace. The Current Sector Address of the file in the Device Table is decremented by the number of sectors specified. If the amount specified is too large, the Current Sector Address is set to the Starting Sector Address of the file. The 'S' parameter is particularly useful in files where all the logical records are of the same length (i.e., have the same number of sectors per logical record). Backspacing with the 'S' parameter is much faster than backspacing over logical records in a file, since the system merely decrements the Current Sector Address in the Device Table by the specified number of sectors, and no disk accesses are required. However, the user must be certain that he knows exactly how many sectors are in each logical record.

Examples:

```
100 DBACKSPACE BEG
100 DBACKSPACE 2*X
100 DBACKSPACE #2, 5S
100 DBACKSPACE #1, BEG
100 DBACKSPACE #A, 10
```

General Form:

```
DSKIP [#n,] { END             }
            { expression [S]  }
```

where:

#n	=	A file number to which the data file is currently assigned ('n' is an integer or numeric variable whose value is from 0 to 6).
END	=	Skip to current end-of-file.
expression	=	Truncated value equals the number of logical records or sectors to be skipped.
S	=	Absolute number of sectors are to be skipped.

Purpose:

The DSKIP statement is used to skip over logical records or sectors in a cataloged disk file. If 'expression' is used alone, the system skips over a number of logical records equal to the truncated value of 'expression', and the Current Sector Address for the file is updated to the starting address of the new logical record. If the 'END' parameter is used, the system skips to the end of the file; i.e., the current sector address for the file is updated to the address of the end-of-file trailer record. Once a DSKIP #n, END statement has been executed, data can be added to the end of the file using DATASAVE DC statements. Note that the DSKIP END statement cannot be used unless a trailer record has previously been written in the file with a DATASAVE DC END statement. DSKIP END results in an Error 82 (No End of File) if no trailer record can be located in the file.

If the 'S' parameter is used, the truncated value of the expression equals the total number of sectors to be skipped. The Current Sector Address of the file is incremented by the number of sectors specified. If the amount specified is too large, the Current Sector Address is set to the Ending Sector Address of the file. The 'S' parameter is particularly useful in files where all logical records are of the same length (i.e., have the same number of sectors per logical record). Skipping with the 'S' parameter is much faster than skipping logical records in a file, since the system merely increments the current address by the specified number of sectors, and no disk accesses are necessary. However, the user must be sure that he knows exactly how many sectors are in each logical record.

Examples:

```
100 DSKIP 4
100 DSKIP #2, END
100 DSKIP END
100 DSKIP #3, 4*X
100 DSKIP #A, 20S
```

LIMITS

General Form:

LIMITS {F / R / T} [#n,] [name,] variable 1, variable 2, variable 3

where:

F	=	Fixed platter, Drive #1, Drive #3.
R	=	Removable platter, Drive #2.
T	=	Either 'F' platter or 'R' platter, depending on device type specified in the device address.
#n	=	A file number to which the disk address is currently assigned ('n' is an integer or numeric variable whose value is from 0 to 6).
name	=	The name of the cataloged data or program file whose limits are to be retrieved. The name is from 1 to 8 characters and is expressed as an alphanumeric variable or literal string in quotes. If 'name' is not specified, limit information on a currently open file (in a file slot) is to be retrieved.
variable 1	=	A numeric variable designated to receive the starting sector address of the file.
variable 2	=	A numeric variable designated to receive the ending sector address of the file.
variable 3	=	A numeric variable designated to receive the number of sectors used by the file, or current sector address of the file.

Purpose:

The LIMITS statement obtains the Beginning and Ending Sector Address and Current Sector Address or number of sectors used for a cataloged file.

If 'name' is specified in the statement, information is taken from the Catalog Index entry for the named file. In this case, variable 3 is set equal to the total number of sectors used by the file.

If 'name' is not specified, information is retrieved from the Device Table entry for the currently open file associated with either the specified file number (if #n is specified), or the default file number (if #n is not specified). In this case, variable 3 is set equal to the Current Sector Address of the file.

LIMITS can be used within a program to find out how much remaining space is left in a file or to get sector address limits of a file.

<u>Limits of a Cataloged File ('name' specified)</u>

If a file name is specified, the LIMITS statement finds the named program or data file on the specified disk and sets variable 1 equal to the Starting Sector Address of the file, variable 2 equal to the Ending Sector Address of the file, and variable 3 equal to the number of sectors currently used by the file. The number of sectors currently being used by the file is accurate only if an end-of-file record has been written in the file. An end-of-file record is written in a data file with a DATASAVE DC END statement.

Therefore, in order to be able to tell how many sectors are used in a data file, the file must be ended with an end-of-file record.

Note that this form of the LIMITS statement alters the file parameters in a slot in the Device Table. If a file number #1 - #6 is included in the LIMITS statement, the parameters in the associated slot are altered. If no file number is used, or if n=0, the parameters in the default slot are altered. The second form of LIMITS, discussed below, does not alter the Device Table.

Examples:

```
100 LIMITS F "PAYROLL", A,B,C
100 LIMITS T A$, S,E,A
100 LIMITS T #A, "DATFIL 1", X,Y,Z(3)
100 LIMITS F #1, "SAM", A,B,C
```

Limits of a Currently Open File ('name' Not Specified)

If a file name is not specified, the LIMITS statement gives the Starting, Ending and Current Sector Addresses of the file currently open at #n or in the default slot. Variable 1 = Starting, variable 2 = Ending, variable 3 = Current. The 'T' parameter must be specified when seeking the LIMITS of a currently open file.

Examples:

```
100 LIMITS T #A(1), A1,A2,A3
100 LIMITS T #5, A,B,C
100 LIMITS T X,Y,Z(2)
```

LIST DC

General Form:

```
LIST DC  ⎧F⎫  ⎡#n  ⎤
         ⎨R⎬  ⎣/xxx⎦
         ⎩T⎭
```

where:

DC	=	A parameter specifying Disk Catalog Mode.
F	=	Fixed platter, Drive #1, Drive #3.
R	=	Removable platter, Drive #2.
T	=	'F' platter or 'R' platter, depending on device type specified.
#n	=	A file number to which the disk address is currently assigned ('n' is an integer or numeric variable from 0 to 6).
/xxx	=	Device address of disk. If neither #n nor /xxx is specified, or if n = 0, the default address (stored opposite #0 in the Device Table) is used. The system default disk address is 310.

Purpose:

The purpose of the LIST DC statement is to display or print out a listing of the information contained in the Catalog Index. When the LIST DC statement is executed, the following information is displayed on the currently selected LIST device:

a. The number of sectors in the Catalog Index.
b. The address of the last sector reserved for the Catalog Area.
c. The address of the last used sector in the Catalog Area.

For each cataloged file, the LIST DC statement outputs the following data:

a. The file name.
b. The file status (S if scratched).
c. The file type (program (P) or data (D)).
d. The Starting Sector Address.
e. The Ending Sector Address.
f. The number of sectors currently used in the file. For a data file, this value is originally set to one, and is updated only when an end-of-file record is written in the file.

Depressing the HALT/STEP key terminates printout of the catalog.

Examples:

```
LIST DC F
LIST DC F #2
LIST DC R
LIST DC T #A
LIST DC F/320
```

General Form:

LOAD DC {F | R | T} [#n, | /xxx,] name

where:

DC	=	A parameter specifying Disk Catalog Mode.
F	=	Fixed platter, Drive #1, Drive #3.
R	=	Removable platter, Drive #2.
T	=	'F' platter or 'R' platter, depending on device type specified in the device address.
#n	=	A file number to which the disk address is currently assigned ('n' is an integer of numeric variable whose value is from 0 to 6).
/xxx	=	The device address of the disk. If neither #n nor /xxx is specified, or if n = 0, the default disk address (stored opposite #0 in the Device Table) is used. The system default disk address is 310.
name	=	The name of the cataloged program to be loaded; the name must be from 1 to 8 characters in length, specified either as an alphanumeric variable or literal string in quotes.

Purpose:

The LOAD DC command is used to load BASIC programs or program segments from the disk. This command causes the system to locate the named program in the catalog, and append it to the program text currently in memory. Programs can be loaded into memory from any disk platter.

LOAD DC can be used to add to program text currently in memory or, if executed following a CLEAR command, to load a new program. An error results if the requested file is not a program file, or if it is not present in the catalog.

Examples:

```
LOAD DC F "PROG 1"
LOAD DC R #2, "TEST1/0"
LOAD DC R /320, "OUTPUT1"
LOAD DC T A$
LOAD DC T #A1, B$
```

LOAD DC (Statement)

General Form:

LOAD DC $\begin{Bmatrix} F \\ R \\ T \end{Bmatrix}$ $\begin{bmatrix} \#n, \\ /xxx, \end{bmatrix}$ name [line number 1] [, line number 2]

where:

DC = A parameter specifying Disk Catalog Mode.

F = Fixed platter, Drive #1, Drive #3.

R = Removable platter, Drive #2.

T = 'F' platter or 'R' platter, depending on device type specified in the device address.

#n = A file number to which the disk address is currently assigned ('n' is an integer or numeric variable whose value is from 0 to 6).

/xxx = The device address of the disk. If neither #n nor /xxx is specified, or if n = 0, the default address (stored opposite #0 in the Device Table) is used. The system default disk address is 310.

name = The name of the cataloged program file to be loaded into memory, expressed either as an alphanumeric variable or a literal string in quotes. The name can be from 1 to 8 characters in length.

line number 1 = The number of the first program line to be deleted from the program currently in memory prior to loading the new program. After loading, execution continues automatically at this line number. An error results if there is no line with this number in the newly loaded program.

line number 2 = The number of the last program line to be deleted from the program currently in memory before the new program is loaded.

Purpose:

The LOAD DC statement loads a BASIC program or program segment into memory from the disk, and automatically executes it. LOAD DC is a BASIC statement which in effect produces an automatic combination of the following BASIC statements and commands:

STOP (stop current program execution)

CLEAR P (clear program text from memory, beginning at 'line number 1' (if specified) and ending at 'line number 2' (if specified); if no line numbers are specified, clear all currently stored program text from memory)

CLEAR N (clear all non-common variables from memory)

LOAD DC (load new program or program segment from disk)

RUN (run new program, beginning at 'line number 1', if specified, or at the lowest program line in memory, if no line numbers are specified)

If only 'line number 1' is specified, the remainder of the currently stored program is deleted, starting with that line number, prior to loading the new program from disk, and execution continues automatically with 'line number 1' of the newly loaded program. If both line numbers are specified, all program lines in memory between and including these lines are cleared prior to loading the new program. If no line numbers are specified, all currently stored program text is cleared, and the newly loaded program is executed from the lowest line number. In all cases, all non-common variables are cleared prior to loading the new program.

The LOAD DC statement permits segmented programs to be run automatically without normal user intervention. Common variables are passed between program segments. If LOAD DC is included on a multistatement line, it must be the last executable statement on the line.

In Immediate Mode, LOAD DC is interpreted as a command (see LOAD DC command).

Programs can be loaded from any disk platter by including the proper parameter in the LOAD DC statement. If 'T' is used as a parameter, the program is loaded from the disk platter designated by the device type in the disk device address (device type 3 designates the 'F' platter; device type B designates the 'R' platter).

Examples:

```
100 LOAD DC R "PROG 1"
100 LOAD DC F #2, "I/OMSTR"
100 LOAD DC F/320, "I/OSUB1" 250, 299
100 LOAD DC R "I/OCNTRL" 500
100 LOAD DC T A$ 100
100 LOAD DC T #X, B$
```

MOVE

General Form:

MOVE [#n, /xxx,] {FR RF}

where:

#n = A file number to which the disk address is currently assigned ('n' is an integer or numeric variable whose value is from 0 to 6).

/xxx = The device address of the disk.
If neither #n nor /xxx is specified, or if n = 0, the default disk address (stored opposite #0 in the Device Table) is used. The system disk default address is 310.

FR = Move and compress the catalog area from the 'F' platter to the 'R' platter.

RF = Move and compress the catalog from the 'R' platter to the 'F' platter.

Purpose:

The purpose of the MOVE statement is to copy the entire catalog from one disk platter to the other, deleting all scratched files from the Catalog Area, and removing the scratched file names from the Catalog Index. After the scratched files are removed, the still-active files are moved up to fill in the vacated sectors in the Catalog Area, and the Starting, Ending, and Current Sector Addresses of all relocated files are automatically altered to reflect the files' new positions in the Catalog. In effect, the MOVE command copies the Catalog Area and Catalog Index, squeezing out all deleted files. Temporary files are not copied.

If the 'FR' parameter is used, the contents of the 'F' platter are compressed and copied to the 'R' platter. If the 'RF' parameter is used, the process takes place from the 'R' platter to the 'F' platter.

Following a MOVE, the user can execute a VERIFY statement to insure that the entire catalog was copied correctly.

When MOVE is executed as either a command or program statement, 1024 bytes of memory must be available for buffering (not occupied by a BASIC program or variables); otherwise, an error results and the MOVE is not performed. The large buffer minimizes the time required for the MOVE operation.

NOTE TO OWNERS OF THE
MODELS 2270-1 and 2270-3:

On the Model 2270-3, it is not possible to MOVE the catalog from Platter #3 to Platter #1 or #2, or vice versa. In order to move the catalog to or from Platter #3, the platter must be physically removed from drive #3 and inserted in drive #1 or drive #2. On the Model 2270-1, MOVE is illegal.

Examples:

```
10 MOVE FR
10 MOVE /320, RF
10 MOVE #2, FR
10 MOVE #C, RF
```

MOVE END

General Form:

$$\text{MOVE END} \begin{Bmatrix} F \\ R \\ T \end{Bmatrix} \begin{bmatrix} \#n \\ /xxx \end{bmatrix} = \text{expression}$$

where:

F	=	Fixed platter, Drive #1, Drive #3.
R	=	Removable platter, Drive #2.
T	=	'F' platter or 'R' platter, depending on the device type specified in the device address.
/xxx	=	The device address of the disk.
#n	=	A file number to which the disk address is currently assigned ('n' is an integer or numeric variable whose value is from 0 to 6).

If neither #n nor /xxx is specified, or if n = 0, the default address (stored opposite #0 in the Device Table) is used. The system default disk address is 310.

Purpose:

The MOVE END statement is used to increase or decrease the size of the Catalog Area on a disk platter. The upper limit of the Catalog Area is initially defined by the END parameter in the SCRATCH DISK statement (see SCRATCH DISK). Once the limit of this area has been set, it can be altered using the MOVE END statement. The truncated value of the 'expression' specifies the sector address of the new end of the Catalog Area. An error results if a previously cataloged file resides at this address, or if the address is higher than the highest legal address on the platter. Note that MOVE END does not alter the size of the Catalog Index.

Examples:

```
MOVE END F = 4799
MOVE END R = .5*L
MOVE END T = X+Y
MOVE END R/320 = 2399
```

General Form:

SAVE DC {F / R / T} [$] [(expression / old file name)] [#n, / /xxx,] [P] new file name [line number 1] [, line number 2]

where:	DC =	A parameter specifying Disk Catalog Mode.
	F =	Fixed platter, Drive #1, Drive #3.
	R =	Removable platter, Drive #2.
	T =	'F' platter or 'R' platter, depending on device type specified in the device address.
	$ =	Read after write.
	expression =	Truncated value equals the number of sectors to reserve in addition to the number required to store the program.
	old file name =	The name of a currently scratched program or data file.
	#n =	A file number to which the disk address is currently assigned ('n' is a digit or numeric variable whose value is from 0 to 6).
	/xxx =	The device address of the disk. If neither #n nor /xxx is used, or if n = 0, the default address (stored opposite #0 in the Device Table) is used. The system default disk address is 310.
	P =	Set the protection bit on the file to be saved.
	new file name =	The name of the program to be saved. The name must be from 1 to 8 characters in length, and may be expressed as an alphanumeric variable or as a literal string in quotes.
	line number 1 =	The first line of program text to be saved.
	line number 2 =	The last line of program text to be saved.

Purpose:

The SAVE DC command causes BASIC programs, or portions of programs, to be recorded on the designated disk platter. The file name, file type (program file), starting sector address, and ending sector address are entered in the Catalog Index, and the program is automatically stored, starting in a location determined by the system on the basis of the current entries in the Catalog Area.

The '$' parameter specifies that a 'read-after-write' verification test be performed to ensure that all program text is written correctly to the disk. The read-after-write check effectively doubles the time required for the SAVE DC operation, however.

Inclusion of the 'expression' parameter instructs the system to reserve a number of sectors in addition to the number actually needed to store the program at the end of the program file. These additional sectors can be used for future expansion of the program. The truncated value of 'expression' equals the number of extra sectors to be reserved. A new program also can be stored over a scratched program or data file on the disk, if the 'old file name' parameter is used. The 'old file name' parameter specifies the name of the scratched file, and the 'new file name' parameter indicates the name of the new program which is to be stored in its place. If the scratched file identified by 'old file name' does not occupy adequate space to hold the new program, an error results. When replacing an old program with a new one on disk, it is possible for 'old file name'and 'new file name' to be identical. Otherwise, 'new file name' must be unique.

If neither the 'old file name' nor the 'expression' parameter is included in the SAVE DC command, the system uses only the exact number of sectors required for the program being stored, and appends the new program file at the current end of the Catalog Area.

The 'new file name' parameter, which specifies the name of the program being saved, can be from one to eight characters in length, expressed as a literal string in quotes (i.e., "PROG 1"), or as the value of an alphanumeric variable (e.g., if A$ = "PROG 2", then A$ can be included as the 'new file name' parameter and the file is automatically named PROG 2).

The 'P' parameter indicates that the program is protected, and cannot be listed or resaved, although it can be loaded and run.

NOTE:

In order to save or list any program after a protected program has been loaded, it is necessary to clear all of memory either by executing a CLEAR command with no parameters, or by MASTER INITIALIZING the system (switching the main power switch on the Power Supply Unit OFF and then ON).

'Line Number 1' and 'line number 2' specify the first and last lines, respectively, of the program in memory which is to be saved. Both of these parameters are optional; if only 'line number 1' is included in the SAVE DC command, all program lines in memory beginning with that line are saved on disk. If neither line number is specified, all program text in memory is saved.

Examples:

```
SAVE DC F "CONVERT"
SAVE DC R "OUTPUT" 300, 500
SAVE DC T $ (100) #2, "OUTPUT 2"
SAVE DC F (A$) /320, B$
SAVE DC T #A, "COORD"
SAVE DC F ("OLD") "NEW"
SAVE DC FP "PROG 1"
```

General Form:

SCRATCH { F / R / T } [#n, / /xxx,] name [,name] . . .

where:

F = Fixed platter, Drive #1, Drive #3.

R = Removable platter, Drive #2.

T = 'F' platter or 'R' platter, depending on device type specified in the device address.

#n = A file number to which the disk address is currently assigned ('n' is an integer or numeric variable whose value is from 0 to 6).

/xxx = The device address of the disk. If neither #n nor /xxx is specified, or if n = 0, the default disk address (stored opposite #0 in the Device Table) is used. The system default disk address is 310.

name = The names of one or more cataloged files (program or data) to be scratched from the catalog. Each name must be from 1 to 8 characters long, and may be expressed as an alphanumeric variable or as a literal string enclosed in quotes.

Purpose:

The SCRATCH statement is used to set the status of the named disk file(s) to a scratched condition. The SCRATCH statement does not remove the files from the catalog; a subsequent listing of the catalog shows the normal information for both scratched and non-scratched files, as well as which files have been scratched. The program text or data in the scratched files is not altered or destroyed by the SCRATCH statement. Once files have been scratched, they cannot be accessed by DATALOAD DC OPEN or LOAD DC statements. They can, however, be renamed by DATASAVE DC OPEN statements or SAVE DC commands, and the sectors utilized by scratched files can be reused to save new programs or data files.

The SCRATCH statement is generally used prior to a MOVE statement. When a MOVE statement is executed, information concerning all scratched files is deleted from the Catalog Index, and the corresponding program text or data is deleted from the Catalog Area (see MOVE).

NOTE:

Until a MOVE is executed, all scratched file names remain in the Catalog Index, even if the space occupied by the files in the Catalog Area has been renamed and reused. In the latter case, the scratched file name no longer appears in a listing of the Catalog Index, but it continues to occupy space in the Index. A scratched file name is removed from the Index only when it is renamed with the same name, or when a MOVE is executed.

Examples:

```
    SCRATCH F "HEADER"
    SCRATCH R #2, "FLD4/15", "FLD10/7"
    SCRATCH R/320, "COLHDR"
10  SCRATCH F A$, B$, C$
10  SCRATCH F #2, "TEMP 1", A$
10  SCRATCH F #A2, "SORT", "MERGER"
```

General Form:

SCRATCH DISK {F / R / T} [#n, / /xxx,] [LS = expression 1 ,] END = expression 2

where:

F	=	Fixed platter, Drive #1, Drive #3.
R	=	Removable platter, Drive #2.
T	=	Either 'F' platter or 'R' platter, depending upon the device type specified in the device address.
#n	=	A file number to which the disk address is currently assigned ('n' is an integer or numeric variable whose value is from 0 to 6).
/xxx	=	The device address of the disk. If neither #n nor /xxx is specified, or if n = 0, the default disk address (stored opposite #0 in the Device Table) is used. The system default disk address is 310.
LS	=	A parameter specifying the number of sectors to be set aside for the Catalog Index.
expression 1	=	An integer or expression whose truncated value is from 1 to 255. If the 'LS' parameter is not included, the size of the Catalog Index is set automatically at 24 sectors.
END	=	A parameter specifying the last (highest) sector address in the Catalog Area.
expression 2	=	An expression whose truncated value must be less than or equal to the last (highest) sector address on the disk.

Purpose:

The SCRATCH DISK statement is used to reserve space for the Catalog Index and Catalog Area on a disk platter (each disk platter must be initialized separately) prior to saving program files or data files on the disk. This space must be reserved prior to the use of any other catalog statement; otherwise, an error is indicated.

When the SCRATCH DISK statement is executed, the system reserves a number of sectors, starting with sector number 0 on the specified platter, for a disk catalog. The 'LS' parameter defines the size of the Catalog Index, and the truncated value of 'expression 1' specifies the number of sectors to be reserved. A maximum of 255 sectors (sectors 0-254) can be reserved for the Index. If the 'LS' parameter is not included in the SCRATCH DISK statement, 24 sectors (sectors 0-23) are reserved automatically for the Index. The entry for each cataloged file in the Catalog Index consists of the file's name and associated sector address parameters; each sector of the Index can hold 16 file entries, with the exception of sector 0, which holds 15 entries (a small portion of sector number 0 contains systems information used to maintain the catalog). When the catalog is initially established, the remainder of sector number 0 and all other sectors reserved for the Catalog Index are filled with zeroes.

The END parameter defines the limit of the Catalog Area on disk. The truncated value of 'expression 2' specifies the address of the last sector to be used for storing cataloged files. The END parameter is particularly useful when temporary work files are to be established, since temporary files must be established outside the Catalog Area. An error will result if the user attempts to establish a temporary file within the Catalog Area.

The end of the Catalog Area can be altered with the MOVE END statement (see MOVE END).

> NOTE:
>
> Although, in general, the Catalog Area can be expanded or retracted when necessary with the MOVE END statement, the size of the Catalog Index cannot be altered once specified without reorganizing the entire catalog. Take special care, therefore, to provide ample space for future expansion when specifying the size of the Catalog Index in the 'LS' parameter.

Examples:

```
    SCRATCH DISK R END = 9791
    SCRATCH DISK F LS = 4, END = 1000
100 SCRATCH DISK F/320, END = X*2
100 SCRATCH DISK T #X, LS = L, END = E
```

General Form:

```
             ⎧F⎫ ⎡#n,  ⎤
      VERIFY ⎨R⎬ ⎢     ⎥   (expression 1, expression 2)
             ⎩T⎭ ⎣/xxx,⎦
```

where:

F	=	Fixed platter, Drive #1, Drive #3.
R	=	Removable platter, Drive #2.
T	=	'F' platter or 'R' platter, depending on the device type specified in the device address.
#n	=	A file number which the disk address is currently assigned ('n' is an integer or numeric variable whose value is from 0 to 6).
/xxx	=	Device address of disk. If neither #n nor /xxx is specified, or if n = 0, the default address (stored opposite #0 in the Device Table) is used. The system default disk address is 310.
expression 1	=	An expression whose truncated value equals the address of the first sector to be verified.
expression 2	=	An expression whose truncated value equals the address of the last sector to be verified.

Purpose:

The VERIFY statement reads all sectors within the specified range from the designated disk platter, and performs cyclic and longitudinal redundancy checks to ensure that information has been written correctly to those sectors. The truncated value of 'expression 1' specifies the address of the first sector to be verified, and the truncated value of 'expression 2' specifies the address of the last sector to be verified. If a cataloged platter is to be verified, 'expression 1' should be zero (the first sector address on the platter), while 'expression 2' should be set equal to the last sector in the Catalog Area. The ending sector address of the Catalog Area can be obtained by listing the Catalog Index (see LIST DC).

If one or more errors are detected, a list of the erroneous sectors is written on the currently selected Console Output device. The HALT/STEP key can be used to terminate the printout of erroneous sectors.

NOTE:

VERIFY can be used in both Automatic File Cataloging and Absolute Sector Addressing modes.

Examples:

```
10 VERIFY F #2, (500,500+L)
10 VERIFY T #A(1), (100,200)
10 VERIFY R (0,1023)
10 VERIFY F/320, (0,2000)
```

ERROR OUTPUT:

```
ERROR IN SECTOR 1097
ERROR IN SECTOR 8012
```

Chapter 9
Absolute Sector Addressing

9.1 INTRODUCTION

Absolute Sector Addressing Mode comprises nine BASIC statements and commands which enable the programmer to read or write information in specific sectors on the disk. No catalog or Catalog Index can be established or maintained in Absolute Sector Addressing Mode (except by user-supplied software), nor is it possible to name programs or data files. Files are identified only by reference to their starting sector addresses. Similarly, individual records must be saved into or loaded from a file by specifying a starting sector address. All file addressing information must be maintained by the programmer; such information is not maintained automatically by the system. Because the disk statements in Absolute Sector Addressing Mode provide direct access to individual sectors, they are referred to as "direct addressing" statements.

The direct addressing statements provide the programmer with a means of writing customized disk operating systems and special file access procedures such as binary searches, sorting routines, etc. which cannot be done efficiently - and, in some cases, which cannot be done at all - with catalog procedures alone. Two classes of statements are available in Absolute Sector Addressing Mode: the DA statements (where "DA" is a mnemonic for "direct address") and the BA statements (where "BA" is a mnemonic for "block address"). Both permit direct access to specific sectors on the disk.

The DA statements can be used to write or read programs or data records beginning at a specified sector on the disk. Multi-sector programs and data records are automatically read or written, just as they are with DC statements. All records saved with a DA statement or command are automatically formatted to contain the standard System 2200 control information (see Chapter 7, Section 7.6), and records loaded with a DA statement or command must contain this format information. Records created by DA statements or commands are, therefore, identical in format to records created by DC (catalog) statements or commands, and records saved in one mode may be retrieved in the other.

The BA statements comprise a special class of statements which read and write exactly one sector (256 bytes) of unformatted data. Records created with a DATASAVE DC or DATASAVE DA statement are automatically formatted by the system to contain certain control information. (Refer to Chapter 7, Section 7.6, for a discussion of the control information automatically included in each sector of a logical record.) When a data record is read from the disk with a DATALOAD DC or DATALOAD DA statement, the system expects to find the control information; a record which does not contain the expected control information cannot be read with a DC or DA statement. When a record is created with a DATASAVE BA statement, however, no control information is written by the system. In this special case, the programmer is free to write his own control information in each record, and to format his records in a way best suited for his application. Records with a non-standard format can be read with a DATALOAD BA statement; they cannot be read with DC or DA statements. DATALOAD BA can also be used to read sectors (program or data) written originally with a DC or DA statement.

Although no catalog or Catalog Index is established or maintained in Absolute Sector Addressing Mode, the Device Table is used as an intermediate storage location for certain sector address parameters used and returned by each direct addressing statement.

The information stored in the Device Table consists of the following items:

1. The starting sector address specified in the direct addressing statement. This value is stored in the Starting Sector Address location in a Device Table slot.

2. The highest possible sector address in the system (32767). This value is stored in the Ending Sector Address location in a Device Table slot.

3. The next sequential sector address (returned to a designated return variable in the direct addressing statement following statement execution). This value is stored in the Current Sector Address location in a Device Table slot.

If a file number (#1-#6) is included in the direct addressing statement, the above values are recorded in the associated slot in the Device Table; otherwise, they are stored in the default slot. Suppose, for example, a program occupying 10 sectors is saved with the command SAVE DA F (101, D). Following execution of this command, the default slot in the Device Table contains the following values:

```
START   = 101
END     = 32767
CURRENT = 111
```

Although this information is not much use to the programmer, it is important to realize that the Device Table is used in this way by direct addressing statements. The programmer must take precautions to avoid a conflict between catalog statements and direct addressing statements in their use of Device Table slots, since the parameters of a currently open cataloged file will be clobbered if the file number associated with those parameters is used in a direct addressing statement.

In addition to reading and writing information on the disk, Absolute Sector Addressing Mode also provides the capability to perform platter-to-platter copy operations and verify the transferred data. The Absolute Sector Addressing statements and commands are:

SAVE DA
LOAD DA (command)
LOAD DA (statement)
DATASAVE DA
DATALOAD DA
DATASAVE BA
DATALOAD BA
COPY
VERIFY

9.2 SPECIFYING SECTOR ADDRESSES

When a data record or program is saved or loaded with a direct addressing statement or command, the starting sector address must be specified by the programmer. The address may be supplied in the form of an expression, or as the value of an alphanumeric variable. If the address is supplied as the value of an alpha variable, the binary value of the first two bytes of that variable is interpreted as the sector address. The value of the expression or alpha variable must, of course, be less than or equal to the last (highest) sector address on the disk platter. After the statement is processed, the system automatically returns the address of the next available sector. A second alpha or numeric variable must be included in the statement to receive this address.

```
SAVE DA F (100,           A)
            ↑             ↑
        Specifies the     After execution of
        address of the    the SAVE DA command,
        first sector on   this variable contains
        the 'F' platter   the address of the next
        to be used to     available sector on the
        store the         'F' platter.
        saved program.
```

In order to economize on the use of memory and disk space, and to facilitate address calculations in binary, the beginning sector address and the next available sector address may be expressed as two-byte binary values (i.e., as the first two bytes of alphanumeric variables). A sector address expressed as a two-character binary number occupies only two bytes of memory or disk storage, while the same address expressed as a decimal value requires eight bytes of memory and nine bytes of disk storage. The savings in storage space gained by expressing the sector address in binary can become appreciable when, for example, key files are established to facilitate random access operations. Typically, a key file contains a list of keys along with the sector addresses of records identified by those keys. In a key file containing, say, 9,000 keys and sector addresses, some 7,000 bytes of disk storage (about 27 sectors) are saved by expressing the sector addresses in binary rather than decimal. If the starting sector address is to be expressed as a binary number, it must be specified as the value of an alphanumeric variable (the first two bytes are used). If the next available sector address is to be returned as a binary number, the receiving variable must be specified as an alphanumeric variable of at least two characters in length.

9.3 STORING AND RETRIEVING PROGRAMS ON DISK IN ABSOLUTE SECTOR ADDRESSING MODE

In Absolute Sector Addressing Mode, the programmer himself must keep track of each program's location on the disk. The starting sector address of the program must be directly specified by the programmer when writing or reading a program on disk; it becomes the responsibility of the programmer, therefore, to ensure that information already recorded on disk is not overwritten by each new program, and that the location of each program is saved for future reference. Because there are few cases in which the advantage to be gained in direct addressing program operations offsets the added complexities involved, program storage and retrieval are not commonly done in Absolute Sector Addressing Mode.

Apart from the important fact that a direct addressing statement must specify an absolute sector address rather than a file name, the SAVE DA and LOAD DA instructions are not remarkably different from their cataloging counterparts, SAVE DC and LOAD DC. Specifically, the format of a program file written on disk with SAVE DA is almost identical to that of a cataloged file written with SAVE DC. In both cases, the program file begins with a one-sector header record, and ends with a trailer record. (In cataloged program files, the header record contains the file name; in program files created with SAVE DA, the header record contains a field of blanks in place of a file name.) An additional sector of control information, the end-of-file control record, is written at the end of every cataloged program file by SAVE DC. This control record is not written in program files recorded with SAVE DA.

The close similarity between the formats of cataloged program files and those created with direct addressing statements makes it possible for programs recorded in catalog mode (with SAVE DC) to be read in direct addressing mode (with LOAD DA). LOAD DA, like LOAD DC, begins reading a program at the header record (the starting sector address of the program must, therefore, be known), and terminates reading when it encounters the trailer record. In this way, the entire program file is automatically read and loaded into memory. In normal operations, there is no advantage to be gained by loading cataloged programs with LOAD DA; it is generally safer and easier to use LOAD DC. The only situation in which it could be advantageous to employ LOAD DA for cataloged program files is recovery from an accident which destroys entries for one or more program files in the Catalog Index, without harming the programs themselves. In such a situation, the programs can be accessed only with direct addressing.

The LOAD DC statement cannot be used to read non-cataloged files recorded with SAVE DA. SAVE DA does not record the file name and sector address parameters in the Catalog Index when a file is saved, and LOAD DC cannot access a program without this information.

Saving Programs on Disk with SAVE DA

Programs are stored on disk in Absolute Sector Addressing Mode with a SAVE DA command. The following items of information must be included in the command:

1. The disk platter on which the program is to be stored (specified as 'F', 'R', or 'T').

2. The address of the first sector on the disk in which the program is to be stored (specified as an expression or alphanumeric variable).

3. A numeric or alphanumeric return variable designated to receive the address of the first free sector following execution of the SAVE DA command.

4. Optionally, one or two line numbers identifying the program lines which are to be saved on disk. If one line number is specified, all program lines beginning at that line are saved on disk. If two line numbers are specified, all program lines between and including those two lines are saved. If no line number is specified, all resident program text is saved.

Example 9-1: Saving Program on Disk with SAVE DA
(No Line Numbers Specified)

```
SAVE DA F (1250,L)
```

This command (SAVE DA is not programmable) causes all program lines currently in memory to be saved on the 'F' disk platter, beginning at sector 1250. As many sectors are used as are needed to store the resident program text. Following execution of the command, the address of the next available sector is returned to numeric variable L as a decimal value. For example, if the program required 10 sectors on disk (sectors 1250-1259), then L = 1260 following execution of the command.

Example 9-2: Saving a Program on Disk with SAVE DA
(Two Line Numbers Specified)

```
SAVE DA R (1300,N) 100, 750
```

SAVE DA causes lines 100 through 750 to be recorded on disk starting at sector 1300, and uses as many sectors as it needs to store the program. When the program is recorded, the address of the next available sector is returned to variable N.

Retrieving Programs from Disk with LOAD DA

The LOAD DA instruction, like its catalog counterpart LOAD DC, is a hybrid having two distinct forms, the LOAD DA command and the LOAD DA statement. As with LOAD DC, the two forms of LOAD DA have significantly different functions, and must be discussed separately. In both forms of LOAD DA, however, the starting sector address of the program to be loaded must be specified. It is important to note in this context that LOAD DA always expects to read a complete program, beginning with a header record, including one or more program records, and ending with a trailer record. For this reason, it is not possible to begin program loading in the middle of a program, or at any point beyond the program header record. For example, if the starting sector address of a program is sector #100, and the starting address specified in a LOAD DA instruction is 101 or beyond, the program is not loaded. In some cases, this situation causes LOAD DA to search forward on the disk for the next sequential program header record, and to automatically load that program; in other cases, the processor simply hangs up, and must be reinitialized with RESET. In any case, this is not a recommended procedure.

The LOAD DA Command

The LOAD DA command causes a program to be read from disk, beginning at a specified sector address, and appended to existing program text in memory. Program lines in memory having the same numbers as lines in the newly loaded program are cleared and replaced by the new lines. Resident program lines with different line numbers are not cleared, however, and remain in memory following the LOAD DA operation. For this reason, resident program text should generally be cleared with a CLEAR or CLEAR P command prior to loading in the new program.

Example 9-3: Loading a Program from Disk with LOAD DA Command

```
CLEAR
LOAD DA F (100, D)
```

The LOAD DA command causes the system to load a BASIC program from the 'F' platter beginning at sector 100 (if sector 100 does not contain a program header record, the results of statement execution are unpredictable). When the program has been loaded, the address of the next sequential sector following the trailer record is returned to variable D. (For example, if the program trailer record resides at sector #112, D = 113 following execution.)

The LOAD DA Statement

The operation of the LOAD DA statement is analogous to that of the LOAD DC statement. LOAD DA permits programs to be loaded from a specified sector location on disk under program control. Prior to loading the program from disk, LOAD DA automatically clears out all or a specified portion of the resident program text, as well as all noncommon variables. (Common variables are not cleared.) Once loaded in memory, the new program is executed automatically.

The LOAD DA statement contains the following parameters:

1. A platter parameter ('F', 'R' or 'T').

2. The starting sector address of the program to be loaded, specified as an expression or alpha variable. This address must be the address of the program header record.

3. A numeric or alphanumeric return variable designated to receive the address of the next sequential sector following the program trailer record. (Note: This variable must be a common variable.)

4. Optionally, one or two program line numbers defining the portion of resident program text which is to be cleared prior to loading the new program. Inclusion of one line number causes all program lines beginning at that line to be cleared. Inclusion of a pair of line numbers causes all program lines between and including the two specified lines to be cleared. Omission of both line numbers causes the totality of resident program text to be cleared.

Example 9-4: Loading Programs from Disk with a LOAD DA Statement (No Line Number Specified)

```
10 COM D
50 LOAD DA F (24,D)
```

Statement 50 causes a program to be loaded from the 'F' platter beginning at sector 24. Prior to loading in the new program, all program text in memory is cleared, along with all non-common variables. After the new program has been loaded, program execution begins automatically at the lowest program line. The address of the next sequential sector following the program trailer record is returned as a decimal value to numeric variable D. For example, if the program trailer record is located in sector #33, then D = 34 following execution of statement 50. (Note: D must have been specified as a common variable in a COM statement prior to execution of the LOAD DA statement.)

Note that the return variable ('D' in the above example) must be a common variable; otherwise, it is cleared along with all other noncommon variables before the program is loaded, and an Error 87 (Common Variable Required) is signalled.

The LOAD DA statement, like LOAD DC, can be used to load program overlays from disk and append them to an existing program in memory. In this case, one or both of the optional line number parameters are specified to define the portion of resident program text which must be cleared prior to loading the program overlay. Note that when one or both line numbers are included, execution of the overlay begins automatically at the first line number specified in the LOAD DA statement. If the new program does not contain a line having the first line number specified, an ERROR 11 (Missing Line Number) is signalled.

If the program overlays are stored in sequential areas of the disk, it is possible to use the same variable to contain the starting sector address and receive the address of the next available sector following statement execution. In this way, the starting sector address is automatically updated to the address of the next available sector every time the LOAD DA statement is executed. Note that this technique must be modified if cataloged programs are read, since a cataloged program has an additional system end-of-file sector following the trailer record which is not read as part of the program by LOAD DA, and the address of this sector will be returned by the LOAD DA statement. For normal processing, it is recommended that cataloged programs be read only with the catalog instruction LOAD DC.

Example 9-5: Loading Program Overlays from the Disk with the LOAD DA Statement (Two Line Numbers Specified)

```
80  COM D
90  D = 24
.
.
.
500 LOAD DA F (D,D) 100,500
```

Statement 500 causes a program to be loaded into memory from the 'F' platter, starting at the sector whose address is stored in D. Prior to loading the program overlay, program lines 100 through 500 are cleared from memory, along with all non-common variables. After the program has been loaded, program execution begins automatically at line 100. Following statement execution, the address of the next available sector is returned to D (however, D must have been specified as a common variable). When Statement 500 is executed a second time, the new value of D is the starting sector address of the second program overlay (assuming that the overlays are stored sequentially on the disk, and that they are not cataloged files.) The second overlay is automatically loaded over the first overlay, and run from line 100. The process can be continued in this way for as long as necessary.

9.4 STORING AND RETRIEVING DATA ON DISK IN ABSOLUTE SECTOR ADDRESSING MODE

In Absolute Sector Addressing Mode, named data files are not maintained by the system, nor are the file parameters stored in the Catalog Index or Device Table. However, the system does write certain sector address information in the default slot (or in one of the other slots, #1 - #6, if a file number is specified in the statement) in the Device Table every time a logical record is saved or loaded with a DA statement. If the referenced file number happens also to be associated with a currently open cataloged file, the parameters of the cataloged file will be wiped out. To avoid this problem, always use different file numbers for direct addressing statements and catalog statements when the two modes are utilized concurrently.

Storing Data on the Disk

Data is stored on the disk in Absolute Sector Addressing Mode with the DATASAVE DA statement. At least four items of information must be provided in the statement:

1. The disk platter on which the data is to be saved (specified by 'F', 'R', or 'T').

2. The address of the first sector on that platter in which the data is to be stored (specified as an expression or alphanumeric variable).

3. A numeric or alphanumeric variable which is to receive the address of the next available sector following statement execution.

4. The data which is to be saved in a record on the disk.

Each DATASAVE DA statement (like DATASAVE DC) saves one logical record, consisting of enough sectors on disk to store all data specified in the argument list. Records saved on the disk with DATASAVE DA are identical in format to those created by DATASAVE DC, and contain the standard System 2200 format information. Records initially saved with DATASAVE DA can therefore be loaded with DATALOAD DC. Note, however, that when DATASAVE DA is used to write a record in a cataloged data file, it does not update the file parameters in the Catalog Index. In normal processing operations, the use of direct addressing statements to alter cataloged files is not recommended.

Example 9-6: Storing Data on Disk with a DATASAVE DA Statement

```
100 B$ = HEX(01E0)
150 DATASAVE DA R (B$,X$) A, B(), C()
```

Statement 150 causes the value of numeric variable A and arrays B() and C() to be stored on the 'R' platter, starting at sector 480 (480 is the decimal equivalent of HEX(01E0), which is the value of B$). One logical record is written containing enough sectors to store all data specified in the argument list. Following the execution of statement 150, X$ is set equal to the binary address of the next available sector. For example, if A, B(), and C() require nine sectors on the disk, the value of X$ following statement execution is HEX(01E9) (decimal equivalence, 489).

If a number of records are to be saved or loaded in sequential sectors on the disk, it is possible to use the same variable to contain the starting sector address and receive the address of the next available sector following statement execution. In this way, the starting sector address is automatically updated to the address of the next available sector following each save or load operation.

Example 9-7: Saving a Number of Data Records in Sequential Areas of the Disk

```
200 DIM B(25)
210 A1 = 50
220 FOR I = 1 TO 25
230 INPUT "VALUES FOR THIS RECORD", B(I)
240 NEXT I
250 DATASAVE DA F(A1,A1) B()
:
:
250 GOTO 220
```

The starting sector address (A1) is initially set to 50. At line 230, the values to be stored in the first record are entered. The first time through the loop, line 250 saves array B() on the 'F' platter beginning at sector 50. When the record has been written, the address of the next available sector is returned in A1. Assuming that B() required ten sectors, A1 is set equal to 60 following execution of statement 250. The second time through the loop, array B() is saved on the 'F' platter beginning at sector 60, since this is the new value of A1. The process may be continued in this way in order to store records in sequential areas of the disk.

After all data records have been saved in a file, the file should be ended with an end-of-file trailer record, which can be used subsequently to test for the end-of-file if the records are read sequentially for processing. In Absolute Sector Addressing Mode, the trailer record is the programmer's only way of protecting himself from reading beyond the legitimate data in a file (unless he designs his own trailer record), since the data file has no absolute limit (as it does in catalog mode). If no trailer record is written, the program may read beyond the limit of legitimate data in the file and retrieve meaningless data from the subsequent, unused sectors. An end-of-file record is written in Absolute Sector Addressing Mode exactly as it is written in Catalog Mode, by specifying the "END" parameter instead of an argument list in a DATASAVE DA statement:

Example 9-8: Writing an End-Of-File Record in a Data File with a DATASAVE DA END Statement

```
180 DIM B(25)
190 FOR I=1 TO 25
200 INPUT "VALUES FOR THIS RECORD", B(I)
210 NEXT I
220 DATASAVE DA R (A1,A1) B()
230 INPUT "IS THIS THE LAST RECORD? (Y OR N)", F$
240 IF F$ = "Y" THEN 350
250 GOTO 210
:
:
350 DATASAVE DA R (A1,A1) END
```

This routine illustrates a simple input loop in which the operator is asked after entering each record if it is the last record. If a response of "N" (or any response other than "Y") is entered, the routine loops back to input another record. When a response of "Y" is entered, however, the routine branches to line 350 and writes an end-of-file record in the file.

When a new record is written into a file which has been ended with a trailer record, the trailer record should be overwritten, and a new trailer record created following all subsequent data saving operations. For example, if the trailer record occupies sector 497 in a file, the next data record should be saved beginning at sector 497, and a new trailer record written following the save operation.

<u>Retrieving Data from Disk</u>

Data is retrieved from a data file on the disk in Absolute Sector Addressing Mode with a DATALOAD DA statement. Four items of information must be specified:

1. The disk platter on which the data is stored (specified by 'F', 'R', or 'T').

2. The address of the first sector on that platter from which data is to be read (specified as an expression or alphanumeric variable).

3. A numeric or alphanumeric return variable designated to receive the address of the next sequential logical record following statement execution.

4. An argument list consisting of one or more alpha or numeric receiving variables, arrays, or array elements designated to receive the data read from the disk.

Example 9-9: Retrieving Data from a Data File on Disk with a DATALOAD DA Statement

```
300 DATALOAD DA R (481,B2) A,B,C
```

Statement 300 causes the system to load data from the 'R' platter beginning at sector 481, and store the data in numeric variables A, B, and C in memory. Enough data is read from the disk to fill al' variables specified in the argument list (unless the trailer record is encountered, at which point reading stops). However, it is recommended that exactly one logical record be read with each DATALOAD DA statement. In order to read one logical record, the argument list of the DATALOAD DA statement must correspond to the argument list of the DATASAVE DA statement which originally saved the record. If only the first few fields in a logical record are loaded, the remaining fields in the record are read but ignored. If the argument list contains more receiving arguments than there are fields in a logical record, values are read from the next sequential logical record until the argument list is filled. The remainder of the second record is then read and ignored. Following statement execution, the return variable B2 is set to the address of the next sequential logical record. Thus, if the record occupies three sectors (481, 482, 483), B2 = 484 following statement execution.

If an end-of-file record has been written in the data file, it is possible to test for the end-of-file condition with an IF END THEN statement. The IF END THEN statement is useful when processing records sequentially from a file, since it terminates reading and initiates a branch to a specified line number when the end-of-file record is read. The end-of-file record is not transferred into the DATALOAD DA argument list, and the value of the return variable in the DATALOAD DA statement is set to the address of the end-of-file record rather than to the next sequential sector. The system is therefore positioned to save a new record over the EOF record if additional data is to be stored in the file.

Example 9-10: Testing for the End-Of-File Condition in a Non-Cataloged Data File

```
400 DATALOAD DA R (B2,B2) A()
410 IF END THEN 500
:
:
490 GOTO 400
500 STOP
```

Statement 400 loads one logical record from the 'R' platter, beginning at the sector whose address is stored in B2, and stores the data in array A(). Statement 410 checks for an end-of-file trailer record (previously written with a DATASAVE DA END statement). If the trailer record is detected, the program skips to statement 500 and stops. If no trailer record is detected, program execution continues normally, with data in A() being processed until, at statement 490, the system is instructed to loop back and load in another record. Note that when the trailer record is read, the receiving variable (B2) is set to the address of the trailer record, not the address of the next consecutive sector.

9.5 THE 'BA' STATEMENTS

Two special statements, DATASAVE BA and DATALOAD BA, enable the programmer to save and load records that do not contain the standard System 2200 control information (such records cannot be saved or loaded with DC or DA statements). Since records saved or loaded with a BA statement are not formatted automatically with System 2200 control information, the programmer is free to write his own control information, and format his records in a manner appropriate to his application. Records which are saved with a DATASAVE BA statement must be loaded with a DATALOAD BA statement. The DATALOAD DC and DATALOAD DA statements cannot be used to read a record which was saved initially with DATASAVE BA. However, DATALOAD BA can be used to read sectors which were written initially with DC or DA statements or commands.

The DATASAVE BA statement writes exactly one sector (256 bytes) of unformatted data from an alphanumeric array into a specified sector on the disk. A single alphanumeric array must be used in the DATASAVE BA argument list; alpha variables, as well as numeric variables and arrays, are illegal. Multiple arguments are not permitted. It is not possible to write a multi-sector record with a single DATASAVE BA statement. If the alpha array in the DATASAVE BA argument list contains more than 256 bytes of data, the additional data is ignored. If the array contains fewer than 256 bytes, the remainder of the sector being addressed is filled with meaningless data. It is therefore always advisable to specify an array which contains at least 256 bytes of data in the DATASAVE BA argument list. Four items of information must be specified in the DATASAVE BA statement:

1. The disk platter on which the data is to be stored (specified by 'F', 'R', or 'T').

2. The address of the sector in which the data is to be written (multi-sector records are not written automatically);

3. A numeric or alphanumeric return variable designated to receive the address of the next consecutive sector following statement execution.

4. One alphanumeric array containing the data to be saved on the disk. (It is recommended that the array contain 256 bytes of data.)

Example 9-11: Writing an Unformatted Sector with DATASAVE BA

```
200 DATASAVE BA F (L$,L$) A$()
```

Statement 200 causes 256 bytes of unformatted data to be transferred from array A$() into the sector on the 'F' platter whose address is stored in alpha variable L$. If A$() contains more than 256 bytes of data, the additional data is ignored. If A$() contains fewer than 256 bytes of data, the remainder of the sector is filled with garbage. Following statement execution, the address of the next consecutive sector is returned to L$ (i.e., if L$ = HEX(01E0) prior to execution of statement 200, then L$ = HEX(01E1) following statement execution).

The DATALOAD BA statement loads exactly one sector (256 bytes) of data from a specified sector on the disk into a specified alphanumeric array in memory (numeric arrays, as well as alpha and numeric variables and array elements, are illegal). The receiving array must be dimensioned to contain at least 256 bytes. If the array contains fewer than 256 bytes, an error is signalled and the data is not transferred; if the array contains more than 256 bytes, the additional bytes are undisturbed. It is not possible to read multi-sector records with the DATALOAD BA statement. The DATALOAD BA statement must include the same four elements specified in the DATASAVE BA statement (i.e., disk platter to be accessed, address of sector to be loaded, variable specified to receive address of next consecutive sector, and alpha array specified to receive data read from disk).

Example 9-12: Reading a Sector from Disk with DATALOAD BA

```
240 DIM A$(16)16
250 DATALOAD BA F (20,L) A$()
```

Statement 250 causes all information stored in sector 20 on the 'F' platter (256 bytes) to be loaded into alpha array A$() in memory. A$() is dimensioned at line 240 to contain 256 bytes of data. If A$() held fewer than 256 bytes, an error (Error 60) would be signalled. Following execution of the statement, the address of the next consecutive sector is returned in numeric variable L (i.e., following statement execution, L=21). If A$() were dimensioned larger than 256 bytes, the additional bytes of A$() would remain unaltered.

> NOTE:
>
> As with the DA statements, the BA statements utilize the default slot in the Device Table (or one of the other slots, #1 - #6, if a file number is specified) to store sector address information. BA statements must, therefore, be assigned different file numbers from DC statements when the two modes are used concurrently.

9.6 PLATTER-TO-PLATTER COPY

Absolute Sector Addressing Mode provides the capability to copy all or part of the contents of one disk platter onto the other with the COPY statement. The entire contents of a disk platter, or any specified portion of its contents, can be copied from one disk platter to the other. Unlike MOVE (see the discussion of MOVE in Chapter 5), COPY transfers all information located on the portion of the disk platter which is to be copied (including scratched and temporary files) to the corresponding sectors on the second platter. The beginning and ending sector addresses of the portion of the disk platter which is to be copied must be specified. If the entire disk platter is to be copied, the starting sector address should be 0 and the ending sector address should be the highest sector address on the platter. If the catalog is to be copied, the Catalog Index must be copied along with the Catalog Area. In this case, the starting sector address must be 0, and the ending sector address must be the last sector in the Catalog Area. The ending sector address of the Catalog Area can be determined by listing the Catalog Index. However, it is recommended that MOVE be used instead of COPY when transferring the catalog from platter to platter (since in that case scratched files are automatically deleted).

Example 9-13: Copying a Disk Platter

```
10 COPY RF (0, 2399)
```

Statement 10 causes the contents of sectors zero through 2399 to be transferred from the 'R' disk platter to the corresponding sectors (0 - 2399) on the 'F' disk platter.

If it is convenient, the starting and ending sector addresses may be expressed as the values of numeric variables or expressions.

Example 9-14: Copying a Disk Platter

```
5  A = 10
10 COPY/320, FR (A,A*100)
```

Statement 10 causes sectors 10 (the value of A) through 1000 (the value of A*100) to be transferred from the 'F' disk platter to the same sectors of the 'R' disk platter. Both platters are located in the disk drive with address 320.

Following a COPY operation, the transferred information should be checked to ensure that it has been transferred correctly. The VERIFY statement is used to perform such a validation check. If the entire contents of the disk platter are copied, the entire platter can be checked by executing a VERIFY statement which specifies sector 0 as the starting address, and the address of the last sector on the platter as the ending address. If only a specific portion of a platter is transferred, the VERIFY statement can be used to verify only that portion of the second platter.

Example 9-15: Verifying Data Transfer Following a COPY Operation

```
10 COPY RF (0,1000)
20 VERIFY F (0,1000)
```

Statement 10 copies sectors zero through 1000 from the 'R' platter to the same sectors on the 'F' platter. Statement 20 verifies the newly-copied sectors 0 - 1000 on the 'F' platter.

If the check performed by VERIFY is positive, the system returns the CRT cursor and colon to the screen, indicating that the information has been copied accurately. If one or more errors are discovered, the system returns an error message indicating which sector(s) did not copy properly, e.g.,:

ERROR IN SECTOR 946.

If you encounter an error following a COPY operation, repeat the COPY. Repeated failure could indicate a faulty disk platter. If the error persists with another platter, call your Wang Service Representative.

VERIFY can be used to verify any portion of a disk platter, or an entire platter, for any reason. It need not be used only in conjunction with COPY. It may be useful, for example, to verify data on a previously recorded platter before the platter is reused. Many programmers verify each platter at the beginning of daily operation. The CRC and LRC checks performed by VERIFY provide an extra measure of protection against the accidental use of invalid data in important applications.

WARNING:

It is important that backup copies of important disk-based data files be maintained and kept up to date. Like other storage media, disk platters can be worn out with repeated use, and they are, of course, subject to accidental damage or destruction. To avoid the necessity of recreating your data base following such a potential disaster, you should always maintain one or more backup platters containing all important files. Non-cataloged files can be copied to a backup platter with the COPY statement. For cataloged files, the MOVE statement should be used. The Model 2270-1, which does not have a COPY capability, should be backed up on tape cassette.

> NOTE TO OWNERS OF THE
> MODEL 2270-1 AND 2270-3:
>
> On the Model 2270-1, the COPY statement is illegal. It is not possible to COPY information from one Model 2270-1 to a second disk unit.
>
> On the Model 2270-3, it is illegal to attempt a COPY operation to or from the #3 drive. If the #3 diskette is to be copied, it must be physically removed from the #3 disk drive and inserted into drive #1 or #2.

9.7 USING ABSOLUTE SECTOR ADDRESSING STATEMENTS IN CONJUNCTION WITH CATALOG PROCEDURES (BINARY SEARCH)

In the concluding paragraph of Chapter 7, it was pointed out that Absolute Sector Addressing statements can be used in conjunction with catalog procedures to develop more versatile and efficient file access techniques. One of the data retrieval techniques most commonly used on data files is the binary search technique. The System 2200 provides a special BASIC verb, LIMITS, which can be used in conjunction with direct addressing statements to perform a binary search on cataloged files. (LIMITS is discussed in Chapter 7, Section 7.7.)

A binary search is a technique for locating a particular record in a file by searching successively smaller segments of the file until the record is found. The procedure is somewhat as follows: the highest and lowest records in the file are first checked; if neither of them is the desired record, the middle record in the file is checked. If the middle record is not the desired record, then the sought-after record must be located either in the top half of the file (that is, between the highest record and the middle record), or in the lower half of the file (that is, between the lowest record and the middle record). The middle record in the appropriate half is then checked, and the process of performing successive bifurcations continues until the record is found (or until it is determined that no such record exists in the file). Clearly, a binary search cannot efficiently be performed if the file is not sorted in ascending or descending order.

The use of a binary search can be illustrated with an example from industry. Consider a small company which maintains a customer file on disk. In the simplest case, each record in the file might contain only three fields, a three-digit customer I.D. number, the customer's name, and the customer's credit rating:

I.D.#	NAME	CREDIT RATING
062	JOHN Q. TRAPP	A1

Figure 9-1. Typical Entry in Customer Credit File

The customer credit file is a cataloged file named CREDIT, in which each record occupies a single sector. The file begins at Sector #100, and is sorted in ascending order on the customer I.D. numbers.

SECTOR #	I.D.#	NAME	CREDIT
100	007	FRANKLIN, FREDERICK	A-2
101	011	GEROME, HERBERT	B-1
102	012	.	.
103	013	.	.
104	017	.	.
105	022	.	.
106	025	.	.
107	037	.	.
108	039	.	.
109	052	.	.
110	055	FRACK, ALFRED R.	A-1
111	062	.	.
112	073	.	.
113	101	.	.
114	111	.	.
115	123	.	.
116	128	MARSH, DAVID H.	C-3

Figure 9-2. Typical Customer Credit File (Sorted in Ascending Order)

As you can see, the file contains 17 records. Suppose, now, that one of the customers, Alfred R. Frack, applies for additional credit. Before granting this credit, the credit manager will want to check Mr. Frack's credit rating. One way to locate Mr. Frack's record is to search sequentially through the file until his customer I.D. (055) is found. In the sample file, this technique involves reading and checking 11 records, or slightly more than one half the total number of records in the file. A faster and more efficient way to find the record is to search the file with a binary search. The procedure is as follows:

1. Begin by checking the first (lowest) record in the file and the last (highest) record, to see if either of them is the desired record. In this case, neither the first record (I.D.#007) nor the last record (I.D.#128) is the desired record.

2. Next, check the middle record in the file. To find the sector address of this record, add the sector address of the last (highest) record in the file (116) to the sector address of the first (lowest) record in the file (100), and take the integer value of the average:

   M = INT((H+L)/2)
   M = INT((116+100)/2)
   M = 108

   For the first search, the highest address is 116 (H=116), and the lowest is 100 (L=100). Thus, M=108. The first sector to be accessed is sector 108.

3. Compare the key of this record (I.D. #039) with the desired key (I.D. #055). Since the desired key 055 is greater than the middle key 039, it must be located in the top half of the file (that is, between sectors 108 and 116).

4. Using the middle sector address (108) as the new low sector address, find the middle record in the top half of the file, midway between sector 108 and sector 116. In this case, INT((108+116)/2)=112.

5. Retrieve sector 112 and compare its key (I.D. #073) with the desired key (I.D. #055). Since 073 is larger than 055, the desired record must be in the lower quarter of this half of the file (i.e., between sector 108 and sector 112). Using sector 112 as the new high address, find the sector midway between 108 and 112. INT((108+112)/2)=110. Compare the key of sector 110 (I.D. #055) with the desired key (I.D. #055). Since the keys match, sector 110 contains the desired record, and the search is finished.

	1st Search: Sector Address	Key		2nd Search			3rd Search	
	100	007						
	101	011						
	102	012						
	103	013						
	104	017						
	105	022						
	106	025						
	107	037						
middle →	108	039		108	039			
	109	052		109	052			
	110	055		110	055			
	111	062		111	062			
	112	073	middle →	112	073		108	039
	113	101		113	101	middle	109	052
	114	111		114	111	and →	110	055
	115	123		115	123	desired	111	062
	116	128		116	128	record	112	073

Figure 9-3. Binary Search Technique

Although this example presumed an odd number of records in the file, the technique is the same for a file which contains an even number of records. A more serious problem is presented by files in which each record consists of two or more sectors. In such a case, the number of sectors in each record must be taken into account when calculating the record addresses on each search. It is impossible to conduct a binary search if the number of sectors per record is not constant.

In order to conduct a binary search on a file, then, there are three requirements:

1. The file must be sorted.
2. The number of sectors per record must be constant.
3. The limits of the file (i.e., beginning and ending sector addresses) must be known.

For cataloged files, the beginning and ending sector addresses can be obtained under program control with the LIMITS statement.

It may be obvious that the ending sector address of a cataloged file should not be used as the upper limit of the file, unless the file is filled with data. Use of the ending sector address as the upper limit when the file is not full decreases the efficiency of the binary search, since one or more searches may be wasted searching the empty sectors between the end-of-file trailer record and the last sector of the file (or, those unused sectors may contain meaningless data - including old program text - which would cause an error when the DATALOAD DA statement attempts to read it). It is generally safer and more efficient to use the address of the last data record as the upper limit of the file in a binary search, since all sectors between the beginning of the file and the last data record are certain to contain valid data. The address of the last data record in a file is computed by subtracting 1 from the address of the end-of-file trailer record. The address of the trailer record can be computed by first executing a LIMITS on the file (with the file name specified), then subtracting 2 from the number of sectors used in the file, and adding this value to the starting sector address of the file. Thus, to determine the address of the trailer record in the file "CREDIT", first execute a LIMITS:

```
20 LIMITS F "CREDIT", A1, A2, A3
```

Since the file name is specified rather than a file number, LIMITS accesses the Catalog Index on the 'F' platter and retrieves the Starting and Ending sector addresses, and Number of Sectors Used, for CREDIT. Variable A1 contains the starting sector address, variable A2 the ending sector address, and variable A3 contains the number of sectors used. The address of the trailer record then is computed with the following formula:

```
T = Starting + (Used -2)
T = A1 + (A3-2)
```

The address of the trailer record is stored in variable 'T'. The sector address of the last data record in the file may now be found merely by subtracting one from the address stored in 'T':

```
H = T-1
```

Here the address of the last data record is stored in variable 'H'. This address is used as the upper limit of the file for the first dichotomy in the binary search. The following example program illustrates the binary search described above on the customer credit information file, "CREDIT".

Example 9-16: Performing a Binary Search on a Cataloged Data File

```
5   REM ****** BINARY SEARCH ROUTINE ******
10      DIM R$3, A$3, F$26, C$4
20      LIMITS F "CREDIT",A1,A2,A3
25  REM ****** COMPUTE ADDRESS OF LAST DATA RECORD ******
30      T = A1+(A3-2)
40      H = T - 1
50  REM ****** ENTER KEY OF DESIRED RECORD ******
60      INPUT "DESIRED I.D.",R$
70  REM ****** READ & CHECK LOWEST RECORD ******
80      DATA LOAD DA F (A1,S) A$,F$,C$
90      IF A$ = R$ THEN 260
100 REM ****** READ & CHECK HIGHEST RECORD ******
110     DATA LOAD DA F (H,S) A$,F$,C$
120     IF A$ = R$ THEN 260
130 REM ****** COMPUTE MIDDLE SECTOR ADDRESS ******
140     M = INT((A1+H)/2)
150 REM ****** READ & CHECK MIDDLE RECORD ******
160     DATA LOAD DA F (M,S) A$,F$,C$
170     IF A$ = R$ THEN 260
180 REM ****** IS DESIRED KEY HIGHER OR LOWER THAN KEY READ? ******
190     IF R$ < A$ THEN 210
200     A1 = M
201     GOTO 230
210     H = M
220 REM ****** HAVE ALL RECORDS BEEN CHECKED? ******
230     IF H = M+1 THEN 280
240     GOTO 140
250 REM ****** RECORD FOUND - PRINT RECORD ******
260     PRINT A$,F$,C$
265     STOP
270 REM ****** RECORD NOT FOUND - PRINT ERROR MESSAGE ******
280     STOP "RECORD NOT IN FILE"
```

Statement 20 performs a LIMITS on the cataloged file CREDIT; the starting sector address of CREDIT is returned to A1, the ending sector address to A2, and the number of sectors used, to A3. Statement 30 calculates the address of the trailer record in CREDIT by subtracting 2 from the number of sectors used (A3), and adding this value to the starting address (A1). The resultant address is stored in T. Statement 40 computes the address of the last data record by subtracting 1 from 'T'. Line 60 is an INPUT statement which requests the key for the desired record. Line 80 loads in the first record of the file; its key is checked against the specified key. If there is no match, the highest record in the file is loaded (line 110), and its key is checked (line 120). If neither the first nor the last record is the desired record, the address of the middle record is computed (line 140), and this record is read and checked. If the middle record does not hold the desired key, the process is repeated on the upper or lower half of the file, depending upon whether the desired key is larger or smaller than the middle record key (lines 190, 200). The process continues either until the desired record is found (in which case it is printed), or until it is determined that no such record exists in the file (in which case an error message is displayed).

9.8 CONCLUSION

Direct addressing statements and commands can be used in conjunction with catalog procedures to develop an efficient and versatile data management system. One technique which might be used in such a system is the binary search technique discussed in the preceding section. A variety of different techniques also are available, and the interested reader is directed to the bibliography in Appendix H for a list of texts which discuss disk file access techniques. The direct addressing statements need not, of course, be regarded as merely supplemental to and supportive of catalog procedures. On the contrary, highly sophisticated and complex data management systems can be constructed in Absolute Sector Addressing Mode exclusively. The bibliography in Appendix H also lists a number of texts which discuss disk management system design concepts and philosophies.

Chapter 10
Absolute Sector Addressing Statements and Commands

10.1 INTRODUCTION

This chapter contains descriptions of and General Forms for the following Absolute Sector Addressing statements and commands, listed alphabetically for ease of reference:

COPY
DATALOAD BA
DATALOAD DA
DATASAVE BA
DATASAVE DA
LOAD DA (command)
LOAD DA (statement)
SAVE DA

10.2 STATEMENT/COMMAND DISTINCTION AND GENERAL RULES OF SYNTAX

Refer to Chapter 8, Section 8.2, for an explanation of the distinction between System 2200 BASIC statements and commands.

Refer to Chapter 8, Section 8.3, for a list of the rules of syntax and notation used in the General Forms.

COPY

General Form:

COPY [#n, | /xxx,] {RF | FR} (expression 1 , expression 2)

where:

- #n = A file number to which the disk address is currently assigned ('n' is an integer or numeric variable whose value is from 0 to 6).
- /xxx = The device address of the disk. If neither #n nor /xxx is specified, or if n = 0, the default disk address (stored opposite #0 in the Device Table) is used. The system default disk address is 310.
- RF = Copy the specified sectors from the 'R' disk platter to the 'F' disk platter.
- FR = Copy the specified sectors from the 'F' disk platter to the 'R' disk platter.
- expression 1 = An expression whose truncated value equals the address of the first sector to be copied.
- expression 2 = An expression whose truncated value equals the address of the last sector to be copied.

Purpose:

The purpose of the COPY statement is to copy information from one disk platter to another. The truncated value of 'expression 1' represents the address of the first sector to be copied, and the truncated value of 'expression 2' represents the address of the last sector to be copied. The information is copied from the first platter to the same sectors on the second platter. If the 'RF' parameter is used, the copying is from the 'R' platter to the 'F' platter. If 'FR' is used, the copying is from the 'F' platter to the 'R' platter.

The COPY statement is generally used to make backup copies of non-cataloged files. When files are copied from one disk to another using COPY, no deletion of scratched files occurs. If COPY is used to copy a catalog, the Catalog Index must be copied along with the entire Catalog Area; 'expression 1' is set to zero in this case, while 'expression 2' is set to the ending sector address of the Catalog Area. The ending sector address of the Catalog Area can be obtained by executing a LIST DC statement. However, it is recommended that MOVE be used instead of COPY to back up a catalog.

When COPY is executed as either a command or program statement, 1024 bytes of System 2200 memory must be available for buffering (that is, at least 1,024 bytes of memory must not be occupied by a BASIC program or variables); otherwise, an error message results and the COPY is not performed. The large buffer minimizes the time required for the COPY operation.

Following the COPY, a VERIFY statement can be executed to insure that the specified information was copied correctly.

> NOTE:
>
> COPY can be used in both Automatic File Cataloging Mode and Absolute Sector Addressing Mode.

Examples:

```
10 COPY RF (0,49)
10 COPY #2, RF (0,X+4)
10 COPY /320, FR (Y*2, Y*2+100)
10 COPY #A, FR (0,1700)
```

> NOTE TO OWNERS OF THE
> MODELS 2270-1 and 2270-3:
>
> On the Model 2270-3, it is not possible to COPY the contents of Platter #3 to Platter #1 or #2, or vice versa. In order to COPY Platter #3, the disk platter must be physically removed from disk drive #3 and inserted into disk drive #1 or #2. On the Model 2270-1, COPY is an illegal statement.

DATALOAD BA

General Form:

DATALOAD BA $\begin{Bmatrix} F \\ R \\ T \end{Bmatrix}$ $\begin{bmatrix} \#n, \\ /xxx, \end{bmatrix}$ (sector address, $\begin{Bmatrix} L \\ L\$ \end{Bmatrix}$) alphanumeric array designator

where:

BA	=	A parameter specifying Absolute Sector Address Mode and block data format.
F	=	Fixed platter, Drive #1, Drive #3.
R	=	Removable platter, Drive #2.
T	=	'F' platter, or 'R' platter, depending on device type specified in the selected device address.
#n	=	A file number to which the disk address is currently assigned ('n' is an integer or numeric variable whose value is from 0 to 6).
/xxx	=	Device address of disk. If neither #n nor /xxx is specified, or if n = 0, the default disk address (stored opposite #0 in the Device Table) is used. The system default disk address is 310.
sector address	=	An expression or alphanumeric variable whose truncated value specifies the sector address of the record to be read. The value of the expression or alpha variable must be less than or equal to the last (highest) sector address on the disk platter.
L	=	A numeric variable which is set to the address of the next sequential sector after the DATALOAD BA statement is processed.
L$	=	A two-byte alphanumeric variable which is set to the binary address of the next sequential sector when the DATALOAD BA statement is processed.
alphanumeric array designator	=	An alphanumeric array name followed by closed parentheses, e.g., A$( ).

Purpose:

The DATALOAD BA statement is used to load one sector of unformatted data from the disk into System 2200 memory. The 'BA' parameter specifies Absolute Sector Addressing Mode and block data format, and is not normally used when the referenced file is a cataloged file. The DATALOAD BA statement reads one sector from the specified disk and sequentially stores the entire 256 bytes in the designated alpha array. No check is made for control bytes normally found in System 2200 data records. An error results if the alpha array is not large enough to hold at least 256 bytes. If the array is larger than 256 bytes, the additional bytes of the array are not affected by the DATALOAD BA operation.

After the statement is executed, the system returns the address of the next consecutive sector, either as a decimal value if a numeric return variable is specified ('L' parameter), or as a two-byte binary value if an alphanumeric return variable is specified ('L$' parameter). This address can be used in a subsequent disk statement or command to provide sequential access to data stored on the disk.

Execution of the DATALOAD BA statement alters the sector address parameters in the Device Table default slot (or in one of the other slots, #1 - #6, if a file number is used in the statement).

Examples:

```
100 DATALOAD BA F (20,L) A$()
100 DATALOAD BA T #2, (B$,B$) B$()
100 DATALOAD BA F /320, (C,C) J$()
100 DATALOAD BA T #A, (A,B) Z$()
```

DATALOAD DA

General Form:

DATALOAD DA $\begin{Bmatrix} F \\ R \\ T \end{Bmatrix}$ $\begin{bmatrix} \#n, \\ /xxx, \end{bmatrix}$ (sector address, $\begin{Bmatrix} L \\ L\$ \end{Bmatrix}$) argument list

where:

DA = A parameter specifying Absolute Sector Address Mode and standard System 2200 data format.

F = Fixed platter, Drive #1, Drive #3.

R = Removable platter, Drive #2.

T = 'F' platter, or 'R' platter, depending on device type specified in the device address.

#n = A file number to which the disk address is currently assigned ('n' is an integer or numeric variable whose value is from 0 to 6).

/xxx = Device address of disk.
If neither #n nor /xxx is specified, or if n = 0, the default disk address (stored opposite #0 in the Device Table) is used. The system default disk address is 310.

sector address = An expression or alphanumeric variable whose truncated value specifies the starting sector address of the record to be loaded. The value of the expression or alpha variable must be less than or equal to the last (highest) sector address on the disk platter.

L = A numeric variable which is set to the address of the next available sector after the DATALOAD DA statement is processed.

L$ = A two-byte string variable which is set to the binary address of the next available sector when the DATALOAD DA statement is processed.

argument list = $\begin{Bmatrix} \text{alphanumeric variable} \\ \text{numeric variable} \\ \text{alpha or numeric array designator} \end{Bmatrix}$ [, { ...

array designator = An array name followed by closed parenthesis, e.g., A( ), B$( ).

Purpose:

DATALOAD DA reads one or more logical records from the disk, starting at the absolute sector address specified. (The records must be formatted with standard System 2200 control information.) The 'DA' parameter specifies direct addressing mode and generally is not used when the referenced data file is a cataloged file. However, Absolute Sector Addressing can be used with cataloged files and may be useful for certain applications (see Section 9.8). The data to be read must be in standard System 2200 format, including the necessary control information (i.e., the data must have been written onto the disk by a DATASAVE DA or DATASAVE DC statement).

The DATALOAD DA statement reads a logical record from the specified disk and assigns the values read to the variables and/or arrays in the argument list sequentially; arrays are filled row by row. If the argument list is not filled, another logical record is read. Data in the logical record not used by the DATALOAD DA statement is read but ignored. If the DATALOAD DA argument list requires more data than is contained in the logical record being read, data is automatically read from the next logical record until the argument list is satisfied. The remainder of the next record is then read but ignored. If an end-of-file (trailer record) is encountered while executing a DATALOAD DA statement, no additional data is read, the next available sector is set to the sector address of the trailer record, and the remaining variables in the argument list remain at their current values. An IF END THEN statement will then cause a valid program transfer.

After the DATALOAD DA statement is executed, the system returns the address of the next sequential logical record, either as a decimal value if a numeric return variable is specified ('L' parameter), or as a binary value if an alphanumeric return variable is specified ('L$' parameter). This address can be used in a subsequent disk statement or command to provide sequential access to data stored on disk.

Data can be read from any disk platter by including the proper parameter ('F' or 'R') in the DATALOAD DA statement. If the 'T' parameter is specified, the platter to be accessed is determined by the device type (3 or B) in the disk device address.

Execution of the DATALOAD DA statement alters the sector address parameters in the Device Table default slot (or in one of the other slots, #1 - #6, if a file number is used in the statement).

Examples:

```
100 DATALOAD DA R (A$,L$) X, Y(), Z$()
100 DATALOAD DA T #3, (20,C) A$, B2$(), M2
100 DATALOAD DA F /320, (D,D) F$(), J
100 DATALOAD DA R (B$,B$) A,B,S()
100 DATALOAD DA T #A, (E,D,) X$()
```

DATASAVE BA

General Form:

DATASAVE BA $\begin{Bmatrix} F \\ R \\ T \end{Bmatrix}$ [$] $\begin{bmatrix} \#n, \\ /xxx, \end{bmatrix}$ (sector address, $\begin{Bmatrix} L \\ L\$ \end{Bmatrix}$) alphanumeric array designator

where:

BA = A parameter specifying Absolute Sector Address Mode and block data format.

F = Fixed platter, Drive #1, Drive #3.

R = Removable platter, Drive #2.

T = 'F' platter or 'R' platter, depending on device type specified in the device address.

$ = Read-after-Write.

#n = A file number to which the disk address is currently assigned ('n' is an integer or numeric variable whose value is from 0 to 6).

/xxx = Device address of disk.

If neither #n nor /xxx is specified, or if n = 0, the default disk address (stored opposite #0 in the Device Table) is used. The system default disk address is 310.

sector address = An expression or alphanumeric variable whose truncated value specifies the sector address at which the record is to be saved. The value of the expression or alpha variable must be less than or equal to the value of the last (highest) sector address on the disk platter.

L = A numeric variable which is set to the address of the next sequential sector after the DATASAVE BA statement is processed.

L$ = A two-byte alphanumeric variable which is set to the binary address of the next sequential sector when the DATASAVE BA statement is processed.

alphanumeric array designator = An alphanumeric array name followed by closed parentheses, e.g., A$( ).

Purpose:

The DATASAVE BA statement is used to save unformatted data on the disk. The 'BA' parameter specifies Absolute Sector Addressing Mode and generally should not be used when the referenced data file is meant to be cataloged. 'BA' also specifies block data format; each DATASAVE BA statement writes one sector with no control information. If the alpha array in the argument list contains more than 256 bytes, only the first 256 bytes are written on disk. If the array contains fewer than 256 bytes, the remainder of the sector is filled with meaningless data.

The DATASAVE BA statement writes data from the specified alpha array into the specified sector on disk. After the statement is executed, the system returns the address of the next sequential sector, either as a decimal value if a numeric return variable is specified ('L' parameter), or as a two-byte binary value if an alphanumeric return variable is specified ('L$' parameter). This address can be used in a subsequent disk statement to permit sequential storage of data on the disk.

Data can be written on any disk platter by including the proper parameter ('F' or 'R') in the DATASAVE BA statement. If the 'T' parameter is specified, the platter to be accessed is determined by the device type (3 or B) in the disk device address.

The '$' parameter specifies that a 'read-after-write' verification check be made on all data written to the disk. This verification check provides added insurance that data is written accurately on the disk, but also doubles the execution time of the DATASAVE BA statement.

Since information written with DATASAVE BA contains no control information, it can be read back only with a DATALOAD BA statement.

Execution of the DATASAVE BA statement alters the sector address parameters in the Device Table default slot (or in one of the other slots, #1 - #6, if a file number is used in the statement).

Examples:

```
100 DATASAVE BA F (L$,L$) A$()
100 DATASAVE BA R $ #3, (20,L) B$()
100 DATASAVE BA F $ /320, (2*1,L) F$()
100 DATASAVE BA T #2, (Q,Q) D$()
```

DATASAVE DA

General Form:

DATASAVE DA { F / R / T } [$] [#n, / /xxx,] (sector address, { L / L$ }) { END / argument list }

where:

DA = A parameter specifying Absolute Sector Address Mode and standard System 2200 data format.

F = Fixed platter, Drive #1, Drive #3.

R = Removable platter, Drive #2.

T = 'F' platter or 'R' platter, depending on device type specified in the device address.

$ = Read-after-write.

#n = A file number to which the disk address is currently assigned ('n' is an integer or numeric variable whose value is from 0 to 6).

/xxx = The device address of the disk.
If neither #n nor /xxx is specified, or if n = 0, the default disk address (stored opposite #0 in the Device Table) is used. The system default disk address is 310.

sector address = An expression or alphanumeric variable whose truncated value specifies the starting sector address of the record to be saved. The value of the expression or alpha variable must be less than or equal to the last (highest) sector address on the disk platter.

L = A numeric variable which is set to the address of the next available sector after the DATASAVE DA statement is processed.

L$ = A two-byte alphanumeric variable which is set to the binary address of the next available sector after the DATASAVE DA statement is processed.

argument list = { alphanumeric variable / literal string / expression / alpha or numeric array designator } [, { }] . . .

array designator = Array name followed by closed parentheses, e.g., A( ), B$( ).

Purpose:

The DATASAVE DA statement is used to save data on the disk in Absolute Sector Addressing Mode. The 'DA' parameter indicates a direct addressing operation; the statement therefore generally is not used when the referenced data file is a cataloged file, since there is a risk the user may unintentionally destroy part of the catalog information. However, direct addressing statements can be used with cataloged files for certain applications (see Section 9.8). The 'END' parameter in a DATASAVE DC statement should never be used for records stored in a cataloged file. There are two important considerations which must be kept in mind when writing a record into a cataloged file with DATASAVE DA. First, the system provides no automatic boundary checking; hence, records can be written past the end of one file and into the beginning of the next without system detection. Second, the "number of sectors used" is not updated in the Catalog Index when a trailer record is written with DATASAVE DA END. Therefore, DSKIP END cannot be used to skip to the end of the file.

The 'DA' parameter specifies that the data in the argument list is to be written in standard System 2200 format, including the necessary control information. Each DATASAVE DA statement writes a logical record consisting of one or more sectors. The DATASAVE DA statement causes the values of variables, expressions, and array elements to be written sequentially onto the specified disk. Arrays are written row by row.

> NOTE:
>
> Each numeric value in the 'argument list' requires 9 bytes on disk; each alphanumeric variable requires the maximum number of characters for which the variable is dimensioned plus 1. Each 256-byte sector also requires three bytes of control information.

If the 'END' parameter is used, a data trailer record is written for the file. This record can be used to test for the end of a file during processing with an IF END THEN statement.

The DATASAVE DA statement writes the data from the argument list onto the disk beginning at the specified sector address. After the statement is executed, the system returns the address of the next available sector, either as a decimal value if a numeric return variable is specified ('L' parameter) or as a two-byte binary value if an alphanumeric return variable is specified ('L$' parameter). This address can be used in subsequent disk statements to provide sequential access to data on the disk.

Data can be written on any disk platter by including the proper parameter ('F' or 'R') in the DATASAVE DA statement. If the 'T' parameter is specified, the platter to be accessed is determined by the device type (3 or B) in the disk device address.

The '$' parameter specifies that a 'read-after-write' verification test be made on all data written to the disk. This verification check provides added insurance that data is written accurately on the disk, but also doubles the execution time of the DATASAVE DA statement.

Execution of the DATASAVE DA statement alters the sector address parameters in the Device Table default slot (or in one of the other slots, #1 - #6, if a file number is used in the statement).

Examples:

```
DATASAVE DA F (20,B) X, Y(), Z$()
DATASAVE DA R $ /320, (C,C) F$(), A()
DATASAVE DA T $ #2, (B$,B$) M$(), "J.DEAN"
DATASAVE DA F (2*M+1,L) J(), K1
DATASAVE DA T (Q,Q) END
DATASAVE DA T #A, (A,B) END
```

LOAD DA

(COMMAND ONLY, NOT PROGRAMMABLE)

General Form:

LOAD DA {F / R / T} [#n, / /xxx,] (sector address, {L / L$})

where:	DA =	A parameter specifying Absolute Sector Address Mode.
	F =	Fixed platter, Drive #1, Drive #3.
	R =	Removable platter, Drive #2.
	T =	'F' platter or 'R' platter, depending on device type specified in device address.
	#n =	A file number to which the disk address is currently assigned ('n' is an integer or numeric variable whose value is from 0 to 6).
	/xxx =	Device address of disk. If neither #n nor /xxx is specified, or if n = 0, the default disk address (stored opposite #0 in the Device Table) is used. The system default disk address is 310.
	sector address =	An expression or alphanumeric variable whose truncated value specifies the starting sector address of the program to be loaded. The value of the expression or alpha variable must be the address of the program header record, and must be less than or equal to the last (highest) sector address on the disk platter.
	L =	A numeric variable which is set to the address of the next available sector after the LOAD DA command is processed.
	L$ =	A two-byte alphanumeric variable which is set to the binary address of the next available sector when the LOAD DA command is processed.

Purpose:

The LOAD DA command is used to load BASIC programs or program segments from the disk in Absolute Sector Addressing Mode. When the LOAD DA command is executed, the program which begins at the specified 'sector address' is read and appended to the current program in memory. (Note that 'sector address' must be the address of a program header record.) The LOAD DA command can be used to add program text to a program currently in memory or, if entered after a CLEAR command, to load a new program from the disk.

After the LOAD DA command is executed, the system returns the address of the next available sector, either as a decimal value if a numeric return variable is specified ('L' parameter), or as a two-byte binary value if an alphanumeric return variable is specified ('L$' parameter). This address can be used in a subsequent disk statement or command to permit sequential access to programs on the disk.

Execution of the LOAD DA command alters the sector address parameters in the Device Table default slot (or in one of the other slots, #1 - #6, if a file number is used in the command).

LOAD DA can also be used as a program statement to chain programs or subroutines (see LOAD DA statement).

Examples:

```
LOAD DA R (24,D)
LOAD DA F (A$,B$)
LOAD DA R /320, (L$,L$)
LOAD DA T #2, (A,B)
LOAD DA R (24,L$)
LOAD DA R (A$,B)
LOAD DA T #A, (C,D)
```

General Form:

LOAD DA {F / R / T} [#n, / /xxx,] (sector address, {L / L$}) [line number 1] [, line number 2]

where:

DA	=	A parameter specifying Absolute Sector Addressing Mode.
F	=	Fixed platter, Drive #1, Drive #3.
R	=	Removable platter, Drive #2.
T	=	'F' platter or 'R' platter, depending on device type specified in the device address.
#n	=	A file number to which the disk address is currently assigned ('n' is an integer or numeric variable whose value is from 0 to 6).
/xxx	=	Device address of disk. If neither #n nor /xxx is specified, or if n = 0, the default disk address (stored opposite #0 in the Device Table) is used. The system default disk address is 310.
sector address	=	An expression or alphanumeric variable whose truncated value specifies the starting sector address of the program which is to be loaded. The value of the expression or alpha variable must be the address of the program header record, and must be less than or equal to the last (highest) sector address on the disk platter.
line number 1	=	The line number of the first line to be deleted from the program currently in memory before loading the new program. After loading, execution continues automatically starting at this line number. An error results if there is no line with this number in the new program.
line number 2	=	The number of the last text line to be deleted from the program currently in memory before loading the new program.
L	=	A numeric variable which is set to the address of the next available sector after the LOAD DA statement is processed.
L$	=	A two-byte alphanumeric variable which is set to the binary address of the next available sector when the LOAD DA statement is processed.

Note: *L or L$* **must** *be a common variable.*

Purpose:

The LOAD DA statement is used to load programs from a specified location on the disk. (Note that the 'sector address' specified must be the address of the program header record.) The 'DA' specifies direct addressing; therefore, the LOAD DA statement is not generally used to load cataloged programs from the disk. LOAD DA is a BASIC program statement which, in effect, produces an automatic combination of the following:

STOP (stop current program execution)

CLEAR P	(clear program text from memory, beginning at 'line number 1' (if specified) and ending at 'line number 2' (if specified); if no line number is specified, clear all program text from memory)
CLEAR N	(clear all non-common variables from memory)
LOAD DA	(load new program or program segment from disk)
RUN	(run new program, beginning at 'line number 1' (if specified); if no line number is specified, run new program from lowest statement line)

The two 'line number' parameters may be used to cause the system to clear a specified portion of resident program text prior to loading in the new program. If both line numbers are specified, all program lines between and including the two specified lines are cleared prior to loading the new program, and execution of the new program begins automatically at 'line number 1'. If only 'line number 1' is specified, the remainder of the resident program is deleted starting with that line number, and execution continues with 'line number 1' of the newly loaded program. If no line numbers are specified, the entire resident program is deleted, and the newly loaded program is executed from its lowest line number. In every case, all non-common variables are cleared. LOAD DA permits segmented programs to be run automatically without normal user intervention, with common variables passed between program segments. If included on a multi-statement line, LOAD DA must be the last executable statement on the line.

In Immediate Mode, LOAD DA is interpreted as a command (see LOAD DA command).

Programs can be loaded from any disk platter by including the proper parameter ('F' or 'R') in the LOAD DA statement. If the 'T' parameter is specified, the platter to be accessed is determined by the device type (3 or B) in the disk device address.

After the program is loaded, the system returns the address of the next sequential sector either as a decimal value, if a numeric return variable is specified ('L' parameter), or as a two-byte binary value, if an alphanumeric return variable is specified ('L$' parameter). This address can be used in a subsequent statement to permit sequential access to programs on the disk.

Execution of the LOAD DA statement alters the sector address parameters in the Device Table default slot (or in one of the other slots, #1 - #6, if a file number is used in the statement).

Examples:

```
 100 LOAD DA F (40,L)
  50 LOAD DA R /320, (L$,L$) 310,450
 530 LOAD DA T #2, (N$,L$) 570
 700 LOAD DA F /320, (L,L)
1020 LOAD DA F (2*I+1,L$) 400
2000 LOAD DA T #B, (C,D)
```

MP-001 sector 19000 - quies Date

General Form:

SAVE DA {F / R / T} [$] [#n, / /xxx,] [P] (sector address, {L / L$}) [line number 1] [, line number 2]

where:

DA =	A parameter specifying Absolute Sector Addressing Mode.
F =	Fixed platter, Drive #1, Drive #3.
R =	Removable platter, Drive #2.
T =	'F' platter or 'R' platter, depending on device type specified in the device address.
$ =	Read-after-write.
#n =	A file number to which the disk address is currently assigned ('n' is an integer or numeric variable whose value is from 0 to 6).
/xxx =	Device address of disk. If neither #n nor /xxx is specified, or if n = 0, the default device address (stored opposite #0 in the Device Table) is used. The system default disk address is 310.
P =	Set the protection bit on the file to be saved.
sector address =	An expression or alphanumeric variable whose truncated value specifies the starting sector address of the program to be saved. The value of the expression or alpha variable must be less than or equal to the last (highest) sector address of the disk platter.
L =	A numeric variable which is set to the address of the next available sector after the SAVE DA command is processed.
L$ =	A two-byte alphanumeric variable which is set to the binary address of the next available sector when the SAVE DA command is processed.
line number 1 =	The number of the first program line to be saved.
line number 2 =	The number of the last program line to be saved.

Purpose:

The SAVE DA command is used to save programs on the disk beginning at a specified location. Because the 'DA' specifies Absolute Sector Addressing Mode, this command should not be used if the program is to be saved under catalog procedures. The SAVE DA command causes BASIC programs (or portions of BASIC programs) to be recorded on the designated platter beginning at the specified sector address. The program cannot be named and can be loaded back into memory only with a LOAD DA statement or command.

After each program is saved, the system returns the address of the next available sector, either as a decimal value if a numeric return variable is specified ('L' parameter), or as a two-byte binary value if an alphanumeric return variable is specified ('L$' parameter). This address can be used in a subsequent disk command to permit the sequential storage of programs on disk.

Execution of the SAVE DA command alters the sector address parameters in the Device Table default slot (or in one of the other slots, #1 - #6, if a file number is used in the command).

Programs can be saved on any disk platter by including the proper parameter ('F' or 'R') in the SAVE DA command. If the 'T' parameter is specified, the platter to be accessed is determined by the device type (3 or B) in the disk device address.

The '$' specifies that a 'read-after-write' verification check be performed on all information written to the disk. This verification check provides added insurance that the program is recorded accurately, but also doubles the execution time of the SAVE DA command.

The 'P' parameter permits the user to protect saved programs. A protected program can be loaded and run, but cannot be listed or resaved.

> NOTE:
>
> In order to save any program on disk after a protected program has been loaded, the user must enter a CLEAR command with no parameters, or Master Initialize the system (i.e., turn main power switch OFF, then ON).

Examples:

```
SAVE DA F (3,L)
SAVE DA R $ /320, P (L,L)
SAVE DA R #2, (A$,A$) 200
SAVE DA T #2, P (A$,A$)
SAVE DA F (2+X,L)
```

Chapter 11
The Disk Multiplexers (Models 2224 and 2230MXA/B)

11.1 INTRODUCTION

The disk multiplexer permits several independent systems to share a single disk unit. A maximum of four independent CPU's can be multiplexed to the same disk unit, thus enabling users with systems in physically separate locations to share a common disk-resident data base, or to maintain independent files on a commonly used disk.

Wang markets two types of disk multiplexers: the Model 2224 "box" multiplexer, and the Model 2230MXA/B "daisy-chain" multiplexer. The Model 2224 consists of a multiplexer chassis into which all participating systems, as well as the disk unit, connect. Communications between all systems and the disk unit are controlled by the multiplexer electronics within this central chassis. The Model 2230MXA/B is, by contrast, a "daisy-chain" multiplexer which has no central chassis. It consists solely of a series of special multiplexer controller boards, one of which, the "master" board, controls all access to the disk unit. The special boards are installed in participating systems, and the systems are connected together to form a chain. Only the system with the master board connects directly to the disk unit. The multiplexers can be used with any disk model.

11.2 THE MODEL 2224 DISK MULTIPLEXER

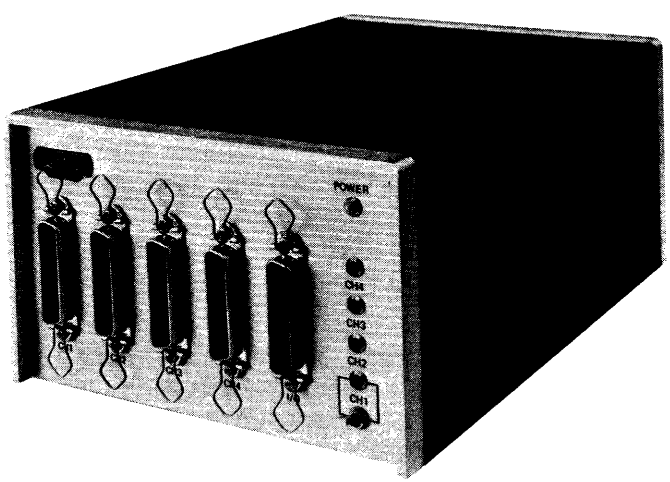


Figure 11-1. Model 2224 Disk Multiplexer

The Model 2224 Disk Multiplexer consists of a central chassis containing the power supply and multiplexer electronics, along with special multiplexer controller boards and connector cables. It is available in three configurations:

- Model 2224-2 - Central chassis plus two controller boards; permits two stations to share a disk unit.
- Model 2224-3 - Central chassis plus three controller boards; permits three stations to share a disk unit.
- Model 2224-4 - Central chassis plus four controller boards; permits four stations to share a disk unit.

The standard cables provided to connect the multiplexer chassis to each participating system are 12 feet (3.7 meters) in length. If one or more stations must be located a greater distance from the disk, longer cables are available as special options. The special connector cables are available in lengths ranging from 100 feet (30.5m) to 500 feet (152.4m), in increments of 100 feet. Part numbers for these special connector cables are listed below:

Cable	Part #
100 ft (30.5m)	120-2224-1
200 ft (61m)	120-2224-2
300 ft (91.5m)	120-2224-3
400 ft (122m)	120-2224-4
500 ft (152.4m)	120-2224-5

Note that two connector cables cannot be spliced together. Each cable has two 36-pin Amphenol plugs, one of which is inserted in a jack in the multiplexer chassis, and the other of which is plugged into a jack on the multiplexer controller board in a participating CPU.

11.3 INSTALLATION OF THE MODEL 2224

Unpacking and Inspection

Carefully unpack your equipment and inspect it for damage. If a unit is damaged, notify the shipping agency at once. Check each piece of equipment received against the purchase order. (Decals specifying model numbers can be found on all Wang equipment, usually on the back of each unit.)

Installation Procedure

> NOTE:
>
> If a connector cable must be routed through conduit or any tight space requiring removal of one of the Amphenol plugs, a Wang Service Representative must be responsible for removing and reattaching the plug. Reattachment of the Amphenol plug is a delicate job which, if done improperly, can impede or prevent proper data transmission along the cable. Contact your local Wang Field Service Office for a quote on this job.

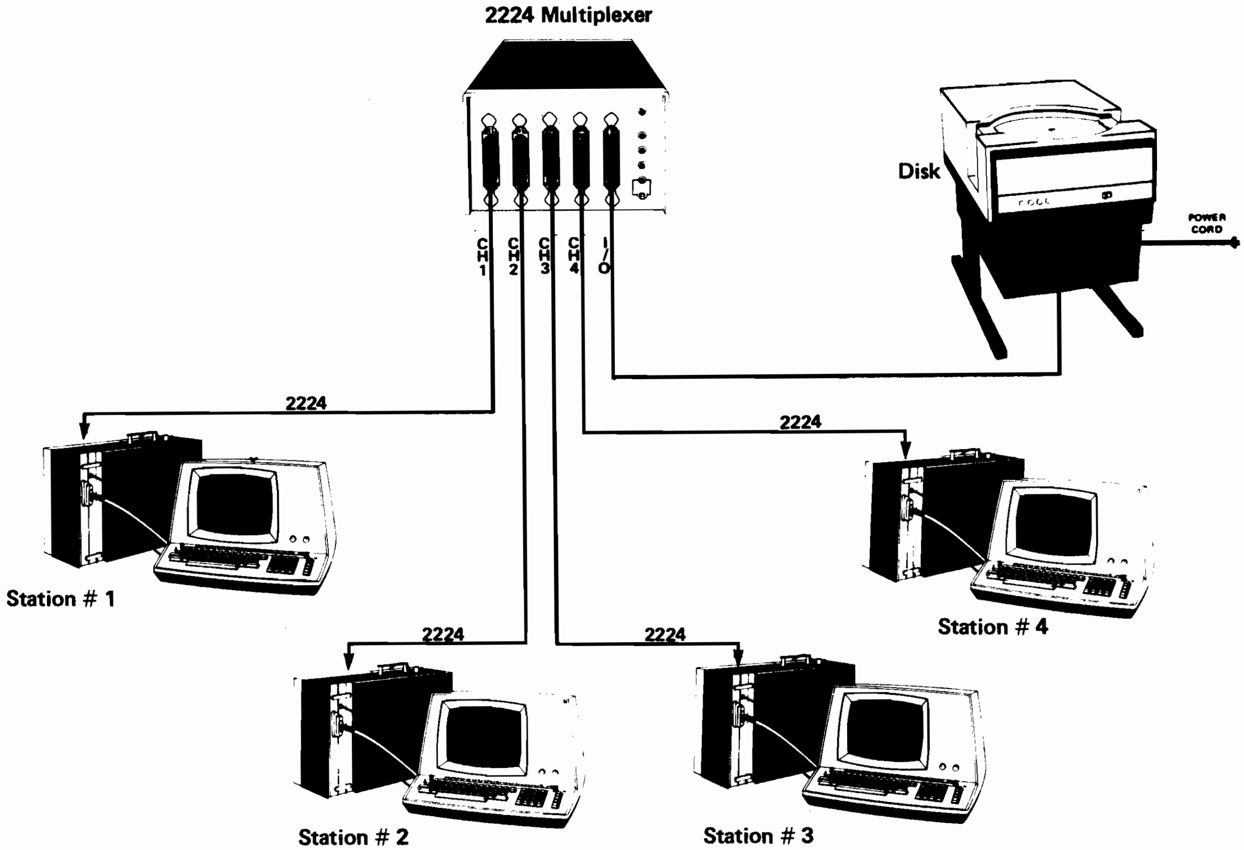


Figure 11.2 Typical System Configuration for Model 2224 Disk Multiplexer, Disk Unit, and Four Stations.

1. Install the Disk Multiplexer controller board in each participating CPU. Any available peripheral slot may be used. In systems which already have a disk controller board, the multiplexer controller board replaces the disk controller board.

2. Plug one end of each connector cable into the multiplexer controller board in a CPU, and plug the other end into one of the four jacks labelled "Channel 1" - "Channel 4" on the front panel of the central chassis. Each participating CPU must be connected to the multiplexer central chassis in this manner.

> NOTE:
>
> The system which is located closest to the disk unit itself should be assigned to Channel #1.

3. Plug the disk connector cable into the fifth jack on the front panel of the central chassis, labelled "I/O".

4. Install each participating system according to the procedure described in the Reference Manual.

5. Plug the power cords from the disk unit and multiplexer into grounded wall sockets. Input power requirements for the disk and multiplexer are 115 VAC, 9 amps, 50/60 Hz ± 1 cycle. (The disk unit is also available in a configuration which requires 230 VAC, 5 amps, 50/60 Hz ± 1 cycle, by special request.)

> NOTE:
>
> When routing the multiplexer connector cables between the multiplexer chassis and the participating systems, avoid exposing a cable to intense electric or magnetic fields, or sources of electronic noise, since they may affect transmission over the cable. In general, you should try to keep the cable away from electrical trunk lines, fluorescent lights, and electric office equipment (such as electric typewriters and tape recorders). If you have a specific question about what to avoid when routing a cable, contact your Wang Service Representative.

Power-On Procedure

1. Switch ON the power switches on the CRT's and other peripherals on each system.

2. Switch ON the power switches on the multiplexer and disk unit.

3. Switch ON the main power switch on the Power Supply Unit (the light on the Power Supply Unit illuminates) in each system. This Master Initializes the system.

4. On the disk unit, the POWER light comes on. The CRT display in each station looks like this:

```
READY
:_
```

5. Press the RESET button on any system keyboard to initialize the Multiplexer/Disk Controller.

The disk unit may now be accessed via the multiplexer. For a discussion of how the multiplexer functions, turn to Section 11.6.

NOTE:

If you experience some difficulty in maintaining valid data transmission over a connector cable, the problem may lie in a coating which can form on the pins in the Amphenol plugs during extended periods of disuse. To remove this coating, simply insert and remove the plug in the jack several times, or cut a piece from a white ink-type eraser small enough to fit between the pins, and use it to clean the surfaces of the pins. Transmission problems can also be created by electrical and magnetic interference in the connector cable.

11.4 THE MODEL 2230MXA/B MULTIPLEXER

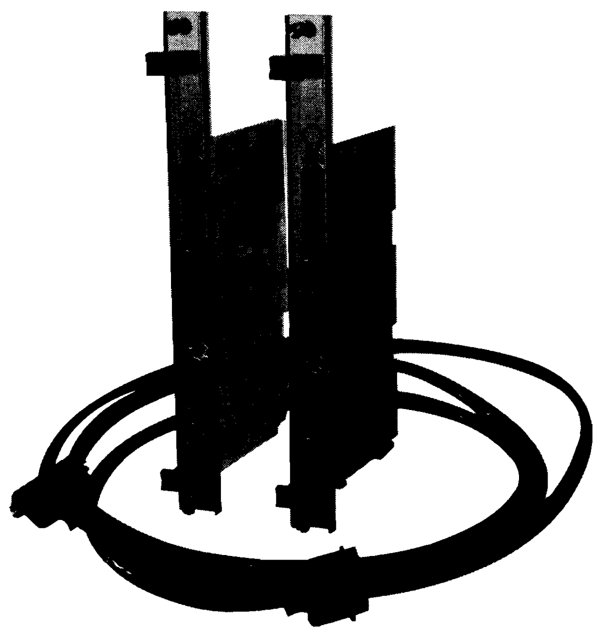


Figure 11-3. Model 2230MXA Master Board and 2230MXB Slave Boards

The Model 2230MXA/B Multiplexer is a "daisy chain" multiplexer consisting of a single 2230MXA master controller board, and one or more 2230MXB slave controller boards. A 12-foot (3.7 meter) connector cable accompanies each slave board. The 2230MXA board and 2230MXB slave boards are purchased separately. The number of slave boards required is determined by the size of the total installation: a master board and one slave board permits two stations to share the disk; a master board and two slave boards permit three stations to share the disk; a master board and three slave boards permit four stations to share the disk.

Unlike the Model 2224, the Model 2230MX does not utilize a central chassis; instead, all necessary electronics are built into the controller boards themselves. (The primary electronics are in the 2230MXA master board.) Each slave board has a 36-pin input connector labeled "MUX INPUT", and a 50-pin output connector labeled "MUX OUTPUT". The master board has a 36-pin input connector, also labeled "MUX INPUT", and a 50-pin connector labeled "DISK".

The connector cables correspondingly have two Amphenol plugs, one of 36 pins and one of 50 pins. The systems are connected together by running cables from the MUX INPUT jack of one CPU to the MUX OUTPUT jack of the next consecutive CPU to form a chain. At the beginning of the chain is the master system (the CPU which has the 2230MXA master board). The disk connector cable plugs into the DISK jack on the master board to complete the chain. (See Figure 11-5.) The master system is the only system which connects directly to the disk unit.

In addition to the standard 12-foot (3.7m) connector cable shipped with each slave board, longer extension cables are available in lengths of 50, 100, and 200 feet (15.5, 31, and 61 meters). The extension cable part numbers are listed below:

Cable Length	Part #
50 ft (15.5m)	120-2225-50
100 ft (31m)	120-2225-1
200 ft (61m)	120-2225-2

These cables are "extension cables" in a literal sense, since they serve as extensions for the standard connector cables; an extension cable cannot be used alone to connect two systems. Each extension cable has two 36-pin Amphenol plugs, one male and one female. The male plug is inserted in the MUX INPUT jack of a slave board, while the female plug must be connected to the 36-pin male plug on a standard connector cable. The 50-pin plug on the other end of the standard cable is then inserted in the MUX OUTPUT jack of a second board. Because the extension cable is combined with the standard cable in this manner, the total length of the cable between two units is always equal to the extension cable length plus 12 feet. (See Figure 11-4.)

In special cases, it is possible to connect two or more extension cables together to create an extension longer than 200 feet. In every case, however, the maximum permissable distance between two systems is 512 feet, and the maximum distance between the first and last systems in the chain is 536 feet. The cable connecting the disk unit to the master CPU is approximately ten feet (3 meters) in length, and cannot be extended.

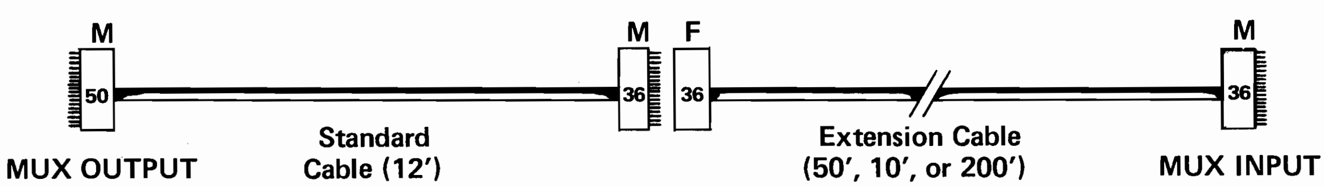


Figure 11-4. Connecting Extension Cable with Standard 12-foot Cable

11.5 INSTALLING THE MODEL 2230MXA/B

Unpacking and Inspection

Carefully unpack your equipment and inspect it for damage. If a unit is damaged, notify the shipping agency at once. Be certain that you have one 2230MXA master board, and the expected number of 2230MXB slave boards.

Installation Procedure

> NOTE:
>
> If a connector cable is to be routed through conduit, or through any tight space requiring removal of one of the Amphenol plugs, it is important that the plug be disconnected and reconnected by a qualified Wang Service Representative. Reconnection of the Amphenol plug is a delicate job which, if done improperly, can impede or prevent data transmission along the line. Contact your Wang Field Service Office for a quote.

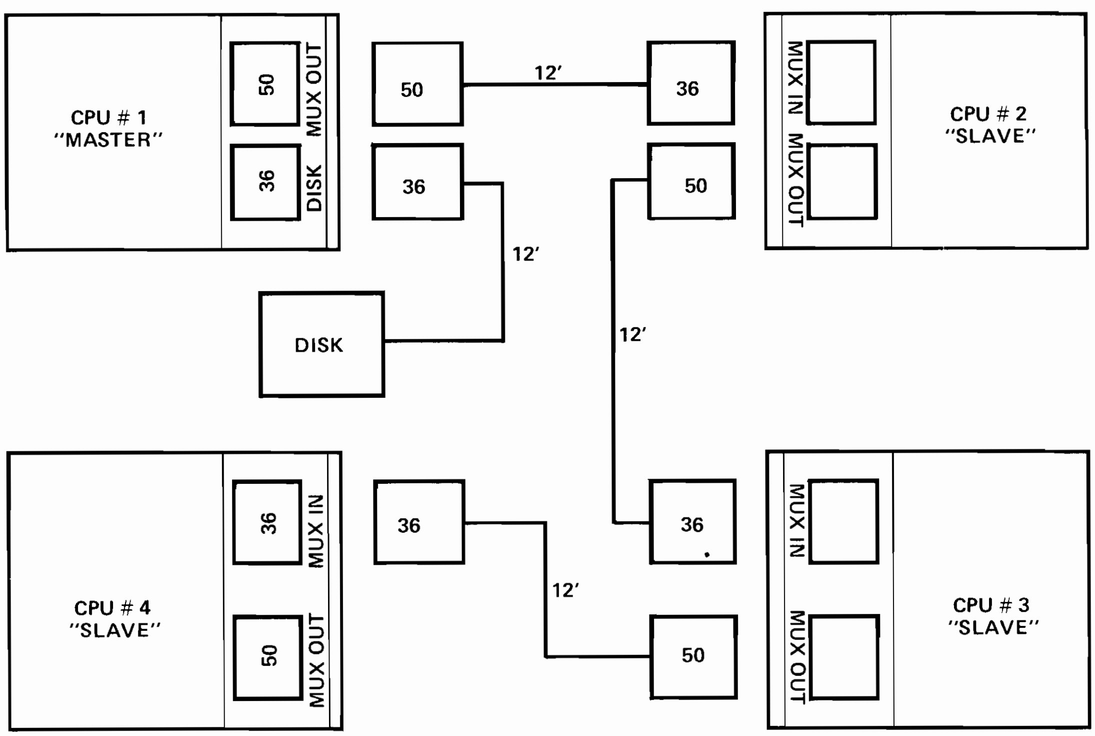


Figure 11-5. Typical System Configuration: Model 2230MXA/B Multiplexer, Disk Unit, and Four Attached CPU's.

1. Install the 2230MXA master controller board in CPU #1 (the system nearest the disk). Install 2230MXB slave boards in the remaining systems. In systems which already have a disk controller board, the multiplexer board replaces the disk board.

2. Plug the disk I/O cable into the jack marked "DISK" on the 2230MXA master board.

3. Insert the 50-pin connector cable plug in the jack labelled "MUX OUTPUT" on the master board. If no extension cable is used, insert the 36-pin plug on the other end of the cable into the MUX INPUT jack in the slave board in CPU #2. If an extension cable is used, plug the standard cable into the extension cable, and plug the extension cable into the MUX INPUT jack. Repeat this procedure for all attached systems.

4. Be sure that all attached systems are properly set up and ready for operation.

5. Plug all power cords into grounded (three-hole) wall sockets. Power requirements for each disk are listed in an Appendix at the back of this manual.

> NOTE:
>
> When routing the multiplexer connector cables between participating systems, take care to avoid exposing a cable to intense electric or magnetic fields, or sources of electronic noise, since they may interfere with data transmission over the cable. In general, you should try to keep the connector cable away from electrical trunk lines, fluorescent lights, and electrical office equipment (such as electric typewriters and tape recorders). If you have a specific question about routing a cable, contact your Wang Service Representative.

Power-On Procedure

1. Switch ON the power switches on all system peripherals, including the disk unit.

2. Switch ON the Main Power Switches on all system power supply units (the light on the power supply unit illuminates).

3. On the disk unit, the POWER light should be illuminated. The CRT display at each station looks like this:

```
READY
:_
```

4. Touch the RESET button on the keyboard of the master system to initialize the multiplexer/disk controller boards.

> NOTE:
>
> When several systems are multiplexed to the same disk with the 2230MXA/B Multiplexer, the master CPU (the CPU with the 2230MXA master board) must be powered ON in order for any other system to access the disk. However, one or more of the slave CPU's (those with 2230MXB slave boards) may be powered OFF without disturbing the operation of the other CPU's.

The disk may now be accessed via the multiplexer from any attached system. Turn to Section 11.6 for an explanation of how the multiplexer operates, and a discussion of some programming considerations.

> NOTE:
>
> If you experience difficulty in maintaining valid data transmission between the disk and one or more systems, the problem may lie with the Amphenol connector plugs. A coating sometimes forms on the pins of a plug during extended periods of disuse. To remove this coating, which may inhibit transmission, simply insert and remove the plug in a jack several times, or cut a piece from a white ink-type eraser small enough to fit between the pins, and use it to clean the surfaces of the pins. Transmission problems also can be created by electrical and magnetic interference in the cables.

11.6 MULTIPLEXER OPERATION

The disk multiplexer controls all communication between participating systems and the disk unit. The multiplexer automatically "polls" each system, beginning with system (or "station") #1, until it finds a system which is attempting to access the disk. At that point, the multiplexer permits the inquiring station to execute one disk statement or command. Following execution of the statement or command, the multiplexer resumes its polling until it encounters another system trying to access the disk. The multiplexer does not monitor the amount of time required to execute each statement, nor does it limit the number of sectors transferred by a statement. A single statement may read or write only one sector, but it is equally possible to carry out multi-sector transfers with one statement. (A MOVE or COPY statement, for example, might transfer an entire disk platter to a second platter.) It is recommended, however, that major file maintenance operations be executed only by a station in Hog Mode (see below). In any case, the system which is executing the statement retains use of the disk until statement execution is completed. Control is then transferred to the next inquiring station. On the Model 2224 multiplexer, a row of lights (one for each channel) on the right-hand front panel of the central chassis wink on and off during operation, indicating which channel has access to the disk at a given moment. The Model 2230MXA/B provides no external indication of which system has access to the disk.

In normal operation, the multiplexer imposes no special demands or conditions upon the programmer. The disk is simply addressed as usual with the appropriate disk statements and commands. If no other systems are accessing the disk, the total execution time of a multi-statement disk operation is not noticeably affected by the multiplexer. If more than one multi-statement disk operation is being carried on at once, however, the time required for each operation is roughly equal to the total time required to execute all operations, since one statement from each system is executed on each pass by the multiplexer.

Although in general all systems attached to the multiplexer gain access to the disk on a statement-by-statement basis, there are cases in which it is desirable to give one system a period of exclusive and uninterrupted access to the disk. During certain critical file maintenance or update procedures, for example, it is important that other systems be prevented from accidentally interfering in the routine, since they might unknowingly overwrite valuable data or pointers, or otherwise confuse the situation. Because operators on remote stations have no way of knowing that critical maintenance procedures are being carried out at any given time, it is necessary to prevent them from unknowingly interrupting a routine by locking out all systems except the one executing the routine. A system which monopolizes the disk in this way is said, somewhat picturesquely, to be "hogging" the disk. Both multiplexers provide a feature which enables a single system to "hog" the disk under program control. The Model 2224 also offers a manual switch-selectable Hog Mode capability.

11.7 THE MANUAL HOG MODE SWITCH (MODEL 2224 ONLY)

The manual Hog Mode switch is located on the lower left-hand corner of the Model 2224 Multiplexer front panel. When the switch is flipped up, it activates the Hog Mode for Channel #1, closing Channels #2-#4 and locking out the systems on those channels. Only the system plugged into Channel #1 has access to the disk. The Hog Mode switch can thus be used to give station #1 exclusive access to the disk for important file maintenance or update operations.

Because the manual Hog Mode switch allows only station #1 to hog the disk, it is recommended that station #1 be used for all major file maintenance procedures. After the procedures are completed, the Hog Mode can be deactivated by flipping the Hog Mode switch down to its normal position. The multiplexer then resumes its normal operation, polling each system in sequence until it finds one trying to access the disk.

11.8 PROGRAMMABLE HOG MODE (MODELS 2224 AND 2230MXA/B)

The programmable Hog Mode feature (available on both multiplexer models) enables any station plugged into the multiplexer to seize control of the disk under program control, and lock out all other stations. A special disk address of 390 (or B90) is used to initiate the programmable Hog Mode. Special Hog Mode device addresses can be generated for other disk units in a system by adding a HEX(80) to the normal disk device address. When a disk statement or command is executed which references this special address, tne system executing that statement automatically gains exclusive control of the disk; no other system can access the disk until the programmable Hog Mode is turned off. For example, the statement

```
10 MOVE/390 FR
```

automatically activates the Hog Mode, locking out all other stations, and initiates a MOVE operation from the 'F' platter to the 'R' platter.

It is also possible to store the special address opposite a slot in the Device Table. In this case, execution of a statement referencing the appropriate file number automatically activates the Hog Mode.

Example 11-1: Activating the Programmable Hog Mode
(Models 2224 and 2230MXA/B Disk Multiplexers)

```
10 SELECT DISK 390
20 MOVE FR
```

Statement 10 stores special disk address 390 in the default slot (opposite #0) in the Device Table. Statement 20 references the default slot (since no file number is specified); thus, execution of statement 20 automatically locks out all other stations from utilizing the disk, and initiates a MOVE operation.

Example 11-2: Activating the Programmable Hog Mode
(Models 2224 and 2230MXA/B Disk Multiplexers)

```
10  SELECT #1 390
:
:
200 DATALOAD DC OPEN F #1, "CUSTFILE"
```

Statement 10 stores special address 390 opposite #1 in the Device Table. Statement 200 references #1, to reopen the file CUSTFILE, thus activating the Hog Mode and locking out all other stations.

Note that the Hog Mode is turned on only when a disk statement or command implicitly or explicitly referencing the special address is executed. Merely storing the special address in the Device Table with a SELECT statement does not initiate the Hog Mode. For example, the statement

```
SELECT #1 390
```

does not activate the Hog Mode. It must be followed by a disk statement which references the Hog Mode address. Thus, the special address (390) can be stored in a particular slot in the Device Table and referenced by a system only when it wishes to monopolize the disk.

> NOTE:
>
> As long as the special Hog Mode address remains in the Device Table, any reference to its associated file number will activate the Hog Mode. Care must be exercised, therefore, in the use of this file number.

When a system has completed the operations requiring exclusive use of the disk, the Hog Mode can be turned off simply by executing a disk statement or command which implicitly or explicitly references the regular disk address. For example, if the regular disk address is 310, the Hog Mode is deactivated by either of the following statements:

```
250 LOAD DC F /310, "PROG 1"
          or
300 SELECT #1 310
310 DATASAVE DC #1, END
```

Example 11-3: Deactivating the Programmable Hog Mode (Models 2224 and 2230MXA/B Disk Multiplexers)

```
10  SELECT #1 390, #3 390
20  DATALOAD DC OPEN #1 "CUSTFILE"
30  DATALOAD DC OPEN #3 "CREDIT"
:
:   (Processing)
:
190 SELECT #1 310, #3 310
200 DATASAVE DC #1, END
210 DATASAVE DC #3, END
```

This example illustrates a portion of a processing routine carried out in Hog Mode. At line 10, the special Hog Mode address is selected to file numbers #1 and #3. At lines 20 and 30, the two files to be used are opened with these file numbers. The Hog Mode is activated when line 20 is executed. Upon completion of processing at line 190, the file numbers are reselected to the normal disk address, 310. The Hog Mode is not deselected at this time, however. Deselection takes place only when the normal address is referenced in a disk statement (line 200).

Notice that it is not sufficient merely to remove the special address from the Device Table or to replace it with the normal address in the Device Table. A disk statement or command must be executed which implicitly or explicitly references the normal disk address in order to turn off the Hog Mode. For example, the statement

```
SELECT #1 310
```

by itself does not deactivate the Hog Mode. It must be followed by a disk statement referencing the regular address:

```
DATASAVE DC #1, END
```

> NOTE:
>
> Any reference to the normal device address during Hog Mode processing will deactivate the Hog Mode. If several file numbers are to be used to reference different files for processing in Hog Mode, each file number must be selected to the special Hog Mode address (e.g., SELECT #1 390, #3 390, #4 390, etc.). The Hog Mode will be deactivated as soon as a file number selected to the normal address is used in a disk statement.

It is also possible to turn off the programmable Hog Mode at any point by keying RESET on the keyboard of the system currently hogging the disk.

> NOTE:
>
> The programmable Hog Mode must be deactivated by the system which is hogging the disk. It cannot be deactivated by another system.

Appendix A
Disk Error Codes

CODE 57

Error: ILLEGAL DISK SECTOR ADDRESS

Cause: Illegal disk sector address specified; value is negative or greater than 32767. (The System 2200 cannot store a sector address greater than 32767.)

Action: Correct the program statement in error.

Example:

```
100 DATASAVE DAF (42000, X) A, B, C.
                            ↑ ERR 57
100 DATASAVE DAF (4200, X) A, B, C       (Possible Correction)
```

CODE 58

Error: EXPECTED DATA RECORD

Cause: A program record or header record was read when a data record was expected.

Action: Correct the program.

Example:

```
100 DATALOAD DAF(0,X) A, B, C
                         ↑ ERR 58
```

CODE 59

Error: ILLEGAL ALPHA VARIABLE FOR SECTOR ADDRESS

Cause: The alphanumeric string variable designated to receive the next available address following execution of the DA or BA instruction is not at least two bytes long.

Action: Dimension the alpha variable to be at least two characters long.

Example:

```
10 DIM A$1
100 DATASAVE DA R (F$, A$) X, Y, Z
                          ↑ ERR 59
10 DIM A$2                                (Possible Correction)
```

CODE 60

Error: ARRAY TOO SMALL

Cause: The alphanumeric array specified in a DATALOAD BA statement does not contain enough space to store the sector of information (256 bytes) being read from the disk. The array must contain at least 256 bytes.

Action: Increase the size of the array.

Example:

```
10 DIM A$(15)
20 DATALOAD BA R (1000,L) A$()
                            ↑ ERR 60
10 DIM A$(16) 16                          (Possible Correction)
```

CODE 61

Error: **DISK HARDWARE ERROR**

Cause: The disk did not recognize or properly respond to the System 2200 during a read or write operation in the proper amount of time.

Action: Run program again. If error persists, re-initialize the disk. If failure recurs, contact Wang service personnel.

Example:

```
100 DATASAVE DCF X,Y,Z
                    ↑ERR 61
```

CODE 62

Error: **FILE FULL**

Cause: The disk sector being addressed is not located within the specified catalog file boundaries. When writing data in a cataloged file, Error 62 signifies that the file is full; Error 62 is also generated if a DSKIP or DBACKSPACE operation has set the current sector address beyond the limits of the file.

Action: Correct the program.

Example:

```
100 DATASAVE DCT#2, A$( ), B$( ), C$( )
                                    ↑ERR 62
```

CODE 63

Error: **MISSING ALPHA ARRAY DESIGNATOR**

Cause: An alpha array designator (e.g., A$( )) was expected. (Block operations for cassette and disk require an alpha array argument.)

Action: Correct the statement in error.

Example:

```
100 DATALOAD BA A$
                 ↑ERR 63
100 DATALOAD BA A$( )         (Possible Correction)
```

CODE 64

Error: **SECTOR NOT ON DISK**

Cause: The disk sector being addressed is not on the disk. (Maximum legal sector address depends upon the model of disk used.)

Action: Correct the program statement in error.

Example:

```
100 MOVEEND F = 10000
                   ↑ERR 64
100 MOVEEND F = 9791          (Possible Correction)
```

CODE 65

Error: DISK HARDWARE MALFUNCTION

Cause: A disk hardware error occurred (i.e., the disk is not in file ready position). This could occur, for example, if the Model 2230 or 2260 is in LOAD mode or power is not turned on. On the Model 2270, the error could result if a disk platter does not move freely within its jacket.

Action: Insure disk is turned on and properly set up for operation. For the Models 2230 and 2260, set the disk into LOAD mode and then back into RUN mode, with the LOAD/RUN selection switch. The error light should then go out. For the Model 2270, make sure that the disk platter moves freely within its jacket and that the drive door is tightly shut. If the error persists, call Wang Field Service Personnel. (Note: the Models 2230 and 2260 should never be left in LOAD mode when running.)

Example:

```
100 DATALOAD DCF A$,B$
                    ↑ERR 65
```

CODE 66

Error: FORMAT KEY ENGAGED

Cause: The disk format key is engaged. (The key is normally engaged only when formatting a disk platter.)

Action: Turn off the format key.

Example:

```
100 DATASAVE DCF X,Y,Z
                    ↑ERR 66
```

CODE 67

Error: DISK FORMAT ERROR

Cause: A disk format error was detected during a disk read or write operation. The disk is not properly formatted so that sector addresses can be read.

Action: Format the disk again. If the error persists, replace the platter.

Example:

```
100 DATALOAD DCF X, Y, Z
                    ↑ERR 67
```

CODE 68

Error: LRC ERROR

Cause: A disk longitudinal redundancy check error occured when reading a sector. The data may have been written incorrectly, or the System 2200/Disk Controller could be malfunctioning.

Action: Run program again. If error persists, re-write the bad sector or replace the platter. If error still persists call Wang Service personnel.

Example:

```
100 DATALOAD DCF A$( )
                    ↑ERR 68
```

CODE 71

Error: CANNOT FIND SECTOR/PROTECTED PLATTER

Cause: A disk seek error occurred; the specified sector could not be found on the disk. On the Model 2270, Error 71 also is signalled if an attempt is made to write on a protected diskette. No data or programs can be recorded on a protected diskette.

Action: Run program again. If error persists, re-initialize (reformat) the disk platter, or replace it. If error still occurs, call Wang Service Personnel. For Model 2270 diskette, make sure write protect hole is covered.

Example:

```
100 DATALOAD DCF A$( )
                    ↑ERR 71
```

CODE 72

Error: CYCLIC READ ERROR

Cause: A cyclic redundancy check error occurred when reading a sector; the sector being addressed has never been written to the disk, or the sector was incorrectly written on the disk (i.e., the disk platter was never initially formatted).

Example:

```
100 MOVEEND F = 8000
                   ↑ERR 72
```

CODE 73

Error: ILLEGAL ALTERING OF A FILE

Cause: The user is attempting to create a file with a name which is already in the Catalog Index, or is attempting to rename or write over an existing scratched file without using the proper syntax.

Action: Use the proper form of the statement. The scratched file name must be referenced.

Example:

```
SAVE DCF "SAM1"
              ↑ERR 73
SAVE DCF ("SAM1") "SAM1"          (Possible Correction)
```

CODE 74

Error: CATALOG END ERROR

Cause: The end of Catalog Area falls within the Catalog Index area, or has been changed by MOVEEND to fall within the area already occupied by cataloged files; or there is no room left in the Catalog Area to store additional files.

Example:

```
SCRATCH DISK F LS=100, END=50
                           ↑ERR 74
SCRATCH DISK F LS=100, END=500      (Possible Correction)
```

CODE 75

Error: COMMAND ONLY (Not Programmable)

Cause: A command is being executed on a numbered statement line within a BASIC program. Commands are not programmable.

Action: Do not use commands as program statements.

Example:

```
10 SAVE DC R "PROG 1"
             ↑ERR 75
```

CODE 77

Error: STARTING SECTOR GREATER THAN ENDING SECTOR

Cause: The starting sector address specified is greater than the ending sector address specified.

Action: Correct the statement in error.

Example:

```
10 COPY FR(1000, 100)
                   ↑ERR 77
10 COPY FR(100, 1000)                    (Possible Correction)
```

CODE 78

Error: FILE NOT SCRATCHED

Cause: The user is attempting to rename a file which has not been scratched.

Action: Scratch the file before renaming it.

Example:

```
SAVE DCF ("LINREG") "LINREG2"
                     ↑ERR 78
SCRATCH F "LINREG"                       (Possible Correction)
SAVE DCF ("LINREG") "LINREG2"
```

CODE 79

Error: FILE ALREADY CATALOGED

Cause: An attempt was made to catalog a file with a name that already exists in the Catalog Index.

Action: Use a different name.

Example:

```
SAVE DCF "MATLIB"
                   ↑ERR 79
SAVE DCF "MATLIB1"                       (Possible Correction)
```

CODE 80

Error: FILE NOT IN CATALOG

Cause: The user is attempting to address a non-existing file name, or load a data file as a program, or open a program file as a data file.

Action: Make sure you are using the correct file name; make sure the proper disk platter is mounted. (List the contents of the Catalog Index on the platter to determine the proper names for all cataloged files.)

Example:

```
LOAD DCF "PRES"
                ↑ERR 80
LOAD DCR "PRES"                          (Possible Correction)
```

CODE 81

Error: /XXX DEVICE SPECIFICATION ILLEGAL

Cause: The /xxx device specification cannot be used in this statement.

Action: Correct the statement in error.

Example:
```
100 DATASAVE DC /310, X
                    ↑ERR 81
100 DATASAVE DC #1, X                    (Possible Correction)
```

CODE 82

Error: NO END OF FILE

Cause: No end-of-file record was written in the file and therefore could not be found in a DSKIP END operation, or an end-of-file record was written with a DATASAVE DA END statement rather than DATASAVE DC END.

Action: Correct the file by writing an end-of-file record with a DATASAVE DC END statement.

Example:
```
100 DSKIP END
          ↑ERR 82
```

CODE 83

Error: DISK HARDWARE FAILURE

Cause: A disk address cannot be properly transferred from the System 2200 to the disk when processing MOVE or COPY.

Action: Run program again. If error persists, replace the platter. If replacing the platter does not correct the problem, call Wang Field Service Personnel.

Example:
```
COPY FR(100,500)
          ↑ERR 83
```

CODE 84

Error: NOT ENOUGH SYSTEM 2200 MEMORY AVAILABLE FOR MOVE OR COPY

Cause: A 1K buffer is required in memory for MOVE or COPY operation. (i.e., 1024 bytes must be available which are not occupied by program text or variables).

Action: Clear out all or part of program or variables before MOVE or COPY.

Example:
```
COPY FR(0, 9000)
          ↑ERR 84
```

CODE 85

Error: READ AFTER WRITE ERROR

Cause: The comparison of read after write to a disk sector failed. The information was not written properly.

Action: Write the information again. If error persists, replace the platter. If replacing the platter does not correct the problem, call Wang Field Service personnel.

Example:
```
100 DATASAVE DCF$ X, Y, Z
                ↑ERR 85
```

CODE 86

Error: FILE NOT OPEN

Cause: The file was not opened.

Action: Open the file before reading from it.

Example:

```
100 DATALOAD DC A$
                 ↑ERR 86
10 DATALOAD DC OPEN F "DATFIL"        (Possible Correction)
```

CODE 87

Error: COMMON VARIABLE REQUIRED

Cause: The variable in the LOAD DA statement, used to receive the sector address of the next available sector after the load, is not a common variable.

Action: Define the variable to be common.

Example:

```
10 LOAD DAR (100,L)
                 ↑ERR 87
5 COM L                               (Possible Correction)
```

CODE 88

Error: CATALOG INDEX FULL

Cause: There is no more room in the Catalog Index for a new name.

Action: Scratch any unwanted files and compress the catalog using a MOVE statement, or mount a new disk platter.

Example:

```
SAVE DCF "PRGM"
               ↑ERR 88
```

Appendix B
Model 2230 Specifications

MODEL 2230

STORAGE CAPACITY

DISK MODEL	SECTORS PER PLATTER	TOTAL SECTORS	BYTES PER PLATTER	TOTAL BYTES
2230-1	2,400	4,800	614,400	1,228,800
2230-2	4,800	9,600	1,228,800	2,457,600
2230-3	9,792	19,584	2,506,752	5,013,504

LEGAL SECTOR ADDRESSES

DISK MODEL	SECTORS PER PLATTER	LOWEST LEGAL ADDRESS	HIGHEST LEGAL ADDRESS
2230-1	2,400	000	2399
2230-2	4,800	000	4799
2230-3	9,792	000	9791

PERFORMANCE

Rotation Speed
All configurations . . . . 1500 rpm

Access Time (Position Head to Track)

Minimum (one track) . . . . 9 ms
Average (across one-half available tracks)
2230-1 . . . . 21 ms
2230-2 . . . . 28 ms
2230-3 . . . . 41 ms
Maximum (across all available tracks)
2230-1 . . . . 42 ms
2230-2 . . . . 56 ms
2230-3 . . . . 82 ms

Latency Time (Platter Rotation to Sector on Track)
Average (one-half revolution) . . . . 20 ms

Read/Write Time
One 256-byte sector (including CPU/controller overhead) . . . . 13 ms

Raw Transfer Rate . . . . 195,000 bytes/sec

Move/Copy Time (Entire Disk Platter)

2230-1	Approx 1.2 min
2230-2	Approx 2.4 min
2230-3	Approx 4.9 min

GENERAL SPECIFICATIONS

Physical Dimensions

Height	32.5 in. (82.6 cm)
Width	17.5 in. (44.5 cm)
Depth	29 in. (73 cm)

Weight	126 lb (57 kg)

Power Requirements

Voltage: 115 or 230 VAC ±10%
50 or 60 Hz ± 1 cycle

Power: 800 watts start-up
425 watts running.

Cabling

10 ft (3m) cable with connector to female receptacle on the CPU

Operating Environment*

50°F - 90°F (10°C - 35°C)
20% - 80% Relative Humidity

*See Appendix E for more detailed information on Model 2230 operating environment and the proper storage environment for Model 2230 disk cartridges.

Appendix C
Model 2260 Specifications

MODEL 2260

STORAGE CAPACITY

SECTORS PER PLATTER	TOTAL SECTORS	BYTES PER PLATTER	TOTAL BYTES
19,584	39,168	5,013,504	10,027,008

LEGAL SECTOR ADDRESSES

TOTAL SECTORS EACH PLATTER	LOWEST LEGAL ADDRESS	HIGHEST LEGAL ADDRESS
19,584	000	19583

PERFORMANCE

Rotation Speed . . . . 2400 rpm

Access Time (Position Head to Track)
- Minimum (one track) . . . . 4.1 ms
- Average (across one-half available tracks) . . . 40 ms
- Maximum (across all tracks) . . . . 80 ms

Latency Time (Platter Rotation to Sector on Track)
- Average (one-half revolution) . . . . 12.5 ms

Read/Write Time
- One 256-byte sector (including CPU/controller overhead) . . . . 13 ms

Raw Transfer Rate . . . . 312,500 bytes/sec

Move/Copy Time (Entire Disk Platter) . . . . Approx 10 min

GENERAL SPECIFICATIONS

Physical Dimensions
- Height . . . . 32.5 in. (82.6 cm)
- Width . . . . 17.5 in. (44.5 cm)
- Depth . . . . 29 in. (73 cm)

Weight . . . . 126 lb (57 kg)

Power Requirements
- Voltage: 115 or 230 VAC ± 10%
  50 or 60 Hz ± 1 cycle

Power: 800 watts start-up.
425 watts running.

Cabling
10 ft (3 m) cable with connector to female receptacle on the CPU

Operating Environment*
50°F - 95°F (10°C - 35°C)
20% - 80% Relative Humidity

*See Appendix E for more detailed information on the Model 2260 operating environment, and the proper storage of Model 2260 disk cartridges.

Appendix D
Model 2270 Specifications

MODEL 2270

STORAGE CAPACITY

DISK MODEL	SECTORS PER PLATTER	TOTAL SECTORS	BYTES PER PLATTER	TOTAL BYTES
2270-1	1,024	1,024	262,144	262,144
2270-2	1,024	2,048	262,144	524,288
2270-3	1,024	3,072	262,144	786,432

LEGAL SECTOR ADDRESSES

DISK MODEL	SECTORS PER PLATTER	LOWEST LEGAL ADDRESS	HIGHEST LEGAL ADDRESS
2270-1,2,3	1,024	000	1023

PERFORMANCE

Rotation Speed
All configurations . . . . 360 rpm

Access Time (Position Head to Track)
Minimum (one track) . . . . 14 ms
Average (across one-half available tracks) . . . 363 ms
Maximum (across all available tracks) . . . . . 726 ms

Latency Time (Platter Rotation to Sector on Track)
Average (one sector read/write, one-half revolution) . . . . 83.3 ms

Read/Write Time
One 256-byte sector (including CPU/controller overhead) . . . . 21.8 ms

Raw Transfer Rate . . . . 30,000 bytes/sec

Move/Copy Time (Entire Disk Platter) . . . . Approx 2 min

GENERAL SPECIFICATIONS

Physical Dimensions
Height . . . . 19 in. (47.5 cm)
Width . . . . 17.5 in. (43.5 cm)
Depth . . . . 16.3 in. (40.8 cm)

Weight . . . . 68 lb (30.6 kg)

Power Requirements
Voltage: 115 or 230 VAC ± 10%
50 or 60 Hz ± 1 cycle

Power: 225 Watts

Cabling
12 ft (3.7 m) cable with connector to female receptacle on disk controller board in the CPU

Operating Environment
50°F - 95°F (10°C - 35°C)
20% - 80% Relative Humidity

Recommended Operating Environment
40% - 60% Relative Humidity

Appendix E
Model 2230/2260 Disk Cartridge Maintenance Information

In order to maintain the original high quality of the disk cartridges, it is important that proper care be observed in their handling and storage. This Appendix lists several recommended procedures for the operation, handling, and storage of the disk cartridge, proper attention to which will ensure the continued dependable and efficient performance of the cartridge.

1. General Handling Precautions

The following general precautions apply:

a) Reassemble the cartridge and bottom cover when the cartridge is not installed in the disk drive.

b) Clean the covers periodically with a clean, lint-free cloth to remove any buildup of dust.

c) Replace cracked, distorted, or otherwise damaged bottom covers.

d) Keep beverages, tobacco, and smoking accessories off the disk unit.

e) Clean the machine room daily using a vacuum cleaner or damp mop. Do not raise dust with cleaning implements such as brooms or feather dusters.

f) Do not expose the cartridge to intense magnetic fields such as those generated by high-current bus bars, cables, and welding transformers. A field intensity of more than 50 gauss may cause loss of information.

g) Do not store the cartridge in direct sunlight.

h) If you drop a cartridge, have it inspected by a Wang Service Representative before attempting to use it on a disk drive.

The cartridge may contain information that is valuable to your installation. Protect it as much as possible.

2. Carrying

The disk cartridge should be carried flat or on its side. To carry the cartridge flat, simply grasp the cartridge handle. To carry a cartridge on its side, hold it with your fingers in the recessed handle compartment, with your thumb gripping the beveled edge on the bottom cover.

3. <u>Labeling</u>

Cartridges may be marked for identification with either a felt tip pen or an adhesive label. Wang recommends that the top surface of the cartridge be used as the labeling surface. You may want to duplicate the label on the outside of the bottom cover to facilitate identification. However, you should refer to the label on the top surface of the cartridge for positive identification of information contained on the cartridge.

Use the following precautions when labeling a cartridge:

a) Use only good quality adhesive labels. Inferior labels could work loose while the cartridge is loaded, and cause severe damage to the read/write heads or the disk surface.

b) Mark the label before you put it on the cartridge.

c) Remove old labels.

d) If the cartridge has been labeled with a felt-tip pen, use isopropyl alcohol and a soft, lint-free cloth to alter or remove old markings.

4. <u>Operating Environment</u>

Disk cartridges that are in frequent use should be stored in the machine room or in a similar environment. Cartridges that have not been stored in the machine room should be conditioned to machine room temperature for two hours prior to use. The conditioning time is necessary to insure accurate track registration, data recording, and data retrieval.

The operating requirements are:

Temperature:	50°F (10°C) to 100°F(37.8°C)
Relative Humidity:	20% to 80%
Maximum wet bulb:	78°F (25.5°C)

5. <u>Storage</u>

A disk cartridge, locked in its bottom cover, forms a sealed storage container. Unless cartridges are to be stored for a long period of time, no further protection is required. Cartridges may be stacked on top of each other (no more than five high) or stored on their sides. Clean, dust-free cabinets made of metal or other fire resistant material provide satisfactory storage facilities for the cartridges.

If the cartridges are to be stored for a long period, they should be repacked in their original shipping containers before storing. This protects the covers from excessive dust and/or dirt accumulation.

6. <u>High Security Storage</u>

Store cartridges containing vitally important data or duplicate master records in a cabinet or storeroom that provides protection against catastrophic damage. The cabinet or storeroom should be insulated to prevent the internal temperature from rising above 150°F (66°C) in case of fire.

7. <u>Storage Environment</u>

   a) For short term requirements, the cartridge should be stored in the machine room or similar environment.

   b) Long term storage:

Temperature	-40°F to 150°F (-40°C to 66°C)
Relative Humidity:	8 to 80%

   If the short term requirements are exceeded, you should condition the cartridge to the machine room environment for two hours prior to use.

8. <u>Shipping and Receiving</u>

Disk cartridges are protected in transit by packaging assemblies designed to withstand normal shipping abuse. Upon receiving a cartridge, examine the shipping container for possible shipping damage. If you find any, have a Wang Service Representative inspect the cartridge prior to installing it on a disk drive. This will eliminate the possibility of damaging the drive or further damaging the cartridge.

Appendix F
Disk Cartridge Compatibility (Models 2230 and 2260)

F.1 Two Types of Disk Cartridges (Removable Disk Platters)

The disk cartridges (Removable Disk Platters) used in the Models 2230 and 2260 Disk Drives are of two different types:

- Cartridges approved for use in the Model 2230. These cartridges are certified for 100 tpi (tracks per inch). The number of available tracks on a cartridge is determined by the Model 2230 configuration in which the cartridge is formatted:

	Tracks per Recording Surface	Total Tracks
Model 2230-1	50	100
Model 2230-2	100	200
Model 2230-3	204	408

On all cartridges approved for the Model 2230, tracks are spaced .01 inch apart on a recording surface.

- Cartridges approved for use in the Model 2260. These cartridges are certified for 200 tpi (tracks per inch). In addition, they are tested to ensure that they contain 408 usable tracks on each recording surface (816 total tracks). The tracks on a cartridge approved for the Model 2260 are spaced .005 inch apart.

F.2. Ordering Information for Model 2230 and 2260 Disk Cartridges

To order disk cartridges for the Models 2230 and 2260, specify one of the following part numbers:

Model 2230 - Approved Cartridge (100 tpi) - 177-0041
Model 2260 - Approved Cartridge (200 tpi) - 177-0062

> WARNING:
>
> Model 2260 owners should be aware that, although Wang Laboratories tests all 408 tracks on each recording surface of our 200 tpi cartridges, cartridges purchased outside of Wang Laboratories may not be tested for the entire 408 tracks (even if the cartridge is certified for 200 tpi). Since the Model 2260 utilizes every available track on the cartridge (no tracks are reserved as "extras"), the loss of even a single faulty track can result in serious problems for the user. Wang therefore recommends that disk cartridges for the Model 2260 be obtained exclusively through Wang Laboratories.

F.3 Differences in Platter Format (Models 2230 and 2260)

Although disk cartridges for the Model 2230 and Model 2260 have the same physical dimensions (15 inches in diameter), and are physically interchangeable between the two disk units, there are differences both in track density and platter format (the arrangement of tracks and sectors on the recording surface) between the two types of cartridges. The differences between the two cartridges are explained below in some detail. Section F.4 of this Appendix describes a special procedure available on the Model 2260 which makes it possible to read a disk cartridge originally formatted and recorded in a Model 2230 disk drive.

> NOTE:
>
> Cartridges approved for use in a Model 2260 carry a label which reads:
>
> APPROVED FOR WANG 2260
> (200 Tracks Per Inch)
>
> Similarly, cartridges approved for use in a Model 2230 carry a label which reads:
>
> APPROVED FOR WANG 630/730/2230
>
> In both cases, the label is affixed in the area of the cartridge handle.

The track density of a Model 2260-approved cartridge is twice as great as that of a Model 2230 cartridge (200 tracks per inch as opposed to 100 tracks per inch). Correspondingly, consecutive tracks on a Model 2260 cartridge are half as far apart (.005 inch) as tracks on a 2230 cartridge (.01 inch). In addition to the difference in track densities, cartridges formatted in the Models 2230 and 2260 have different sector arrangements.

During the platter formatting procedure (described in Chapter 2), the number and location of the sectors on a disk platter are assigned by the disk drive. A specified number of usable tracks (the number depending upon the disk model) are subdivided into 24 sectors each, and each sector is assigned a unique sector address. When a platter is formatted in the Model 2230, the disk writes sequential sector addresses into every sixth sector (see Chapter 1, Section 1.9). Sequential sectors are therefore staggered one-quarter of a track, or six physical sectors, apart on a platter formatted in the Model 2230. The staggered arrangement of sectors enables the system to read as many as four sequential sectors in a single revolution during certain large data transfer operations such as MOVE or COPY. On the much faster Model 2260, sequential sectors are staggered one-half track, or 12 sectors, apart for the same reason.

F.4 Accessing a Model 2230 Cartridge (Removable Platter) in a Model 2260 Disk Drive

The different track densities of the Model 2230 and Model 2260 disk cartridges, as well as the different platter formats employed by the two disk models, dictate that disk cartridges used in the two disks are not compatible. It was anticipated, however, that some Model 2230 owners might wish to trade up to the larger Model 2260; for their convenience, the Model 2260 has a special feature which makes is possible to load a Model 2230 cartridge into the 2260 Disk Drive and access it for reading data and programs. This special feature enables the programmer who trades in a Model 2230 for a Model 2260 to transfer his data base (recorded on one or more cartridges) from the Model 2230 to the Model 2260. The need to recreate the entire data base for the larger disk is thereby avoided.

In order to access a Model 2230 disk cartridge in a Model 2260 Disk Drive, a special disk device address of 350 must be used. The special address causes the Model 2260 to initiate a routine which simulates the Model 2230 track spacing and platter format for reading on the Removable Platter. The programmer is then able to read information from the cartridge in Model 2230 format with standard disk statements, while the cartridge is loaded in a Model 2260 drive. For example, the statement LIST DC R /350 generates a listing of the Catalog Index from a Model 2230 cartridge loaded in a Model 2260 drive. Similarly, the special address can be assigned to a file number, and referenced indirectly in statements which do not permit the use of a device address. For example, the statements

```
SELECT #3 350
DATALOAD DC OPEN R #3, "TEST"
```

could be used to reopen the file TEST on a Model 2230 cartridge in a Model 2260 Disk Drive.

If the Model 2260 is the second, or subsequent, disk drive in a system, the special address must be calculated by adding a HEX(40) to the disk device address. For example, if the address of the Model 2260 is 320, the special address is 360; if the normal address is 330, the special address is 370, etc. Note that the special address should not be used to access the Fixed Platter of the Model 2260, nor should it be used to access a Removable Platter formatted in the Model 2260 itself.

> NOTE:
>
> The special procedure described above applies only to reading a 2230 cartridge in a 2260 drive; information cannot be written on a Model 2230 cartridge by the 2260 disk unit.

F.5 Model 2260 Cartridge in a Model 2230 Disk Drive

A disk cartridge formatted and recorded in the Model 2260 Disk Drive can be physically loaded into a Model 2230 drive, but it cannot be accessed for reading or recording. Unlike the Model 2260, the Model 2230 has no capability to compensate for the differences in platter format between the two disk models. A cartridge formatted in the Model 2260 can, of course, be reformatted in the Model 2230 and used as a standard Model 2230 cartridge. Note, however, that the reformatting procedure wipes out all information stored on the cartridge, and the additional tracks on the Model 2260 cartridge are not used by the Model 2230 when the cartridge is reformatted.

F.6 Illegality of MOVE and COPY (Model 2230 Cartridge to Model 2260 Fixed Platter)

The MOVE and COPY statements cannot be used to transfer information between a Model 2230 disk cartridge and the Fixed Disk Platter of the Model 2260. The illegality of MOVE and COPY in this case results from the fact that only one disk address can be specified in a MOVE or COPY statement, while two different addresses are required to access the two platters (the special address for the Model 2230 cartridge, and the normal disk address for the Fixed Platter). Information can, however, be transferred between the Fixed and Removable Platters with any one of a number of short programs. The brief routine below utilizes the DATASAVE BA statements to copy a range of sectors specified by the user from the Removable to the Fixed Platter, in effect simulating a COPY RF operation.

```
10 INPUT "BEGINNING AND ENDING SECTOR ADDRESSES", A,B
20 DIM A$ (16) 16
30 SELECT #3 350, $4 310
40 DATALOAD BA R #3, (A,A) A$()
50 DATASAVE BA F #4, (A-1, L) A$()
60 IF A < = B THEN 40
70 STOP "END OF COPY"
```

This routine requires only that the operator enter the first and last sectors he wishes to copy. All sectors between and including those sectors are copied from the Model 2230-formatted Removable Platter to the Model 2260 Fixed Platter. Note that the special address (350) is used to access the Removable Platter, while the normal disk address (310) is used to access the Fixed Platter. Information cannot be copied in the opposite direction (Model 2260 Fixed Platter to 2230 cartridge).

It is evident that when cataloged files are copied with this routine, scratched files are not automatically deleted from the catalog. In order to delete scratched files, the Model 2230 cartridge must be replaced with a cartridge formatted by the Model 2260, and the catalog must be moved back onto the cartridge with a MOVE statement.

Appendix G
A Glossary of Disk Terminology

absolute sector address - An address permanently assigned to a disk sector.

Absolute Sector Addressing Mode - A mode of disk operation which enables the programmer to address individual sectors on disk. Also referred to as 'direct addressing' mode.

access - See 'disk access' and 'file access'.

argument - In a DATASAVE DC or DA statement, a discrete value, specified directly (as a numeric value or literal string in quotes) or indirectly (as the value of a variable or array element). Each argument occupies a single field in the record on disk, and is separated from neighboring fields by a Start-of-Value (SOV) bytes. In a DATALOAD DC or DA statement, each receiving variable or array element which receives one value when the record is read from disk is regarded as a receiving argument. For the most part, multiple arguments in a statement must be separated by commas; however, when an array designator is used to specify an entire array, each element of the array is regarded as a separate argument.

argument list - The list of all arguments in a DATASAVE DC/DA or DATALOAD DC/DA statement.

Automatic File Cataloging Mode - A mode of disk operation in which the names and locations of files on disk are maintained automatically by the system in a Catalog Index.

binary address - A sector address expressed as a two-byte binary number.

binary search - A dichotomizing search in which the number of records in the file is divided into two equal parts at each step in the search.

blocked records - Two or more short records stored in one sector. Since the minimum length of any record is, from the system's point of view, one sector, the blocking of multiple records in a single sector must be a function of user's software.

Catalog Area	- The area on a disk platter reserved for the storage of cataloged files.
Catalog Index	- An index containing names and pointers for each cataloged file in the Catalog Area.
command	- A BASIC statement which cannot be executed on a numbered statement line. See 'statement'.
control byte	- Any of several special bytes created automatically by the system to help it keep track of data stored on the disk, and which are completely transparent to the user's software. See also 'start-of-value control byte' and 'sector control byte'.
cyclic redundancy check	- A special checksum test automatically performed by the disk unit on all data read from the disk. Abbreviated CRC.
cylinder	- On the Models 2230 and 2260, the number of sectors which can be accessed without repositioning the access arm (96 sectors).
data file	- A collection of related data records treated as a logical unit. For example, an inventory file contains a number of inventory records, each of which in turn consists of specified items of information about a particular item in the inventory. In catalog mode, a data file can be opened or reopened by name.
data record	- See 'logical data record'.
default address	- The device address for a System 2200 peripheral which is used automatically by the system when no other address is specified. For the disk unit, the system default address is 310. The disk default address is always stored opposite the default file number (#0) in the Device Table, and may be changed temporarily with a SELECT DISK statement. However, the system default address (310) is automatically returned to the default slot upon Master

Initialization. See also 'device address'.

default file number - The file number in the Device Table automatically used by the system when a disk statement or command is executed which does not specify a file number. The default file number is always #0, and cannot be changed. The default disk address is always stored in the slot opposite the default file number. See also 'default address', 'Device Table', and 'file number'.

device address - A three-digit hexadecimal code used by the CPU to identify each peripheral device. The device address is set in the controller board for each peripheral either at the factory or by a Wang Service Representative, and should be clearly printed on the top surface of the controller board. See also 'default address', 'device type', and 'unit device address'.

Device Table - A special section of memory used to store disk device addresses and sector address parameters for currently open files on disk. The Device Table is located within the 696 bytes of memory reserved for "housekeeping" purposes by the System 2200 CPU. It consists of seven rows, or "slots", identified by file numbers #0 - #6. A device address and a set of sector address parameters for an open file can be stored in each slot. The slots opposite file numbers #1 - #6 are also used for other I/O devices in addition to the disk (such as tape cassette drives, paper tape readers, and card readers). The default slot (opposite #0) is used only for the disk, however. Default addresses for other I/O devices are stored in another section of memory. See also 'default address', 'default file number', and 'file number'.

device type - The first digit of the three-digit device address. For the disk unit, the device type can be either '3' or 'B' (e.g., 3XX, or BXX). When used in conjunction with the 'T' parameter, the device type determines which disk platter in a multi-platter disk unit is to be accessed. In this case, a device type of '3' identifies the 'F' disk platter, while a device type of 'B' identifies the 'R' disk platter. For the Model 2270-1 Single Removable Diskette Drive, and for the third platter of the Model 2270-3, a device type of 'B' is illegal. See also 'device address' and 'unit device address'.

disk access - Any disk read or write operation. See also 'file access'.

disk drive - 1. Broadly, a disk unit containing one or more disk platters. 2. More specifically, the assembly (consisting of drive motor, spindle, and access arm(s)) which drives the disk platter(s) and is activated by a single disk command. In the Models 2230 and 2260, both platters are driven by a single disk drive; in the Model 2270, each platter is driven by an independent drive. See also 'disk platter'.

disk latency period - The period of time which elapses from the time the read/write head positions itself to a track until the desired sector in that track rotates to the read/write head's position. Disk latency time is determined by the rotation speed of the disk unit. Latency time may be important for random access operations; it is generally negligible in sequential access operations. See also 'track access time'.

disk platter - The flat, circular plastic or metal plate which is coated on its recording surface with a magnetic substance such as iron oxide, and which serves as the storage medium in a disk unit. Each platter in a Model 2230 or 2260 has two recording surfaces; each platter in the Model 2270 has only a single recording surface.

Electronic Data Processing - The storage and processing of usually significant volumes of data by electronic devices such as electronic digital computers and associated peripheral devices. Abbreviated EDP.

ending sector address - The address of the last sector in a file or multi-sector logical record. See 'starting sector address' and 'absolute sector address'.

end-of-file trailer record - A special record, one sector in length, which marks the end of currently stored data in a data file. The end-of-file record is created with a DATASAVE DC END or DATASAVE DA END statement. Creation of an end-of-file trailer record in a cataloged file automatically causes the 'used' column in the Catalog Index to be updated, and enables the programmer to check for the end-of-file with an IF END THEN statement, or to skip to the end-of-file of a cataloged file with a DSKIP END statement.

expression - A numeric value (e.g., '1234'), operation (e.g., 'A*B↑2'), variable (e.g., 'N') or array element (e.g., 'N(3)').

field - 1. An individual item of data within a logical data record on the disk. Each argument in the DATASAVE DC or DATASAVE DA argument list is recorded as a single field (marked off by SOV control bytes) in the logical record created by the statement.
2. A specified section of a record reserved for a particular type of information. For example, a 'key field' consists of a number of bytes located at a specific place in a record which always holds the key value for the record.

file - A collection of related records treated as a logical unit. Files may be of two types, program files and data files. In catalog mode, files can be created and accessed by name. See 'data file' and 'program file'.

file access - 1. Any disk operation in which information (programs or data) is read from or written in a file on disk.
2. Any disk operation which results in positioning the read/write head to a location preparatory to reading or writing information in a file. See also 'disk access'.

file number - One of the seven numbers #0 - #6 associated with slots in the Device Table, and used to identify currently open files on disk. File numbers #1 - #6 are also used to identify tape files. A file number is always preceded by a "#" symbol. See 'default file number' and 'Device Table'.

hashing technique - A technique for storing and accessing information on disk in which a specialized algorithm, called a "hash function", is used to convert a record's key value into an absolute sector address, which is then used as the location at which the record is stored. This technique is used by the system in catalog mode to store file names in the Catalog Index.

header record - A record containing special control information and preceding all other records in a file. Every program file saved on disk begins with a one-sector header record. In cataloged programs, the header contains the program name, along with catalog system control information. Data files on disk have no header record, but cataloged data files do have a system control record at the end of the file which serves the same purpose as a header. See 'trailer record' and 'system control record'.

Hog Mode - A mode of disk multiplexer operation in which one station obtains exclusive access to the disk, while all other stations are locked out.

key field - A field in a record on the disk consisting of one or more bytes, and containing the key value for that record. See 'field' and 'key value'.

key value - A numeric or alphanumeric value in a record used to identify the record for purposes of access and control. See 'key field', 'sort', and 'hashing technique'.

logical data record - A data record on the disk created by a DATASAVE DC or DATASAVE DA statement which occupies one or more sectors, and contains all of the data from the DATASAVE DC or DATASAVE DA argument list. See also 'record' and 'data file'.

logical record - See 'logical data record'.

longitudinal redundancy check - A checksum test performed by the system on each sector of data read from the disk. Abbreviated LRC.

multiplexing - A process of allocating disk time to a number of systems by sequentially interleaving disk operations from the various inquiring systems.

multi-volume file - A file occupying two or more disk platters (or tape cassettes). Each separate platter is considered a different "volume" of the file. Each volume must be carefully identified with a file name and a volume number.

parameter - An element in a BASIC statement or command which follows the BASIC verb, and whose function and meaning are defined for the purposes of the statement. Parameters may be of two types, constant (or fixed) and variable. The value of a fixed parameter is predefined and cannot be altered by the user. The value of a variable parameter is specified by the user, although there are normally certain limitations imposed upon the range of values which may be assigned to a particular parameter. A fixed parameter is always indicated in the general form of a statement or command as an uppercase letter (e.g., 'P', 'DC', 'S', etc.), while a variable parameter is indicated with a lowercase letter (e.g., 'xxx', 'n') or described with a lowercase literal string (e.g., 'name', 'sector address', etc.).

pointer - An absolute sector address or displacement which "points" to the location of a record on the disk.

program file - A file on disk consisting of a single BASIC program or program segment, and optionally also containing extra sectors reserved for possible future expansion of the program. A program file always begins with a header record and ends with a trailer record. In catalog mode, a program file can be saved and loaded by name.

program record - A sector in a program file between the header record and the trailer record which contains program text. See 'header record' and 'trailer record'.

protect parameter - A special parameter ('P') used to protect programs saved on disk or tape.

protected program - A program on disk or tape which can be loaded and run, but cannot be listed or resaved.

read-after-write verification - An optional verification check which can be performed on each sector of data as it is written on the disk. The read-after-write check is specified by including the dollar sign ('$') parameter in a disk statement or command. However, a read-after-write check effectively doubles the execution time of the disk operation.

read/write head - An electromagnetic recording head which reads and writes information on the recording surface of a disk platter.

record - A collection of related items of data treated as a logical unit. See 'logical data record' and 'data file'.

sector - The basic unit of storage on a disk platter, consisting of a data field with a fixed length of 256 bytes, an absolute sector address, and certain control information. Each sector is regarded as a discrete unit, and is directly accessible by the system.

sector control bytes - Special control bytes containing system control information which are written automatically by the system into each sector of a logical data record and each program record stored on disk. Each sector in a logical data record contains three sector control bytes; each one-sector program record in a program file contains two sector control bytes. The sector control bytes are transparent to the user software.

sort - 1. To arrange data sequentially in ascending or descending order.
2. To sequentially order logical data records in a file based upon the key values of the records.
3. The act of performing a sorting operation.

starting sector address - The address of the first sector in a file or multi-sector logical record. See also 'ending sector address'.

start-of-value control byte - A control byte created automatically by the system independent of user software, and prefixed to each field in a logical record when the record is written with a DATASAVE DC or DATASAVE DA statement. This control byte separates fields within a record and marks the beginning of each new field. The start-of-value bytes are not automatically written when a DATASAVE BA statement is executed. Abbreviated SOV.

statement - 1. Broadly, a generic term for all System 2200 BASIC instructions. The class of statements is divided into two sub-classes: programmable statements (also called simply 'statements'), and non-programmable statements (generally referred to as 'commands').
2. In a narrower sense, the term 'statement' is often used as a synonym for 'programmable statement' to denote only those statements which can be executed on a numbered program line. See 'command'.

system control record - A special record one sector in length which always occupies the last sector of a cataloged data file, and contains control information and pointers for the file. A system control record is automatically created and updated by the system for each data file maintained in catalog mode; it is completely transparent to the user's software.

temporary files - Files established outside the Catalog Area on a disk, generally for the storage of transient data. Temporary files cannot be named, and no entry is listed for them in the Catalog Index.

They can, however, be accessed with catalog procedures.

track - Any of the concentric circular electromagnetic paths into which the recording surface of a disk platter is divided. Each track, in turn, is subdivided into a number of sectors. The number of tracks on a platter differs according to the disk model and configuration. See 'sector' and 'disk platter'.

track access time - The time required for the access assembly to move the read/write head from its current position to the track containing the desired sector. For random access operations, the track access time may become significant if the sectors to be accessed are scattered on widely separated tracks. For most sequential access operations, however, the track access time is negligible. See also 'disk latency time'.

trailer record - 1. In program files, the sector immediately following the last program record. The trailer record contains control information, written automatically by the system, along with the last few lines of program text.
2. In data files, a special record created by specifying the 'END' parameter in a DATASAVE DC or DATASAVE DA statement, to mark the limit of valid data in the file. Also referred to as an "end-of-file" trailer record. See 'end-of-file trailer record'.

unit device address - The last two digits of the three-digit device address (e.g., X10, X20, X50, etc.), which identify individual disk units when more than one is attached to the same system. See 'device address', and 'device type'.

work files - See 'temporary files'.

Appendix H
Bibliography

The techniques involved in creating, maintaining, and accessing disk-based data files have been, and continue to be, the subjects of an extensive number of textbooks and articles. The authors included in this bibliography approach the programming problems associated with disk storage from a variety of different perspectives, and with varying degrees of sophistication. In general, however, the bibliography has been heavily weighted toward the relative novice, although in all cases some background in programming is required.

It is suggested that the programmer with little or no experience in disk operations begin with a text which provides a general survey of the standard types of disk file structures and access techniques. (The titles identified with asterisks provide such a survey at an introductory or intermediate level.) The number of disk storage and access techniques which have been developed over the last 10 or 15 years is considerable, even if one restricts oneself only to the "standard" techniques, and each has particular strengths and weaknesses which make it suitable for some applications and most unsuitable for others. Armed with an overview of the available systems and techniques, the programmer will be in a position to determine which of them most appropriately suit his own application. He then can proceed to a textbook or article which treats the chosen technique(s) in greater depth.

1. Bosco, R.L., Data Bases, Computers, and the Social Sciences (Wiley-Interscience, New York, 1970).

2. Brooks, F.P., and K.E. Iverson, Automatic Data Processing (John Wiley and Sons, New York, 1963).

3. Clemenson, W.D., "File Organization and Search Techniques," Annual Review of Information Science and Technology, Volume 1, Ed. C. Cuadra (John Wiley and Sons, New York, 1966).

4. Daley, R.C., and P.G. Newmann, "A General Purpose File System for Secondary Storage," Proceedings of the AFIPS 1965 Fall Joint Computer Conference, Volume 27, Part 1 (Spartan Books, New York).

5. Dodd, G.G., "Elements of Data Management Systems," Computer Surveys, Volume 1, No. 2, June 1966.

*6. Forsythe, A.I., and T.A. Keenan, E.I. Organick, and W. Stenberg, Computer Science: A First Course (John Wiley and Sons, New York, 1969).

7. Gear, C.W., Computer Organization and Programming (McGraw Hill, New York, 1969).

8. Gruenberger, F. (Ed.), Critical Factors in Data Management (Prentice-Hall, Englewood Cliffs, N.J., 1969).

9. Hsiao, D. and Harary, F., "A Formal System for Information Retrieval from Files," Communications of the ACM, Volume 13, Number 2 (February, 1970).

10. Hull, T.E. and D.F. Day, Computers and Problem Solving (Addison-Wesley (Canada) Ltd., Don Mills, Ontario, 1970).

11. Iverson, K.E., A Programming Language (John Wiley and Sons, New York, 1962).

12. Johnson, L.R., "Indirect Chaining Method for Addressing on Secondary Keys," Communications of the ACM, Volume 4, Number 4 (May 1961).

13. Korfhage, R.R., Logic and Algorithms (John Wiley and Sons, New York, 1966).

14. Knuth, D.E., The Art of Computer Programming, Volumes I and III (Addison-Wesley, Reading, Mass., 1968).

*15. Lefkowitz, D., File Structures for On-Line Systems (Spartan Books, New York, 1969).

16. Lowe, T.C., "The Influence of Data-Base Characteristics and Usage on Direct Access File Organization," Journal of the ACM, Volume 15, Number 4 (October, 1968).

17. Martin, J., Design of Real-Time Computer Systems (Prentice-Hall, Englewood Cliffs, N.J., 1967).

18. Mauer, W.D., "An Improved Hash Code for Scatter Storage," Communications of the ACM, Volume 11, Number 1 (January, 1968).

19. McIlroy, M.D., "A Variant Method of File Searching," Communications of the ACM, Volume 6, Number 1 (January, 1963).

20. Meadow, C.T., The Analysis of Information Systems (John Wiley and Sons, New York, 1967).

21. Morris, R., "Scatter Storage Techniques," Communications of the ACM, Volume 11, Number 1, (January, 1968).

22. Peterson, W.W., "Addressing for Random-Access Storage," IBM Journal of Research and Development, Volume 1 (1957).

23. Rosove, P.E., Developing Computer-Based Information Systems (John Wiley and Sons, New York, 1967).

24. Williams, W.F., Principles of Automatic Information Retrieval (The Business Press, Elmhurst, Illinois, 1968).

*25. Yourdon, E., Design of On-Line Computer Systems (Prentice-Hall, Englewood Cliffs, N.J., 1972).

*Titles marked with an asterisk are intermediate-level texts recommended for programmers with limited background in disk operations.

Appendix I
Equipment Guarantee and Preventive Maintenance Information

GUARANTEE

Your equipment is guaranteed from defects in materials and workmanship for a period of ninety days (one year for State and Federal Governments).

MAINTENANCE

It is recommended that your equipment be serviced semi-annually. Wang Laboratories offers a Maintenance Agreement which automatically ensures proper servicing. If no Maintenance Agreement is purchased, all servicing must be requested by the customer. A Maintenance Agreement protects your investment and offers the following benefits:

Preventive Maintenance:

Semi-annually, your equipment is inspected for worn parts, lubricated, cleaned, and updated with engineering changes, if any. Preventive maintenance minimizes "downtime" by anticipating repairs before they are necessary.

Fixed Annual Cost:

When you buy a Maintenance Agreement, you issue only one purchase order for service for an entire year and receive one annual billing, or more frequent billing, if desired.

Further information regarding a Maintenance Agreement can be obtained from your local Wang Sales/Service Office.

NOTE:

Wang Laboratories, Inc. can neither guarantee nor honor maintenance agreements for any equipment modified by the user. Damage to equipment incurred as a result of such modification becomes the financial responsibility of the user.

Appendix J
System 2200 Error Codes

ABBREVIATED ERROR MESSAGES

ERR 01 TEXT OVERFLOW
ERR 02 TABLE OVERFLOW
ERR 03 MATH ERROR
ERR 04 MISSING LEFT PARENTHESIS
ERR 05 MISSING RIGHT PARENTHESIS
ERR 06 MISSING EQUALS SIGN
ERR 07 MISSING QUOTATION MARKS
ERR 08 UNDEFINED FN FUNCTION
ERR 09 ILLEGAL FN USAGE
ERR 10 INCOMPLETE STATEMENT
ERR 11 MISSING LINE NUMBER OR CONTINUE ILLEGAL
ERR 12 MISSING STATEMENT TEXT
ERR 13 MISSING OR ILLEGAL INTEGER
ERR 14 MISSING RELATION OPERATOR
ERR 15 MISSING EXPRESSION
ERR 16 MISSING SCALAR
ERR 17 MISSING ARRAY
ERR 18 ILLEGAL VALUE
ERR 19 MISSING NUMBER
ERR 20 ILLEGAL NUMBER FORMAT
ERR 21 MISSING LETTER OR DIGIT
ERR 22 UNDEFINED ARRAY VARIABLE
ERR 23 NO PROGRAM STATEMENTS
ERR 24 ILLEGAL IMMEDIATE MODE STATEMENT
ERR 25 ILLEGAL GOSUB/RETURN USAGE
ERR 26 ILLEGAL FOR/NEXT USAGE
ERR 27 INSUFFICIENT DATA
ERR 28 DATA REFERENCE BEYOND LIMITS
ERR 29 ILLEGAL DATA FORMAT
ERR 30 ILLEGAL COMMON ASSIGNMENT
ERR 31 ILLEGAL LINE NUMBER
ERR 33 MISSING HEX DIGIT
ERR 34 TAPE READ ERROR
ERR 35 MISSING COMMA OR SEMICOLON
ERR 36 ILLEGAL IMAGE STATEMENT
ERR 37 STATEMENT NOT IMAGE STATEMENT
ERR 38 ILLEGAL FLOATING POINT FORMAT
ERR 39 MISSING LITERAL STRING
ERR 40 MISSING ALPHANUMERIC VARIABLE
ERR 41 ILLEGAL STR(ARGUMENTS
ERR 42 FILE NAME TOO LONG
ERR 43 WRONG VARIABLE TYPE
ERR 44 PROGRAM PROTECTED
ERR 45 PROGRAM LINE TOO LONG
ERR 46 NEW STARTING STATEMENT NUMBER TOO LOW
ERR 47 ILLEGAL OR UNDEFINED DEVICE SPECIFICATION
ERR 48 UNDEFINED SPECIAL FUNCTION KEY
ERR 49 END OF TAPE
ERR 50 PROTECTED TAPE
ERR 51 ILLEGAL STATEMENT
ERR 52 EXPECTED DATA (NONHEADER) RECORD
ERR 53 ILLEGAL USE OF HEX FUNCTION
ERR 54 ILLEGAL PLOT ARGUMENT
ERR 55 ILLEGAL BT ARGUMENT
ERR 56 NUMBER EXCEEDS IMAGE FORMAT
ERR 57 ILLEGAL VALUE
ERR 58 EXPECTED DATA RECORD
ERR 59 ILLEGAL ALPHA VARIABLE
ERR 60 ARRAY TOO SMALL
ERR 61 TRANSIENT DISK HARDWARE ERROR
ERR 62 FILE FULL
ERR 63 MISSING ALPHA ARRAY DESIGNATOR
ERR 64 SECTOR NOT ON DISK OR DISK NOT SCRATCHED
ERR 65 DISK HARDWARE MALFUNCTION
ERR 66 FORMAT KEY ENGAGED
ERR 67 DISK FORMAT ERROR
ERR 68 LRC ERROR
ERR 71 CANNOT FIND SECTOR
ERR 72 CYCLIC READ ERROR
ERR 73 ILLEGAL ALTERING OF A FILE
ERR 74 CATALOG END ERROR
ERR 75 COMMAND ONLY (NOT PROGRAMMABLE)
ERR 76 MISSING < OR > (PLOT STATEMENT)
ERR 77 STARTING SECTOR > ENDING SECTOR
ERR 78 FILE NOT SCRATCHED
ERR 79 FILE ALREADY CATALOGED
ERR 80 FILE NOT IN CATALOG
ERR 81 /XYY DEVICE SPECIFICATION ILLEGAL
ERR 82 NO END OF FILE
ERR 83 DISK HARDWARE ERROR
ERR 84 NOT ENOUGH MEMORY FOR MOVE OR COPY
ERR 85 READ AFTER WRITE ERROR
ERR 86 FILE NOT OPEN
ERR 87 COMMON VARIABLE REQUIRED
ERR 88 LIBRARY INDEX FULL
ERR 89 MATRIX NOT SQUARE
ERR 90 MATRIX OPERANDS NOT COMPATIBLE
ERR 91 ILLEGAL MATRIX OPERAND
ERR 92 ILLEGAL REDIMENSIONING OF ARRAY
ERR 93 SINGULAR MATRIX
ERR 94 MISSING ASTERISK
ERR 95 ILLEGAL MICROCOMMAND OR FIELD/ DELIMITER SPECIFICATION
ERR 96 MISSING ARG 3 BUFFER
ERR 97 VARIABLE OR ARRAY TOO SMALL
ERR 98 ILLEGAL ARRAY DELIMITERS
ERR=1 MISSING NUMERIC ARRAY NAME
ERR=2 ARRAY TOO LARGE
ERR=3 ILLEGAL DIMENSIONS

INDEX

Access Assembly 5, 6, 7
Access Time . . . . . 11, Appendices B and C
Address 3, 9, 10, 11, 38, 209
Amphenol Plug 177, 181
Argument 52, 209
Argument List 48, 51-53, 55, 57-58, 209

Backup Platters, Importance of 65, 147
Basic Rules of Syntax 104-105
Binary Search 148-152, 209

Cartridge Bowl 19
Catalog 37, 38, 39
Catalog Area 38, 39
Catalog Index 38, 39
Catalog Index, Sample Listing 45, 63
Catalog Procedures 37
Catalog, Initialization of 38-40
Chaining Programs from Disk 43-44
Command 103-104, 210
Control Information 96-97
COPY 146-147, 156
COPY Examples 146, 147, 157
Current Sector Address 76-79
Cylinder Concept (Models 2230 & 2260) . . 13

Data File 46, 47, 51
Data Record 48, 51, 52, 56, 58, 96
DATALOAD BA 144, 145, 160
DATALOAD BA Examples 145, 161
DATALOAD DA 142-144, 160
DATALOAD DA Examples 143, 144, 161
DATALOAD DC 55-58, 106
DATALOAD DC Examples 56, 57, 106
DATALOAD DC OPEN 54-55, 107
DATALOAD DC OPEN Examples . . . 55, 108
DATASAVE BA 144-145, 162
DATASAVE BA Examples 145, 163
DATASAVE DA 140-142, 164
DATASAVE DA Examples 141, 142, 165
DATASAVE DC 46, 48-51, 109
DATASAVE DC CLOSE 79-80, 111
DATASAVE DC CLOSE Examples . . 79, 80, 111
DATASAVE DC END 49, 50
DATASAVE DC END Examples 50
DATASAVE DC Examples . . . 49, 50, 51, 110
DATASAVE DC OPEN . 46, 47, 93, 94, 95, 112
DATASAVE DC OPEN Examples 47, 94, 95, 113
DBACKSPACE 58-60, 81-82, 114
DBACKSPACE Examples 60, 82, 114
Default Disk Address 34, 68, 69, 85
Default File Number 68, 69
Device Selection 69, 70

Device Table 68, 70-86
Device Type 82-84
Direct Access 3
Disk Cartridge (Models 2230 &
2260) . . . . See "Removable Disk Platter"
Disk Cartridge, Handling &
Storage of . . . . 23, Appendix E
Diskette (Model 2270) 5, 6, 26-30, 31
Diskette, Handling and Storage of 31
Disk Device Address . 34, 68, 69, 82, 83, 86, 87
Disk Latency Period 11
DSKIP 58-60, 81-82, 115
DSKIP Examples 59, 61, 82, 115
Dust Cover 20, 21, 23

Electronic Data Processing (EDP) 2
'END' Parameter 49, 142
End-of-File Trailer Record . 49, 51, 61, 142, 218
Ending Sector Address 213
Extension Cables (Disk Multiplexers) . . 174, 178

'F' Parameter 33, 34
Field 52
File Numbers 68, 69, 70, 74, 75, 76
Fixed Disk Platter 3, 17
Format Errors (Models 2230 & 2260) 18
Format Errors (Model 2270) 29
Formatting Procedure (Models 2230 & 2260) . 17
Formatting Procedure (Model 2270) 26

Hierarchy of Data 46
Hog Mode, Programmable 182-185
Hog Mode, Manual 182
Hog Mode, Programmagle 182-185
Hollerith, Herman 1

IF END THEN 61, 143, 144
Indirect Addressing of Disk Unit . . . 34, 69-70
Inter-Field Gaps 97-98
Iron Oxide, Used as a Recording Medium . . 7, 8

Keyed File Access Method (KFAM) 46

LIMITS 99-101, 116
LIMITS Examples 100, 101, 117
LIST DC 44-45, 118
LIST DC Examples 44, 118
LOAD DA Command 138, 167
LOAD DA Examples . . 138, 139, 140, 167, 168
LOAD DA Statement 138, 168
LOAD DC Command 41-42, 119
LOAD DC Command Examples 42, 119
LOAD DC Statement 42-44, 120
LOAD DC Statement Examples . . . 43, 44, 121
Loading & Formatting Diskettes 26-29

INDEX (Cont.)

Loading Procedure, Disk Cartridge (Models 2230, 2260) 21-23
Logical Record 46, 48, 51-53, 56, 57, 58, 59, 60, 141, 143, 214
'LS' Parameter 39

Machine Interlocks 24
Maintenance Check (Models 2230 & 2260) . . 24
MOVE 63-64, 122
MOVE Examples 64, 123
Multiple Disk Units, Addressing Scheme for 86-87
Multiplexer Operation 181

Overlaying Programs from Disk 44, 139

Program File 40, 89-90, 136
Protect Parameter 126, 171, 216

'R' Parameter 33, 34
Random Access 3
Random Access Device 3
Read-After-Write 110, 112, 125, 163, 165, 171, 216
Read/Write Head 5, 6
Removable Disk Platter . . . . . 3, 17, 18, 23, ... Appendices E, F

'S' Parameter 81, 82
SAVE DA 137, 170
SAVE DA Examples 137, 170
SAVE DC 40-41, 89-90, 125
SAVE DC Examples 40, 41, 90, 126
SCRATCH 62-63, 127
SCRATCH DISK 38-40, 129
SCRATCH DISK Examples 39, 130
Scratched File 62, 63, 64, 93-95
Scratched Files, Reusing 93-95
Sector 8, 216
Sector Address 8, 38
Sector Arrangement in a Track 11
Sector Control Bytes 8, 96, 216
Sector Layout 12
Sector Numbering 9, 10, 38

To help us to provide you with the best manuals possible, please make your comments and suggestions concerning this publication on the form below. Then detach, fold, tape closed and mail to us. All comments and suggestions become the property of Wang Laboratories, Inc. For a reply, be sure to include your name and address. Your cooperation is appreciated.

700-3159G

TITLE OF MANUAL: **2200 DISK REFERENCE MANUAL**

COMMENTS:

Fold

Fold

(Please tape. Postal regulations prohibit the use of staples.)

WANG

Fold

FIRST CLASS
PERMIT NO. 16
Tewksbury, Mass.

BUSINESS REPLY MAIL

NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

– POSTAGE WILL BE PAID BY –

WANG LABORATORIES, INC.
ONE INDUSTRIAL AVENUE
LOWELL, MASSACHUSETTS 01851

Attention: Technical Writing Department

Fold

Cut along dotted line.

Printed in U.S.A.
13-1019

WANG LABORATORIES (CANADA) LTD.
49 Valleybrook Drive
Don Mills, Ontario M3B 2S6
TELEPHONE (416) 449-2175
Telex: 069-66546

WANG EUROPE S.A./N.V.
(European Headquarters)
250, Avenue Louise
1050 Brussels, Belgium
TELEPHONE 02/640.37.80
Telex: 12430-12398

WANG EUROPE S.A./N.V.
(Belgian Sales)
350, Avenue Louise
1050 Brussels, Belgium
TELEPHONE 02/648.91.00
Telex: 62691

WANG DO BRASIL COMPUTADORES LTDA.
Praca Olavo Bilac No. 28
SL1801/1803
Rio de Janeiro, Centro, RJ, Brasil
TELEPHONE 232-7503, 232-7026

WANG COMPUTERS (SO. AFRICA) PTY. LTD.
Corner of Allen Rd. & Garden St.
Bordeaux, Transvaal
Republic of South Africa
TELEPHONE (011) 48-6123
Telex: 960-83297

WANG INTERNATIONAL TRADE, INC.
One Industrial Avenue
Lowell, Massachusetts 01851
TELEPHONE (617) 851-4111
Telex: 94-7421

WANG SKANDINAVISKA AB
Pyramidvaegen 9A
S-171 36 Solna, Sweden
TELEPHONE 08/27 27 98
Telex: 11498

WANG COMPUTER LTD.
Shindaiso Building No. 5
2-10-7 Dogenzaka Shibuya-Ku
Tokyo, Japan
TELEPHONE (03) 464-0644
Telex: 2424909 WCL TKO J

WANG NEDERLAND B.V.
Produktieweg 1
Ijsselstein, Netherlands
TELEPHONE (03408) 41.84
Telex: 47579

WANG PACIFIC LTD.
9th Floor, Lap Heng House
47-50, Gloucester Road
Hong Kong
TELEPHONE 5-274641
Telex: 74879 Wang HX

WANG INDUSTRIAL CO., LTD.
7, Tun Hwa South Road
Sun Start Tun Hwa Bldg.
Taipei, Taiwan, China
TELEPHONE 7522068, 7814181-3
Telex: 21713

WANG GESELLSCHAFT MBH
Murlingengasse 7
A-1120 Vienna, Austria
TELEPHONE 85.85.33
Telex: 74640 Wang a

WANG GESELLSCHAFT MBH
Wiedner Hauptstrasse 68
A-1040 Vienna, Austria
TELEPHONE 57.94.20
Telex: 76424 Wang a

WANG S.A./A.G.
Markusstrasse 20
Postfach 423
CH 8042 Zurich 6, Switzerland
TELEPHONE 41-1-60 50 20
Telex: 59151

WANG COMPUTER PTY. LTD.
55 Herbert Street
St. Leonards, 2065, Australia
TELEPHONE 439-3511
Telex: 24569

WANG ELECTRONICS LTD.
Argyle House, 3rd Floor
Joel Street
Northwood Hills
Middlesex, HA6 INS, England
TELEPHONE (09274) 28211
Telex: 923498

WANG FRANCE S.A.R.L.
Tour Gallieni, 1
78/80 Ave. Gallieni
93170 Bagnolet, France
TELEPHONE 33.1.3602211
Telex: 680958F

WANG LABORATORIES GmbH
Moselstrasse 4
6000 Frankfurt AM Main
Postfach 16826
West Germany
TELEPHONE (0611) 252061
Telex: 04-16246

WANG DE PANAMA (CPEC) S.A.
Apartado 6425
Calle 45E, No. 9N. Bella Vista
Panama 5, Panama
TELEPHONE 69-0855, 69-0857
Telex: 3282243

WANG COMPUTER LTD.
302 Great North Road
Grey Lynn, Auckland
New Zealand
TELEPHONE Auckland 762-219
Telex: CAPENG 2826

WANG COMPUTER PTE., LTD.
Suite 1801-1808, 18th Floor
Tunas Building, 114 Anson Road
Singapore 2, Republic of Singapore
TELEPHONE 2218044, 45, 46
Telex: RS 24160 WANGSIN

WANG COMPUTER SERVICES
One Industrial Avenue
Lowell, Massachusetts 01851
TELEPHONE (617) 851-4111
TWX 710-343-6769
Telex: 94-7421

DATA CENTER DIVISION
20 South Avenue
Burlington, Massachusetts 01803
TELEPHONE (617) 272-8550

WANG LABORATORIES, INC.
ONE INDUSTRIAL AVENUE, LOWELL, MASSACHUSETTS 01851, TEL. (617) 851-4111, TWX 710 343-6769, TELEX 94-7421

Printed in U.S.A.
700-3159G
8-77-2.5M
Price: see current list