

HONEYWELL

SOFTWARE

MOD 1 (MSR)

DATA MANAGEMENT SUBSYSTEM

GENERAL SYSTEM:

SERIES 200/OPERATING SYSTEM - MOD 1
(MASS STORAGE RESIDENT)

SUBJECT:

Programming and Operating Procedures for
the Data Management Subsystem of the Mod 1
(MSR) Operating System.

SPECIAL
INSTRUCTIONS:

This edition completely supersedes the manual
of the same name dated May 31, 1968. It is
one of a series of manuals describing the Mod 1
(MSR) Operating System. Refer to the Preface
for other related information. The portions of
this publication containing new and changed in-
formation are indicated on page iii.

INCLUDES UPDATE PAGES PUBLISHED AS ADDENDUM NO. 1 ON
AUGUST 29, 1969, AND ADDENDUM NO. 2 ON JANUARY 5, 1970.

DATE: December 3, 1968

FILE NO.: 123.6005.141C.5-618*

0644
10M
10170

Printed in U.S.A.

*Underscoring denotes Order Number.

PREFACE

This manual describes the Data Management Subsystem of the Series 200/Operating System - Mod 1 (Mass Storage Resident). Besides this manual, other pertinent publications include the following:

Mod 1 (MSR) Operating System Summary Description (Order No. 615);

Supervisor (Order No. 616);

Program Development Subsystem (Order No. 617);

Utility Routines (Order No. 619); and

Operating Procedures (Order No. 620).

The introductory bulletin cited above is prerequisite reading to this manual and the other manuals listed. In addition, a publications guide is provided in the introductory bulletin to aid the reader in his study of the system.

Section I describes the basic elements of the Data Management Subsystem. Section II gives the concepts relating to data and volume conventions and the rules of file organization. Section III describes the input/output routines associated with data files, and, finally, Section IV describes the file support routines. A series of appendices offers other topics of related interest to the reader.

Mod 1 (MSR) Data Management Subsystem is a coded system designed to extend the power of Series 200 in the area of data control. It is supported by comprehensive documentation and training; periodic program maintenance and, where feasible, improvements are furnished for the current version of the system, provided it is not modified by the user.

Copyright 1970
Honeywell Inc.
Wellesley Hills, Massachusetts 02181

#5-618

NEW AND CHANGED INFORMATION

This edition incorporates a number of additions and changes reflecting the added capability of the Mod 1 (MSR) Operating System to process volumes containing bad tracks. (This capability is fully described in the Utility Routines manual.) These changes occur for the most part in Sections III and IV, particularly in tables listing halt codes, console messages, and exits. Because of the addition of this capability, it has also been necessary to expand the entries in Tables A-1 and A-2, Volume Label and Volume Directory.

Besides the information noted above, the following changes have also been made:

Section III. Logical I/O C.

A supplementary list is typed out at the console giving information relevant to device condition messages and file I/O condition messages.

Section IV. File Support C.

The ability to load or unload a sequential file to a direct access file on mass storage.

File Support diagnostics for the 5040 halt have been expanded.

A number of minor corrections have been made to the text, and, wherever necessary, explanations have been clarified or expanded.

TABLE OF CONTENTS

		Page
Section I	Introduction	1-1
	Data Management Conventions	1-1
	Logical I/O C Program	1-4
	File Support C Program	1-5
	Job Control Language for Data Management Subsystem	1-6
	Equipment Requirements for Data Management Subsystem	1-7
	Required Equipment	1-7
	Additional Usable Equipment	1-7
Section II	Data Management Conventions	2-1
	Volume Conventions	2-1
	Formatting and Volume Preparation	2-3
	Bootstrap Records	2-3
	Volume Label	2-3
	Volume Directory	2-3
	Data Conventions	2-4
	Allocation Conventions	2-6
	Units of Allocation	2-6
	Track-Linking Records	2-8
	File Organization Conventions	2-8
	Sequential File Organization	2-8
	Allocation	2-9
	Data Structure	2-9
	Indexed Sequential File Organization	2-9
	Allocation	2-10
	File Structure	2-11
	Prime Data Area	2-11
	Index Areas	2-11
	Overflow Areas	2-12
	Directly Processing an Indexed Sequential File	2-13
	Data Item Status Character	2-20
	Direct Access File Organization	2-20
	Allocation	2-23
	File Organization	2-23
	Data Area	2-23
	Overflow Areas	2-23
	Direct Access Files and Keys	2-24
	Data Item Status Character	2-25
	Cumulative Loading of a Direct Access File	2-25
	Processing Conventions	2-26
	Sequential or Direct Processing	2-26
Volume Processing Functions	2-26	
File Processing Functions	2-27	
Backup Procedures	2-27	
Logical Backup	2-27	
Physical Backup	2-27	
Section III	Logical I/O C	3-1
	Mass Storage Input/Output Control Macro Routine (MIOC)	3-2

TABLE OF CONTENTS (cont)

	Page
Section III (cont)	
File Description Macro Routine (MCA).....	3-2
Communication Area Service Macro Routines (MLCA and MUCA).....	3-2
Action Macro Routines	3-2
Summary of Logical I/O C Macro Routines	3-2
File Processing Modes.....	3-4
Input/Output Processing Mode.....	3-4
Input-Only Processing Mode	3-4
Output-Only Processing Mode	3-4
Action Macro Processing Functions	3-4
Opening Files	3-6
Opening Sequential Files	3-6
Opening Partitioned Sequential Files	3-8
Opening an Indexed Sequential File	3-8
Opening Direct Access Files	3-9
Closing Files.....	3-10
Closing Sequential and Partitioned Sequential Files.....	3-10
Closing Indexed Sequential and Direct Access Files	3-10
Retrieving Items in Files	3-11
Retrieving Items in Sequential and Partitioned Sequential Files	3-11
Retrieving Items in Indexed Sequential Files.....	3-12
Retrieving Items in Direct Access Files	3-13
Replacing Items in Files.....	3-15
Replacing Items in Sequential and Partitioned Sequential Files	3-16
Replacing Items in Indexed Sequential Files	3-16
Replacing Items in Direct Access Files.....	3-16
Putting Items to Sequential and Partitioned Sequential Files..	3-17
Action Macro Calls (for Partitioned Sequential Files Only)...	3-17
Set Processing to Beginning of Specified Member (SETM). 3-17	3-17
End Processing of Current Member (ENDM).....	3-18
Alter Status of Member (MALTER).....	3-19
Release Complete File to Unused State (MSREL).....	3-19
Inserting Items in Files	3-20
Inserting Items in Indexed Sequential Files	3-20
Inserting Items in Direct Access Files.....	3-21
Deleting Items from Files	3-21
Seeking a Desired Cylinder.....	3-22
Setting Processing to a Specified Location	3-22
Program Organization	3-22
Language Elements for Logical I/O C.....	3-23
Input/Output Control Macro Routine (MIOC).....	3-26
MIOC Macro Call.....	3-26
Parameters of MIOC Macro Call.....	3-36
File Description Macro Routine (MCA)	3-38
MCA Macro Call	3-38.1
Communication Area Service Macro Routines (MLCA and MUCA).....	3-50

TABLE OF CONTENTS (cont)

	Page
Section III (cont)	
Mass Storage Load Communication Area Macro Call (MLCA).....	3-51
Mass Storage Unload Communication Area Macro Call (MUCA).....	3-51
Communication Area Field Designators.....	3-52
Action Macro Calls	3-54
Open (MSOPEN)	3-55
Close (MSCLOS)	3-56
Get (MSGET).....	3-56
Replace (MSREP)	3-58
Insert (MSINS)	3-58
Delete (MSDEL)	3-59
Put (MSPUT)	3-59
Set Member (SETM)	3-59
End Member (ENDM)	3-60
Alter Member (MALTER)	3-60
Release (MSREL)	3-61
Set Location (SETL)	3-62
Seek (MSEEK).....	3-62
Programmer's Preparation Information for Logical I/O C	3-64
Logical I/O C Memory Requirements.....	3-64
Program Organization	3-64
MIOC Segmentation	3-65
MIOC Restrictions.....	3-67
Physical I/O C Relationships with MIOC	3-69
Physical I/O C Relationships with MCA	3-69
Address Mode	3-69
Index Registers	3-69
Read/Write Channel Utilization	3-70
Direct Access Addressing.....	3-70
Item Key Specification	3-71
Direct Access.....	3-71
Indexed Sequential	3-72
Exits and Halts.....	3-72
Operating Procedures for Logical I/O C	3-77
Control Panel Operating Procedures	3-77
Console Typewriter Operating Procedures	3-82
Section IV	
File Support C	4-1
General Description of File Support C.....	4-1
Foreground/Background Processing of File Support C	4-1
Functions of File Support C.....	4-2
Allocate	4-2
Deallocate	4-2
Load	4-2
Unload.....	4-2
Map	4-2

TABLE OF CONTENTS (cont)

	Page
Section IV (cont)	
Map Description of a File.....	4-2
Map Expired Files	4-6
Map Unused Areas	4-6
Considerations.....	4-6
Number of Functions Performed	4-6
Block and Record Sizes Within 12K Memory	4-6
Job Control Language for File Support C.....	4-8
Execute Statement.....	4-8
Job Control for a Single Operation	4-8
Job Control for a Sequence of Operations.....	4-9
Allocate Function	4-9
Job Control Language for Allocate Function	4-10
Execute Statement	4-11
Function Statement.....	4-12
File Statement.....	4-12
File-Name Parameter	4-12
File-Organization Parameter.....	4-12
General Overflow Parameter.....	4-13
Item-Key Parameter	4-13
Password Parameter	4-13
File-Expiration Date Parameter	4-14
Protection-Status Parameter.....	4-14
Device-Address Parameter	4-14
Size Statement	4-15
Record-Length Parameter	4-15
Item-Length Parameter	4-15
Block-Size Parameter.....	4-16
Bucket-Size Parameter.....	4-16
Index-Size Parameter	4-16
Cylinder Overflow-Size Parameter.....	4-16
String-Size Parameter.....	4-17
Units Statement	4-17
Volume-Name Parameter.....	4-17
Master/Cylinder Index Parameter	4-18
Overflow Parameter	4-18
Data Unit of Allocation	4-19
FROM Parameter.....	4-19
TO Parameter.....	4-19
Member Statement.....	4-19
Member-Name Parameter	4-19
Member-Length Parameter	4-20
File Statement for the List File	4-20
Device-Address Parameters	4-20
Day Statement	4-21
Job Control Language Example for Allocate Function....	4-21
Summary of Job Control Statements for Allocate	
Function	4-24
Deallocate Function	4-28

TABLE OF CONTENTS (cont)

	Page
Section IV (cont)	
Job Control Language for Deallocate Function.....	4-28
Execute Statement.....	4-28
Function Statement.....	4-29
Volume Statement.....	4-29
Volume-Name Parameter.....	4-29
File Statement.....	4-29
File-Name Parameter.....	4-29
Expiration-Date Check Parameter.....	4-29
Password Parameter.....	4-30
Device-Address Parameter.....	4-30
Day Statement.....	4-31
Job Control Language Example for Deallocate Function..	4-31
Summary of Job Control Statements for Deallocate	
Function.....	4-33
Load and Unload Functions.....	4-33
Job Control Language for Load and Unload Functions....	4-35
Execute Statement.....	4-36
Function Statement.....	4-36
File Statements.....	4-36
In/Out Parameter.....	4-37
File-Name Parameter.....	4-37
Device-Type Parameter.....	4-37
Device-Address Parameter.....	4-38
Item-Length Parameter.....	4-39
Record-Length Parameter.....	4-39
Banner-Character Parameter.....	4-40
Parity Parameter.....	4-40
Padding-Character Parameter.....	4-41
Mode Parameter.....	4-41
Password Parameter.....	4-41
Bucket-Addressing Parameter.....	4-42
Protection-Status Parameter.....	4-42
Imbed Parameter.....	4-43
Release Parameter.....	4-43
Report-Number Parameter.....	4-43
Member Statements.....	4-44
Member-Name Parameter.....	4-44
Exits Statement.....	4-44
Program-Segment-Name Parameter.....	4-45
Low-Memory-Address Parameter.....	4-45
Job Control Language Examples for Load and Unload	
Functions.....	4-45
Summary of Job Control Statements for Load and Unload	
Functions.....	4-48
Map Function.....	4-51
Job Control Language for Map Function.....	4-51
Execute Statement.....	4-51
Function Statement.....	4-51

TABLE OF CONTENTS (cont)

	Page
Section IV (cont)	
Volume Statement	4-52
Volume-Name Parameter	4-52
Device-Address Parameter.....	4-52
File Statement	4-53
Day Statement	4-53
File Statement for the List File	4-53
Device-Type Parameter.....	4-54
Device-Address Parameter.....	4-54
Job Control Language Examples for Map Function	4-54
Summary of Job Control Statements for Map Function....	4-55
Programmer's Preparation Information for File Support C..	4-57
File Considerations	4-57
Direct Access Files	4-57
Unloading a Direct Access File	4-57
Loading a Direct Access File	4-57
Sequential Files.....	4-57
Partitioned Sequential Files.....	4-58
Unloading a Partitioned Sequential File	4-58
Unloading by File	4-58
Unloading Selected Members	4-58
Loading a Partitioned Sequential File	4-58
Loading by File.....	4-58
Loading Selected Members.....	4-58
Processing a Partitioned Sequential File by Mem- ber Names	4-58
Loading from Mass Storage to Mass Storage.....	4-59
Indexed Sequential Files	4-59
Allocating an Indexed Sequential File	4-59
Loading an Indexed Sequential File	4-59
Unloading an Indexed Sequential File.....	4-60
Mixed File Organizations	4-60
Loading or Unloading	4-60
Own-Coding Considerations	4-60
Structure of Own-Coding Routine.....	4-61
Own-Coding Considerations for Tape-Resident Operation.....	4-61
Own-Coding Communication with Load/Unload Function.....	4-62
Omitting Items from the Output File	4-62
Invalid Bucket Addresses.....	4-62
Insufficient Space.....	4-62
Entrance to General Overflow	4-63
Key Out of Sequence	4-63
Tape and Card File Considerations	4-63
1/2-Inch Tape Formats.....	4-63
Header Label	4-63
Data Records	4-65
Padding Items.....	4-66

TABLE OF CONTENTS (cont)

	Page
Section IV (cont)	
Trailer Label	4-66
Tape Marks	4-66
Card File Formats	4-66
Header Label.....	4-66
Data Items	4-67
Trailer Label	4-67
Unloading Mass Storage Files onto Printer	4-67
Operating Procedures for File Support C	4-68
Loading File Support C	4-68
Mod 1 (MSR) Operating System.....	4-68
Mod 1 (TR) Operating System	4-70
Protection of Mass Storage During Execution of File Support C	4-70
Protection During Allocate	4-70
Protection During Deallocate	4-70
Protection During Load/Unload	4-70
Protection During Map	4-70
Operator Control and Messages for File Support C.....	4-70
Operator Control with Control Panel.....	4-70
Peripheral Conditions	4-71
File Related Conditions	4-72
Job Control File Conditions	4-72
Conditions Specific to File Support C.....	4-77
Operator Control with Console Typewriter	4-83
Peripheral Conditions	4-83
File-Related Conditions	4-84
Job Control File Conditions	4-84
Typewriter Messages Specific to File Support C...	4-86
Failure During Allocation and Deallocation.....	4-93
Failure During Allocation.....	4-93
Failure During Deallocation.....	4-94
Appendix A	
Volume Label and Volume Directory.....	A-1
Appendix B	
Partitioning a Sequential File	B-1
Appendix C	
File Design and Allocation.....	C-1
File Design Criteria	C-1
Application Considerations	C-1
File Additions.....	C-1
File Inquiries	C-1
Random Versus Sequential Files	C-2
Random Plus Sequential Files	C-2
General File Design Considerations.....	C-2
Block Size.....	C-2
Assignment of Units of Allocation	C-3
Multivolume File Processing	C-3
Assignment of Files to be Processed Concurrently...	C-4
Sequential File Considerations	C-4

TABLE OF CONTENTS (cont)

	Page
Appendix C (cont)	
Allocation.....	C-4
Direct Access File Considerations	C-10
Bucket Size and Overflow	C-10
Allocation.....	C-12
Indexed Sequential File Considerations.....	C-17
Design Considerations.....	C-17
Item Sequence	C-17
Distribution and Volatility.....	C-17
Types of Overflow.....	C-17
Optimization	C-18
Optimizing Access Time	C-18
Optimizing Storage Capacity.....	C-20
Comprising Between Access Time and Storage Capacity.....	C-22
Allocation.....	C-24
Data Cylinders Required	C-24
Tracks Required for Master/Cylinder Index.....	C-25
Appendix D	
Physical I/O C	D-1
Use of Physical I/O C	D-1
Read Action	D-2
Write Action	D-2
Wait Action	D-2
Restore Action	D-2
Verify Action.....	D-2
Seek Action	D-2
Detailed Description of Physical I/O C Macro Routines	D-3
Control Macro Routine (MPIOC)	D-3
Communication Area Macro Routine (MPCA).....	D-3
Communication Area Service Macro Calls (MLCA and MUCA)	D-3
Action Macro Routines	D-4
Language Elements of Physical I/O C.....	D-4
Control Macro Call (MPIOC)	D-4
Parameters of MPIOC Macro Call	D-4
Communication Area Macro Call (MPCA).....	D-6
Parameters of the MPCA Macro Call	D-6
Communication Area Service Macro Calls (MLCA and MUCA)	D-9
Action Macro Calls	D-10
Read Action Macro Call.....	D-10
Write Action Macro Call	D-10
Wait Action Macro Call	D-11
Restore Action Macro Call	D-11
Verify Action Macro Call	D-11
Seek Action Macro Call	D-11
Programmer's Preparation Information for Physical I/O C..	D-12
Address Mode	D-12

TABLE OF CONTENTS (cont)

	Page
Appendix D (cont)	
Read/Write Channel Utilization.....	D-12
Special Considerations for Specifying Parameters	D-13
Use of Index Registers.....	D-13
Peripheral Address Assignment and RWC Configura- tion Considerations.....	D-13
Fixed Peripheral Address Assignment	D-13
Variable Peripheral Address Assignment.....	D-13
Considerations for MPIOC Parameter Specification.....	D-14
Suffix Character.....	D-14
Peripheral Address Assignment	D-14
Device Protection	D-14
Considerations for MPCA Parameter Specification.....	D-14
File Prefix	D-15
Suffix of Related MPIOC	D-15
Buffer Address (AAD).....	D-15
User's Uncorrectable Error Routine Entrance (EAD)..	D-15
Type of Read or Write (TRW).....	D-15
Control Unit Current Address and Status	D-16
Considerations for Action Macro Routines	D-17
Read Action Macro Routine.....	D-17
Write Action Macro Routine	D-17
Verify Action Macro Routine	D-18
Wait Action Macro Routine	D-18
Restore Action Macro Routine	D-18
LOKDEV Action Macro Routine	D-18
Handling Track Linking Records	D-18
User's Uncorrectable Error Routine	D-19
Error Type Indicator (ERI).....	D-19
Address Register Contents at Time of Error Exit (EDF).....	D-20
Re-execution of Correction Procedure.....	D-20
Bypass Error Condition.....	D-21
Issue New Action Macro Call	D-21
Operating Procedures for Physical I/O C	D-21
Appendix E	
Randomizing Techniques	E-1
Randomizing Addressing.....	E-1
Prime Number Division	E-1
Square Enfold and Extract	E-2
Radix Conversion	E-4
Nonnumeric Item Keys	E-5
Multifield Keys	E-6
Frequency Analysis	E-7
Appendix F	
Mass Storage File Protection.....	F-1
File Protection.....	F-1
Write Protection	F-1
Password Protection.....	F-2
Appendix G	
Terminal Files.....	G-1
Creation of Terminal Files	G-1

TABLE OF CONTENTS (cont)

		Page
Appendix G (cont)	Print-Image Files	G-2
	Card-Image Files	G-2

LIST OF ILLUSTRATIONS

Figure 2-1.	Disk Pack Cylinder Concept - Type 259 Disk Pack Drives.....	2-2
Figure 2-2.	Relationship Between Items and Records.....	2-5
Figure 2-3.	Relationship Between Items, Records, and Blocks	2-5
Figure 2-4.	Illustration of Units of Allocation - Type 261 or Type 262 Disk File.....	2-7
Figure 2-5.	Relationship Between Items of the Master and Cylinder Index	2-14
Figure 2-6.	Relationship Between String Index Items and the Data Area of a Cylinder	2-15
Figure 2-7.	Insertion of Items into a String	2-16
Figure 2-8.	Deletion of an Item from a String	2-21
Figure 2-9.	Using the Item Position of a Deleted Item	2-22
Figure 2-10.	Relationship Between Items, Records, Blocks, and Buckets....	2-24
Figure 3-1.	Omission of Single Parameter from Macro Call.....	3-25
Figure 3-2.	Omission of Consecutive Parameters from Macro Call	3-25
Figure 3-3.	Program Segment Loading.....	3-68
Figure 4-1.	Format of File Support C Execute Statement	4-8
Figure 4-2.	Job Control Statements for Allocation of Files.....	4-10
Figure 4-3.	Job Control Statements for Deallocate Function.....	4-28
Figure 4-4.	Job Control Statements for Loading and Unloading Files	4-35
Figure 4-5.	Listing of Sample Unload-to-Printer Function	4-69
Figure B-1.	Sequential File Using Partitioning Option.....	B-4
Figure D-1.	MPCA Control Unit Current Address and Status Field	D-16

LIST OF TABLES

Table 3-1.	Summary of Logical I/O C Macro Routines.....	3-3
Table 3-2.	Action Macro Calls for Each File Type in Each Processing Mode	3-6
Table 3-3.	Summary of MSGET Macro Functions for Direct Access Files	3-15
Table 3-4.	Parameters of MIOC Macro Call.....	3-27
Table 3-5.	Summary of MIOC Parameter Values.....	3-36
Table 3-6.	Parameters of MCA Macro Call	3-39
Table 3-7.	Summary of MCA Parameter Values.....	3-49
Table 3-8.	Mnemonic Designators for Communication Area Fields.....	3-52
Table 3-9.	Summary of Action Macro Call Coding.....	3-63
Table 3-10.	MIOC Segmentation.....	3-66
Table 3-11.	Exit and Return Codes for Volume Directory Exits.....	3-73
Table 3-12.	Exit and Return Codes for Member Index Exits	3-75
Table 3-13.	Exit and Return Codes for Data Exits.....	3-75

LIST OF TABLES (cont)

	Page
Table 3-14.	Exit and Return Codes for Device Exits 3-76
Table 3-15.	Halt Codes for Logical I/O C 3-78
Table 3-16.	Console Typewriter Pause Codes and Messages for Logical I/O C 3-83
Table 4-1.	Available Memory per I/O Media for 12K Configuration 4-7
Table 4-2.	Summary of Job Control Statements for Allocation Function ... 4-25
Table 4-3.	Summary of Job Control Statements for Deallocate Function ... 4-34
Table 4-4.	Minimum Device Requirements for Mass Storage File Organizations 4-39
Table 4-5.	Summary of Job Control Statements for Loan/Unload Functions 4-49
Table 4-6.	Summary of Job Control Statements for Map Function 4-56
Table 4-7.	Conditions Related to Non-Mass Storage File 4-71
Table 4-8.	Job Control Halt Codes 4-73
Table 4-9.	File Support Diagnostics for 5040 Halt 4-74
Table 4-10.	File Support C Halts 4-78
Table 4-11.	Typewriter Messages for Conditions Related to Non-Mass Storage Files 4-83
Table 4-12.	Job Control File Console Typewriter Messages 4-85
Table 4-13.	Typewriter Messages Specific to File Support C 4-86
Table A-1.	Volume Label A-2
Table A-2.	Volume Directory A-3
Table B-1.	Fields of Member Index Items B-2
Table B-2.	Fields of First Item in Member Index B-2
Table B-3.	Fields of Last Item in Member Index B-3
Table C-1.	Optimum Record Size - Types 155, 258, 259, 273, 259A, and 259B Disk Pack Drives C-5
Table C-2.	Optimum Record Size - Type 261 or Type 262 Disk Files C-7
Table C-3.	Overflow Probabilities C-11
Table C-4.	Cylinder Overflow as Percentage of Data Area C-13
Table C-5.	Example-Optimization for an Indexed Sequential File C-23
Table C-6.	Example-Summary of Optimum Points C-23
Table D-1.	Parameters of MPIOC Macro Call D-4
Table D-2.	Parameters of MPCA Macro Call D-6.1
Table D-3.	Mnemonic Designators for MLCA and MUCA D-9
Table D-4.	Corrective Action for User's Error Routine D-21
Table E-1.	Prime Numbers E-3

SECTION I INTRODUCTION

This manual describes the Data Management Subsystem of the Series/200 Operating System - Mod 1 (Mass Storage Resident). Data management, as described herein, involves: conventions established by Honeywell for the organization of data, a method of processing data in files stored on mass storage devices, and a means of transferring data files to or from mass storage devices. The established conventions include those for preparing a volume for use, data organization, reserving space on a volume to store a file, and file organization.

To process data in files stored on mass storage devices, a means of accessing the entire file must be available. The input/output control program provided by Honeywell, called Logical I/O C, supplies the programmer with this capability. To load and unload data files using mass storage devices, a method of reserving (or allocating) space for the files must be available. The File Support C program, included in the operating system, enables the programmer to perform these functions and, in addition, enables him to delete (or deallocate) files from mass storage. To simplify the programmer's task when he performs these functions, a job control language which is common to the entire operating system is used. The programs included in the Data Management Subsystem are fully compatible with the bad track handling procedures described in Appendix B of the manual, Mod 1 (MSR) Utility Routines.

DATA MANAGEMENT CONVENTIONS

The fundamental concept of the Data Management Subsystem is that all data to be processed by the operating system is organized according to one set of conventions. The conventions established for this operating system involve the mass storage volume, data organization, allocation of space, and file organization.

A volume is a unit of peripheral storage, in this case, a disk pack. Volumes are composed physically of disk surfaces and logically of cylinders as described in Section II of this manual. The volume conventions established to ready a volume for use in the system involve preparing the volume, the volume label, and the volume directory. All mass storage volumes used in this operating system must be prepared before data is written on them.

Volume preparation is performed by the Volume Preparation C program described in the manual Mod 1 (MSR) Utility Routines (Order Number 619). The File Support C allocate function (described later) reserves space on a mass storage volume so that it is capable of accepting the data file for storage. To ensure that the correct volume is being used, each volume has a label. The volume label contains the name of the volume and a code indicating the type of disk pack being

used. This information is written into the volume label by the Volume Preparation C program. The volume directory, also established by the Volume Preparation C program, contains the names of all files stored on the volume, a description of each file, and information about the size and location of each file.

The data organization conventions established for this operating system involve defining the units of data and distinguishing between logical and physical units. The units of data are items, records, blocks, and files. An item is a logical unit of data; it is the basic unit of information for a data processing program. A record is a physical unit of data written between two gaps (interrecord gaps) on a track. A block is a group of one or more records that is transferred to and from mass storage as a unit. A block contains one or more items. A file is a collection of logically related items; it is the largest unit of data that can be stored and retrieved by the operating system.

The conventions established for allocating space on a volume to store a file involve the concept of "unit of allocation." The unit of allocation is the basic element in designating the volume area that is assigned to store a file; it specifies the beginning cylinder and track numbers and the ending cylinder and track numbers between which the unit of allocation is stored.

The file organization conventions established for the Mod 1 (MSR) Operating System involve defining the types of file organization that can be used. At present, the operating system accepts three basic types of file organization: sequential, indexed sequential, and direct access. The organization of a file predetermines the methods that can be used to process it.

The sequential file is organized so that items are accessed sequentially, i. e., the items are retrieved in the same sequence in which they were written. This method of accessing items corresponds to that used in magnetic tape processing. Thus, any function operating upon a sequential file can process only one volume at any given time. Regardless of the number of devices assigned to a sequential file, the second volume can be processed only after processing of the first volume is completed. A single exception to this procedure occurs when a sort is performed, and the item address is present. In this case, the input (sequential) file must be on-line, since the sort must reaccess each item in the input file in a random manner.

An indexed sequential file is organized so that each item can be processed directly, sequentially, or in combination both directly and sequentially.

When an indexed sequential file is loaded by File Support C or processed directly, all volumes of the file must be on-line at all times. An item key is provided to the input/output rou-

tines. The item containing this key is located through the indexes. Items must be identified by a contiguous set of characters within an item. The item identifiers are called "item keys" or simply "keys." A key can be any number of characters long and can appear anywhere within an item. However, each item key in an indexed sequential file must be the same length and appear in the same position within each item of the file.

The File Support C load function builds three indexes for the file. The indexes built are a master index, a cylinder index, and a string index. Overflow areas are initialized at load time. The indexes are subsequently used in accessing an item in the file. To insert a new item, the system simply locates the two items in the file which immediately precede and follow the new item (based on the value of the new item's key), and the new item is placed between them. Inserting items can cause items to overflow the data area of the file; in such a case, the overflowing items are stored in the overflow areas, and the string index entries are adjusted to indicate this.

The logical sequence of items in an indexed sequential file does not necessarily correspond to the physical placement of the items. When an indexed sequential file is processed sequentially, items are retrieved in logical sequence from the beginning. The volume(s) containing the master/cylinder index and general overflow area must be on-line, whereas the volumes containing only data are processed one at a time. However, File Support C requires that all volumes be on-line when the file is loaded.

In some applications, an indexed sequential file can be processed both directly and sequentially. One example of such a combination is when processing is to begin at some point other than the beginning of the file, but is to be sequential thenceforth. All volumes must be on-line at all times.

A direct access file is organized to provide fast access to items. The file is normally processed directly; items are retrieved as needed without reading intervening items on the file. All volumes must be on-line.

A direct access file can also be processed sequentially if it is necessary to process every item in the file, for example, if data is to be unloaded from mass storage onto tape. Volumes are processed one at a time, and items are processed according to their physical placement on the file.

The structure of a direct access file is in terms of buckets. A bucket is an area (defined by the programmer) that contains one or more items. When a bucket contains more than one

item, there need not be a relationship between the items. However, use of a randomizing routine may produce identical addresses for items which may, in turn, be used to form a bucket. A bucket and a block (defined earlier) can be the same size, or a bucket can contain more than one block.

A direct access file is divided into a data area, a cylinder overflow area, and a general overflow area. (The overflow areas are optional.) Because items can be inserted into a direct access file, any bucket in the file can become completely filled with items. To accommodate additional items, an area can be reserved at the end of each cylinder. This area is the cylinder overflow area and is used to store the overflow items from the data area. The possibility exists that items can overflow the cylinder overflow area. In this case, these items are stored in a general overflow area which, if present, is the last cylinder in each unit of allocation.

Because items can be inserted and deleted in a direct access or an indexed sequential file, a method of determining the status of an item position is required. An item position can be used, unused, or it can contain an item that has been deleted. When an item position is used, it is called an active item position; when unused, it is called inactive. The last character of each item in a direct access or indexed sequential file serves as the status character, indicating whether the item position is active, inactive, or contains a deleted item.

LOGICAL I/O C PROGRAM

The Logical I/O C program provides a method of accessing files and data stored on mass storage devices. For this operating system, Logical I/O C consists of a set of macro routines that the assembly-language programmer calls for in his source-language program. The called routines are specialized (i. e., tailored to a specific need) and assembled in his machine-executable program by Mass Storage Easycoder Assembler C, described in the manual Operating System - Mod 1 (Mass Storage Resident) — Program Development Subsystem (Order Number 617). Logical I/O C can be specialized and assembled using Library Processor D and Easycoder Assembler D of the Mod 1 (TR) Operating System.

Logical I/O C consists of four types of macro routines used for: control, file description, communication area service, and action. The control macro routine is called the mass storage input/output control (MIOC) macro routine; it provides general control over the entire input/output process. The file description macro routine is called the mass storage communication area (MCA) macro routine; it sets up a communication area for a file in which all values necessary to describe the file and the processing options are stored. Pertinent portions of the communication area are available to the programmer and can be altered by him through the use of

the communication area service macro routines. Unloading any field of the communication area available to the programmer is done through the mass storage unload communication area (MUCA) macro routine. Altering any available field of the communication area is done through the mass storage load communication area (MLCA) macro routine. The action macro routines that the programmer includes in the main line of his coding cause the various functions of Logical I/O C to be performed.

The file processing functions that Logical I/O C performs are summarized below.

1. Open or close a file, verifying and updating the file's directory information.
2. Get, put, or replace individual items in a file, blocking and unblocking as necessary.
3. Insert an item into or delete an item from a direct access or an indexed sequential file.
4. Directly access items in a direct access or an indexed sequential file.
5. Establish linkage to the program Physical I/O C. Physical I/O C reads and writes data, detects errors, and (if possible) corrects errors.
6. Provide exits to user-written label and error routines.
7. Control the overlapping of central processor and input/output operations.
8. Terminate sequential processing on one volume and switch to the next volume.
9. Allow other processing or peripheral data transfers to occur during cylinder-to-cylinder access time (seek time) of a disk device.

FILE SUPPORT C PROGRAM

The File Support C program performs frequently desired functions on mass storage files. These functions are as follows.

1. Allocation of files to be stored on mass storage volumes.
2. Deallocation of files stored on mass storage volumes.
3. Loading files onto mass storage volumes.
4. Unloading files from mass storage volumes.
5. Listing the contents of the volume directory or the unassigned tracks of a volume.

The allocate function is used by the programmer to assign areas of one or more volumes for storing a file and to automatically update each volume directory accordingly. This function also initializes a newly allocated file automatically. The deallocate function removes all entries for a file from the directory of each volume on which the file exists. This makes all areas used by this file available for future allocation. The load function is used by the programmer to load a mass storage file from cards, tape, or another mass storage file. The unload function is used to unload a mass storage file onto cards, tape, printer, or another mass storage file. The map function is used by the programmer to obtain a printed listing based on the contents of the volume directory.

All File Support C routines are automatically specialized at execution time. This specialization is based on parameters supplied by the programmer in the job control statements. Therefore, it is not necessary for the programmer to perform an assembly operation to specialize these routines.

JOB CONTROL LANGUAGE FOR DATA MANAGEMENT SUBSYSTEM

The job control language used in this manual is shown with certain typographic conventions that aid in describing the language precisely. The typographic conventions are as follows.

1. Use of upper case. Letters or words (that are written in upper case), numbers, and almost all punctuation marks represent literal information that the programmer must write exactly as shown in an accompanying illustration or example.
2. Use of lower case. Letters or words (including hyphenated words) that are written without capital letters are generic expressions that represent information which the programmer must supply. The generic expressions name or describe a quantity; the programmer must supply the actual value.
3. Use of braces. Braces ({ }) enclose information from which the programmer must make a choice. In other words, enclosed in braces is a list of expressions and the programmer must use one of these expressions. For example, the parameter used to specify the type of file organization for which space is being reserved by the allocate function is shown below.

$\text{ORG} = \left(\begin{array}{c} \text{SEQ} \\ \text{PART} \\ \text{DIR} \\ \text{IND} \end{array} \right),$

The programmer chooses the file organization appropriate for the file being allocated; for example, sequential organization is indicated as follows.

ORG=SEQ,

4. Use of ellipses. An ellipsis (. . .) indicates that an expression can be repeated. Each repetition has the same format as the first expression. For example, specification of FROM and TO parameters for the File Support C allocate function is indicated as follows.

FROM=(c, t), TO=(c, t), . . .

The ellipsis indicates that more than one pair of FROM and TO parameters can be written. For example, the programmer might write the following for two units of allocation.

FROM=(1, 0), TO=(9, 9), FROM=(75, 0), TO=(99, 9),

EQUIPMENT REQUIREMENTS FOR DATA MANAGEMENT SUBSYSTEM

Any peripheral devices used by the Data Management Subsystem, either for system files or data files, may be assigned to peripheral addresses in either the first or second I/O sector (with the exception of the job control file which cannot be reassigned). The following paragraphs describe the required equipment and the additional equipment usable in the Mod 1 (MSR) Operating System.

Required Equipment

A Series 200 central processor

Advanced Programming Instructions

One card reader

One printer

12,288 characters of main memory

One mass storage device and associated control unit, selected from any of the following combinations:

Control Unit	Device Type
157C	155
257C	155
260	258, 259, 261, 262, 273
257	258, 259, 273
257-1	258, 259, 273
257A	259A
257B	259B

Additional Usable Equipment

One card punch

One or more Type 204B Magnetic Tape Unit(s)

One Type 220 Console with additional 4,096 characters of memory

Up to 32,768 characters of main memory

Second I/O sector

As many as 8 mass storage devices on each available control unit.

)

0

1

)

2

3

)

SECTION II

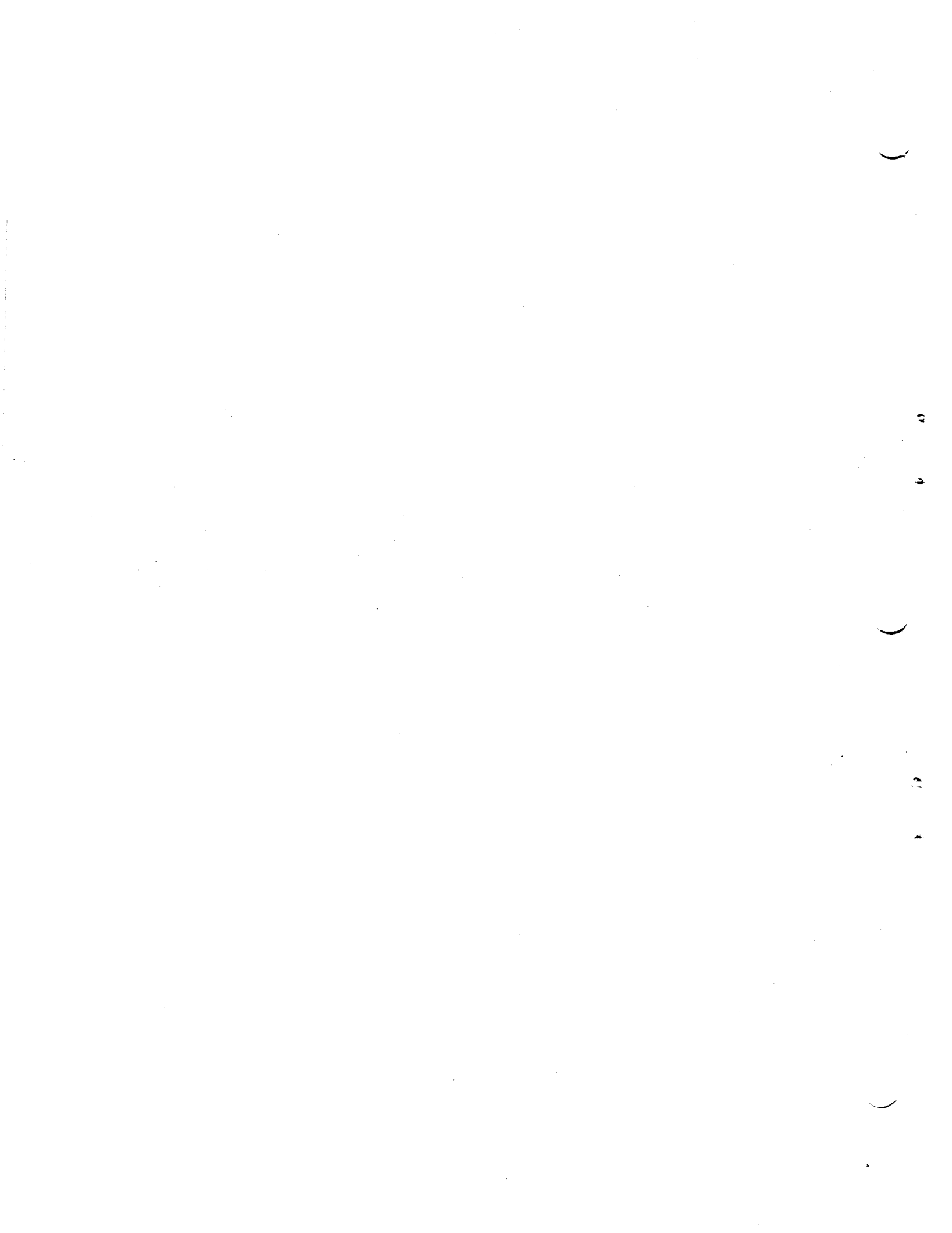
DATA MANAGEMENT CONVENTIONS

The basic concept in data management is that all data in a system is organized according to one set of conventions. The set of conventions established for the Mod 1 (MSR) Operating System involves volume, data, and allocation conventions, as well as conventions for organizing and processing files. All files in the system, including those supplied by Honeywell for the use of system programs and those supplied by the user, obey the data management conventions. These files are managed (created and accessed) by a common set of Honeywell-supplied programs (i.e., File Support C, Logical I/O C, and Physical I/O C programs). The data management conventions that have been established for the Mod 1 (MSR) Operating System provide full user control over data, efficiency in processing data, and ease in programming and operating.

VOLUME CONVENTIONS

A volume is a unit of peripheral storage. In the Mod 1 (MSR) Operating System, a disk pack is defined as a volume. Mass storage volumes are composed physically of disk surfaces and logically of cylinders, as shown by the illustration of a Type 259 Disk Pack Drive in Figure 2-1. Each disk surface has a series of concentric recording bands called tracks. With the Type 155 Disk Pack Drive data is recorded on both sides of one disk. Thus, there are two tracks (00 and 01) per cylinder. With the Types 258, 259, 259A, and 259B Disk Pack Drives data is recorded on both sides of all disks except for the top and bottom disks. Data is recorded on the underside of the top disk and on the top side of the bottom disk. There are six disks, and, thus, there are ten tracks (00 through 09) in each cylinder. The Type 273 Disk Pack Drive contains 11 disks, and, thus, there are 20 tracks (00 through 19) in each cylinder. The Types 261 and 262 Disk Files contain 128 tracks per cylinder. (For a detailed description of Types 155, 258, 259, 259A, and 259B Disk Pack Drives and the Types 261 and 262 Disk Files, see Direct-Access Devices and Controls, Order No. 514.)

Cylinders are composed of tracks arranged vertically above and below each other on different, but adjacent, disk surfaces. Cylinders are numbered consecutively from the outermost (cylinder 000) to the innermost (cylinder 103 or 202 or 127). The number of tracks per cylinder and the number of cylinders per drive for the various mass storage devices are shown below.



SECTION II. DATA MANAGEMENT CONVENTIONS

Device Type	Tracks per Cylinder	Cylinders per Drive
155	2	203
258	10	104
259	10	203
273	20	203
259A	10	203
259B	10	203
261	128	128
262	128	128

NOTES: 1. The last four cylinders of Type 258 Disk Pack Drive; the last three cylinders of Types 155, 259, 273, 259A, and 259B Disk Pack Drives; and the last three cylinders of the Types 261 and 262 Disk Files are all reserved for system use.
 2. Type 262 Disk Files are treated in all respects as two Type 261 Disk Files.

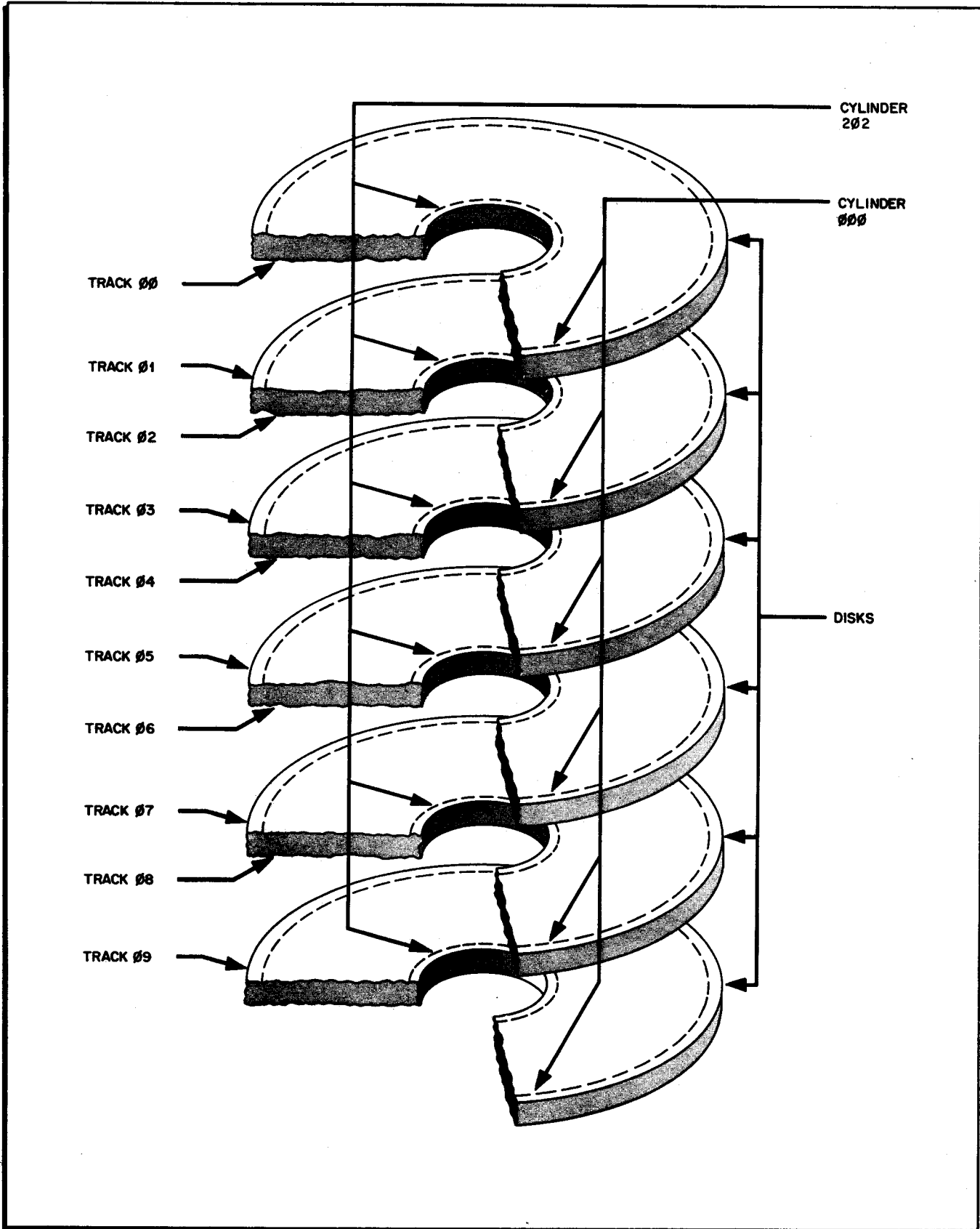


Figure 2-1. Disk Pack Cylinder Concept-Type 259 Disk Pack Drive

The volume conventions established for the Mod 1 (MSR) Operating System are concerned with volume preparation, bootstrap records, the volume label, and the volume directory.

Formatting and Volume Preparation

All mass storage volumes in the operating system must be formatted before data can be written on them. Formatting establishes the size and record number of each record on a track. Initially, each volume is formatted throughout all tracks on the volume by Volume Preparation C, a program which is supplied by Honeywell as a utility routine for the operating system and which must be executed whenever a volume is introduced into the system. Volume Preparation C formats each track to ensure the quality of the recording surfaces and creates the volume label and volume directory.

All records other than the last record on a track must be the same size. The last record on each track must be a track-linking record. The records are numbered in sequence, starting with record number 0000_g.

All system files, unless specifically noted, are formatted with a Mod 1 (MSR) standard 250-character record size. Examples of such are bootstrap records, volume label, volume directory, executable program file, library file, program development work files, and sort work files. Portions of a volume that are assigned to files are automatically reformatted whenever the files are allocated.

Bootstrap Records

The Bootstrap Generator C program, supplied by Honeywell as a utility routine for the operating system, is used to create the bootstrap routine on mass storage volumes any time after a volume has been prepared for use by Volume Preparation C. The bootstrap routine is used for loading the system from mass storage into memory. The operating system writes the bootstrap routine on the first track of the volume. Therefore, the first track of each volume (cylinder 000, track 00) is not available to the user.

Volume Label

The unique identification for the volume is contained in the volume label. The volume label consists of one 250-character record. It is written as the first record (record 0) on the second track (cylinder 000, track 01) of each volume by Volume Preparation C. The contents of the volume label are listed in Appendix A of this manual.

Volume Directory

The volume directory is written by Volume Preparation C and maintained by File Support C.

With the exception of the Type 155 Disk Pack Drive it begins on the third track (cylinder 000, track 02) and can occupy from three to seven tracks of each volume. For the Type 155 Disk Pack Drive it begins on cylinder 000, track 01, record 1 and occupies all records through cylinder 001, track 00 and 01. The volume directory is a catalog of all files stored in whole or in part on the volume. Three sequential files make up the volume directory. The first file in the volume directory contains the names of all the files stored on the volume. This file is called *VOLNAMES* and, in addition to the names of the files, it contains the addresses of the file description and the file allocation for each file on the volume. The second file in the volume directory contains the description of each file stored on the volume. This file is called *VOLDESCR* and is made up of three distinct areas: an area for general information, an area for labeling information, and an area for file organization information. For each file, the general information area of *VOLDESCR* contains information such as the type of file organization, item and record size, and the blocking factor. The labeling information area of *VOLDESCR* for each file contains information such as the creation date and number of the file, and the modification date and number for the file. The area of *VOLDESCR* that contains information on the file's organization has entries such as the length of the index and the number blocks in the file for sequential files and the key length and position for direct access files. The third file in the volume directory is called *VOLALLOC* and describes the allocation of each file stored on the volume. If the file is continued on another volume, *VOLALLOC* identifies that volume. The complete contents of the volume directory are listed in Appendix A of this manual.

DATA CONVENTIONS

The data conventions established for the operating system involve defining the units of data and distinguishing between logical and physical units of data. Also included in the following discussion are the relationships between the units of data. The units of data are: items, records, blocks, and files.

An item is a logical unit of data. It is the basic unit of information for a data processing program and is the smallest logical unit of data operated on by the input/output control programs. For example, an item can be a single policy in an insurance policy file or an individual's account in a master payroll file. The maximum item size, regardless of the type of file organization, is 4,095 characters.

A record is a physical unit of data. It is that data written between two interrecord gaps on a track. All data records on a track, just like all data records in a file, must be the same physical size. A single record is the smallest physical unit of data that is operated on by the input/output control programs. The record size is the prime determinant of the number of data characters that can fit on a track.

A record can contain one item, more than one item, or a portion of an item. The total number of characters in a given item need not be contained in a single record; i. e., the item's characters may be interrupted by the interrecord gap (IRG). For example, if the record size is 250 characters and the item size is 100 characters, two records contain five items. This relationship is illustrated in Figure 2-2. Procedures for determining the optimum record size can be found in Appendix C.

RECORD 0 250 CH.			I R G	RECORD 1 250 CH.		
ITEM 0 100 CH.	ITEM 1 100 CH.	ITEM 2 50 CH.		ITEM 2 50 CH.	ITEM 3 100 CH.	ITEM 4 100 CH.

Figure 2-2. Relationship between Items and Records

A block is a physical unit of data. Blocks are defined as a whole number of records transferred either to or from main memory by a single data transfer operation. A block can contain one or more physical records, and block size is determined by the user. A block may be contained entirely on one track or may begin on one track and end on another; however, it cannot begin on one cylinder and end on another. Since it is the contents of a block (i. e., records) that are transferred to or from memory, a buffer should not be smaller than a block. A block also contains a whole number of items. The number of items in a block is known as the item-blocking factor and may range from 1 through 4,095. The relationship between blocks, records, and items is shown in Figure 2-3. Efficient use of mass storage capacity is dependent on optimal relationship of records to track and blocks to cylinder.

BLOCK 0 900 CH.						I R G	BLOCK 1 900 CH.					
RECORD 0 450 CH.			RECORD 1 450 CH.				RECORD 2 450 CH.			RECORD 3 450 CH.		
ITEM 0 180 CH.	ITEM 1 180 CH.	ITEM 2 90 CH.	ITEM 2 90 CH.	ITEM 3 180 CH.	ITEM 4 180 CH.	ITEM 5 180 CH.	ITEM 6 180 CH.	ITEM 7 90 CH.	ITEM 7 90 CH.	ITEM 8 180 CH.	ITEM 9 180 CH.	

Figure 2-3. Relationship between Items, Records, and Blocks

A file is a logical unit of data comprising a collection of logically related items. A file is the largest unit of data that can be stored and retrieved by the operating system. A multivolume file may exist on volumes assigned to both the first and second I/O sectors; all volumes must be of the same device class, as shown in the following.

SECTION II. DATA MANAGEMENT CONVENTIONS

Device Class	Device Type
A	Types 258, 259, 273, 259A, or 259B Disk Pack Drives
B	Type 155 Disk Pack Drive
C	Type 261 or Type 262 Disk File

NOTE: One control unit can control both class A, and Class C devices, but one file cannot be allocated to different class devices because they have different capacity parameters (characters per track, track per cylinder). Also, one file cannot be allocated across devices of different transfer rates.

The Type 259A Disk Pack Drive has a lower data transfer rate than the other class A device, but otherwise it is treated similarly by the operating system software.

ALLOCATION CONVENTIONS

A file can be stored on one volume, or a file can span up to 8 volumes. In the latter case, portions of the total file are stored on individual volumes. When the entire file is stored on one volume, it is a single-volume file. When the entire file is stored on more than one volume, it is a multivolume file and can be assigned to devices connected to the first, second, or both I/O sectors. The portion of a file stored on one volume (in either single-volume or multivolume files) is called a "file volume."

Units of Allocation

A "unit of allocation" is the basic element in the designation of an area of the volume assigned to the file. The unit of allocation is of the form C1T1C2T2, where C is a cylinder address and T is a track address. C1 is the first cylinder of the unit of allocation, and C2 is the last cylinder of the unit. T1 is the first track used on all cylinders between C1 and C2, and T2 is the last track used on these cylinders. The operating system maintains the status of the units of allocation on each volume through the file allocation index *VOLALLOC*, (see page 2-4) and a new file cannot be completely allocated if its units of allocation for any volume conflict with the units of allocation for any file currently allocated on that volume. (See "Failure During Allocation and Deallocation," in Section IV.)

NOTE: In general, it is recommended that, for Types 155, 258, 259 or 273 Disk Pack Drives, the units of allocation of a file be made a full cylinder wide (two, ten, or twenty tracks). The fewer the number of tracks per cylinder, the greater the number of accesses to a new cylinder required. An unnecessary increase in the number of such accesses can increase the time required to process a file. However, a file can be any track width as long as all of its data units of allocation have the same width. If files are small or are in communication with other mass storage files, cylinder can be shared.

When a file has more than one unit of allocation, a unit of allocation for that file cannot start on the cylinder on which the immediately preceding unit of allocation for that file ended. See Figure 2-4 for examples.

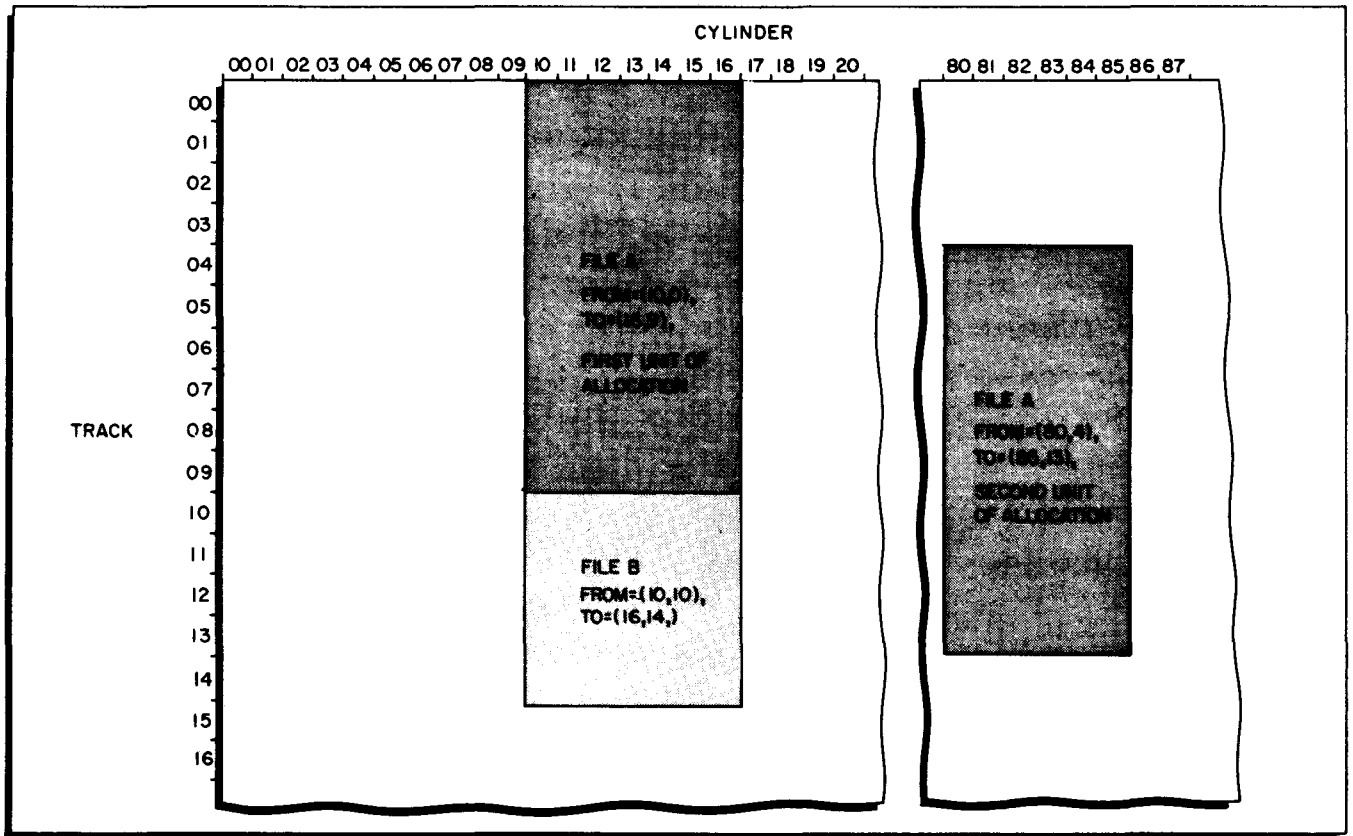


Figure 2-4. Illustration of Units of Allocation-
Type 261 or Type 262 Disk File

A multivolume file can use as many as eight volumes. However, all volumes of a multivolume file must be of the same device class.

A single-volume file can have up to six units of allocation. Any given file volume used to store a multivolume file also can have up to six units of allocation, but the maximum number of units of allocation for an entire multivolume file is sixteen.

All units of allocation for a multivolume file must be assigned consecutively by named volume. That is, units of allocation may not be assigned on volume A, volume B, volume C, and then on volume A again. All of volume A's units of allocation must be assigned before any units of allocation for volume B, etc. are assigned. In this example, volume A has the lowest volume sequence number, volume B the next higher, etc.

If the unit of allocation ($C_1T_1C_2T_2$) for a file on a Type 261 or Type 262 Disk File were 10-00-16-09, it could be shown graphically as in Figure 2-4 (File A, first unit). In this illustration, the mass storage area is shown as if it had been rolled out into a plane. Portions of two files are shown to illustrate the uniform track-width requirement for the data units.

Track-Linking Records

The last record on each track must be a track-linking record. Within each unit of allocation, a track-linking record points to record 0 of the next successive track. The last track-linking record in the unit points to record 0 of the first track of the next unit of allocation on the volume. If no succeeding unit of allocation exists on the volume, the track-linking record points to record 0 of the physically successive track. For direct access files, the last track-linking record in a file volume contains the address of record 0 of the first allocated track of the last cylinder of the file volume.

FILE ORGANIZATION CONVENTIONS

A file is organized according to the method used to access its items. This, in turn, pre-determines how the file can be processed. That is, if a file is organized one way, its items can be accessed directly (going directly from one item to another without accessing all the intervening items), and new items can be inserted into the file. If, however, a file is organized another way, items must be accessed serially, and new items cannot be inserted into the file. Thus, the method of file organization controls how the file can be processed.

Sequential File Organization

A sequential file is organized to permit accessing of each item in physical sequence. Thus, items are retrieved in the same sequence that they were written. This type of file organization is intended primarily for a file in which most of the items are processed each time the file is used. Note that to process this type of file, the transaction input must be sorted to conform to the sequence of items in the file. This is because each time this type of file is processed, the first item is accessed first, and then each succeeding item is accessed in turn. New items cannot be inserted into a file of this type, but existing items can be replaced. There is no need to uniquely identify each item in the file.

If the sequential file partitioning option is used (see Appendix B), there are several additional advantages to sequential file organization. With this option, the sequential file can be segmented into a number of smaller files, called "members." Immediate access to the beginning of any member is possible. Item capacity of each member is independent, but file parameters are uniform throughout all members of the file.

ALLOCATION

All the units of allocation for a sequential file are used for data, and all the units of allocation for one file on one volume are collectively called a file volume. The procedure for determining the unit or units of allocation for a sequential file is described in Appendix C of this manual. Note that when the partitioning option is used for a sequential file, that file must be allocated on only one volume. A multivolume partitioned sequential file cannot be allocated or processed.

DATA STRUCTURE

The data in a sequential file consists of one physically continuous stream of items. Processing is done in physical sequence (which corresponds to logical sequence). The end of data in a sequential file is marked by an item starting with \square EOD \dagger (7625462477 octal). This item appears in the last data item position of each file volume filled with data and after the last data item of the file. All tracks allocated to a sequential file are used to store data, with the terminal item position of each file volume reserved for \square EOD \dagger .

Indexed Sequential File Organization

An indexed sequential file is organized so that each item can be accessed in logical sequence (like a sequential file) and directly. However, when an indexed sequential file is processed sequentially, items are retrieved in logical sequence and not necessarily in physical sequence (unlike a sequential file). To process an indexed sequential file, each item must be uniquely identified. The item identifiers used are called item keys or just simply keys. An item key is a contiguous set of characters within an item. A key can be any number of characters long and can appear anywhere within an item. However, each item key in an indexed sequential file must be the same length and appear in the same position within each item of the file, as determined by the user when the file is allocated. In addition, the key of each succeeding item of the file must be greater (in terms of the binary collating sequence) than that of the item immediately preceding it. In other words, the first item in the file must have a key whose value (in terms of the binary collating sequence) is the smallest, and each succeeding item in the file must have a key whose value (in terms of the binary collating sequence) is larger than that of the preceding item.

Based on the sequential ordering of items and keys in the file, the operating system builds three levels of indexes for the file. These are called the master index, the cylinder index, and the string index. The indexes are used in directly accessing an item in the file. To directly access an item in an indexed sequential file, an item key must be provided to the input/output routines. This key is located in the master index to point to the appropriate block in the cylinder index. The cylinder index, in turn, points to the appropriate cylinder's string index. The string index points to the first block of a string containing the item. Since an item can be accessed in this manner, a new item can be inserted into the file. To insert a new item, the system locates

the two items in the file which immediately precede and follow the new item (based on the value of the new item's key), and inserts the new item between them. Inserting items can cause items to overflow the data area of the file, in which case, the overflowing items are stored in the overflow areas, and the string indexes are adjusted to indicate this accordingly.

To sum up the capabilities afforded by the indexed sequential file organization, the user can: retrieve (access) and update each item in the file in logical sequence, as in the sequential file organization; retrieve and update items in any sequence, negating the requirements of sorting the transaction input to the sequence of the file; add new items to the file, with all adjustments to the indexes being handled automatically; delete an item from the file, and then reuse the item position; and retrieve any item in the file, and then all subsequent items in a logical sequence.

ALLOCATION

An indexed sequential file must have a minimum of three units of allocation. The first unit of allocation must be for the master/cylinder index; the second must be for the general overflow area; and at least one subsequent unit of allocation must be for data. Of course, on a single-volume indexed sequential file, all units of allocation will be on one volume, but the sequence of units of allocation must be as stated above: master/cylinder index, general overflow, and then data. The units of allocation for the master/cylinder index and general overflow may be any width (tracks per cylinder); but a unit cannot begin on the same cylinder as that which ended the previous unit of allocation. All data units must be of uniform track width. Four methods of allocating multivolume indexed sequential files are shown below.

Method 1.	Master/Cylinder Index Overflow Data	on volume A on volume B on volumes C through H
Method 2.	Master/Cylinder Index Overflow and Data Data	on volume A } on volume B on volumes C through H
Method 3.	Master/Cylinder Index Overflow and Data Data	} on volume A on volumes B through H
Method 4.	Master/Cylinder Index and Overflow Data	} on volume A on volumes B through H

As can be seen, the unit of allocation for the master/cylinder index and that for the general overflow cannot be separated by a unit of allocation for data; but the unit of allocation for the master/cylinder index need not be on the same volume as that for the general overflow. However, the master/cylinder index must be allocated first, and the general overflow must be allocated immediately after the master/cylinder index and before any data areas are allocated.

The procedure for determining the units of allocation for an indexed sequential file is described in Appendix C of this manual.

FILE STRUCTURE

An indexed sequential file is divided into three distinct kinds of areas: prime data, index, and overflow. The prime data area is the area initially loaded with data. There are two kinds of index areas, one for the master/cylinder index and one for the string indexes. There are two types of overflow areas, one for cylinder overflow and one for general overflow.

Prime Data Area

The prime data area is made up of any number of "strings" that will fit on the data portion of a cylinder. A string consists of any number of blocks, as specified by the user. Block length must not exceed one track. Formulas for computing the optimum string size can be found in Appendix C. The items are entered into each string in succession, according to the sequenced input file, when an indexed sequential file is loaded. Each cylinder allocated to the data area contains (1) a string index with as many items as there are strings of data on the cylinder and (2) one or more strings of data, each of which is a user-determined number of blocks long, followed optionally by (3), a cylinder overflow area. The cylinders allocated for data may or may not contain cylinder overflow areas.

An additional feature of the prime data area is that the user can specify a number of items per string as "imbedded overflow." Specifying imbedded overflow causes empty item spaces to be left at the end of each string loaded in the data area. When new items are inserted into a string, these item spaces can be used before either the cylinder or general overflow areas need be used.

Index Areas

The index areas (the master/cylinder index area and the string index areas) contain the master index, the cylinder index, and the string indexes. The master index and the cylinder index are allocated as a single unit, the master/cylinder index. The string indexes are not allocated as such; they are built when the file is loaded. Only those cylinders actually containing data will contain a string index. Those cylinders beyond the last cylinder actually loaded

are unavailable for data. Formulas for computing the size of the master/cylinder index can be found in Appendix C.

The master index is the highest level of index for the file and it contains one item for each block of the cylinder index. The block size of the master index is the same as the block size the user specifies for the data area. Each item in the master index contains an address field, a status field, and an item key field. The contents and purpose of these fields is explained in the paragraph entitled "Directly Processing an Indexed Sequential File," which follows this description of file structure.

The cylinder index is an intermediate-level index, and it contains one item for each loaded data cylinder in each data unit of allocation for the file. The block size of the cylinder index is the same as the block size that the user specifies for the data area. Each item in the cylinder index contains an address field, a status field, and an item key field. These fields are explained later. An option is available to have some or all blocks of the cylinder index area resident in main memory, providing higher processing speed.

The string indexes are the lowest-level index for the file. Each string index contains one item for each string of data loaded on a data cylinder of the file, and each string index is always at the beginning of that portion of each cylinder allocated for data. String indexes do not exist on cylinders beyond the last cylinder loaded with data. The block size of the string index is the same as the block size allocated for the file. Each item in a string index contains five fields: an address field, a status field, two key fields, and a reserved field. These fields are explained later.

Overflow Areas

The two overflow areas, the cylinder overflow area and the general overflow area, are used to store data items that either overflow the data strings or overflow the cylinder overflow areas. These areas are initialized by the File Support C load function but are used only in subsequent processing. Cylinder overflow areas are not required to be present; but when the user specifies that there is to be cylinder overflow, the area is made up of a number of tracks at the end of the allocated portion of each cylinder in the data unit of allocation. The number of tracks used for cylinder overflow can be one or more, as determined by the user when the file is allocated. The general overflow area, on the other hand, is required as a separate unit of allocation. The general overflow can be any track width (tracks per cylinder) and any number of cylinders, but it cannot begin on the cylinder on which the immediately preceding unit of allocation for the file ended, i. e., the master/cylinder index. Different files can share cylinders. Thus, the master/cylinder indexes of several different files may be on one cylinder, while the general overflow units for the same files share other cylinders.

Items that go into the cylinder overflow areas are all those items whose key values are greater than the highest key value for a string's data area and less than or equal to the highest key value of the string's associated cylinder overflow area. All items in each cylinder overflow area are always entered in ascending binary collating sequence with respect to their item key values.

Items that go into the general overflow area are all those items that cannot be contained in a data string or in the data string's cylinder overflow area. The general overflow area contains all the items that overflow from all the cylinder overflow areas. The items in the general overflow area also are always entered in ascending binary collating sequence with respect to their item key values.

DIRECTLY PROCESSING AN INDEXED SEQUENTIAL FILE

To retrieve an item directly from an indexed sequential file, the user supplies an item key value and specifies MSGET. Input/Output routines then begin searching the master index from its beginning for the item containing a key value equal to or greater than that supplied by the user. When the item in the master index is located, input/output routines are directed by that item's address field to the appropriate block in the cylinder index. This block is then searched from its beginning for the item equal to or greater than that supplied by the user. When the appropriate item in the cylinder index is located, input/output routines are directed by that item's address field to the relative volume and data cylinder string index whose highest key value is equal to or greater than that supplied by the user. The relationship between the items of the master index and the cylinder index is illustrated in Figure 2-5. The string index is then searched for the item containing the key value equal to or greater than that supplied. When the correct item is found, input/output routines are directed to either the correct block in the prime data area or the appropriate overflow area. Figure 2-6 shows the relationship between string index items and the data area of the cylinder immediately after loading. The contents of string index items is shown in greater detail in Figure 2-7. Addresses are shown in decimal but are recorded in binary.

Figure 2-7 also illustrates the insertion of items into a string. As can be seen in these illustrations, items are inserted in logical order based on the value of their keys. Figure 2-7 (Sheets 1, 2, and 3) illustrates how the string is filled. Notice in these illustrations that the key values in the string index item for this string do not change. In Figure 2-7 (Sheet 4), however, the value of the highest key currently in the string is changed by the input/output routines. The value of the highest key associated with the string is never changed. Two key values are maintained in each string index item to enable input/output routines to determine whether to search the string or the overflow area.

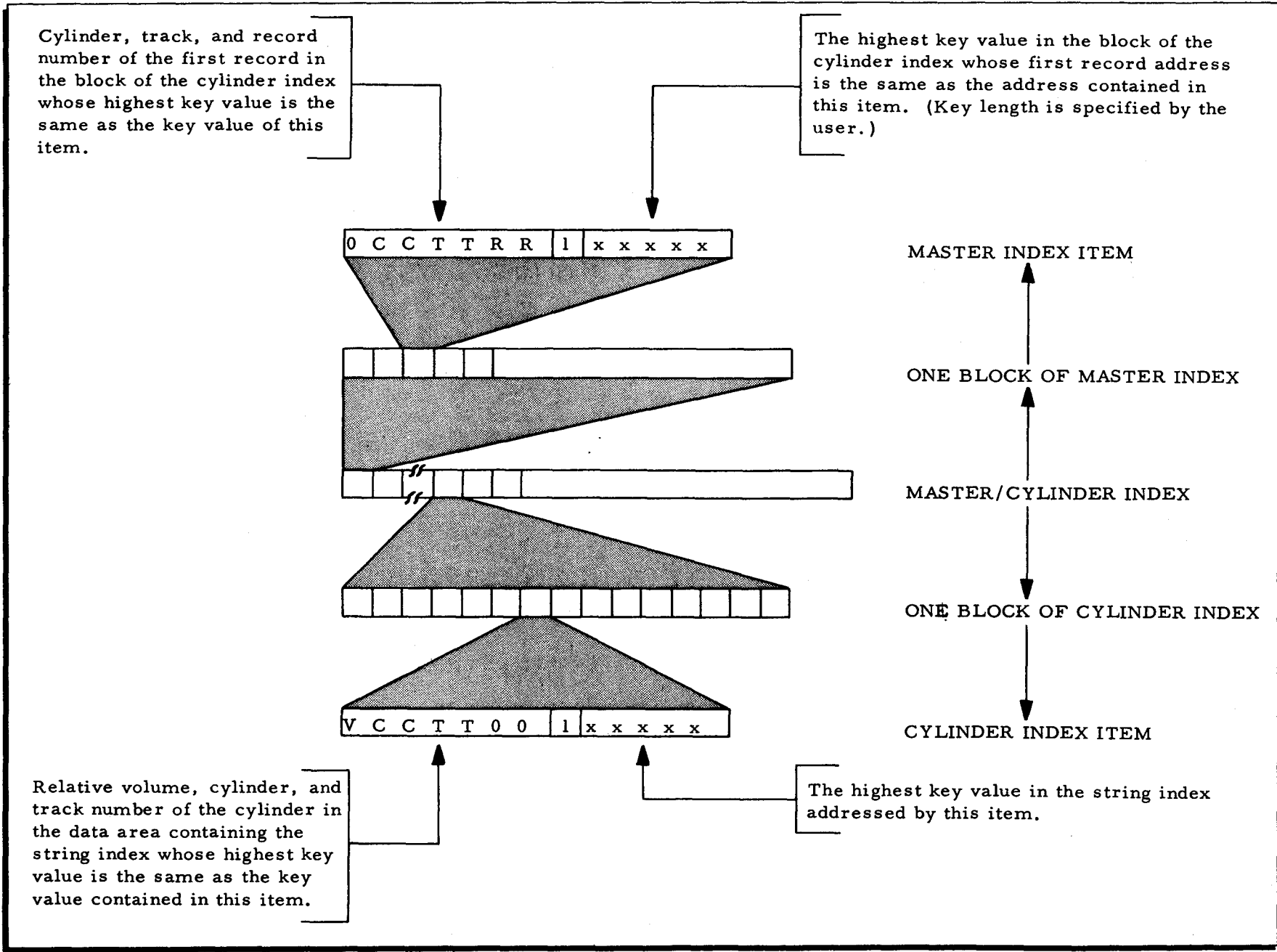


Figure 2-5. Relationship Between Items of the Master and Cylinder Index

2-14

#5-618

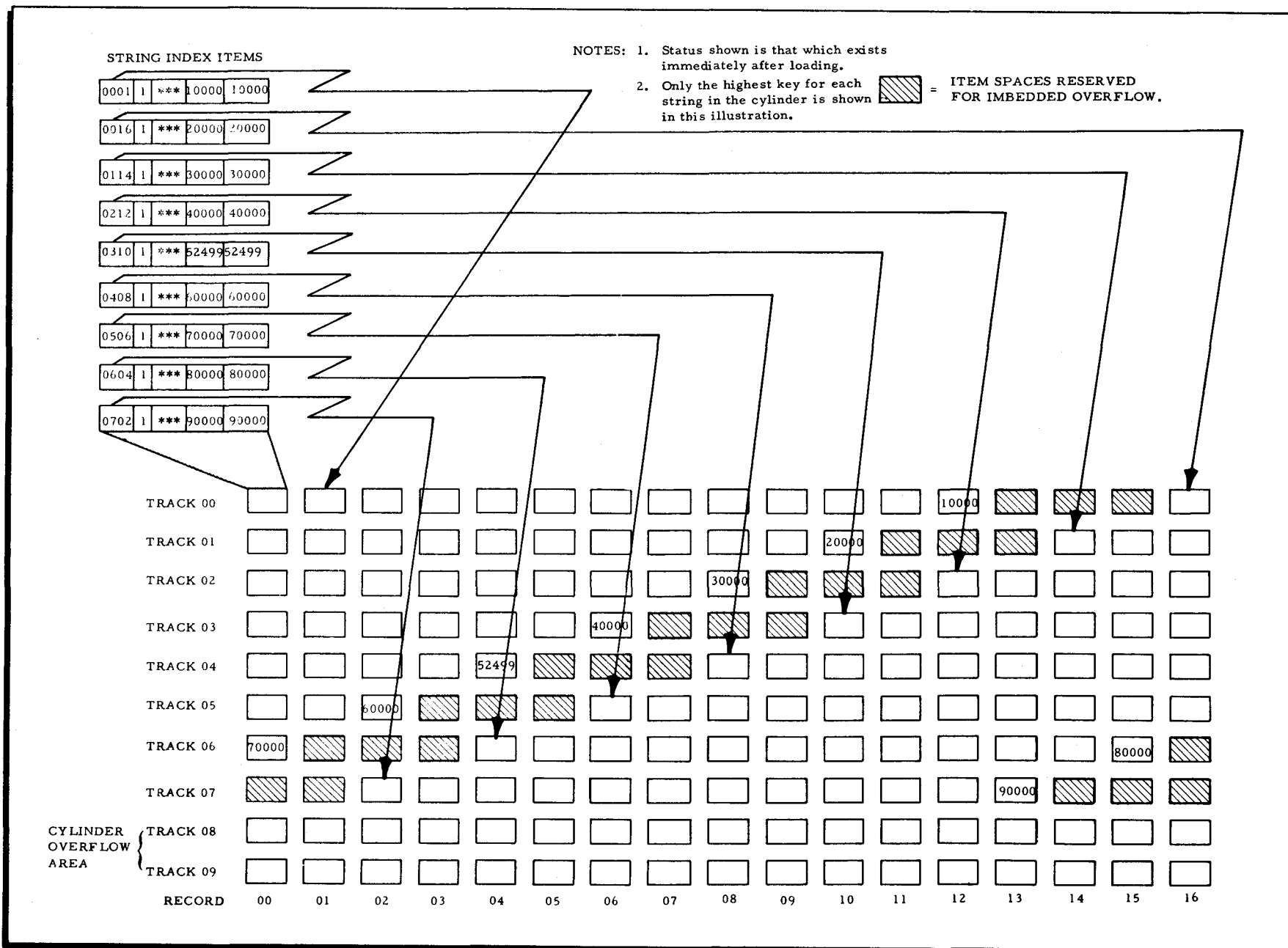


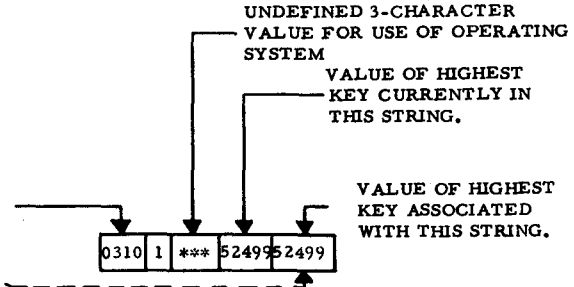
Figure 2-6. Relationship Between String Index Items and the Data Area of a Cylinder

STRING INDEX ITEMS

0001	1	***	10000	10000
0016	1	***	20000	20000
0114	1	***	30000	30000
0212	1	***	40000	40000
0310	1	***	52499	52499
0408	1	***	60000	60000
0506	1	***	70000	70000
0604	1	***	80000	80000
0702	1	***	90000	90000

- NOTES: 1. Contents of string index items are in binary but are shown in decimal format for ease of explanation.
2. Keys of the 12 items loaded are shown in string five. Note that three item spaces are reserved for imbedded overflow.

TRACK AND RECORD NUMBER OF THE FIRST RECORD IN THE STRING WHOSE HIGHEST KEY VALUE IS THE SAME AS



= ITEM SPACES RESERVED FOR IMBEDDED OVERFLOW.

TRACK 00													10000					
TRACK 01											20000							
TRACK 02								30000										
TRACK 03						40000				51000	51100	51200	51300	51400	51600	51786		
TRACK 04	51900	52165	52216	52343	52499													
TRACK 05			60000															
TRACK 06	70000													80000				
TRACK 07													90000					
CYLINDER OVERFLOW AREA	TRACK 08																	
	TRACK 09																	
	RECORD	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16

2-16

#5-618

Figure 2-7. Insertion of Items Into a String (Sheet 1)

STRING INDEX ITEMS

0001	1	***	10000	10000
------	---	-----	-------	-------

0016	1	***	20000	20000
------	---	-----	-------	-------

0114	1	***	30000	30000
------	---	-----	-------	-------

0212	1	***	40000	40000
------	---	-----	-------	-------

0310	1	***	52499	52499
------	---	-----	-------	-------

0408	1	***	60000	60000
------	---	-----	-------	-------

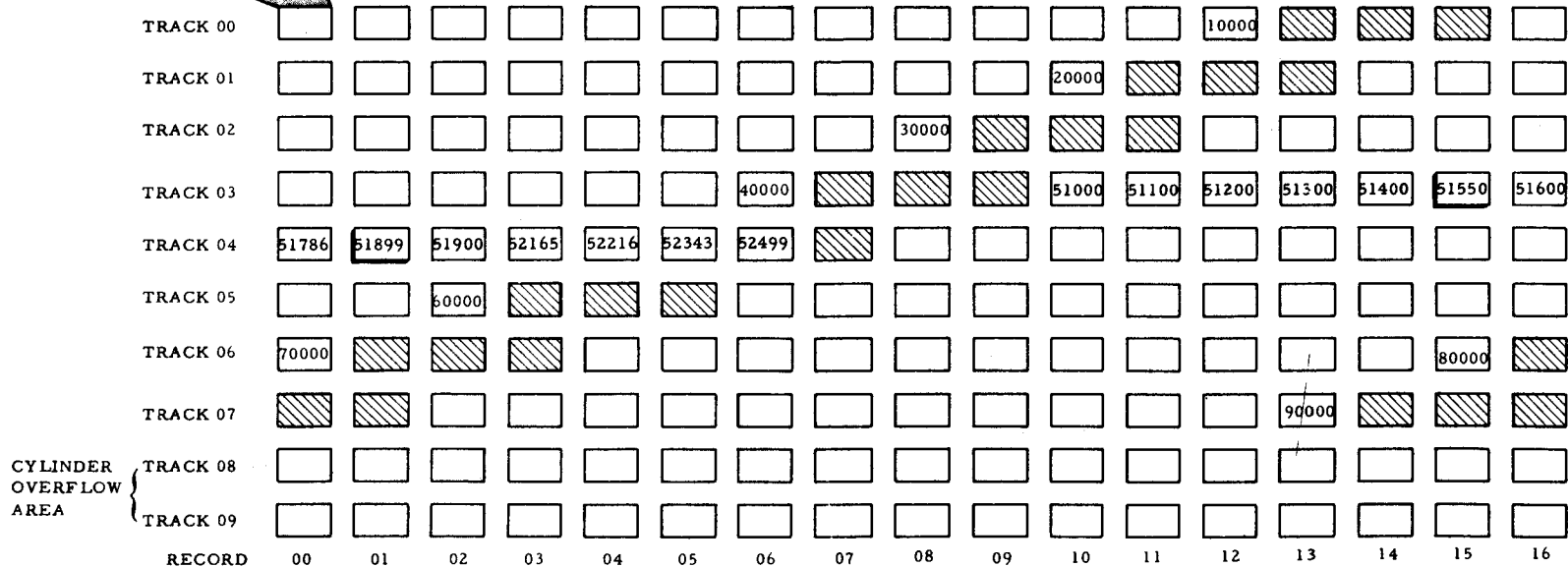
0506	1	***	70000	70000
------	---	-----	-------	-------

0604	1	***	80000	80000
------	---	-----	-------	-------

0702	1	***	90000	90000
------	---	-----	-------	-------

NOTE: Two items have been inserted:
51550 and 51899. No overflow
has occurred.

 = ITEM SPACES RESERVED FOR IMBEDDED OVERFLOW.



2-17

#5-618

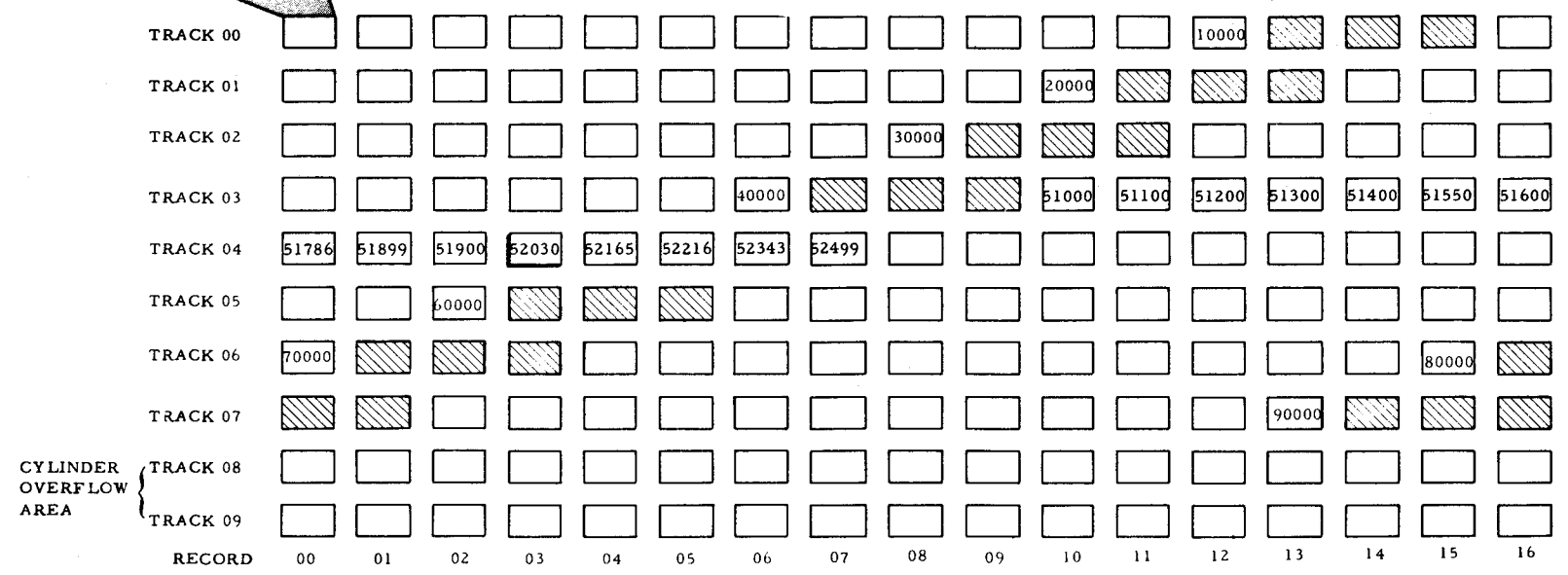
Figure 2-7. Insertion of Items Into a String (Sheet 2)

STRING INDEX ITEMS

0001	1	***	10000	10000
0016	1	***	20000	20000
0114	1	***	30000	30000
0212	1	***	40000	40000
0310	1	***	52499	52499
0408	1	***	60000	60000
0506	1	***	70000	70000
0604	1	***	80000	80000
0702	1	***	90000	90000

NOTE: One item has been inserted:
52030. No overflow has
occurred.

 = ITEM SPACES RESERVED
FOR IMBEDDED OVERFLOW.



2-18

#5-618

























Figure 2-7. Insertion of Items Into a String (Sheet 3)

STRING INDEX ITEMS

0001	1	***	10000	10000
0016	1	***	20000	20000
0114	1	***	30000	30000
0212	1	***	40000	40000
0310	1	***	52216	52499
0408	1	***	60000	60000
0506	1	***	70000	70000
0604	1	***	80000	80000
0702	1	***	90000	90000

NOTE: Items 51050 and 51901 have been inserted. Items 52343 and 52499 are placed in cylinder overflow. Also note adjustment to string index item for the fifth string.

 = ITEM SPACES RESERVED FOR IMBEDDED OVERFLOW.

TRACK 00													10000				
TRACK 01											20000						
TRACK 02								30000									
TRACK 03						40000				51000	51050	51100	51200	51300	51400	51550	
TRACK 04	51600	51786	51899	51900	51901	52030	52165	52216									
TRACK 05			60000														
TRACK 06	70000														80000		
TRACK 07													90000				
CYLINDER OVERFLOW AREA	TRACK 08	52343	52499														
	TRACK 09																
RECORD	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16

2-19

#5-618

Figure 2-7. Insertion of Items Into a String (Sheet 4)

Figures 2-8 and 2-9 show how the item position of a deleted item can be used. Notice that the key of the item inserted (Figure 2-9) is less than that of the item following the deleted item (Figure 2-8). If the key of the inserted item had been greater than that of the item following the deleted item and less than that of the last item in the string (52216), it would have been inserted in the string, and the item with the key 52216 would have been inserted in the cylinder overflow area. Also, the value of the highest key currently in the string would have been changed in the string index item.

DATA ITEM STATUS CHARACTER

In indexed sequential files, an area on mass storage that can contain an item is called an item position. Because the indexed sequential file organization offers the ability to insert and delete items, it is necessary to distinguish between an item position that contains an item and one that does not. To accomplish this, an item position is defined as having two parts: the data portion (including the item key) and the status character part. An item position, therefore, is one character longer than the data portion. When designing indexed sequential files, the user must include the status character in the item size computations.

The input/output routines use the status character to indicate whether the item position is unused (inactive), contains an active item, or contains an item that has been deleted. When an indexed sequential file is allocated and before any data is recorded in the file, the status character of every item position is 77 (octal). The octal values the status character can have for an indexed sequential file after the file has been loaded are as follows:

- 01 = Active,
- 41 = Deleted or imbedded, and
- 42 = Inactive. (If this status character appears in the prime data area, it indicates that the current block is one beyond the last active string of the file. If it appears in an overflow area, it indicates that an active item was never inserted into or beyond this position.)

Direct Access File Organization

The direct access file is organized to provide fast access to items that are not to be retrieved sequentially. A direct access file is organized principally in terms of buckets. Buckets are user-defined areas that may contain one or more items. When a bucket contains more than one item, there is no ordering of the items' within that bucket.

A bucket and a block (as defined previously) may be the same size, or a bucket may contain more than one block. Note that a large bucket may increase the access time to a given item but may decrease the probability of overflow. A smaller bucket, however, might reduce the access

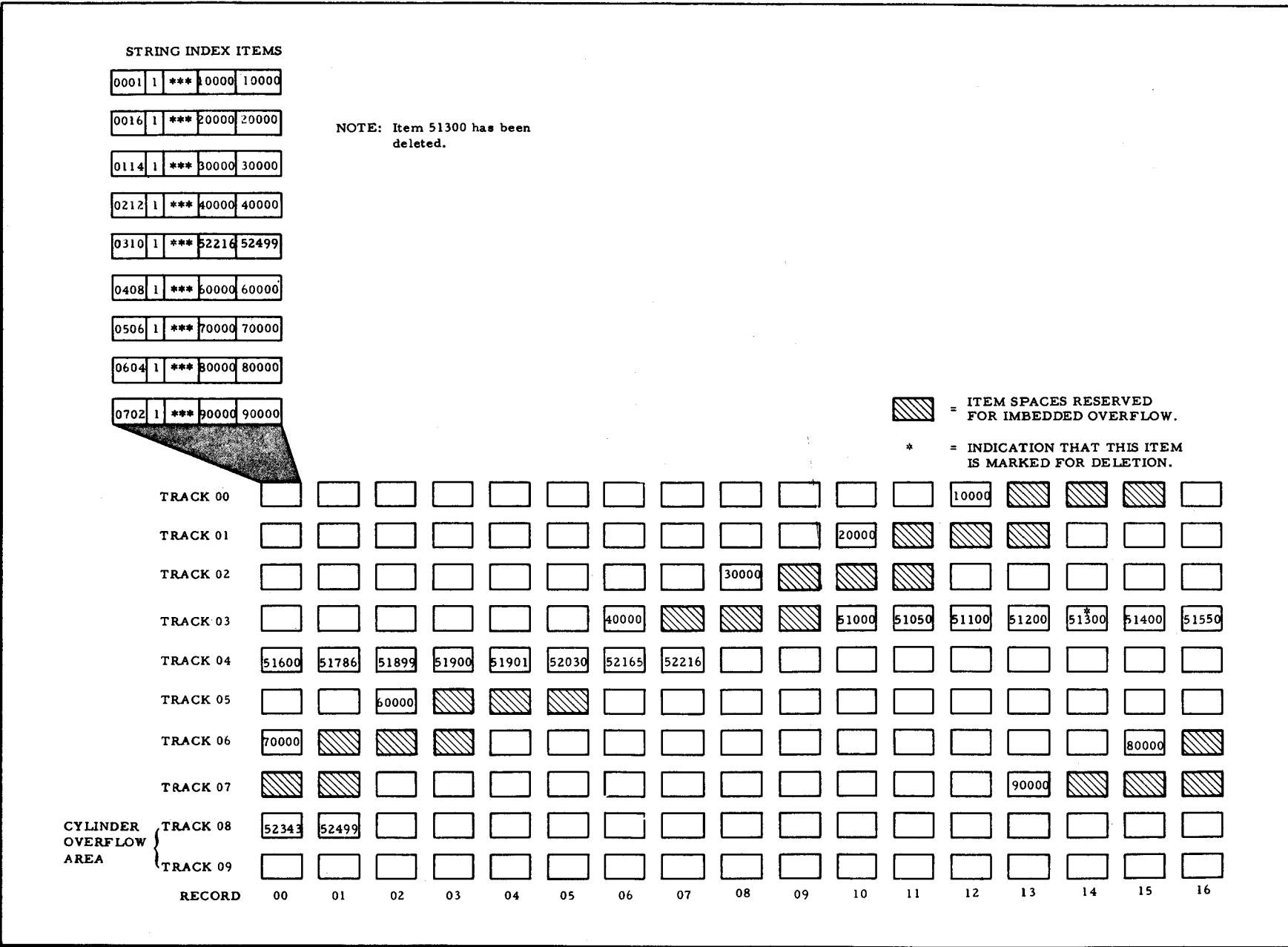


Figure 2-8. Deletion of an Item from a String

STRING INDEX ITEMS

0001	1	***	10000	10000
------	---	-----	-------	-------

0016	1	***	20000	20000
------	---	-----	-------	-------

0114	1	***	30000	30000
------	---	-----	-------	-------

0212	1	***	40000	40000
------	---	-----	-------	-------

0310	1	***	52216	52499
------	---	-----	-------	-------

0408	1	***	60000	60000
------	---	-----	-------	-------

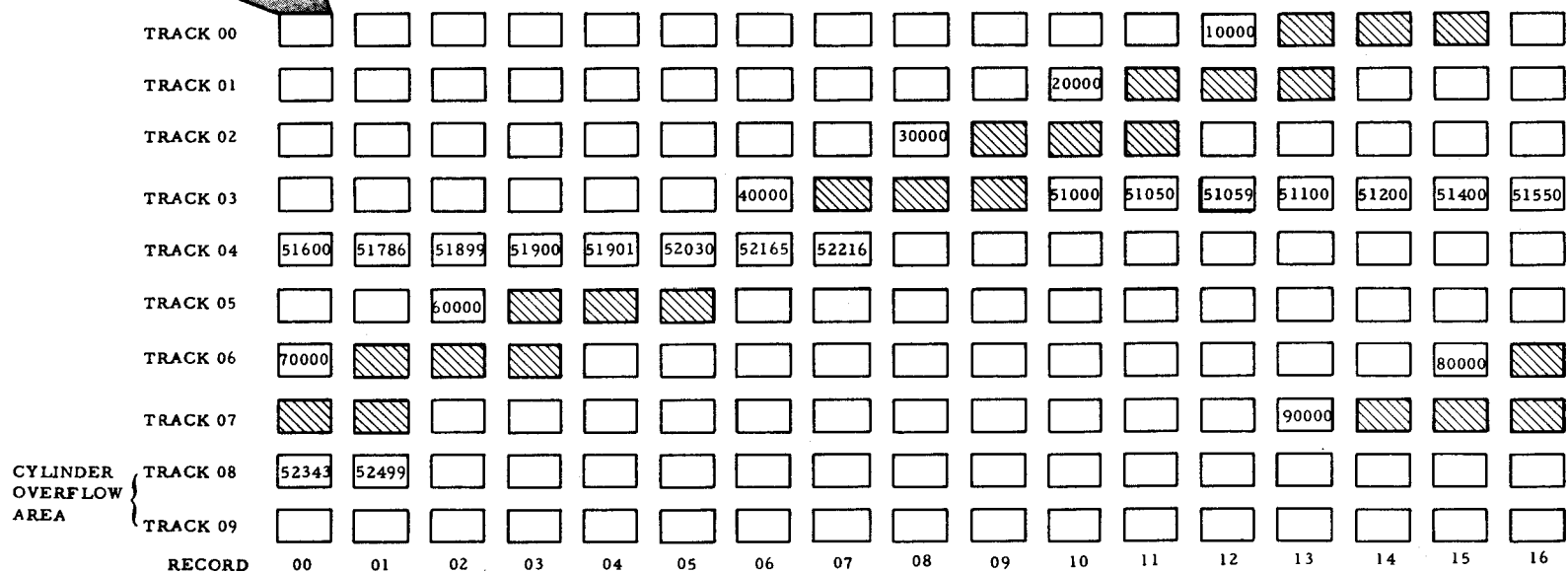
0506	1	***	70000	70000
------	---	-----	-------	-------

0604	1	***	80000	80000
------	---	-----	-------	-------

0702	1	***	90000	90000
------	---	-----	-------	-------

NOTE: Item 51059 has been inserted. No overflow due to previous deletion.

 = ITEM SPACES RESERVED FOR IMBEDDED OVERFLOW.



2-22

#5-618

Figure 2-9. Using the Item Position of a Deleted Item

time to any given item in the bucket, since the area to be searched is less than that in a large bucket.

ALLOCATION

Allocation of a direct access file is done in terms of the unit of allocation, as previously defined on page 2-6. There are no restrictions to the allocation of space for direct access files. Formulas for calculating a unit of allocation for a direct access file are described in Appendix C of this manual.

FILE ORGANIZATION

The direct access file is divided into two areas: data area and overflow area.

Data Area

The data area of any cylinder allocated to a direct access file consists of those tracks of the unit of allocation for that cylinder, minus those tracks within the unit of allocation reserved for cylinder overflow. Cylinder overflow is optional; if not requested, the data area of the cylinder consists of all the tracks in the unit of allocation for that cylinder.

Within the data area, the file is divided into buckets. A bucket's address is the address of the first record within that bucket. (A bucket can contain one or more blocks, and a block can contain one or more items.) When a bucket contains more than one item, there need be no logical relationship between the items, except that through some means (such as randomizing), the address of that bucket was specified as belonging to those items.

The size of a bucket cannot be greater than one cylinder and a bucket cannot begin on one cylinder and end on the next. A bucket is processed as though it flowed directly into the cylinder overflow area (if any) and then into the general overflow area.

One relationship between items, records, blocks, and buckets is illustrated in Figure 2-10A; a second relationship is illustrated in Figure 2-10B.

Overflow Areas

There are two types of overflow areas: the cylinder overflow area and the general overflow area. Each cylinder overflow area is used to accommodate items that overflow the buckets in the data area of that cylinder. The cylinder overflow area is optional; when specified, the user defines a number of tracks to be set aside at the end of each cylinder in the units of allocation for the file. The general overflow area is also optional; but, if specified, it is used to

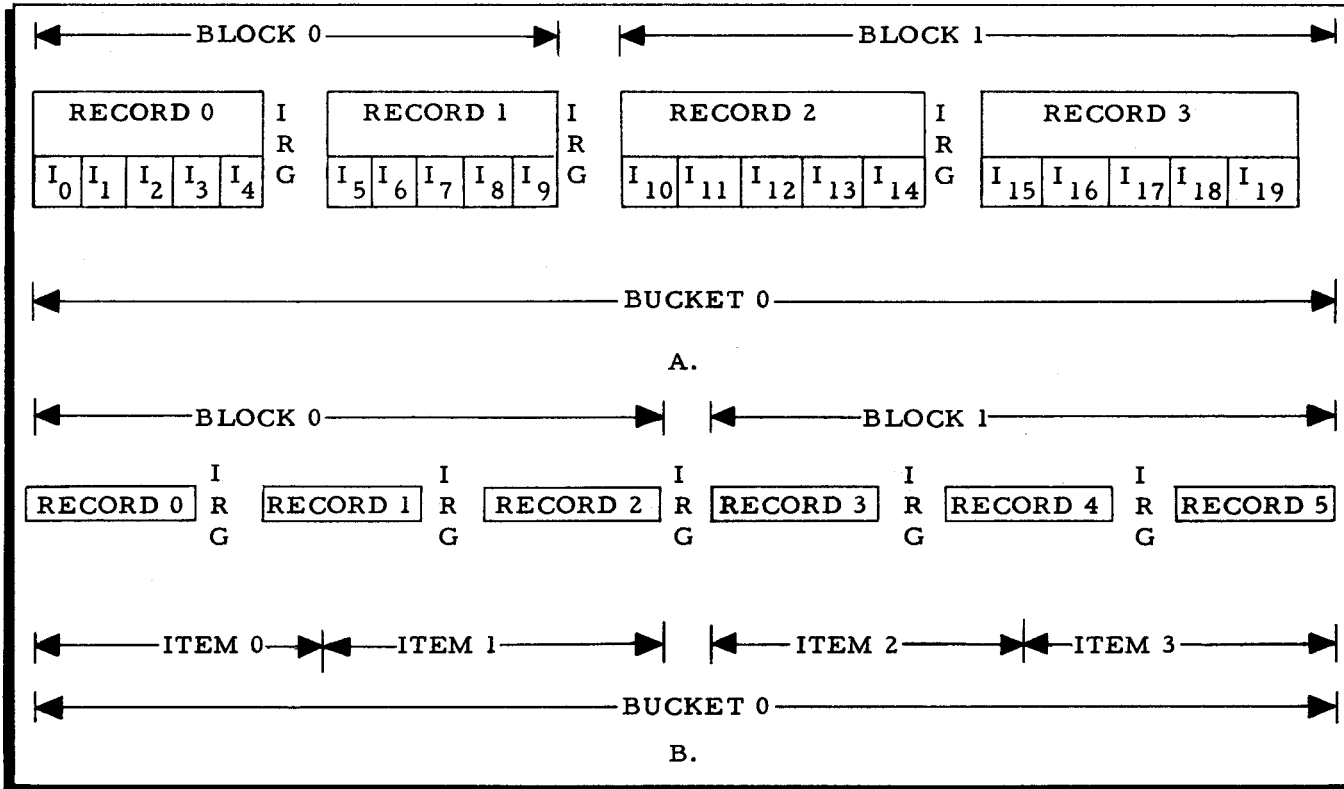


Figure 2-10. Relationship between Items, Records, Blocks, and Buckets

accommodate items that overflow the cylinder overflow areas (if any). If cylinder overflow is not specified, items overflowing any bucket in a unit of allocation enter the general overflow area. When general overflow is specified, the last cylinder of each unit of allocation for the file is used as the general overflow area. All tracks allocated on that cylinder are used for overflow.

DIRECT ACCESS FILES AND KEYS

The meaning of the word "key" depends on the context in which it is used. To define direct access file structure and processing, it is necessary to distinguish between the various uses of the word key. The following list provides the definitions used in this manual:

1. **Actual Key.** The actual, absolute (physical) address of the bucket, in terms of device, pack, cylinder, track, and record.
2. **Relative Key.** The number of a bucket, relative to the beginning of the file. The first bucket in a file is numbered 0. The input/output (I/O) program converts the relative key supplied by the user into the actual key for the item.
3. **Item Key.** The identification field (e. g., part number or employee number) of an item. This field must consist of contiguous characters, but its length and location within the item are determined by the user when the file is allocated.

Use of a relative key relieves the programmer of the following considerations: device address, intervening overflow areas, multiple units of allocation, and multivolume file factors.

Directly accessing an item normally requires that the user provide the bucket address (either relative or actual) and the item key to the input/output routines. The input/output routines locate the beginning of the bucket and then search through it for the item with the specified item key. If the item is not found in the bucket, the search continues through the cylinder overflow area (if any) and, if necessary, through the general overflow area (if any).

DATA ITEM STATUS CHARACTER

In a direct access file, an area on mass storage that can contain an item is called an item position. Because the direct access file organization offers the ability to insert and delete items, it is necessary to distinguish between an item position that contains an item and one that does not. To accomplish this, an item position is defined as having two parts: the data portion (including the item key) and the status character. An item position, therefore, is one character longer than the data portion. When designing a direct access file, the user must include the status character in the item size computations.

The input/output routines use the status character to indicate whether the item position is unused (inactive), contains an active item, or contains an item that has been deleted. When a direct access file is allocated and before any data is recorded in the file, the status character of every item position is set to inactive. The octal values which the status character can assume are as follows:

- 01 = Active;
- 41 = Deleted;
- 77 = Inactive;
- 00 = Active, last block in file volume;
- 40 = Deleted, last block in file volume; and
- 76 = Inactive, last block in file volume.

CUMULATIVE LOADING OF A DIRECT ACCESS FILE

Because the placement of any item in a direct access file is independent of the placement of any other item, the contents of a direct access file can be accumulated into the file through a series of separate load operations over any period of time.

The data source and the item format can vary for each load operation. It is the responsibility of the user's own-code program to standardize the item format for proper placement within the direct access file. (See Section IV.)

PROCESSING CONVENTIONS

Volume processing conventions within the Mod 1 (MSR) Operating System are designed to provide standardized operating procedures which ensure maximum flexibility and operational convenience. These conventions are dependent upon two factors: whether processing is direct or sequential and which functions are being performed.

Sequential or Direct Processing

Data in mass storage files can generally be processed either sequentially or directly. Sequential processing can be in physical sequence (e.g., direct access organization), in which case there is no attempt to order data, and items are processed as they are encountered. Alternatively, sequential processing can be in logical sequence (e.g., indexed sequential organization). In this case, items are processed according to the sequence of some item key field.

When a file is processed sequentially, each volume is operated on as needed; the second volume is not used until processing of a prior volume is completed.

A sequential file is always processed sequentially. Direct access and indexed sequential files can be processed using the minimum number of volumes if files are processed sequentially from the beginning.

Direct processing can be achieved by maintaining a unique address for each item and accessing an item by referring directly to that address. Alternatively, it is often convenient to provide a range of addresses which may contain several items. In this case, retrieval is achieved by beginning at the first address within this range and searching for the desired item. Both an address and an identifier are required. In some cases, these elements are identical. For example, when indexed sequential file organization is used, the key is applied to locate an address from the indexes as well as to identify the item upon which the search is conducted.

When a file is processed directly, all volumes of the file must be online.

Volume Processing Functions

Volume Preparation C, Mass Storage Edit C, Bootstrap Generator C, Disk/Tape Copy, and the File Support C map function deal with volumes as their basic unit of operation. These functions are not oriented to the concept of a file in its usual sense. Thus, whenever these functions are to be performed on more than one volume, each volume is treated individually and requires a separate execution of the program.

File Processing Functions

Most of the remaining functions of the Mod 1 (MSR) Operating System are oriented to the concept of a file. The file may be on one volume or it may be on several volumes. Some functions process only single-volume files (e. g., Library File Update C and Executable Program File Update C).

Certain file-oriented operations are independent of file organization and mode of processing. For example, File Support C allocation and deallocation are handled similarly for all file types, irrespective of the mode of processing. These functions are performed cyclically upon all volumes of a file; i. e., if devices a, b, and c are assigned, allocation or deallocation follows the order: a, b, c, a, b, c, a, etc.

BACKUP PROCEDURES

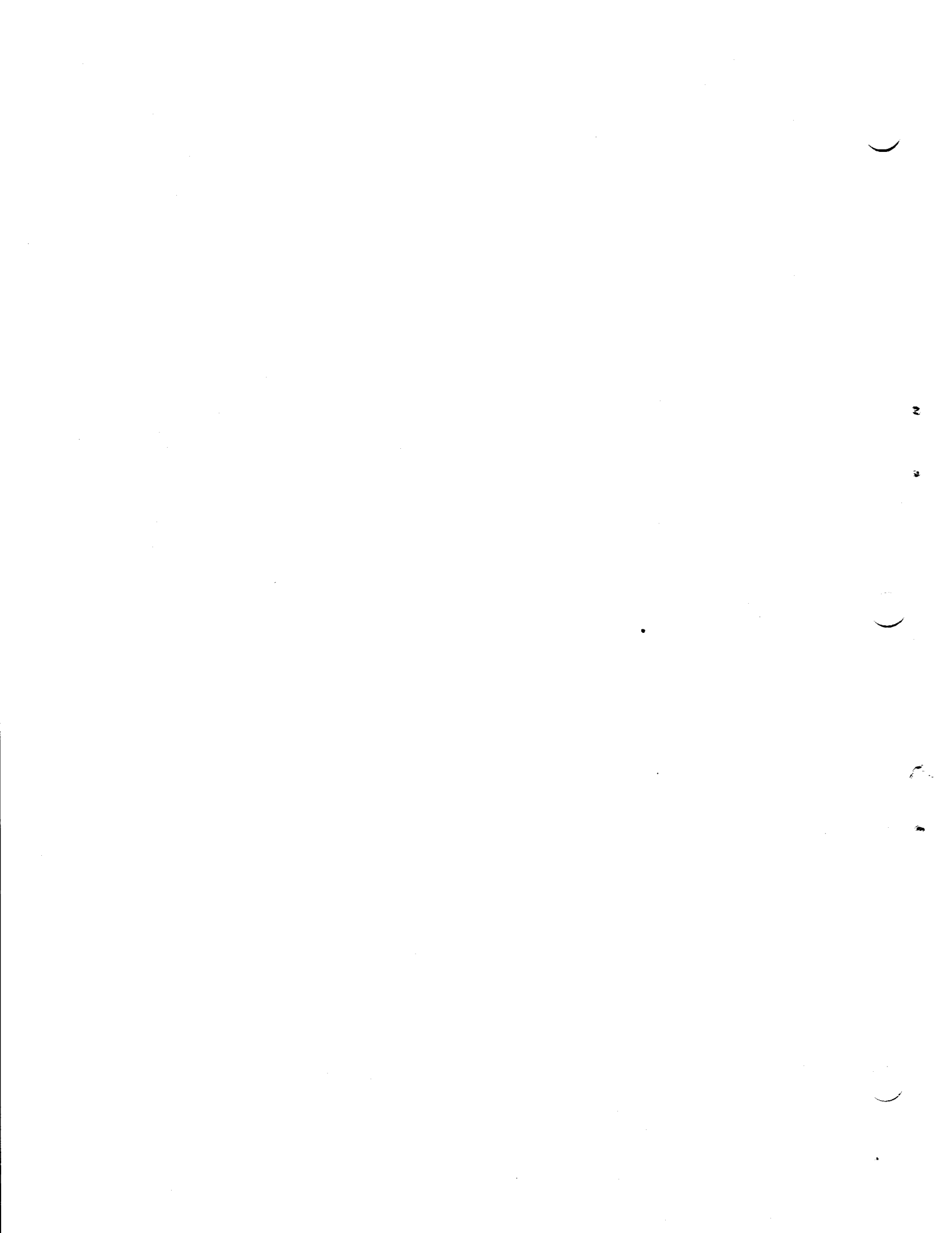
Logical Backup

When a file requires reorganization (perhaps because of a significant change in required capacity) and the data content must be preserved, the contents of the file can be temporarily stored on another medium or another disk by means of the File Support C load/unload program (see Section IV). When the reorganization has been accomplished, via reallocation of the file, the data can then be restored by the File Support C load/unload program.

Physical Backup

When it is desired to save physical areas of a volume in full track format, with or without regard to individual file boundaries, the Disk/Tape Copy program can be used to store the physical contents of entire tracks either onto a tape or onto another disk. (See Utility Routines manual, Section VI.)

Disk/Tape Copy operates on a full track at a time and thus offers a faster method of data transfer. However, the user must give careful consideration to maintaining a valid correspondence between the volume directory and the file areas on the volume. *



SECTION III

LOGICAL I/O C

Logical I/O C consists of a set of macro routines designed to access files residing on mass storage and to operate on the data within those files. The assembly-language programmer calls for and specializes these routines in his source-language program, and they are assembled into his machine-executable program through the use of Mass Storage EasyCoder Assembler C of the Program Development Subsystem.

The functions performed by Logical I/O C are summarized in the following list.

1. Open or close a file, verifying and updating the file directory information.
2. Get, put, or replace individual items in a file, blocking and unblocking as necessary.
3. Insert an item into or delete an item from a direct access or indexed sequential file.
4. Directly access items in a direct access or indexed sequential file.
5. Establish linkage to Physical I/O C. Physical I/O C, which reads and writes data, detects errors, and (if possible) corrects errors, is described in Appendix D of this manual.
6. Provide exits to user-written label and error routines.
7. Ensure the simultaneity of central processor and input/output operations.
8. Terminate sequential processing on one volume and switch to the next volume.
9. Allow other processing (including peripheral data transfers) to occur during cylinder-to-cylinder access time (seek time) of a disk device.

Logical I/O C is composed of four types of macro routines. These are the control, file description, communication area service, and action macro routines. The control macro routine is called the mass storage input/output control (MIOC) macro routine and provides general control over the entire input/output process. The file description macro routine (MCA) is called the mass storage communication area macro routine and sets up a communication area for the file being processed in which all values necessary to describe the file and the processing options are stored. Pertinent portions of the communication area are available to the user and can be altered by him through the use of the communication area service macro routines (see page 3-2). Unloading any applicable field of the communication area is performed by the unload communication area service (MUCA) macro routine; altering any applicable field of the communication area is performed by the load communication area service (MLCA) macro routine. The action macro routines are included in the main line of the user's coding to cause the various functions of the input/output routines to be performed.

MASS STORAGE INPUT/OUTPUT CONTROL MACRO ROUTINE (MIOC)

The mass storage input/output control macro routine (MIOC) is a segmentable series of subroutines that are specialized at assembly time to accommodate all input/output functions requested within a given program. Unnecessary coding is eliminated from each subroutine on the basis of user-specified parameters. The control macro routine further specializes itself at execution time on the basis of a specific description of a file that the programmer creates with the file description macro routine.

A single MIOC can process one or more files. These files can be on different mass storage device types used concurrently. The control macro routine performs all necessary interface functions between the user's program, the user's exit routines, and Physical I/O C.

FILE DESCRIPTION MACRO ROUTINE (MCA)

The file description macro routine (MCA) creates a communication area for each file that is to be processed by a given program. All values required to describe a file's organization and structure are placed in the file's communication area. There are two sources for these values: (1) the file description area of the volume directory (*VOLDESCR*) and (2) those values specified by the programmer at assembly time. Most of the values that are placed in the file's communication area are taken from the *VOLDESCR* entry for the file and placed in the communication area at execution time. Those values that cannot be placed in the communication area at execution time are specified by the programmer when he writes the MCA macro call; these values are placed in the file's communication area at assembly time.

COMMUNICATION AREA SERVICE MACRO ROUTINES (MLCA AND MUCA)

The macro routines which service the communication area (MLCA and MUCA) load information into and interrogate certain fields of the communication area. Using these macro routines, the programmer can alter the contents of certain fields of the communication area without knowing its structure.

ACTION MACRO ROUTINES

The programmer includes action macro routines (summarized in Table 3-1) in the main line of his coding to cause the various functions of Logical I/O C to be performed. The file processing functions that the action macro routines perform are described in detail later in this section.

SUMMARY OF LOGICAL I/O C MACRO ROUTINES

The macro routines that make up Logical I/O C are listed in Table 3-1. This table lists

SECTION III. LOGICAL I/O C

each macro routine according to its type, shows each macro name, and gives a brief description of the general function performed by each macro routine.

Table 3-1. Summary of Logical I/O C Macro Routines

Macro Type	Macro Name	General Function Performed
Input/Output Control	MIOC	Provides general control over the entire input/output process.
File Description	MCA	Sets up a communication area in which all values necessary to describe a file and the processing options are stored.
Communication Area Service	MLCA	Used to alter applicable fields of information in the communication area.
	MUCA	Used to unload applicable fields of information in the communication area.
Action	MSOPEN	Opens a file for processing.
	MSCLOS	Closes a file after processing.
	MSGET	Retrieves an item in the file.
	MSPUT	Delivers items sequentially from memory to mass storage.
	MSREP	Replaces the last item retrieved.
	MSINS	Inserts an item into an indexed sequential or a direct access file.
	MSDEL	Deletes the last item retrieved from an indexed sequential or a direct access file.
	MSEEK	Positions read/write heads to a desired cylinder of a direct access or indexed sequential file allowing other data processing to occur.
	SETM*	Sets processing to the beginning of the specified member.
	ENDM*	Stops processing of the current member.
	MALTER*	Changes the specified member of a file as directed.
	MSREL*	Frees the area occupied by a partitioned sequential file.
SETL**	Sets processing to a specific location of the file for sequential delivery of items.	

*Applies to partitioned sequential files only.
 **Applies to indexed sequential files only.

FILE PROCESSING MODES

There are three file processing modes available: input/output processing, input-only processing, and output-only processing. Sequential and partitioned sequential files can be processed in all three modes, but in certain processing modes, certain functions are not applicable. Indexed sequential and direct access files can be processed only in the input/output and the input-only modes.

Input/Output Processing Mode

In this processing mode, the user can both read data items from the file (input) and write data items to the file (output). With sequential files, this mode is used when it is desired to read data and then update some or all of the data read. With direct access and indexed sequential files, the input/output processing mode is used for the same purpose as with sequential files, or it may be used to insert new items into the file or delete items from the file.

Input-Only Processing Mode

In this processing mode, the user can only read data items from the file (input); he cannot write data items onto the file. This protects the file from undesired alteration.

Output-Only Processing Mode

In this processing mode, the user can only write data items onto a sequential file or partitioned sequential file (output); he cannot read data items from the file.

ACTION MACRO PROCESSING FUNCTIONS

Each function is identified by the name of the action macro call that requests the function. These names are shown in this description in upper case letters. For example, the function of opening a file for processing is requested by the MSOPEN macro call.

The functions the action macro routines perform can be divided into two groups: file-control functions and item-handling functions. The file-control functions are:

1. Opening files for processing,
2. Closing files after processing,
3. Starting processing at the beginning of a specified member,
4. Ending processing of a member,
5. Altering the status of a member,
6. Releasing a file to an unused state,
7. Starting processing at a specified location of the file, and

8. Seeking a specific cylinder of a mass storage volume, preparatory to actual processing of data resident on that cylinder, while allowing other processing to occur during the seek.

In the above list, 1 and 2 apply to all file organizations; 3, 4, 5, and 6 apply only to partitioned sequential files; 7 applies only to indexed sequential files; and 8 applies to indexed sequential or direct access files. Item-handling functions are as follows:

1. Retrieving an item,
2. Replacing an item,
3. Putting items sequentially onto the mass storage file.
4. Inserting items, and
5. Deleting items.

In this list, 1 and 2 apply to all file organizations; 3 applies only to sequential and partitioned sequential files; and 4 and 5 apply only to indexed sequential and direct access files.

Because the various types of files can be processed in more than one mode, certain functions of Logical I/O C action macro calls are not always applicable. The applicable functions for each processing mode are shown in Table 3-2. In Table 3-2, the functions are represented by the action macro call that requests the function. These macro calls are defined as follows:

1. MSOPEN - opens a file for processing,
2. MSCLOS - closes a file after processing,
3. SETM - sets processing to the beginning of the specified member,
4. ENDM - ends processing of a member,
5. MSREL - releases a file to an unused state,
6. SETL - sets processing to the location specified for sequential delivery of items,
7. MALTER - alters the status of a member,
8. MSGET - gets (retrieves) an item in a file,
9. MSREP - replaces the last item retrieved in a file,
10. MSPUT - delivers items sequentially to a mass storage file,
11. MSINS - inserts an item into a file,
12. MSDEL - deletes an item from a file, and
13. MSEEK - seeks a cylinder for subsequent input/output processing of a file allocated to that cylinder.

Note that whenever the phrase "an exit is taken" is used in the subsequent descriptions of the functions, the exits are optional. The only exception to this is that end-of-data exit must

be specified. When an optional exit is not specified for a particular function, the action either continues under default conditions or provides notice to the operator.

Table 3-2. Action Macro Calls for Each File Type in Each Processing Mode

Processing Mode	Action Macro Calls			
	Sequential File Organization	Partitioned Sequential File Organization	Indexed Sequential File Organization	Direct Access File Organization
Input/Output Mode	MSOPEN MSCLOS MSGET MSREP	MSOPEN MSCLOS MSGET MSREP SETM ENDM MALTER MSREL	MSOPEN MSCLOS MSGET MSREP MSINS MSDEL SETL MSEEK	MSOPEN MSCLOS MSGET MSREP MSINS MSDEL MSEEK
Input-Only Mode	MSOPEN MSCLOS MSGET	MSOPEN MSCLOS MSGET SETM ENDM	MSOPEN MSCLOS MSGET SETL	MSOPEN MSCLOS MSGET
Output-Only Mode	MSOPEN MSCLOS MSPUT	MSOPEN MSCLOS MSPUT SETM ENDM MALTER MSREL	Not Applicable	

Opening Files

While the process of opening each of the four file types is similar, not all the steps performed for one type are performed for another. Similarly, additional steps are performed when opening multivolume files.

OPENING SEQUENTIAL FILES

Sequential files are requested to be opened for processing in the mode specified in the MSOPEN macro call. In executing the open function for sequential files, Logical I/O C performs the following steps.

1. Logical I/O C attempts to locate the file requested in the MSOPEN macro call in the name portion (*VOLNAMES*) of the volume directory. If the file name cannot be located in *VOLNAMES* an exit is taken. A return to the open function from the exit indicates that a new volume has been mounted and that a new attempt to open the specified file is to be made.
2. After the file name is located in *VOLNAMES*, Logical I/O C verifies that the file-volume sequence number is zero. If it is not, an exit is taken.

The user can return to the open function from the exit requesting a new attempt to open (implying that a new volume has been mounted) or return to the open function from the exit requesting that the volume be accepted regardless of its file-volume sequence number. (The option to accept the volume regardless of its sequence number is not available in the output-only mode.)

3. After verifying the file-volume sequence number, Logical I/O C locates the file's description in *VOLDESCR*. If password checking for the file is specified, the password for that file is compared to the user's password field referred to by the file's communication area. If password checking for the file is not specified, Logical I/O C verifies that the password field for the file contains all blanks. If either password check fails, an exit is taken. The only allowable return from an exit because of a failure of a password check is for the user to attempt to reopen the file. The password check can fail for any of the following reasons:
 - a. If the user specified password checking in the MCA macro call and the file is not protected by a password;
 - b. If the password for the file supplied by the user (via parameter 21 of the MCA macro call) is not identical to that for the file; or
 - c. If the user does not include a password in the MCA macro call and the file is protected by a password.
4. When the password check is completed successfully, an exit is taken so the user can examine *VOLDESCR*. A return to the open function from this exit indicates either that the open function for this file is to continue or the file is rejected by the user and that an attempt to open a new file is to be made. If the open function for the original file is to continue and if the processing mode specified for the file in the MSOPEN macro call is either input/output or output-only, *VOLDESCR* is written back onto the mass storage device by Logical I/O C.
5. If the specified processing mode is input/output or input-only, Logical I/O C verifies (through the data status field in *VOLDESCR*) that the file volume contains data. If the file volume does not contain data, an exit is taken. The user can either issue a new macro call for a different function, or the user can return to the open function from the exit requesting that a new attempt to open be made (implying that a new volume has been mounted).
6. At this point, Logical I/O C checks the labeling information in *VOLDESCR* and moves all information required by other Logical I/O C functions from *VOLDESCR* into the file's communication area.
7. If the specified processing mode is output-only and if the item-handling mode is specified as locate, Logical I/O C moves the address of the left end of the first item location in the current buffer to the user-specified field defining the next location into which an item is to be placed.
8. At this point, Logical I/O C sets the following indicators in the file's communication area:
 - a. An indicator is set specifying whether or not this file volume is the last file volume of the file (this information is contained in the *VOLALLOC* entry for the file);
 - b. An indicator is set specifying that the file is opened; and
 - c. An indicator is set specifying the appropriate processing mode.

OPENING PARTITIONED SEQUENTIAL FILES

Partitioned sequential files are requested to be opened for processing by the MSOPEN macro call. Unlike opening a sequential file, the processing mode for the partitioned sequential file is not specified until a SETM macro call is issued. In executing the open function for partitioned sequential files, Logical I/O C performs the following steps.

1. Logical I/O C attempts to locate the file requested in the MSOPEN macro call in *VOLNAMES*. If the file name cannot be located, an exit is taken. A return to the open function from this exit implies that a new volume has been mounted and a new attempt to open the file is to be made.
2. After the specified file name is located in *VOLNAMES*, Logical I/O C locates the file's description in *VOLDESCR*. If password checking for the file is specified, Logical I/O C performs the password check as described for opening sequential files. If password checking is not specified, Logical I/O C verifies that the password field for the file contains all blanks. If either password check fails, an exit is taken as described for opening a sequential file.
3. When the password check is completed successfully, an exit to the user is taken so that the user can examine *VOLDESCR*. A return to the open function from this exit indicates either that the open function for this file is to continue or the file is rejected by the user and that an attempt to open a new file is to be made. If the open function for the original file is to continue and if the update action is specified in the MSOPEN macro call, Logical I/O C writes *VOLDESCR* back onto the mass storage device.
4. At this point, Logical I/O C checks the labeling information in *VOLDESCR* and moves all information required by other Logical I/O C functions from *VOLDESCR* into the file's communication area. Logical I/O C then sets an indicator in the file's communication area specifying that the file is opened.

OPENING AN INDEXED SEQUENTIAL FILE

Like sequential files, indexed sequential files are requested to be opened for processing in the mode specified in the MSOPEN macro call. In executing the open function for indexed sequential files, Logical I/O C performs the following steps.

1. Logical I/O C attempts to locate the file requested in the MSOPEN macro call in the name portion of the volume directory. If the file name cannot be located in *VOLNAMES*, an exit is taken. A return to the open function from this exit implies that a new volume has been mounted and a new attempt to open the specified file is to be made.
2. After the file name is located in *VOLNAMES*, Logical I/O C verifies that the file-volume sequence number is one greater than the sequence number of the last file volume (the first file-volume sequence number is zero). If the file-volume sequence number is not one greater than the last file-volume sequence number, an exit is taken. A return to the open function from this exit causes Logical I/O C to try again to open the file.
3. After the file-volume sequence number checking, Logical I/O C performs the password check as described for sequential files.

4. When the password check is successfully completed and if the volume contains the first file volume being opened (file-volume sequence number zero), an exit is taken so the user can examine the *VOLDESCR* entry for the file. The user can return to the open function from this exit specifying that this open function is to continue, or a return can be made specifying that a new attempt to open the file is to be made.
5. If the open function is to continue, Logical I/O C moves all pertinent data from the *VOLDESCR* entry for the file into the file's communication area and updates the *VOLDESCR* modification number and date.
6. If the processing mode specified is input/output, Logical I/O C writes *VOLDESCR* back onto the mass storage device.
7. Next, Logical I/O C processes the *VOLALLOC* entry for the file. In doing this, it moves the master/cylinder index and the general overflow entries from *VOLALLOC* into the file's communication area and the data units of allocation entries into the user provided units of allocation table.
8. Logical I/O C then sets an indicator in the file's communication area specifying the appropriate processing mode.
9. If processing is to be sequential from the beginning of the file and a prime data unit of allocation has been processed, Logical I/O C proceeds to step 10.

NOTE: If processing is not to be sequential from the beginning of the file, Logical I/O C returns to step 1 and repeats the process until all file volumes that contain data have been opened. After all file volumes have been opened, Logical I/O C proceeds to step 10.

10. Logical I/O C sets an indicator in the file's communication area specifying that the file has been opened in the appropriate processing mode.
11. If a resident cylinder index has been requested, Logical I/O C reads the specified number of blocks of the cylinder index into the specified area in main memory.
12. If the MSOPEN macro call specified that processing is to be sequential from the beginning of the file, Logical I/O C issues a SETL macro call for the beginning of the file.
13. Logical I/O C returns to the main line of the user's coding.

OPENING DIRECT ACCESS FILES

Like sequential and indexed sequential files, direct access files are requested to be opened for processing in the mode specified in the MSOPEN macro call. In executing the open function for direct access files, Logical I/O C performs the same first six steps as performed for opening an indexed sequential file; it then performs the following steps.

1. Logical I/O C processes the *VOLALLOC* entry for the file. In doing this, it checks each entry in *VOLALLOC* for the file and moves each entry to the next available position in the user-supplied units of allocation table. If Logical I/O C detects a discrepancy while processing *VOLALLOC*, an exit is taken. The only allowable return to the open function from this exit is to try again to open the file.

2. Logical I/O C then sets an indicator in the file's communication area specifying the appropriate processing mode.
3. If the file is being opened for sequential processing from its beginning, Logical I/O C sets processing to the beginning of data and returns to the user's main line coding.

NOTE: If the file is not being opened for sequential processing from its beginning, Logical I/O C returns to step 1 and repeats the process for each file volume until all file volumes have been opened. Logical I/O C then sets processing to the beginning of data and returns to the user's main line coding.

Closing Files

The process for closing sequential and partitioned sequential files is similar, and the process of closing indexed sequential and direct access files is identical.

CLOSING SEQUENTIAL AND PARTITIONED SEQUENTIAL FILES

Sequential and partitioned sequential files are requested to be closed by the MSCLOS macro call. In executing the close function for these files, Logical I/O C performs the following steps.

NOTE: Steps 1 through 4 apply only to sequential files.

1. If the file was processed in the output-only mode, Logical I/O C ensures that all buffers have been written onto the mass storage device and that the item following the last item written is an end-of-data item. (End-of-data items are signified by \square EOD \dagger in the first five locations of the item.)
2. If the file was processed in the input/output mode, Logical I/O C ensures that the current buffers have been written back onto the mass storage device if an MSREP has been issued for an item in these buffers.
3. If the file was processed in the input/output or output-only mode, Logical I/O C moves the current item count into the file's *VOLDESCR* item.
4. If the file was processed in the output-only mode, Logical I/O C sets the data status indicator to 00 in the last file volume that contains data.
5. Next, an exit is taken so the user can examine *VOLDESCR* for the file. A normal return from the exit to the close function causes Logical I/O C to write *VOLDESCR* back onto the mass storage device if the file was processed in the input/output or output-only mode (or update mode in partitioned sequential files).

CLOSING INDEXED SEQUENTIAL AND DIRECT ACCESS FILES

Indexed sequential and direct access files also are requested to be closed by the MSCLOS macro call. The following steps are performed by Logical I/O C when executing the close function for indexed sequential and direct access files.

1. If the file was processed in the input/output mode and an MSREP, MSINS, or MSDEL was issued for an item in the current buffers, Logical I/O C writes those buffers back onto the mass storage device.
2. If the current file volume was processed sequentially from its beginning, Logical I/O C updates the item count in *VOLDESCR* by adding the net change in items to the current file volume. If the file was not processed sequentially from its beginning, Logical I/O C updates the item count in *VOLDESCR* by adding the net change in items since the file was opened to the last volume in the file.
3. If the file was processed sequentially from its beginning, an exit is taken so the user can examine the current *VOLDESCR*. If the file was not processed sequentially from its beginning, an exit is taken so the user can examine the last *VOLDESCR* in the file. A normal return from the exit to the close function causes Logical I/O C to write *VOLDESCR* back onto the mass storage device if the file was processed in the input/output mode.

Retrieving Items in Files

The process of retrieving items in sequential and partitioned sequential files is similar. The process of getting items in indexed sequential files and in direct access files is significantly different.

RETRIEVING ITEMS IN SEQUENTIAL AND PARTITIONED SEQUENTIAL FILES

The request to retrieve an item in a sequential or partitioned sequential file is the issuance of the MSGET macro call. Logical I/O C can perform the get function for sequential and partitioned sequential files only when the file is being processed in either the input/output mode or in the input-only mode. The process is the same for both file types in either mode, except that when the file is processed in the input-only mode, step 2 does not apply. In executing the get function for sequential and partitioned sequential files, Logical I/O C performs the following steps.

NOTE: A data block is read into memory only after the first MSGET is issued.

1. If the next sequential item is in the current buffer, Logical I/O C begins to get the item at step 4; if not, it begins at step 2.
2. If an MSREP macro call has been issued for an item in the current buffer, Logical I/O C writes the current buffer back onto the mass storage device.
3. Depending on the buffering method (single or double), Logical I/O C causes the current buffer to contain the next sequential block from the mass storage device.
4. If the next sequential item is an end-of-data item and if an indicator in the file's communication area specifies that more data follows on another volume, Logical I/O C sets processing to the next file volume.
5. If the next sequential item is an end-of-data item and if the current file volume is the last one, an end-of-data exit to the user is taken to indicate this. When this exit is taken by Logical I/O C, the user cannot return to the get function. An MSCLOS (or ENDM if the file is partitioned) must be the next action macro call issued for this file.

6. If the next sequential item is not an end-of-data item and if "move item handling" has been specified, Logical I/O C delivers the item to the user-supplied work area. If "locate item delivery" has been specified, Logical I/O C delivers the address of the leftmost character of the item in the current buffer to the user-supplied address field. After either of these deliveries has been made, Logical I/O C returns to the user's main line coding, ensuring that the mass storage address of the item is available to the user.

RETRIEVING ITEMS IN INDEXED SEQUENTIAL FILES

The macro call that is used to request the retrieval of an item in an indexed sequential file is MSGET. Logical I/O C retrieves an item either sequentially or directly (randomly), as directed by the MSGET macro call. The item retrieved is delivered to the user-supplied item work area in the move-item-handling mode, and the address of the leftmost character of the item is delivered to the user-supplied address storage area in the locate-item-handling mode.

If the user requests key verification and the locate-item-handling mode is being used, Logical I/O C moves the item's key to the user-supplied key storage area so that, if the user issues an MSREP macro call for the item, Logical I/O C can verify the item's key before replacing the item in the string.

In executing the sequential get function for items in indexed sequential files, Logical I/O C performs the following steps.

NOTES: 1. To perform the sequential get function, the user must have previously issued a SETL macro call, or specified LIMVOL in the MSOPEN macro call.

2. Step 2, below, applies only to input/output processing.

1. If the next sequential item is in the current buffer, Logical I/O C delivers that item to the user and returns to the user's main line coding. If the file is being processed in the input/output mode and if the item-handling mode is locate, Logical I/O C moves the item's key to the user-supplied key storage area if the user requested key verification.
2. If the current buffer is no longer required and if an MSREP or an MSDEL has been issued for an item in that buffer, Logical I/O C writes the block back onto mass storage.
3. If the last item delivered to the user was the final item on the current file volume and if that file volume is not the last volume of the file, Logical I/O C activates the next sequential file volume.
4. If there is no more data on the current file volume or in the general overflow area, and if this is the last volume in the file, Logical I/O C exits to the user indicating that the end of data has been reached. The user cannot return to this get function when this exit is taken by Logical I/O C.
5. If more data remains on the current file volume, Logical I/O C reads the block containing the next required item into memory and delivers the next sequential item to the user as in step 1.

In executing the random get function for items in indexed sequential files, Logical I/O C performs the following steps.

NOTE: Step 1 applies only to input/output processing.

1. If an MSREP or an MSDEL has been issued for some previously retrieved item, Logical I/O C ensures that the block containing that item has been written back onto the mass storage device.
2. If the last action macro call issued for this file was an MSEEK, Logical I/O C compares the item key of the MSEEK with the item key of the MSGET. If they are the same, Logical I/O C clears the seek indicator and proceeds to step 8; otherwise, it proceeds to step 3.
3. If the cylinder index is resident, Logical I/O C omits steps 4 and 5.
4. Logical I/O C searches the master index for the first item whose key value is greater than or equal to the desired item's key.
5. Logical I/O C then searches the cylinder index block determined in step 4 for the first item whose key is greater than or equal to the desired item's key.
6. When there is a resident cylinder index, Logical I/O C searches it for the desired item's key. If the key falls outside the limits of the resident cylinder index, the master index is searched and the block containing the cylinder index for the desired item's key is read into the last block area of the resident cylinder index; then, the resident cylinder index is searched again for the desired item's key.
7. Logical I/O C seeks the volume and cylinder located by the above steps.
8. The string index is searched from its beginning for the item whose key is equal to or greater than the desired item's key.
9. The prime data area or the cylinder overflow area is searched for the desired item. Logical I/O C will continue the overflow search into the general overflow area if necessary.
10. The item is considered not located if:
 - a. An item with a key value greater than the desired item's key value is located in any area,
 - b. An item status character of 41 or 42 is detected in an item with an equal key value, or
 - c. The physical end of the general overflow area is detected.
11. When the desired item is located, Logical I/O C delivers it to the user in either of the two item-handling modes. If key verification was specified and the file was processed in the input/output mode using the locate-item-handling mode, Logical I/O C moves the item key into the user-supplied key storage area. After the item is delivered to the user, Logical I/O C returns to the user's main line coding ensuring that the current item's mass storage address is available to the user in the file's communication area.

RETRIEVING ITEMS IN DIRECT ACCESS FILES

To retrieve an item in a direct access file, the user can supply the bucket address and the item key, just the item key, just the bucket address, or neither the bucket address or item key. The macro call used to request the retrieval of an item in a direct access file is MSGET. Table 3-3 summarizes the MSGET macro functions for direct access files.

NOTE: When an item is retrieved in a direct access file and delivered to the user by Logical I/O C, Logical I/O C returns to the user's main line coding ensuring that the item's mass storage address is available to the user in the file's communication area.

In executing the get function when both the bucket address and the item key are supplied in the MSGET macro call, Logical I/O C performs the following steps:

NOTE: Step 1 applies only to input/output processing.

1. If an MSREP, MSDEL, or MSINS has been issued for some item previously retrieved or to be inserted, Logical I/O C ensures that the block containing that item has been written back onto the mass storage device.
2. Logical I/O C searches each undeleted item position in the specified bucket for the item with the specified key. When the cylinder overflow area is entered, Logical I/O C sets an indicator in the file's communication area specifying this. It sets another indicator when the general overflow area is entered.
3. When the end of the overflow area(s) is detected or when an inactive item position is encountered, the item is not located and an exit is taken. When Logical I/O C takes this exit, the user cannot return to this get function.
4. When the item is located, Logical I/O C delivers it to the user in either the move or locate-item-handling mode.

In executing the get function when only the item key is supplied, Logical I/O C performs the same steps as outlined when the bucket address and item key are specified, except that Logical I/O C begins searching for the item at the next sequential item location in the current bucket.

In executing the get function when only the bucket address is supplied, Logical I/O C performs the following steps.

NOTE: Step 1 applies only to input/output processing.

1. If an MSREP, MSDEL, or MSINS has been issued for some item previously retrieved or to be inserted, Logical I/O C ensures that the block containing that item has been written back onto the mass storage device.
2. Logical I/O C searches sequentially from the beginning of the specified bucket for the next undeleted item, without regard for its key.
3. The search is continued through the file in the following sequence.
 - a. The remaining buckets on the current cylinder are searched.
 - b. The current cylinder's overflow area is searched.
 - c. Steps a. and b. are repeated for all subsequent cylinders in the current unit of allocation.
 - d. The general overflow area of the current unit of allocation is searched.

- e. Steps a., b., c., and d. are repeated for all subsequent units of allocation on the current file volume.
 - f. When a file volume has been exhausted, Logical I/O C activates a new file volume for sequential processing as in steps a., b., c., and d.
4. If an undeleted item location cannot be located by Logical I/O C, an exit is taken.
 5. When an undeleted item location is located, Logical I/O C delivers the item and returns to the user's main line coding.

In executing the get function when neither the bucket address nor the item key is supplied by the user, Logical I/O C searches as it does when only the bucket address is supplied, but it begins searching for the next sequential undeleted item with the next sequential item in the current bucket.

Table 3-3. Summary of MSGET Macro Functions for Direct Access Files

	BUCKET ADDRESS SPECIFIED	YES	YES	NO	NO
	ITEM KEY SPECIFIED	YES	NO	YES	NO
FUNCTIONS	Start at beginning of this bucket	X	X		
	Start from current bucket position			X	X
	Search for specified item key	X		X	
	Search for next sequential active item		X		X
WHEN BUCKET IS EXHAUSTED	Continue search into overflow area(s)	X		X	
	Continue search into next contiguous area (bucket or overflow)		X		X

NOTE: An MSGET with only a key specified may be executed following an MSGET with a bucket and key specified or another MSGET with only a key specified.

Replacing Items in Files

The process of replacing items in files is similar for sequential and partitioned sequential files, but it is significantly different for indexed sequential and direct access files. In all cases in which an item is replaced, the replaced item is the item last retrieved by Logical I/O C through the MSGET function.

REPLACING ITEMS IN SEQUENTIAL AND PARTITIONED SEQUENTIAL FILES

An item can only be replaced when the file is processed in the input/output mode. The replace function is requested by the MSREP macro call. The following steps are performed by Logical I/O C when executing the replace function for items in sequential and partitioned sequential files.

1. Logical I/O C sets an indicator in the file's communication area specifying that an MSREP has been issued for the item last retrieved by the get function. This ensures that the current buffer is written back onto the mass storage device after it is exhausted but before it is overlaid by a new block.
2. If the move-item-handling mode is specified, Logical I/O C overlays the item in the current buffer to which the last MSGET referred with the item in the user-supplied item work area.

REPLACING ITEMS IN INDEXED SEQUENTIAL FILES

In an indexed sequential file, an item can be replaced only when the file is processed in the input/output mode. To execute the replace function for items in an indexed sequential file, Logical I/O C performs the following steps.

1. If key verification was requested, Logical I/O C ensures that the key of the last item retrieved is the same as that of the replacement item as follows:
 - a. In the move-item-handling mode, Logical I/O C compares the key of the item in the buffer with the key of the item in the user's item work area; and
 - b. In the locate-item-handling mode, Logical I/O C compares the key of the item in the buffer with the key in the user-supplied item key storage area.

If either of these checks fails, an exit to the user is taken. When Logical I/O C takes this exit, it does not expect a return to this replace function.

2. If processing of the file is in the move-item-handling mode, Logical I/O C moves the item from a user-supplied item storage area to its proper place in the current buffer.
3. Logical I/O C then sets an indicator in the file's communication area specifying that the current buffer must be written back onto the mass storage device before it is overlaid in memory. Logical I/O C then returns to the user's main line coding.

REPLACING ITEMS IN DIRECT ACCESS FILES

In a direct access file, items can only be replaced when the file is processed in the input/output mode. In executing the replace function for direct access files, Logical I/O C performs the following steps.

1. Logical I/O C sets an indicator in the file's communication area that an MSREP macro call has been issued for an item in the current block. This ensures that the block will be written back onto the mass storage device before it is overlaid in memory.

2. If the move-item-handling mode is specified, Logical I/O C overlays the item in the current buffer to which the last MSGET referred with the item in the user-supplied item work area.

Putting Items to Sequential and Partitioned Sequential Files

The put function is initiated by the MSPUT macro call and can be used only when a sequential file is being processed in the output-only mode. Note that when processing is in the locate-item delivery mode, the MSOPEN or SETM macro call issued previously causes an initial item delivery address to be placed in the user's linkage address field.

To perform the put function, Logical I/O C performs the following steps.

1. If the file is being processed in the move-item delivery mode, the user's item is moved into the current buffer.
2. Logical I/O C then determines if there is room in the current buffer for another item. If there is no room, Logical I/O C skips to step 4.
3. If the file is being processed in the locate-item delivery mode, the address of the leftmost location of the next available item position is moved to the user's address field. At this point, Logical I/O C returns to the user's main line coding.
4. Logical I/O C determines if there is room in the file or the current member for another block after the current block. If there is no room, an exit to the user is made. Logical I/O C does not expect a return to the put function if this exit is taken. The last item for which the user issued an MSPUT macro call is overlaid by an end-of-data item when an MSCLOS or ENDM macro call is issued. (The last item for which the user issued an MSPUT macro call will still exist in the user's work area.) The next action issued for this file or member must be an MSCLOS or an ENDM macro call.
5. If there is room for another block in the file or member after the current block, the current block is written onto the mass storage device, pointers are set to the new current buffer, and Logical I/O C returns to step 3.

Action Macro Calls (for Partitioned Sequential Files Only)

SET PROCESSING TO BEGINNING OF SPECIFIED MEMBER (SETM)

The set member function is initiated by the SETM macro call. When this action is performed, processing is set to the beginning of the member specified in the SETM macro call in the processing mode designated.

When performing the set member function for a member that is to be processed in the input-only mode, Logical I/O C performs the following steps.

1. Logical I/O C attempts to locate a member index entry for an undeleted member whose name is the same as that specified. If no such name exists, an exit to the user is made to indicate this condition. Logical I/O C does not expect a return to the SETM macro routine if this exit is taken; however, a new action can follow if this exit is taken.

2. When the member index entry for the specified member is located, Logical I/O C sets the address of the member's first item in the file's communication area.
3. A processing mode indicator is then set in the file's communication area, and Logical I/O C returns to the user's main line coding.

When the member is to be processed in the input/output mode, Logical I/O C performs the set member function as described for the input-only mode, except that the processing mode indicator is set to input/output processing in the file's communication area.

When the member is to be processed in the output-only mode, Logical I/O C performs the following steps in executing the set member function.

1. Logical I/O C attempts to locate a member index entry for an undeleted member whose name is the same as that specified. If this entry is found, Logical I/O C skips to step 5.
2. If Logical I/O C cannot locate a member index entry for an undeleted member whose name is the same as that specified, it verifies that there is room in the member index and that there is data space for another entry. If room is not available in either place, exits to the user are available. If either exit is taken, the user cannot return to this set member function, but he can issue a new action macro call.
3. If room exists for another entry, an indicator is set in the file's communication area to signify that a new member is being created.
4. Logical I/O C then sets the address of the first item of the unused area into the file's communication area and performs step 8.
5. Logical I/O C checks the status of the member to see that it is available for output-only processing. If the member is unavailable for output-only processing, Logical I/O C exits to the user, allowing no return to this set member function but allowing a new SETM or an MSCLOS macro call to be issued.
6. If processing is in the locate item delivery mode, the address of the leftmost end of the buffer area (into which the user's first item is to be placed) is set into the user's address field.
7. The mass storage address of the member's first item is moved into the file's communication area.
8. The processing mode indicator in the file's communication area is set to indicate output-only processing, and Logical I/O C returns to the user's main line coding.

END PROCESSING OF CURRENT MEMBER (ENDM)

The end-member processing function is initiated by the ENDM macro call. When performing the end-member processing function, Logical I/O C performs the series of steps listed below.

1. If the current member was processed in the input/output or output-only mode, Logical I/O C ensures that all buffers have been retranscribed

to the mass storage device and, in the case of output-only processing, that an end-of-data item has been generated.

2. If the current member was processed in the output-only mode and if it was just created (i. e. , it is a new member of the file), Logical I/O C generates a member index entry for the new member and decreases the length of the unused area entry for the new member index. If necessary, Logical I/O C also generates a new "end-of-data in the member index" entry for the file.
3. Logical I/O C sets an indicator in the file's communication area, indicating that no member is open for processing, and returns to the user's main line coding.

ALTER STATUS OF MEMBER (MALTER)

The alter member function is initiated by the MALTER macro call to change the status of a member to: (1) deleted, (2) make the member available for output-only processing, (3) make the member unavailable for output-only processing, or (4) rename the member. Note that a change in member status, a change in member name, or both actions can be specified. When the alter member function is performed, Logical I/O C performs the following series of steps.

1. Logical I/O C attempts to locate the specified member name in the member index. Deleted entries in the member index are not examined. If the member name cannot be located in the member index, Logical I/O C exits to the user, indicating the reason for the exit. Logical I/O C does not expect a return to this alter member function if this exit is taken.
2. After locating the member name in the member index, Logical I/O C alters the member as specified by the user.
 - a. The member's status is changed to "available" for output-only processing.
 - b. The member's status is changed to "unavailable" for output-only processing.
 - c. The member's status is changed to "deleted" after Logical I/O C verifies that the member is available for output-only processing. If the member is not available for output-only processing, Logical I/O C exits to the user to indicate this condition and does not change the status of the member to "deleted."
 - d. The member's name is changed to the new name specified in the MALTER macro call.

RELEASE COMPLETE FILE TO UNUSED STATE (MSREL)

The release function is initiated by the MSREL macro call. When this function is performed, the specified partitioned sequential file is released so that no members exist and the complete data area of the file is available for reuse. Note that the file must be opened before it can be released and that, when the release function is performed, verification of the availability status of currently active members is not made. In performing the release function, Logical I/O C performs the following steps.

1. The end-of-index entry in the member index is moved to the second position in the index.
2. The unused area entry in the member index (the first entry in the index) is set to point to the first data block in the file.

Inserting Items in Files

Items can be inserted in indexed sequential and direct access files. The insert function is requested by the MSINS macro call.

INSERTING ITEMS IN INDEXED SEQUENTIAL FILES

NOTE: MSINS can only be used in the input/output mode.

To insert an item in an indexed sequential file, Logical I/O C performs the following steps.

1. If the MSREP or an MSDEL has been issued for some previously retrieved item, Logical I/O C ensures that the block containing that item has been written back onto the mass storage device. If an MSEEK was the last action issued for this file, Logical I/O C verifies that the item key for this MSINS is the same as that for the prior MSEEK and clears the Seek indicator.
2. Using the master, cylinder, and string indexes, Logical I/O C searches for the item position into which the user's item should be inserted. (If part of the cylinder index is resident in memory and the item key does not fall within the value range of the resident portion, the appropriate block of the cylinder index area for the user's item is then read into the last block position of the resident cylinder index area.) If an item position cannot be found, an exit to the user is available to indicate this. If Logical I/O C takes this exit, it does not expect a return to this insert function.
3. If the located item position contains an active item whose key is equal to the key of the user's item, Logical I/O C considers the user's key to be a duplicate item. If this happens, an exit is available to indicate this. If Logical I/O C takes this exit, it does not expect a return to this insert function.
4. Logical I/O C inserts the user's item into the located item position, saving the item originally at that position if necessary.
5. Logical I/O C displaces items as necessary to find an inactive or deleted item position to allow for the extra item in the file. This process of displacement may progress from the prime data area to the cylinder overflow area and then to the general overflow area. All items displaced are maintained in the proper ascending order of item key. If the cylinder or general overflow area is used, Logical I/O C sets indicators in the file's communication area to specify this for later interrogation by the user. If the high key value of a prime data area string is altered, Logical I/O C updates the string index item of the string accordingly.
6. If the general overflow area is exhausted during the displacement process, an exit is available to the user. The user's item has been inserted into the file, but the last item in the general overflow area has been displaced out of the file. The displaced item is available to the user at the time of the exit. If Logical I/O C takes this exit, it does not expect a return to this insert function.

7. An option is available providing two special exits to the user during execution of the MSINS macro routine for an indexed sequential file. If selected, these exits occur while processing each data block affected by the insert. The first exit occurs prior to inserting or moving items in the block, and the second occurs prior to writing the block back onto mass storage after inserting and moving of items has been done.

INSERTING ITEMS IN DIRECT ACCESS FILES

The insert function is initiated by the MSINS macro call and can only be used with direct access files being processed in the input/output mode.

- NOTES:
1. Regardless of the parameters of the MSINS macro instruction, searching for an available item position is always from the bucket to the cylinder overflow area (if any) to the general overflow area (if any). When an item position is not available, Logical I/O C exits to the user to indicate this condition. A return to this insert function is not anticipated by Logical I/O C if this exit is taken.
 2. When an insert operation is performed, an item is always moved from the user's work area into a buffer regardless of the specified item-handling mode.
 3. Duplicate item key checking is not done by the insert function because the programmer may check for duplicate by issuing an MSGET macro call before each MSINS macro call.

When the programmer specifies the bucket into which he wants to insert an item, Logical I/O C performs the following steps.

1. If an MSREP, MSDEL, or MSINS has been issued for some item previously retrieved or to be inserted, Logical I/O C ensures that the block containing that item has been written back onto the mass storage device.
2. Starting at the beginning of the specified bucket, Logical I/O C searches for the first available item position. When the cylinder overflow area is entered, an indicator is set in the file's communication area. Another indicator is set when the general overflow area is entered.
3. When the first available item position is located, Logical I/O C places the item into it.

When the programmer does not specify the bucket into which he wants to place an item, Logical I/O C starts searching for an available item position, beginning with the current item position in the bucket, and the search is continued as in steps 2 and 3 above.

Deleting Items from Files

Items can be deleted from indexed sequential and direct access files. The delete function is requested by the MSDEL macro call.

To delete items from indexed sequential or direct access files, Logical I/O C sets to "deleted" the status character of the item to which the last MSGET macro call referred. It also sets an indicator in the file's communication area specifying that the current block is to be written back onto the mass storage device.

Seeking a Desired Cylinder

In processing a direct access or indexed sequential file, the read/write heads of a disk device can be positioned to the cylinder containing desired data by use of the MSEEK macro routine. A subsequent data transfer to or from that cylinder can be performed after other processing has occurred while the read/write heads were moving into position. Cylinder-to-cylinder access time on one volume can thus be overlapped, for example, with data transfer activities on other volumes.

The programmer supplies the bucket or item key in the MSEEK macro call, and Logical I/O C initiates a seek for the appropriate cylinder on the appropriate volume.

Setting Processing to a Specified Location

The set processing function sets processing to the first item position in an indexed sequential file whose key value is equal to or greater than a key value supplied by the user. This function is requested by the SETL macro call. In executing the set processing function, Logical I/O C performs the following steps.

1. If a delete or a replace function was performed for some previously retrieved item, Logical I/O C ensures that the block containing the item has been written back onto the mass storage device.
2. Logical I/O C then searches (as described for the random get function for items in indexed sequential files) for the first item position equal to or greater than that specified by the user. If Logical I/O C cannot locate an item position, an exit is taken. When Logical I/O C takes this exit, it does not expect a return to this SETL function.
3. Logical I/O C then sets indicators, addresses, and key values in the file's communication area to refer to the next item in each data area.
4. When this is completed, Logical I/O C returns to the user's main line coding.

PROGRAM ORGANIZATION

As stated earlier in this section, the MIOC macro routine is a segmentable series of sub-routines (see page 3-2). All the routines that make up MIOC can be resident in memory at the same time, or MIOC can be segmented so that certain infrequently used routines remain on mass storage until required. When segmentation is specified by the programmer, assignment of segment names and segment loading is handled automatically by MIOC.

NOTE: A program which calls a segmented MIOC must have a segment name specified by a SEG statement.

When requested, the following routines of MIOC are resident at all times and occupy a non-overlayable portion of memory:

1. Common coding,
2. Physical I/O control,
3. Get,
4. Replace,
5. Delete, and
6. Put.

When the programmer specifies that MIOC is to be segmented, designation of the insert routine as resident or nonresident is arbitrary, but the following routines are nonresident when requested. These are brought into a common overlay area in memory (defined by MIOC) as each is required. The common overlay area defined by MIOC is reserved on the basis of the memory requirement for the largest nonresident routine:

1. Open,
2. Set member,
3. End member,
4. Close,
5. Alter member,
6. Release, and
7. Set location.

NOTE: When segmentation is specified, the open function normally is brought into the common overlay area in several segments. These are loaded one after another as the function progresses. The programmer, however, can request that the open function be brought into the common overlay area as a single segment.

LANGUAGE ELEMENTS FOR LOGICAL I/O C

Each of the macro routines described previously can be included in an EasyCoder source-language program simply by writing a macro call for each desired routine. At assembly time, these macro routines for which macro calls are included in the source-language program are specialized and included in the machine-executable output of the assembler. The macro routines are specialized, i. e., they are converted from general to special routines which are tailored to the particular use desired. This is accomplished by means of parameters included in or excluded from the macro call. The specialized macro routines are included in the program at the point at

which the macro call was written. The macro calls described later in this paragraph constitute the language elements of Logical I/O C.

The type, location, operation code, and operands fields of the EasyCoder coding form are significant when writing a macro call. A macro call is identified as such by the contents of the type field (column 6 on the coding form). Whenever a single macro call is written on more than one line of the coding form, all lines used for that call must contain the letter C in the type field, except that the last line used must contain the letter L. In the case in which a single macro call is contained entirely within one line, that line must contain the letter L in the type field. The letter C in the type field signifies that a given line (even if it is the first line) of a macro call is a continuation line. The letter L in the type field signifies that the line (even if it is the only line) is the last line of a macro call.

The name of the macro routine that performs the desired function is written in the operation code field (columns 15 through 20 on the coding form). This name must be written on the first (or only) line of the macro call. The macro routines are stored in the library file according to their names, and the name written in the operation code field is used to locate the macro routine in the library. No line, except the first line of a macro call, may contain the macro routine name in the operation code field.

The location field (columns 8 through 14 on the coding form) on the first line of the macro call contains parameter 0 of that macro routine. The value of this parameter is defined by each macro call. Note that parameter 0 of any macro call is the only parameter that need not be followed by a comma.

The remaining parameter values for a given macro call are written in the operands field (columns 21 through 62 on the coding form). A macro call may contain as many as 63 parameters. All parameters except 0, i. e., parameters 1 through 63, are written in numeric order in the operands field, starting at column 21. A parameter value may be up to 40 characters in length and may be composed of any set of characters except the comma. The comma is used to terminate each parameter and, therefore, cannot be a character in a parameter value. Each parameter written in the operands field must be terminated by a comma. If a parameter is to be omitted from a macro call, its terminating comma must follow the terminating comma of the preceding parameter. To illustrate this, the macro call for the get function (when used with direct access files) can contain as many as three parameters in the operands field. That is, the programmer can specify a file tag and a bucket address, a file tag and an item key, or both bucket address and item key as well as the file tag. If the programmer intends to specify only the file tag and the item key, he omits the bucket address (parameter 2) and includes its terminating comma, as shown in Figure 3-1.

CARD NUMBER	Y	T	R	LOCATION	OPERATION CODE	OPERANDS	
						14 15	20 21
1							
2							
3							
4							
5			L		MSGET	F1,,IKEY,	
6							
7							
8							
9							
10							

Figure 3-1. Omission of Single Parameter from Macro Call

In Figure 3-1, the name of the macro routine which performs the get function is MSGET and is written in the operation code field. Because this macro call only occupies one line on the coding form, the type field contains an L, indicating that this is the last line of this macro call. Parameter 0 of this macro call, normally written in the location field, is omitted. Parameter 1 has the value F1 and identifies the file to which this get function is directed. Parameter 2, the bucket address, is omitted, but its terminating comma is included. Parameter 3 has the value IKEY which points to the location at which the identification field of the item is located. Note that if the terminating comma of parameter 2 were omitted, the assembler would interpret the item key as being the bucket address.

When the macro call contains several parameters, as in the case of the MIOC macro call, another method of omitting parameters can be used in conjunction with the method just described. To illustrate this, the MIOC macro routine does not presently use parameters 5 through 9. The programmer can write the macro call for MIOC as shown in Figure 3-2.

CARD NUMBER	Y	T	R	LOCATION	OPERATION CODE	OPERANDS	
						14 15	20 21
1							
2			C	MINE	MIOC	\$,1,,1,	
3			L		1,0	A,	
4							

Figure 3-2. Omission of Consecutive Parameters from Macro Call

In Figure 3-2, parameter 0 contains the value MINE which, at assembly time, is equated with the lowest memory location that this MIOC routine occupies. Parameter 1 has the value \$ which identifies this unique specialization of the MIOC routine. Parameter 2 has the value 1, signifying that the program that contains this specialized MIOC routine will process sequential files in the input/output mode. Parameter 3 is omitted, but its terminating comma is included. The omission of parameter 3 actually indicates that this specialized MIOC routine does not process partitioned sequential files. Parameter 4 has the value 1, indicating that this specialization

of the MIOC routine will process direct access files in the input/output mode. Notice that parameter 4 is followed by a comma and that parameters 5 through 9 are omitted. These are omitted by writing on the next line of the coding form (in the first two columns of the operation code field) the number of the next parameter to be included and the value of that parameter, starting in the first column of the operands field. Only a consecutive series of parameters can be omitted in this manner. The programmer continues writing parameters consecutively in the operands field, using as many additional lines as required, until all of the parameters of the macro call have been accounted for. Notice in Figure 3-2 that the first line of the macro call contains a C in the type field and that the last line contains an L in this same field.

Input/Output Control Macro Routine (MIOC)

One MIOC macro call is required for each program that uses the facilities of Logical I/O C. When the programmer specializes more than one MIOC in a single program, each MIOC must originate at the same memory location. By including more than one MIOC in a single program, various file requirements can be handled. However, only one MIOC can exist in memory at any given time. The method of achieving uniqueness between tags when more than one specialization of MIOC is used in a single program is explained in Table 3-4 in the description of MIOC parameter 1.

MIOC MACRO CALL

The following example illustrates the method of coding the MIOC macro call.

CARD NUMBER	T	M	LOCATION	OPERATION CODE	OPERANDS																								
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
			L	anytag	MIOC	parameter 01, ..., parameter n,																							

Table 3-4. Parameters of MIOC Macro Call

Number	Name	Value	Function	Comments														
00	Base	Anytag	Tag is equated with the lowest memory location occupied by the MIOC macro routine.	Optional.														
01	Unique character	See "Comments" column for valid characters; see also Note 5.	A single character incorporated in each tag used by this specialization of MIOC. Used to achieve tag uniqueness when more than one specialization of MIOC is being used in a single program and to ensure that a user tag does not duplicate any tag in MIOC.	Required for each MIOC macro call. Valid characters are shown below. <table border="1" data-bbox="1506 576 1910 820"> <thead> <tr> <th>Key Punch</th> <th>Print Symbol</th> </tr> </thead> <tbody> <tr> <td>(+, 8, 5)</td> <td>%</td> </tr> <tr> <td>(+, 8, 6)</td> <td>■</td> </tr> <tr> <td>(-, 8, 3)</td> <td>\$</td> </tr> <tr> <td>(-, 8, 5)</td> <td>"</td> </tr> <tr> <td>(0, 1)</td> <td>/</td> </tr> <tr> <td>(0, 8, 5)</td> <td>C_R</td> </tr> </tbody> </table>	Key Punch	Print Symbol	(+, 8, 5)	%	(+, 8, 6)	■	(-, 8, 3)	\$	(-, 8, 5)	"	(0, 1)	/	(0, 8, 5)	C _R
Key Punch	Print Symbol																	
(+, 8, 5)	%																	
(+, 8, 6)	■																	
(-, 8, 3)	\$																	
(-, 8, 5)	"																	
(0, 1)	/																	
(0, 8, 5)	C _R																	
02	Sequential file functions	Δ	Sequential files will not be processed by this specialization of MIOC.	Coding pertaining only to sequential files is eliminated.														
		1	Input/output or both input-only and input/output processing.															
		2	Output-only processing.	Code indicates type of sequential file (including partitioned sequential) processing. When the parameter is not omitted, one of these codes must be used.														
		3	Input-only processing.															
		4	Input/output and output-only or all three types of processing.															
		5	Input-only and output-only processing.															
03	Sequential file options	Δ	The sequential files to be processed are not partitioned.															
		PARTITION	At least one of the sequential files to be processed is partitioned.	Cannot be used when parameter 02 is blank.														

Table 3-4 (cont). Parameters of MIOC Marco Call

Number	Name	Value	Function	Comments
04	Direct access functions	Δ	Direct access files will not be processed by this specialization of the MIOC macro routine.	Coding pertaining only to direct access files is eliminated.
		1 (see note 2)	Input/output or both input-only and input/output processing.	Code indicates the type of direct access file processing. When the parameter is not omitted, one of these codes must be used.
		2	Input-only processing.	
05	Direct access processing mode	Δ or BOTH	Both sequential and random processing of direct access files is required of this MIOC.	This parameter is valid only if parameter 04 is not blank.
		SEQUENTIAL	This specialization of MIOC only requires sequential processing of direct access files.	The MSINS macro call cannot be used and the MSGET macro calls used cannot contain bucket address or key values.
		RANDOM	This specialization of MIOC only requires direct processing of direct access files.	The MSINS macro call can be used and the MSGET macro calls must specify at least a key value.
06	Indexed sequential file functions	Δ	Indexed sequential files are not processed by this specialization of this MIOC.	Coding pertaining to indexed sequential files is eliminated.
		1 (see note 2)	Input/output or both input/output and input-only.	Code indicates the type of indexed sequential file processing. When the parameter is not omitted, one of these must be used.
		2	Input-only processing.	
07	Indexed sequential processing mode	Δ or BOTH	Both sequential and random processing of indexed sequential files is required of this MIOC.	This parameter is valid only if parameter 06 is not blank.
		SEQUENTIAL	This specialization of MIOC only requires sequential processing of indexed sequential files.	The MSINS macro call cannot be used and the MSGET macro calls used cannot contain key values.
		RANDOM	This specialization of MIOC only requires direct processing of indexed sequential files.	The MSINS macro call can be used and the MSGET macro calls must contain a key value.

3-28

#5-618

Table 3-4 (cont). Parameters of MIOC Macro Call

Number	Name	Value	Function	Comments
08	Residence of cylinder index	Δ	Cylinder index for an indexed sequential file is not to be resident in main memory.	Must be blank if parameter 06 (above) is blank.
		RESIDENT	Cylinder index is to be at least partially resident.	
09	Seek indicator	Δ	Indicates the MSEEK action macro routine will not be called.	Must be blank if both parameters 04 and 06 (above) are blank.
		1	Specifies that the MSEEK action macro routine will be used only on a direct access file.	
		2	Specifies that the MSEEK action macro routine will be used only on an indexed sequential file.	
		3	Specifies that the MSEEK action macro routine may be used on both direct access and indexed sequential files.	
10	Segmentation (see note 3)	Δ	In this specialization, MIOC will not be segmented.	Any letter (A-Z) can be used. This letter is assigned as the first character of each segment generated by MIOC. MIOC assigns the second character.
		X	In this specialization, MIOC will be segmented.	
11	Insert coding (see note 2)	Δ	The direct access and indexed sequential files processed by this specialization of MIOC do not require the insert function coding.	Must be blank when parameter 04 and 06 are blank or 2.
		RESIDENT	Insert coding for the direct access and indexed sequential files processed by the MIOC will be resident.	Insert coding will be resident when parameter 10 is blank.
		SEGMENT	Insert coding will be nonresident for this specialization of MIOC.	Insert coding will occupy the common overlay area, when applicable.

Table 3-4 (cont). Parameters of MIOC Macro Call

Number	Name	Value	Function	Comments
12	SETM-ENDM overlay structure	Δ	If segmentation is specified, the SETM and ENDM macro coding is segmented so that each routine is a separate overlay.	Parameter 12 must be blank if parameter 3 is blank.
		COMBINE	The SETM and ENDM macro routines are brought into the common overlay area together.	
13	Direct access bucket addressing	Δ or RELATIVE	Direct access bucket addresses are relative.	Addresses are specified in binary.
		ACTUAL	Direct access bucket addresses are actual.	
		BOTH	Both relative and actual direct access bucket addresses are used.	
14	Multiple MIOC indicator	Δ	This program has only this one specialization of MIOC.	If the program has more than one MIOC specialization, each specialization must have its own unique character and each must originate at the same memory location.
		MULTIMIOCS	This program has more than this one specialization of MIOC.	
15	Open Segmentation	Δ	The open macro routine will be segmented in a way that will require the least amount of memory.	This parameter has no significance if parameter 10 is blank (Δ).
		COMBINE	The open macro routine will be a single continuous segment in the common overlay area.	
16	Alter member coding requirements	Δ	The coding for the alter member function is required in this MIOC.	This parameter has no significance when parameter 03 is blank.
		NOMALTER	This MIOC does not require the coding for the alter member function.	
17	Release member coding requirements	Δ	The coding for the release member function is required by this MIOC.	This parameter has no significance when parameter 03 is blank.
		NORELEASE	This MIOC does not require the coding for the release member function.	

3-30

#5-618

Table 3-4 (cont). Parameters of MIOC Macro Call

Number	Name	Value	Function	Comments
18	Multiple volume coding requirements	Δ or SINGLEVOL	This specialization of MIOC will not process multivolume files.	If at least one file processed by this MIOC is a multivolume file, MULTIVOL must be specified.
		MULTIVOL	This specialization of MIOC will process multivolume files.	
19 through 24		Not Applicable		These parameters are reserved for the use of the operating system.
25	Key verification requirements	Δ	Key verification is not used on indexed sequential files being processed by this MIOC.	The use of the key verification option is recommended when items are to be replaced in indexed sequential files.
		KEYVER	Key verification is to be made on items being replaced in indexed sequential files being processed by this MIOC.	
26 through 28		Not Applicable		These parameters are reserved for the use of the operating system.
29	Address mode	3 or Δ	MIOC will be assembled in 3-character address mode.	The 4-character address mode Supervisor must be used at execution time.
		4	MIOC will be assembled in 4-character address mode.	
30	Operator control file device type	Δ	A control panel is used by the operator control file.	
		220	A Type 220 Console keyboard/type-writer is used by the operator control file.	
31		Not Applicable		This parameter is reserved for the use of the operating system.

3-31

#5-618

Table 3-4 (cont). Parameters of MIOC Macro Call

Number	Name	Value	Function	Comments
32	Buffer modes	Δ or DOUBLE	File processing functions will utilize two buffers.	See note 1 at end of table. Only one buffer is used in processing indexed sequential files.
		SINGLE	File processing functions will utilize one buffer.	
		BOTH	Both single and double buffering is required for files processed by this MIOC.	
33	Item-handling mode	Δ or LOCATE	Items are to remain in input buffers for user processing, and the user will place items in output buffers.	Locate mode is more efficient when items are not usually to be moved from one area of memory to another but will merely be interrogated and/or updated.
		MOVE	Items are to be moved to or from the input/output buffers from or to a user-supplied item work area.	
		BOTH	Files processed by this MIOC require both the locate- and the move-item-handling modes.	
34 through 45		Not Applicable		These parameters are reserved for use of the operating system.
46	Special insert exits	Δ	There are no special exits (as defined below) taken during an insert to an indexed sequential file.	The data exit (parameter 43 of MCA) is used for these exits.
		SPEC-EXITS	During an insert to an indexed sequential file, the MSINS action macro routine takes two exits to the user while processing each data block. The first exit occurs prior to inserting or moving items within a block. The second exits occurs prior to writing a block to mass storage after the inserting and moving of items has been done.	

3-32

#5-618

Table 3-4 (cont). Parameters of MIOC Macro Call

Number	Name	Value	Function	Comments
47 through 49		Not Applicable		These parameters are reserved for use of the operating system.
50	Physical I/O C requirements	Δ or CALL	This MIOC will call Physical I/O C for specialization on the basis of parameters 51 through 56.	A Physical I/O C macro routine (MPIOC) that this MIOC can use must exist in the program.
		PRESENT	The user has called the appropriate MPIOC macro routine, and the parameter values used to specialize it are the same as the values of parameters 51 through 56.	
51 through 56	Physical I/O C parameter set	(see below)	These values are used when this MIOC macro routine calls MPIOC. They are the same as parameters 01 through 06 of MPIOC.	The user is required to specify either the parameter values he has used in his MPIOC macro call (when parameter 50 = PRESENT), or the parameter values he wants MIOC to use (when parameter 50 = Δ or CALL).
51	Suffix	See parameter 01 for valid characters; see also Note 5.	A unique suffix for all tags in MPIOC macro routine.	Required. May be the same as parameter 01 of any MIOC in the program. Must be the same as parameter 01 of the MPIOC that will be in memory when this MIOC is in memory.
52	Peripheral address assignment	Δ	Honeywell-recommended peripheral address assignment for the mass storage control (04 octal).	
		xx (octal)	Peripheral address assignment to which the mass storage control applicable to this MIOC is attached.	
53	Write verification	Δ	Automatic verification coding is not included.	
		V	Write verification is to be done on some file being processed by this MIOC.	

Table 3-4 (cont). Parameters of MIOC Macro Call

Number	Name	Value	Function	Comments
54	Control of more than one PCU	Δ	PCU number and R/W channel will be specialized at assembly time using parameter 52 and may not be changed without reassembling.	The value of PCU contained in a communications area (MCA) is ignored.
		M	PCU number and R/W channel will be specialized from the current communications area (MCA) at execution time.	The value for PCU in MIOC parameter 52 is ignored. This value allows one MIOC to use more than one PCU.
55	RWC definition	Δ	Read/write channel automatically specialized at assembly time depending on parameter 52. (When parameter 52 is blank or ≤ 7, an octal 56 is generated. When parameter 52 >7, an octal 76 is generated.)	This parameter is meaningless if parameter 54 = M.
		xx (octal)	Specifies read/write channel configuration to be used for all data transfers. Cannot be changed without reassembly. Must correspond to PCU sector specified by parameter 52 and must include read/write channel 3 of that sector. See note 4, below, for permissible character values.	Must correspond to PCU sector specified by parameter 52 and must include read/write channel 3 of that sector. (Correspondence is to actual <u>sector</u> and not to value of <u>sector bits</u> , which are coded differently in RWC and PCU variants.)
56	MSEEK indicator	Δ	The MSEEK action macro routine cannot be called by this specialization of MIOC.	
		SEEK	The MSEEK action macro routine may be called for an indexed sequential or direct access file.	
<p>NOTES: 1. A buffer is a user-defined area which Logical I/O C uses for reading and writing blocks onto mass storage. The user specification of two buffers for use by a single file sometimes increases the efficiency of file processing routines. This increase in efficiency is dependent on the size of blocks, the amount of processing that will take place for each item in the block, the hardware characteristics which either allow or disallow a transfer of more than one block per disk revolution, and the type of I/O function which normally is requested for this file.</p>				

3-34

#5-618

SECTION III. LOGICAL I/O C

Table 3-4 (cont). Parameters of MIOC Macro Call

NOTES: When processing several files simultaneously, the user should specify separate buffers for each file.
(cont)

2. If inserting is required, parameter 11 cannot be blank.
3. If MIOC is segmented, the program within which it is called must specify a segment name.
4. Permissible octal values for parameter 55 are follows: 53, 54, 55, 56, (I/O sector 1); and 73, 74, 75, and 76 (I/O sector 2). Selection of a value depends primarily on the options available with the user's equipment configuration. For example, 53 and 73 are possible only for a type 259A Disk Pack Drive, and 54 and 74 assign a channel capacity greater than necessary. The usual values are 56 and 76. Reference should be made to the table in the user's Series 200 Programmers' Reference Manual for PDT I/O Control Character C1.
5. When specializing the MIOC macro instruction to function in a program that also contains IOCC (I/O Combination C) routines, the programmer must not use the single character \$ (i. e., keypunch -, 8, 3) as the value for parameter 01 or 51.

8/29/69

3-35

#5-618

PARAMETERS OF MIOC MACRO CALL

Table 3-4 lists the parameters of the MIOC macro call. Note that the function of most MIOC parameters is to insert into or eliminate from MIOC certain subroutines. Thus, a given specialization of MIOC makes it as small as possible. A summary of the parameters of the MIOC macro call is provided in Table 3-5.

Table 3-5. Summary of MIOC Parameter Values

Number	Name	Value	Function
00	Base	Anytag	Equated with MIOC lowest memory location.
01	Unique character	(print characters) %, □, \$, ", /, or C _R	Tag which uniquely identifies MIOC. Must be specified. (See Table 3-4, Note 5.)
02	Sequential file functions	Δ, 1, 2, 3, 4, or 5	Specifies how sequential files are to be processed. When left blank, no sequential files are to be processed.
03	Sequential file options	Δ or PARTITION	Specifies whether sequential files are partitioned or not. When left blank, partitioning option is not used.
04	Direct access file functions	Δ, 1, or 2	Specifies how direct access files are to be processed. When left blank no direct access files are to be processed.
05	Direct access processing mode	Δ, BOTH, SEQUENTIAL, or RANDOM	Specifies the processing mode for direct access files. When left blank, the file can be processed either directly or sequentially.
06	Indexed sequential file functions	Δ, 1, or 2	Specifies how indexed sequential files are to be processed. When left blank, indexed sequential files are not processed.
07	Indexed sequential processing modes	Δ, BOTH, SEQUENTIAL, or RANDOM	Specifies the processing mode for indexed sequential files. When left blank, the file can be processed in either mode.
08	Residence of cylinder index	Δ or RESIDENT	Specifies whether blocks of the cylinder index for an indexed sequential file are to be resident in main memory.
09	Seek indicator	Δ, 1, 2, or 3	Specifies whether MSEEK action macro routine is to be called or not, and whether for direct access and/or indexed sequential files.
10	Segmentation	Δ, or A through Z	Specifies whether or not MIOC will be segmented. When left blank, segmentation is not to be used.

SECTION III. LOGICAL I/O C

Table 3-5 (cont). Summary of MIOC Parameter Values

Number	Name	Value	Function
11	Insert coding	Δ, RESIDENT, or SEGMENT	Specifies whether or not insert macro coding is resident. When left blank there will be no insert coding.
12	SETM-ENDM overlay structure	Δ or COMBINE	Specifies how SETM and ENDM macro routines are brought into the common overlay area. When left blank, both functions are brought into the overlay area individually.
13	Direct access bucket addressing	Δ, RELATIVE, ACTUAL, or BOTH	Specifies how direct access bucket addresses are supplied. When left blank, relative bucket addressing is used.
14	Multiple MIOC indicator	Δ or MULTIMIOCS	Specifies whether there is more than one MIOC macro routine in the program. When left blank, only one MIOC macro routine is in the program.
15	Open segmentation	Δ or COMBINE	Specifies how the open macro coding is brought into the common overlay area. When left blank, optimum segmentation is achieved.
16	Alter member coding requirement	Δ or NOMALTER	Specifies whether the coding for the alter member function is used. When left blank, the MALTER macro call can be used.
17	Release member coding requirement	Δ or NORELEASE	Specifies whether the coding for the release member function is used. When left blank, the MSREL macro call can be used.
18	Multiple volume coding requirement	Δ, SINGLEVOL, or MULTIVOL	Specifies whether multivolume files are processed by this MIOC. When left blank, only single volume files are processed.
25	Key verification requirement	Δ or KEYVER	Specifies whether key verification in indexed sequential files is required. When left blank, key verification is not used.
29	Address mode	Δ, 3, or 4	Specifies whether MIOC is to be assembled in 3-character or 4-character address mode. When left blank, 3-character address mode is used.
30	Operator control file device type	Δ or 220	Specifies whether a control panel or Type 220 Console typewriter is used by the operator.
32	Buffer modes	Δ, DOUBLE, SINGLE, or BOTH	Buffering modes to be used with this MIOC. When left blank, double buffering is used.
33	Item-handling mode	Δ, MOVE or LOCATE, or BOTH	Method of delivering items to the user to be used in this MIOC. When left blank, the locate mode is used.

Table 3-5 (cont). Summary of MIOC Parameter Values

Number	Name	Value	Function
46	Special insert exits	Δ or SPEC-EXITS	Specifies whether or not the set of two special exits will be taken during processing of each data block by the MSINS macro routine when called for an indexed sequential file.
50	Physical I/O C requirements	Δ , CALL, or PRESENT	Specifies how the appropriate MPIOC macro is called or specialized. When left blank, MIOC calls MPIOC.
51	Suffix	See parameter 01	Identifies the MPIOC tag. Must be specified.
52	Peripheral address assignment	Δ , xx (octal)	When left blank, 04 (octal) is used.
53	Write verification	Δ or V	Specifies whether write verification is required. When left blank, write verification is not required.
54	Control of more than one peripheral address assignment	Δ or M	Specifies how the address of the peripheral control unit is specialized. When left blank, only one control unit is used.
55	RWC definition	Δ or xx (octal)	Specifies read/write channel when parameter 54 = Δ . Cannot be changed without reassembly.
56	MSEEK indicator	Δ or SEEK	Specifies whether or not the MSEEK action macro routine may be called.
57	LOKDEV	Δ or LOKDEV	Specifies whether or not the LOKDEV action may be called.

File Description Macro Routine (MCA)

One file description (MCA) macro call is required for each file to be processed by MIOC. The MCA macro call automatically generates a Physical I/OC file description macro call (MPCA). The communication area set up by MPCA has the same file tag that is specified in parameter 00 of the MCA macro call. (Physical I/O C is described in Appendix D of this manual.) The programmer can interrogate certain fields of the communication area set up by MPCA, but he should never alter the contents of these fields, since they are used by Logical I/O C.

All volumes of one file must be included in the same device class. The device classes follow.

Class	Device Type
A	258, 259, 273, 259A, 259B
B	155
C	261, 262

MCA MACRO CALL

The following coded example illustrates the method of writing the MCA macro call.

CARD NUMBER				M A C R O	LOCATION	OPERATION CODE	OPERANDS		
1	2	3	4					5	6
				L	tag	MCA	parameter 01, ..., parameter n,		

The parameters required for MCA and their standard values are listed in Table 3-6. A summary of the MCA parameter values is provided in Table 3-7.

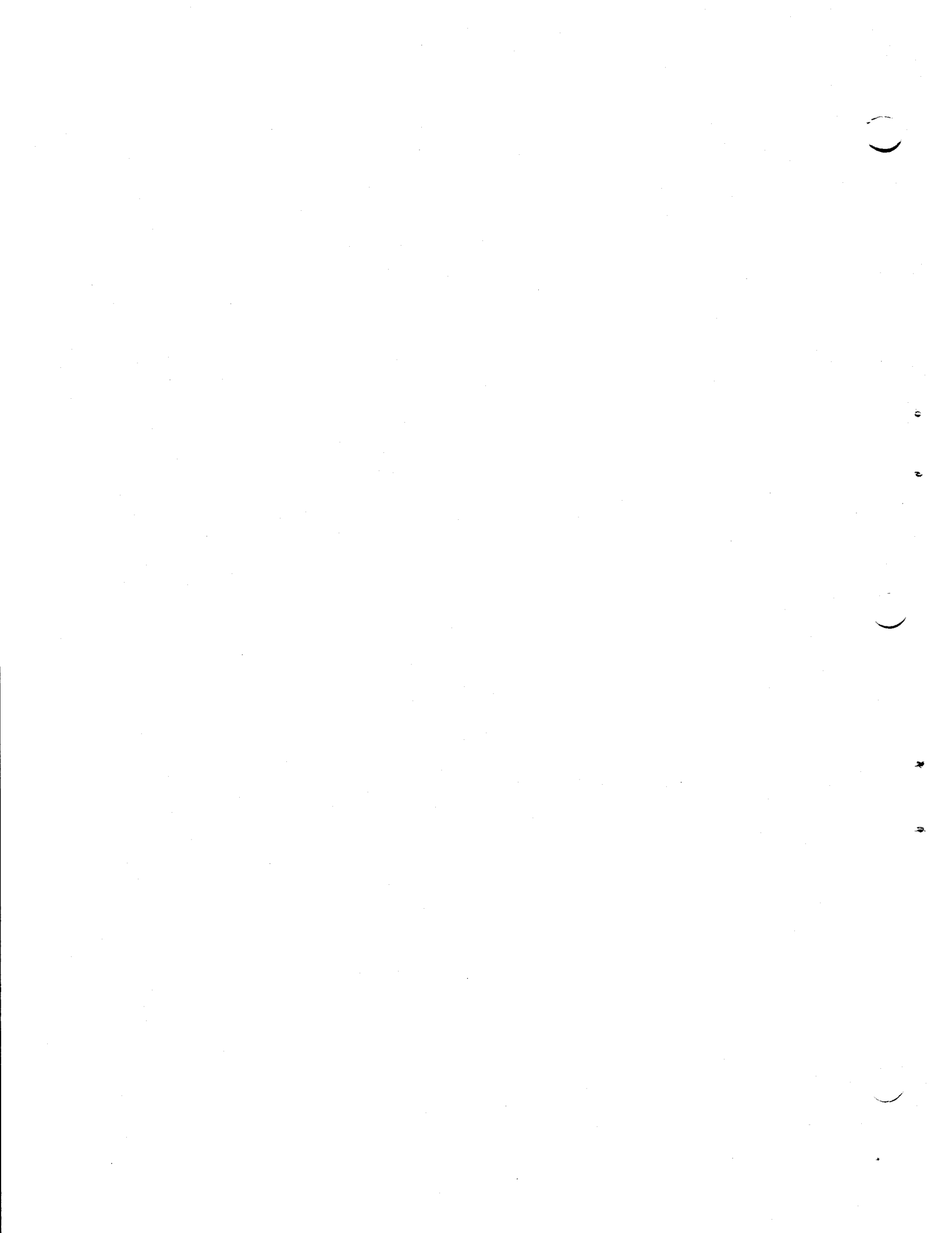


Table 3-6. Parameters of MCA Macro Call

Number	Name	Value	Function	Comments
00	File tag	One, two, or three characters	Used to achieve unique identification of the communication area for this file.	Required. The programmer uses this tag in action macro calls when referring to this file.
01	Unique MIOC character	See parameter 01 of MIOC for possible values.	Associates this communication area with the appropriate MIOC.	Required. Must be the same as parameter 01 of same MIOC call in this program.
02	Volume address	Tag	Specifies the direct address of the leftmost character of a user-supplied table which contains the device addresses of the volumes containing the file.	See Note 1 at end of table for the format of the user-supplied table.
03 through 09		Not applicable		These parameters are reserved for the use of the operating system.
10	I/O buffer address	Tag	Specifies the address of the leftmost character of a user-supplied buffer to be used for data transfers.	See Note 2 at end of table for the format of the user-supplied buffer.
11	Alternate buffer	Δ	This file is processed in the single buffer mode.	If this parameter is assigned a value, double buffering is used. See Note 2 at end of table for the format of the buffer. Must be blank for indexed sequential files.
		Tag	The address of a second user-supplied buffer.	
12	Item delivery mode	Δ or LOCATE	The address of the item in the buffer is delivered to the user.	Optional. When left blank, the locate-item-delivery mode is used.
		MOVE	Items are delivered from/to the buffer to/from the user-supplied work area.	

SECTION III. LOGICAL I/O C

Table 3-6 (cont). Parameters of MCA Macro Call

Number	Name	Value	Function	Comments
13	Item linkage	Tag	Specifies the direct address of the right end of a user-supplied address storage area (i. e., of an index register or a DSA).	Required. See Note 3 at end of table.
14	Insert item linkage	Δ	No inserting will be done.	Applies only to direct access and indexed sequential file processing. See Note 4 at end of table.
		Tag	Specifies the direct address of the right end of user-supplied address storage area that points to the leftmost character of a user-supplied work area.	
15	Insert item work area	Δ	Items will not be inserted into indexed sequential files.	See Note 8 at the end of this table.
		Tag	Points to the leftmost character of an item storage area reserved by the user.	
16	Second insert item work area	Δ	Items will not be inserted into indexed sequential files.	See Note 8 at the end of this table.
		Tag	Points to the leftmost character of a second item storage area reserved by the user.	
17	Units of allocation	Δ	Specifies that this file has one data unit of allocation. Not valid for indexed sequential files.	Optional, except for indexed sequential files. When left blank, Logical I/O C generates a single field table. See Note 5 at end of table for the format of the user-supplied units of allocation table.
		Tag	Specifies the direct address of the left end of a user-supplied table into which the units of allocation for this file are placed when this file is opened. Mandatory for indexed sequential files.	
18	Direct access bucket addressing mode	Δ or RELATIVE	Buckets are relatively addressed in binary for this file.	Optional. When left blank, the relative bucket addressing mode is assumed.
		ACTUAL	The actual key in binary is supplied for buckets in this file.	

SECTION III. LOGICAL I/O C

Table 3-6 (cont). Parameters of MCA Macro Call

Number	Name	Value	Function	Comments
19	Not Applicable			This parameter is reserved for the use of the operating system.
20	File name	Up to 10 characters	Specifies the file name.	Required. Must be the same as the file name stored in the volume directory.
21	Password	Δ	The file is not protected by a password.	Optional. If a password is specified, the field containing the password must be word marked at the leftmost location. The password supplied for the file must be exactly the same as that assigned to the file when it was allocated.
		Tag	Specifies the address of the right end of a user-supplied field into which the user has placed the password for the file.	
22 through 24	Not Applicable			These parameters are reserved for the use of the operating system.
25	Key verification requirements	Δ	Key verification is not required for indexed sequential files.	Parameter 12 must be LOCATE. Key storage area must be the length of an item key and must be word-marked in its leftmost location. Parameter 12 must be MOVE.
		Tag	Specifies a direct address referencing the rightmost character of a user-provided key storage area.	
		KEYVER	Key verification is required every time the replace function is requested.	
26	Sequential key work area	Δ	Specifies that indexed sequential files will not be processed sequentially.	See Note 9 at the end of this table.

Table 3-6 (cont). Parameters of MCA Macro Call

Number	Name	Value	Function	Comments
26 (cont)	Sequential keywork area	Tag	Points to the direct address of the rightmost character of a user-supplied area for the storage of item keys from each current data area of the file.	Tag is the direct address of the right side of the key storage area.
27	Resident cylinder index area	Δ	Specifies that the cylinder index of an indexed sequential file is not resident in main memory.	No punctuation is allowed in this area, since it is for I/O use only.
		Tag	This tag defines the leftmost location of the resident cylinder index area reserved by the user. The size of the area is that required for the number of blocks specified by MCA parameter 28, plus three characters.	
28	Size of resident cylinder index area	Δ	Specifies that the cylinder index is not resident.	
		n (decimal integer)	n specifies the number of blocks of the cylinder index that are to be resident.	
29	MSEEK key work area	Δ	The MSEEK macro routine is not to be called for an indexed sequential file.	This parameter must not be blank if the MSEEK routine is to be called for an indexed sequential file. The MSEEK key work area must have a word mark on the leftmost location. The size of the area is that of one item key.
		Tag	This tag specifies the direct address of the rightmost location of the MSEEK key work area. This area is generated by the user for storage of one item key when the MSEEK macro routine is called for an indexed sequential file. The contents of the area are compared with the item key of a subsequent MSGET routine, to verify that both routines refer to the same item.	

Table 3-6 (cont). Parameters of MCA Macro Call

Number	Name	Value	Function	Comments
30	Physical I/O C suffix		Serves as the suffix character for the MPIOC macro call used for this MIOC.	Required. Must be the same as the unique character specified in parameter 51 of the MIOC macro call with which this MCA is associated.
31	Protection	Δ	Permits no writing.	Optional. When left blank, the file is protected from any writing (see Note 6 at end of table). 02 = Permit data write. 06 = Permit A-file write. 12 = Permit B-file write. 16 = Permit A-and B-file write. 00 = Permit no write.
		xx (octal)	Specifies the protection to be observed for this file.	
32	Verification requirements	Δ	Verification is not required for this file.	Optional. When left blank, automatic verification is not done.
		VERIFY	All output data transfers for this file are automatically verified.	
33 through 39	Not Applicable			These parameters are reserved for the use of the operating system.
40	Volume directory exit	Tag	See Note 7 at end of table. This exit is available whenever the information to be conveyed pertains to the volume directory.	Reason for exit: 1. Exit from open function for user interrogation and alteration of *VOLDESCR*. 2. Exit from close function for user interrogation and alteration of *VOLDESCR*.

Table 3-6 (cont). Parameters of MCA Macro Call

Number	Name	Value	Function	Comments
40 (cont)	Volume directory exit			<p>3. Unable to locate specified file.</p> <p>4. Password check failure.</p> <p>5. Password checking not specified in MCA, but password exists for the file.</p> <p>6. User's unit of allocation table has overflowed.</p> <p>7. Invalid format in *VOLALLOC*.</p> <p>8. Open found a file volume whose sequence number is not one greater than the last.</p> <p>9. Exit from open to user when the device address table is not long enough for the required file volumes in the file.</p> <p>10. The current file volume being processed as input/output does not contain data.</p> <p>11. A new file volume has to be opened, and no entries remain in the device address table.</p> <p>NOTE: For cases 1 and 2, APD points to the left end of the *VOLDESCR* item.</p>

Table 3-6 (cont). Parameters of MCA Macro Call

Number	Name	Value	Function	Comments
41	Index exit	Tag	See Note 7 at end of table. This exit is available whenever the information to be conveyed is pertinent to a particular file's member index.	Reason for exit: <ol style="list-style-type: none"> 1. The set member function is unable to locate the member. 2. The alter function is unable to locate the member. 3. No room in index for the set member function to create another member. 4. The member is unavailable to the set member function for output-only processing. 5. The member is unavailable to the alter function for deletion.
42	Not Applicable			This parameter is reserved for the use of the operating system.
43	Data exit	Tag	See Note 7 at end of table. This exit is taken whenever the information to be conveyed pertains to the file's data.	Reason for exit: <ol style="list-style-type: none"> 1. End-of-data item detected on an input function. 2. There is no room to output the last item requested by the put function. 3. No data blocks remain for the creation of a new member. 4. Unable to locate an item key in an attempt to get an item.

Table 3-6 (cont). Parameters of MCA Macro Call

Number	Name	Value	Function	Comments
43 (cont)	Data exit			5. Unable to locate an available item position while attempting to insert. 6. An invalid direct access bucket address has been specified. 7. Indexed sequential insert has inserted an item and finds that all the overflow areas are full. 8. Indexed sequential key verification failed while replacing an item. 9. Indexed sequential insert discovers a duplicate item key. 10. Set of two special exits has been requested during indexed sequential inserts by parameter 46 of MIOC.
44	Device exit	Tag	See Note 7 at end of table. This exit is taken whenever the information to be conveyed is pertinent to the mass storage device currently being used for this file.	A list of causes for this exit is contained in Table 3-14.

Table 3-6 (cont). Parameters of MCA Macro Call

- NOTES:
1. The format of the user-supplied table that contains the device address of the volumes containing the file must be constructed as follows.
 - a. There must be as many entries in the table as there are devices associated with the file.
 - b. Each entry must be three characters long and be word marked at its leftmost location.
 - c. There must be a record mark one character location to the right of the last entry.
 - d. The format of each entry in the table is "ppdd00" (octal), where:
pp = address of peripheral control unit and
dd = device number.
 2. The format of the user-supplied input/output buffer is as follows.
 - a. The buffer must be as long as a block of information.
 - b. There must be three record-marked character locations to the right of the buffer. No other record marks may appear in the buffer.
 - c. Item marks may be set only within the current item of a buffer and only when the locate mode is being used. They must be cleared before Logical I/O C is reentered.
 - d. Any punctuation on the rightmost data character in the buffer may be cleared by Logical I/O C. If the user requires punctuation in this field, it is his responsibility to restore it.
 - e. Word marks may not exist in a key field in a buffer except for the leftmost position of a direct access key field at the time Logical I/O C is entered. Word marks may not exist in the buffer for an indexed sequential file at the time Logical I/O C is entered.
 - f. Punctuation cannot exist in a buffer used for partitioned sequential processing for the set member, end member, alter member, or release member function.
 3. In the move-item-handling mode, the address storage area points to the leftmost character of a user-supplied work area where Logical I/O C places and retrieves items. This work area must be the length of one item and cannot contain item marks at the time Logical I/O C is entered. If key verification of replaced items on indexed sequential files is to be performed, a word mark can be set at the leftmost character of the key area and nowhere else. In the locate-item-handling mode, the address storage area locates for the user the leftmost character of the current item position in the current buffer.
 4. The work area to which the address storage area points must be the length of one item; the work area cannot contain item marks at the time Logical I/O C is entered. An item to be inserted must be placed in the work area by the user. The value of parameter 14 (to which this note refers) may be the same as the value of parameter 13.
 5. The format of the user-supplied unit of allocation table is as follows.
 - a. There must be at least as many fields in this table as the maximum number of units of allocation for the file.

Table 3-6 (cont). Parameters of MCA Macro Call

NOTES:
(cont).

- b. Each field in the table must be eight characters long and must contain a word mark in its leftmost location.
 - c. There must be a record mark in the character location immediately to the right of the last field in the table.
6. Possible values for parameter 31 are as follows.
- a. 02 = permit data write,
 - b. 06 = permit A-file write,
 - c. 12 = permit B-file write,
 - d. 16 = permit A- and B-file write, and
 - e. 00 = permit no write.

These possible values are specified in octal, as shown. Recall that the control unit protection switches must be set to agree with the value chosen. For a further description of file protection, see Appendix F of this manual.

7. This note applies to parameters 40, 41, 43, and 44. These parameters constitute the four major exit categories. The user is required to interrogate a code to determine the exact cause of any given exit. This code is set into a 1-character DCW instruction that the user is required to generate at one memory location less than the entrance point of each exit routine. Before returning from the user's exit routine, the programmer sets up another code in the same DCW location that indicates the desired action. The tag, which the programmer specifies for these exits, points to a user-supplied routine that must: (1) save the return address, (2) interrogate a code for the cause of the exit, (3) take appropriate action, (4) set up a return code, and (5) return to MIOC (when applicable). When the programmer does not accommodate a particular exit, MIOC uses a standard value to continue (if possible) or notifies the operator, depending on the meaning of the exit.

At the time the exit is taken, the MCA field named "APD" points to data pertinent to the exit (see Tables 3-9, 3-10, 3-11, and 3-12).

8. When items are being inserted into indexed sequential files, two item storage work areas are required. Parameter 15 points to the first area. Since this area is specifically for the use of Logical I/O C, punctuation cannot be present in this area whenever Logical I/O C is entered. Punctuation does not exist in the area upon the normal return to the user's coding from Logical I/O C. The tag of parameter 16 may point to an identical area, or it may point to the same area that parameter 14 points to. If it does point to the same area that parameter 14 points to, however, that area must not contain punctuation when inserting is being done; when Logical I/O C returns to the user's coding, the item in the work area is not preserved.
9. The user-supplied item key storage area must have the following format.
- a. For files with cylinder overflow, three fields are required. Each field must be the length of an item key, and each field must contain a word mark in its leftmost location;
 - b. Or, for files without cylinder overflow, two fields are required. These fields must be the same length as an item key and must be word-marked in their leftmost location.

SECTION III. LOGICAL I/O C

Table 3-7. Summary of MCA Parameter Values

Number	Name	Value	Function
00	File prefix	One, two, or three characters	Used to achieve unique identification of this MCA communication area.
01	Unique MIOC character	See parameter 01 of MIOC for possible values	Associates this communication area with the appropriate MIOC.
02	Volume address	Tag	Specifies the device address of the volumes containing the file.
10	I/O buffer address	Tag	Specifies the address of the user-supplied input/output buffer.
11	Alternate buffer	Δ or tag	When left blank, the file is processed in the single buffer mode. The tag specifies the address of the user-supplied input/output buffer.
12	Item-delivery mode	Δ, LOCATE, or MOVE	Specifies the item delivery mode. When left blank, the locate mode is used.
13	Item linkage	Tag	Specifies the address of a user-supplied address storage area.
14	Insert item linkage	Δ or tag	When left blank, no inserting is done. The tag specifies the address of a storage area that points to the insert item.
15	Insert item work area	Δ or tag	When left blank, items are not inserted into indexed sequential files. Tag points to the leftmost character of an item storage area.
16	Second insert item work area	Δ or tag	When left blank, items are not inserted into indexed sequential files. Tag points to the leftmost character of the second item storage area.
17	Units of allocation	Δ or tag	When left blank, the file has only one unit of allocation, except for indexed sequential files. The tag specifies the address of the user-supplied units of allocation table.
18	Direct access bucket addressing mode	Δ, RELATIVE, or ACTUAL	Specifies the mode of addressing for direct access file. When left blank, the relative address mode is used.
20	File name	Up to ten characters	Specifies the file name.
21	Password	Δ or tag	When left blank, Logical I/O C checks for a blank password field in *VOLDESCR*. The tag specifies the address of the password the user is supplying for the password check.

Table 3-7 (cont). Summary of MCA Parameter Values

Number	Name	Value	Function
25	Key verification requirements	Δ , KEYVER, or tag	Specifies whether or not key verification is used.
26	Sequential key work area	Δ or tag	Specifies whether or not indexed sequential files are processed sequentially.
27	Resident cylinder index area	Δ or tag	Specifies whether or not the cylinder index for an indexed sequential file is resident in main memory. If resident, the tag defines the leftmost location of the cylinder index area.
28	Resident cylinder index area size	n or Δ	Specifies number of blocks of the cylinder index that are resident. "n" equals any number desired, expressed as a decimal integer. Blank if the cylinder index is not resident.
29	MSEEK key work area	Δ or tag	Specifies whether or not the MSEEK action macro routine will be called for an indexed sequential file. The tag defines the rightmost location of a user-generated area for storage of one item key.
30	Physical I/O C suffix	x	Serves as the suffix character for MPIOC macro call used for this MIOC.
31	Protection	Δ , 00, 02, 06, 12, or 16, (octal)	Specifies the protection to be used for this file. When left blank, 00 is used.
32	Verification requirements	Δ or VERIFY	Specifies whether or not output data transfers for this file will be verified. When left blank, verification is not done.
40	Volume directory exit	Tag or Δ	Specifies the address of a user-supplied exit routine. When left blank, this exit is not taken.
41	Index exit	Tag or Δ	Specifies the address of a user-supplied exit routine. When left blank, this exit is not taken.
43	Data exit	Tag or Δ	Specifies the address of a user-supplied exit routine. When left blank, this exit is not taken.
44	Device exit	Tag or Δ	Specifies the address of a user-supplied exit routine. When left blank, this exit is not taken.

Communication Area Service Macro Routines (MLCA and MUCA)

There are two communication area service macro routines: MLCA alters the contents of certain fields of the communication area, and MUCA moves the values of certain fields to the user's own storage area.

MASS STORAGE LOAD COMMUNICATION AREA MACRO CALL (MLCA)

The MLCA macro call provides the programmer with the capability of updating the contents of certain fields in the communication area. To alter the contents of a particular field, the programmer must associate the field's mnemonic designator with a main memory address. A mnemonic designator is a tag which Logical I/O C appends to the communication area tag that the programmer specified as parameter 00 of this file's MCA call. The main memory address is the address of the value to be placed in the communication area field. The MLCA macro routine moves the user's field to the associated field in the communication area. As many of these pairs of mnemonic designators and main memory addresses as are required can be specified in a single MLCA macro call. The following example illustrates the coding of the MLCA macro call.

CARD NUMBER	Y	T	M	R	LOCATION	OPERATION CODE	OPERANDS	
							1 2 3 4 5 6 7 8	14 15 20 21 42 63 80
1					C	CHANGE MLCA	FL1, CHGVER, VER, MYPRT, PRT,	
2					C		ZERERR, ECT, BUF, PBL,	
3					L		NEWNAM, FID,	
4								

In this example, parameter 01 (FL1) is a file prefix and is identical with parameter 00 of the MCA macro call which generates this particular communication area. Parameters 02, 04, 06, 08, and 10 indicate the addresses of main memory locations that contain information to be placed in the communication area fields identified by the mnemonic designators VER, PRT, ECT, PBL, and FID. A complete list of the mnemonic designators is given in Table 3-8.

Parameters 02 through 63 of the MLCA macro call are treated in pairs. The first unit of the pair is the main memory address containing the value to be placed in the communication area, and the second unit of the pair is the mnemonic designator of the communication area field to be updated. The first omitted (blank) main memory address terminates the MLCA function. The order in which the pairs are specified is not significant unless one field is to overlie another.

MASS STORAGE UNLOAD COMMUNICATION AREA MACRO CALL (MUCA)

The MUCA macro call provides the programmer with the ability to access the contents of certain fields in the communication area. The use of this macro call corresponds to that of the MLCA, except that the transfer of information is from the communication area to main memory. The following example illustrates the method of coding the MUCA macro call.

SECTION III. LOGICAL I/O C

CARD NUMBER	Y	M	R	LOCATION	OPERATION CODE	OPERANDS	
						1415	2021
1				L UNLOAD MUCA	FL1, LSTADR, RIC,		
2							
3							
4							
5							
6							
7							
8							
9							
10							
11							

COMMUNICATION AREA FIELD DESIGNATORS

Table 3-8 lists the communication area fields that can be altered or interrogated by the MLCA and MUCA macro calls, respectively. These fields are identified by the mnemonic designator that the programmer specifies in the macro call.

Each alteration or interrogation of a field in the communication area is performed by an Extended Move (EXM) instruction, moving from right to left (data bits only). The move is stopped by an A-field word mark. Thus, in the case of MLCA, the user's word mark terminates the move; in the case of MUCA, the word mark in the communication area field terminates the move. Caution must be used in setting up the field in main memory and in using address arithmetic on the mnemonic designators.

For certain fields, either the MLCA or the MUCA macro call cannot be used. This restriction is indicated in Table 3-8.

Table 3-8. Mnemonic Designators for Communication Area Fields

Field Name	Mnemonic Designators	Contents
Current address	CAD	CAD is an 8-character field in the format DPCCTRR. It is the actual address being used for the current data transfer operation. The MLCA macro call cannot be used for this field.
Protection	PRT	PRT is a single-character field (not word marked) that reflects the protection the programmer requested through parameter 31 of the MCA macro call. The programmer can alter this field with an MLCA (using octal values as shown for parameter 31 in Table 3-6), since it is the programmer's word mark that terminates the move. The MUCA macro call cannot be used for this field.
Error count	ECT	ECT is a 1-character field that shows the cumulative number of rereads and rewrites that have been necessary for the file.

Table 3-8 (cont). Mnemonic Designators for Communication Area Fields

Field Name	Mnemonic Designators	Contents
Read/write channel	RWC	If parameter 54 of MIOC specifies more than one control unit (i.e., contains M), then RWC is a 1-character DCW which defines the read/write channel to be used by Physical I/O C for subsequent operations. The read/write channel can be altered only when the peripheral control unit number differs from that used in a preceding operation. If parameter 54 of MIOC specifies one control unit (i.e., is blank), then RWC cannot be used.
Current peripheral buffer	PBL	PBL is a DSA referring to the left end of the programmer's buffer which most recently has had a data transfer issued to or from it. This field cannot be altered after the file is opened.
Current buffer	CBL	CBL is a DSA that points to the left end of the programmer's buffer that is most recently receiving items or delivering items to the programmer. When single buffering is specified, this field must be the same as PBL. The field cannot be altered after the file is opened.
Address of pertinent data	APD	Any time an exit is taken that allows the programmer to interrogate data retrieved by Logical I/O C, the DSA designated by APD points to that data. Only the MUCA macro call may be issued for this field.
Next user instruction	WCF	Whenever a user exit is taken, WCF (a DSA) points to the return address in the user's code from the last action call issued. The MLCA macro call cannot be used for this field.
File identification	FID	FID is a 10-character field containing the file name that the user specified as parameter 20 of the MCA macro call.
Write verify indicator	VER	The 1-character field VER indicates whether verification is required for the file. An octal 40 indicates that verification is required. An octal 00 indicates that no verification is required. This field may be altered at any time.
Bucket addressing mode	BKA	BKA is a 1-character field which, when equal to octal 40, indicates that bucket addresses are actual. When this field is equal to octal 00, it indicates that bucket addresses are relative.
Overflow indicator	OVF	The 1-character field OVF indicates whether an indexed sequential or a direct access random function has overflowed into either the cylinder or the general overflow areas. A bit is set each time such a situation occurs. It is always reset to zero at the beginning of any direct access function. The values of this field (in octal) are as follows:

Table 3-8 (cont). Mnemonic Designators for Communication Area Fields

Field Name	Mnemonic Designators	Contents
Overflow Indicator (cont)		00 = no overflow, 40 = cylinder overflow, 20 = general overflow, and 60 = both cylinder and general overflow.
Current item address	RIC	RIC is a 10-character field that shows the address of the last item retrieved by the user. The format of the address is DPCCTTRRII (see note).
Relative volume number	RVL	RVL is a 1-character field that contains the relative volume number of the volume currently being processed.
<p>NOTE: D = device number. P = pack number. CCTTRR = mass storage record address for the first record of the item's block, a track-linking record which points to the first record of the item's block, or the first record of a partial portion of the cylinder previous to the cylinder containing the item's block. II = relative item position within the block.</p> <p>If the block resides on a substitute track, CCTT contains the address of the substitute track when one of the following conditions exists:</p> <ol style="list-style-type: none"> 1. When sequential processing is being performed on any file type, and the block is not the first block on the substitute track; or, 2. When direct-access processing is being performed, and the block is not the first block on the substitute track, and is not the first block in a bucket, prime data string, or cylinder overflow area. 		

Action Macro Calls

An action macro call is a request from the user for a particular input/output function. The call is placed in line in the user's coding whenever he desires that function. The coding generated from such a call specifies the communication area (MCA) applicable to the call, designates the operation of the requested function, and conveys any additional information that the function may need (e.g., the addresses of the bucket and item key in direct access files).

The following list defines terms frequently used in describing the action calls. The action macro calls are summarized in Tables 3-2 and 3-9.

1. Location field tag. Whenever the user specifies a tag as parameter 00 of an action macro call, that tag will be equated to the operation code of the first generated instruction of the action macro routine. This feature is provided so that the user may branch directly to the coding generated by an action macro call.
2. File tag. The file tag is a 1-, 2-, or 3-character tag for the file to which the action is directed. This tag must be the same as parameter 00 of the appropriate MCA macro call.
3. Bucket tag. The bucket tag is the address of the right end of a user-supplied field containing a bucket address. The format of this field is defined on page 3-71.

4. Key tag. The key tag is the address of the right end of a user-supplied field containing an item key. The format of this field is defined on pages 3-72 and 3-73.
5. Member-name tag. The member-name tag is the address of the right end of a user-supplied 14-character field containing the name of the desired member. This field must be word-marked at its leftmost location.
6. New-name tag. The new-name tag is the address of the right end of a user-supplied 14-character field containing the new name for the member. This field must be word-marked at its leftmost location.

The file tag applies to every action macro call and is either the first or only parameter of each call. The bucket tags apply only to some action macro routines that can process direct access files. The key tags apply only to some action macro routines that can process indexed sequential and direct access files. The member name and new name tags apply only to some action macro calls that pertain to partitioned sequential files.

OPEN (MSOPEN)

The MSOPEN macro call is used to open a file for processing. MSOPEN is coded as illustrated in the following example.

CARD NUMBER	Y	W	OPERATION CODE	OPERANDS
1 2 3 4 5 6 7 8	14 15	20 21	62 63	80
1	L	anytag	MSOPEN	file-tag, { IN/OUT } LIMVOL,
2				{ IN
3				{ OUT
4				{ UPDATE
5				{ Δ
6				

When opening a sequential file, IN/OUT, IN, or OUT must be specified. When opening a partitioned sequential file, UPDATE must be specified if processing of members is to be done in the input/output or output-only mode. Blank (Δ) must be specified if processing of all members is to be done in the input-only mode. When opening a direct access or indexed sequential file, either IN/OUT or IN must be specified. To open an indexed sequential or direct access file for sequential processing from its beginning, LIMVOL may be specified. The following examples illustrate the MSOPEN macro call coding for opening each type of file.

EXAMPLE 1: Opening a sequential file for input-only processing. In this example, the file tag is the 3-character value of parameter 01, i.e., IM1. Parameter 02 has the value IN. Note the L in column 6.

CARD NUMBER	Y	W	OPERATION CODE	OPERANDS
1 2 3 4 5 6 7 8	14 15	20 21	62 63	80
1	L		MSOPEN	IM1, IN,
2				

EXAMPLE 2: Opening a partitioned sequential file in which processing of all members will be in the input-only mode. In this example, the value of parameter 01 is the single character I. Parameter 02 is blank.

CARD NUMBER	Y	M	OPERATION CODE	OPERANDS
1 2 3 4 5 6 7 8	14 15	20 21	62 63	80
1	L		MSOPENI,	

In this example, if any member of the partitioned sequential file was to be processed in the input/output mode or output-only mode, then UPDATE would have to be the value of parameter 02.

EXAMPLE 3: Opening a direct access file for processing the input/output mode. In this example, the file tag is the 2-character value of parameter 01, i. e., AA. Parameter 02 has the value IN/OUT.

CARD NUMBER	Y	M	OPERATION CODE	OPERANDS
1 2 3 4 5 6 7 8	14 15	20 21	62 63	80
1	L		MSOPENAA, IN/OUT,	

EXAMPLE 4: To open a multivolume indexed sequential or direct access file for sequential processing from its beginning, the open macro call is coded as follows. The value of parameter 03, LIMVOL, specifies the minimum number of file volumes to be opened initially. Note that if the file is indexed sequential, the file volume or volumes containing the master/cylinder index and the general overflow must always be online. Subsequent file volumes are opened as they are encountered.

CARD NUMBER	Y	M	OPERATION CODE	OPERANDS
1 2 3 4 5 6 7 8	14 15	20 21	62 63	80
1	L		MSOPENFL1, IN/OUT, LIMVOL,	

CLOSE (MSCLOS)

The MSCLOS macro call is used to close all types of files. MSCLOS is coded as illustrated in the following example.

CARD NUMBER	Y	M	OPERATION CODE	OPERANDS
1 2 3 4 5 6 7 8	14 15	20 21	62 63	80
1	L		MSCLOS PDQ,	

In this example, it can be seen that the MSCLOS macro call requires only the file-tag parameter. Note that when the file being processed is a partitioned sequential file, the MSCLOS macro call must be preceded by the ENDM macro call when a SETM macro call has been previously issued for a member of the file.

GET (MSGET)

The MSGET macro call is used to retrieve items from the file being processed. MSGET is coded as illustrated in the following example.

SECTION III. LOGICAL I/O C

CARD NUMBER		Y	X	LOCATION	OPERATION CODE	OPERANDS	
1	2	3	4	5	6	7	8
1					MSGET	file-tag,	(bucket-tag, key-tag)
2							bucket-tag
3							Δ, key-tag
4							NEXT
5							key-tag
6							A
7							

In processing sequential files, only the file tag can be used. In processing direct access files, any of the values shown (except the last two) for parameters 02 and 03 may be used. The bucket tag value can be either relative or direct. Notice that when only a key tag is specified for direct access files, the value of parameter 02 must be blank, and its terminating comma must be present. The value NEXT must be specified for direct access files when neither a bucket tag nor a key tag is supplied. In processing indexed sequential files sequentially, only the file tag can be used. For random processing of indexed sequential files, the key tag must be specified as parameter 02. For a description of the searching sequence for the desired item in a direct access file, refer to page 3-13.

EXAMPLE 1: To get an item in a sequential file, or sequentially in an indexed sequential file, the MSGET macro call requires only the file tag. In this illustration, the value of the file tag is XYZ.

CARD NUMBER		Y	X	LOCATION	OPERATION CODE	OPERANDS	
1	2	3	4	5	6	7	8
1					MSGET	XYZ,	
2							

EXAMPLE 2: To get an item in a direct access file, a value for parameter 01 and for parameter 02 is required. The value of parameter 02 can, of course, be blank; but if it is blank, a value of parameter 03 is required. To illustrate this, the following coding shows the file tag as MOD, the bucket-tag value as unspecified, and the key tag value as IKEY, which is the address of the right-most location of the field containing the item key.

CARD NUMBER		Y	X	LOCATION	OPERATION CODE	OPERANDS	
1	2	3	4	5	6	7	8
1					MSGET	MOD, Δ, IKEY,	
2							

EXAMPLE 3: To get the next sequential item in the bucket currently being processed, the programmer codes the MSGET macro call with only the value of the file tag specified for parameter 01 and with NEXT as the value of parameter 02. In this illustration, the file-tag value is WED.

CARD NUMBER		Y	X	LOCATION	OPERATION CODE	OPERANDS	
1	2	3	4	5	6	7	8
1					MSGET	WED, NEXT,	
2							

EXAMPLE 4: To randomly get an item in an indexed sequential file, the programmer codes the MSGET macro call with the file tag specified in parameter 01 and the key tag specified in parameter 02, as shown below. In this example, the file tag is FL1 and the key tag is MYITEM.

SECTION III. LOGICAL I/O C

CARD NUMBER		OPERATION CODE	OPERANDS											
1	2	3	4	5	6	7	8	14	15	20	21	62	63	80
					L									

REPLACE (MSREP)

The MSREP macro call is used to replace that last item retrieved from a file. MSREP is coded as illustrated in the following example.

CARD NUMBER		OPERATION CODE	OPERANDS											
1	2	3	4	5	6	7	8	14	15	20	21	62	63	80
					L									

In the coded example of the MSREP macro call above, it can be seen that the call requires only the file-tag parameter. This is shown as UAR in the example.

INSERT (MSINS)

The MSINS macro call is used to insert items into a direct access or indexed sequential file. MSINS cannot be used with sequential files.¹ The insert macro call is coded as shown in the following two examples. Example 1 is for direct access files, and example 2 is for indexed sequential files.

CARD NUMBER		OPERATION CODE	OPERANDS											
1	2	3	4	5	6	7	8	14	15	20	21	62	63	80
					L									

CARD NUMBER		OPERATION CODE	OPERANDS											
1	2	3	4	5	6	7	8	14	15	20	21	62	63	80
					L									

MSINS requires the file tag; the bucket tag can be either specified or unspecified. For a description of the inserting process for direct access files, refer to "Inserting Items in Direct Access Files" in this section. The following examples show both methods of coding this macro call for direct access files. In each example, the same file tag (LOT) is used. In the first example, the bucket tag is specified as BTAG; and, in the second example, it is unspecified.

CARD NUMBER		OPERATION CODE	OPERANDS											
1	2	3	4	5	6	7	8	14	15	20	21	62	63	80
					L									

CARD NUMBER		OPERATION CODE	OPERANDS											
1	2	3	4	5	6	7	8	14	15	20	21	62	63	80
					L									

¹ Parameter 11 of MIOC must be specified as SEGMENT or RESIDENT if the insert call is to be issued.

The following example shows the MSINS macro call coded for indexed sequential files. In this example, the file tag is specified as FLX and the key tag as IKEY.

CARD NUMBER		Y P R	M A R K	LOCATION	OPERATION CODE	OPERANDS								
1	2	3	4	5	6	7	8	14	15	20	21	62	63	80
					L									

DELETE (MSDEL)

The MSDEL macro call is used to delete the last item retrieved from a direct access or indexed sequential file. MSDEL cannot be used with sequential file organization. The MSDEL macro call is coded as shown in the following example. For this macro call, only the file tag is required; in the example, it is shown as AB.

CARD NUMBER		Y P R	M A R K	LOCATION	OPERATION CODE	OPERANDS								
1	2	3	4	5	6	7	8	14	15	20	21	62	63	80
					L									

PUT (MSPUT)

The MSPUT macro call is used to deliver items sequentially to the file. This macro call can be used only with the sequential file organization. MSPUT is coded as shown in the following example. For this macro call, only the file tag is required; in the example, it is shown as X.

CARD NUMBER		Y P R	M A R K	LOCATION	OPERATION CODE	OPERANDS								
1	2	3	4	5	6	7	8	14	15	20	21	62	63	80
					L									

SET MEMBER (SETM)

The SETM macro call is used to begin processing of the member specified in the macro call. SETM can be used only with partitioned sequential file organization; it is coded as shown in the following example.

CARD NUMBER		Y P R	M A R K	LOCATION	OPERATION CODE	OPERANDS								
1	2	3	4	5	6	7	8	14	15	20	21	62	63	80
					L									

SETM requires the file tag, the member name tag, and the processing mode parameter. The following example illustrates the coding of the SETM macro call.

EXAMPLE 1: In this example, the member to be opened for processing is tagged MEMTAG and the processing mode is to be the input/output mode.

CARD NUMBER		Y M E R	LOCATION	OPERATION CODE	OPERANDS																								
1	2					3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
		L		SETM	ABC, MEMTAG, IN/OUT,																								

END MEMBER (ENDM)

The ENDM macro call is used to stop processing of the current member. ENDM applies only to partitioned sequential files and is coded as shown in the following example. For this macro call, only the file tag is required.

CARD NUMBER		Y M E R	LOCATION	OPERATION CODE	OPERANDS																								
1	2					3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
		L		ENDM	file-tag,																								

ALTER MEMBER (MALTER)

The MALTER macro call is used to change the specified member of a partitioned sequential file, as directed by the parameters of the macro call. MALTER can be used only with partitioned sequential files and is coded as shown in the following example.

CARD NUMBER		Y M E R	LOCATION	OPERATION CODE	OPERANDS																								
1	2					3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
		L		MALTER	file-tag, member-name-tag, (AVAIL) new-name-tag, (UNAVAIL), (DELETE)																								

MALTER requires the file tag, the member name tag, a change in member status or a change in member name. A change in member status, a change in member name, or both changes can be specified. The following examples illustrate the coding of the MALTER macro call.

EXAMPLE 1: In this example, the member's status is changed from "available for output processing" to "unavailable for output processing." The member name tag is EFG, and the file tag is HIJ.

CARD NUMBER		Y M D P R	A R C H	LOCATION	OPERATION CODE	OPERANDS								
1	2					3	4	5	6	7	8	14	15	20
1				L		MALTERHIJ, EFG, UNAVAIL,								
2														
3														
4														
5														
6														
7														
8														
9														
10														
11														
12														
13														
14														
15														
16														
17														
18														
19														

EXAMPLE 2: In this example, the member's status is changed to "available for output only processing," and its name is changed to the contents of a field tagged NEW that contains the new member name. The file tag for this file is KLM, and the member name tag is NOP.

CARD NUMBER		Y M D P R	A R C H	LOCATION	OPERATION CODE	OPERANDS								
1	2					3	4	5	6	7	8	14	15	20
1				L		MALTERKLM, NOP, AVAIL, NEW,								
2														

EXAMPLE 3: In this example, the old member is deleted. The field containing the new member name is tagged NEW, file tag is RST, and the member name tag is UVW.

CARD NUMBER		Y M D P R	A R C H	LOCATION	OPERATION CODE	OPERANDS								
1	2					3	4	5	6	7	8	14	15	20
1				L		MALTERRST, UVW, NEW,								
2														
3														
4														

RELEASE (MSREL)

The MSREL macro call is used to restore the complete area occupied by a partitioned sequential file to an unused status. MSREL can be used only with partitioned sequential files and is coded as shown in the following example.

CARD NUMBER		Y M D P R	A R C H	LOCATION	OPERATION CODE	OPERANDS								
1	2					3	4	5	6	7	8	14	15	20
1				L		MSREL file-tag,								
2														

SET LOCATION (SETL)

The SETL macro call is used to start processing of an indexed sequential file at a specified location. The location is specified by the value of parameter 02. When this macro call is used, the file is processed sequentially from the location specified. SETL can only be used with indexed sequential files and is coded as shown in the following example.

CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS
1 2 3 4 5	6 7 8	14 15	20 21	62 63	80
1	L			SETL	file-tag, key-tag,

SEEK (MSEEK)

The MSEEK macro call is used to position the read/write heads of a disk device on a specified cylinder of that device. The Seek function does not cause the disk control to become busy. A Seek may be performed for one disk device while the disk control is busy with data transfer or other activities of another disk device connected to it. Whenever the user issues an MSEEK macro call, a return is made to the main line of his coding while the Seek function is being executed. If a subsequent MSGET macro call is issued for the same cylinder, the read/write heads will be correctly positioned. Efficient use of the MSEEK macro call can significantly improve access time; for example, the Seek time required for execution of an MSGET macro routine can be considerably reduced or eliminated.

The MSEEK macro call can be in either of two formats as shown in the following. The first is for a direct access file; the second is for an indexed sequential file.

CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS
1 2 3 4 5	6 7 8	14 15	20 21	62 63	80
1	L			MSEEK	file-tag, bucket-tag,

CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS
1 2 3 4 5	6 7 8	14 15	20 21	62 63	80
1	L			MSEEK	file-tag, key-tag,
2					
3					
4					
5					
6					
7					
8					

Table 3-9 is a summary of action macro call coding.

SECTION III. LOGICAL I/O C

Table 3-9. Summary of Action Macro Call Coding

Command	Parameter 01	Parameter 02	Parameter 03	Parameter 04	Notes
MSOPEN	file-tag	IN/OUT IN OUT UPDATE Δ	LIMVOL		1, 3 1 2 2 2 7
MSCLOS	file-tag				
MSGET	file-tag	bucket-tag bucket-tag Δ NEXT key-tag Δ	key-tag Δ key-tag		4 4 4 4, 5 8 8, 9, 10
MSREP	file-tag				
MSINS	file-tag	bucket-tag Δ key-tag			4 4 8
MSDEL	file-tag				
MSPUT	file-tag				
SETL	file-tag	key-tag			
MSEEK	file-tag	bucket-tag key-tag			4 8
SETM	file-tag	member-name-tag	IN/OUT IN OUT		
ENDM	file-tag				
MALTER	file-tag	member-name-tag	AVAIL UNAVAIL DELETE	new-name-tag	6 6 6 6
MSREL	file-tag				

Table 3-9 (cont). Summary of Action Macro Call Coding

- NOTES:
1. Either IN/OUT, IN, or OUT must be specified when opening a sequential file.
 2. UPDATE must be specified when opening a partitioned sequential file if processing of members will be input/output or output-only. BLANK must be specified if processing of all members will be input-only.
 3. Either IN/OUT or IN must be specified when opening a direct access or indexed sequential file.
 4. Valid only for a direct access file.
 5. NEXT is required for a direct access file when neither bucket nor item key is specified.
 6. Either a change in status (parameter 03) or a change in name (parameter 04) must be specified. Both parameters 03 and 04 may be used.
 7. Applies only to direct access and indexed sequential files. Open function will open the minimum number of volumes to allow sequential processing of the file from its beginning.
 8. Valid only for indexed sequential files.
 9. Required for sequential files.
 10. Required for sequential get function in indexed sequential files.

PROGRAMMER'S PREPARATION INFORMATION FOR LOGICAL I/O C

The following paragraphs contain general and detailed information to assist the programmer in using Logical I/O C. The subjects covered in these paragraphs are: Logical I/O C memory requirements, program organization, read/write channel utilization, address mode, index registers, direct access addressing, direct access item key specification, and exit and halt codes.

Logical I/O C Memory Requirements

Depending on the number and nature of the functions required, the minimum memory requirement for Logical I/O C is 3,500 characters. This figure assumes that MIOC is segmented and that the memory locations required for Physical I/O C are included.

Program Organization

The routines making up Logical I/O C are designed to take a minimum number of memory locations in any given situation. This is accomplished first by generating only the required coding for processing a given program's files, and, secondly, by segmenting the coding for

those functions that are required infrequently during program execution. Thus, while the coding to open or close a file is required in any given program, this coding in a segmented program is loaded into memory only when the programmer issues an action macro call for one of these functions. A multiphase program can further reduce the input/output memory requirements by specializing separate MIOCs with different processing capabilities for each phase. In multiphase programs, tag uniqueness is ensured, since a unique character for all tags of each MIOC is specified by the programmer. The unique tag capability allows any other macro routine in the operating system to be specialized into the same program. Each MIOC called into a given program must originate at the same memory location if they process a common MCA table.

MIOC SEGMENTATION

If the programs incorporating Logical I/O C are to be loaded from mass storage or tape, it is generally advantageous to use the segmentation option. If the programs are to be loaded from a card deck, it is suggested that the segmentation option not be used without first carefully reading the following paragraphs. Segmentation is accomplished by assigning any letter of the alphabet as the parameter 10 value of the MIOC macro call.

When segmentation is desired, the program using Logical I/O C must specify segment names to Mass Storage Easycode Assembler C. Then, during assembly of the segment that contains the MIOC macro call, Logical I/O C takes control of assembly segmentation until all the coding for the requested resident and nonresident functions has been generated. The coding for the resident functions is generated in the same segment of the program that contains the MIOC macro call. The coding for each nonresident function requested is generated in separate segments. Of this nonresident coding, the first segment is x1, where x is the letter assigned as the parameter 10 value of the MIOC macro call. The second segment may be x2, the third x3, etc., until all the nonresident function coding is generated. The last segment generated, always xZ, consists of any coding supplied by the programmer that follows the call for MIOC in the segment that contained the MIOC macro call. Segment xZ appears, regardless of whether coding supplied by the programmer followed the MIOC macro call in its respective segment. This means that if the segment containing the MIOC macro call contains coding after the MIOC call, this coding is assembled in a segment different than the original. For the names of MIOC segments and their respective positions on a binary run file, refer to Table 3-10.

If the call to the Supervisor to load the segment containing MIOC is made in the normal start mode, loading proceeds up to the end of the resident MIOC coding. At that point, there is an Execute statement generated at assembly time by MIOC. This statement causes control to be returned to a MIOC subroutine that requests the Supervisor to load the last MIOC segment, xZ, without altering any communication area fields other than the segment name field. When the

Supervisor completes this loading, control is returned to the location specified in the programmer's Execute (or END) statement for the segment containing the macro call for MIOC. Note that, in this case, the programmer cannot assume that his original segment name will be preserved in the Supervisor's communication area.

When the call to the Supervisor to load the segment containing the MIOC macro call is made in the return or special start mode, coding following the MIOC call is not loaded. When coding does follow the MIOC call in the segment containing the MIOC call, it is the programmer's responsibility to load that coding. This is accomplished by a request to load segment xZ.

For a description of the Supervisor's normal, return, and special starting modes, refer to the manual, Mod 1 (MSR) Supervisor (Order No. 616).

Figure 3-3 illustrates the principles of program segment loading by the Supervisor. In the normal starting mode, segment 01 would be loaded, followed by segment BZ. In the special or return starting mode, only segment 01 would be loaded. Note that B is assumed to be the value assigned to parameter 10 of the MIOC macro call and that the programmer-originated segment containing the MIOC macro call is defined as segment 01.

Table 3-10. MIOC Segmentation

If Parameter 10 Implies Segmentation, (P10 = x), the function:	Is Contained Within These Segment Names When:
<u>Open</u> VOLNAMES and VOLDESCR Processing VOLALLOC processing Direct access precalculations Indexed sequential precalculations Common subroutines	Parameter 15 = Δ Parameter 15 = COMBINE X1 X1 X2 X3 - omitted when parameter 4 = Δ X4 - omitted when parameter 6 = Δ X5 - omitted when parameter 6 = Δ X7
Set member End member	Included only when partitioned sequential files exist ----- Parameter 12 = Parameter 12 = COMBINE XC XC XS
Set location	Included only for sequential processing of indexed sequential files XE
Insert	Included only if segmentation of insert coding is specified (i. e., parameter 11 = segment) XG

Table 3-10 (cont). MIOC Segmentation

If Parameter 10 Implies Segmentation, (P10 = x), the function:	Is Contained Within These Segment Names When:
<u>Swap</u> Common subroutines Close volume processing Open volume processing	Included only if multivolume files exist (i. e., parameter 18 = MULTIVOL) XM XN {XO {XP
MALTER Release	Included only when partitioned sequential files exist XU XV
Close	XY
Remaining user coding	XZ
NOTE: Segments are found in alphanumeric order on a binary run file.	

MIOC RESTRICTIONS

To accomplish segment loading, MIOC must utilize certain fields of the Supervisor's communication area and make certain assumptions about other fields.

The following fields of the Supervisor's communication area are altered during the loading of nonresident functions. These fields are restored to their original values, however, as soon as a particular loading sequence is completed.

1. Segment name field: The segment name field (locations 112 and 113 octal) are altered to contain the segment name of the currently needed segment.
2. Location 152 (octal): Location 152 octal is altered, on the basis of the last requested nonresident function, to ensure that searching for the next requested nonresident function is performed most efficiently. This is done to ensure compatibility with the Mod 1 (TR) Operating System.
3. Start mode field: The start mode field (location 160 octal) is altered to the return start mode.
4. Search mode field: When the program's search mode includes visibility, Logical I/O C always searches by program and segment name and by visibility. When visibility is not included, Logical I/O C always searches by program and segment name. This is accomplished by preserving the leftmost bit of the search mode field (location 157 octal) and altering the five rightmost bits to indicate 20 (octal).

Certain assumptions are made by Logical I/O C concerning the contents of other fields of the Supervisor's communication area. These assumptions are included in the following list.

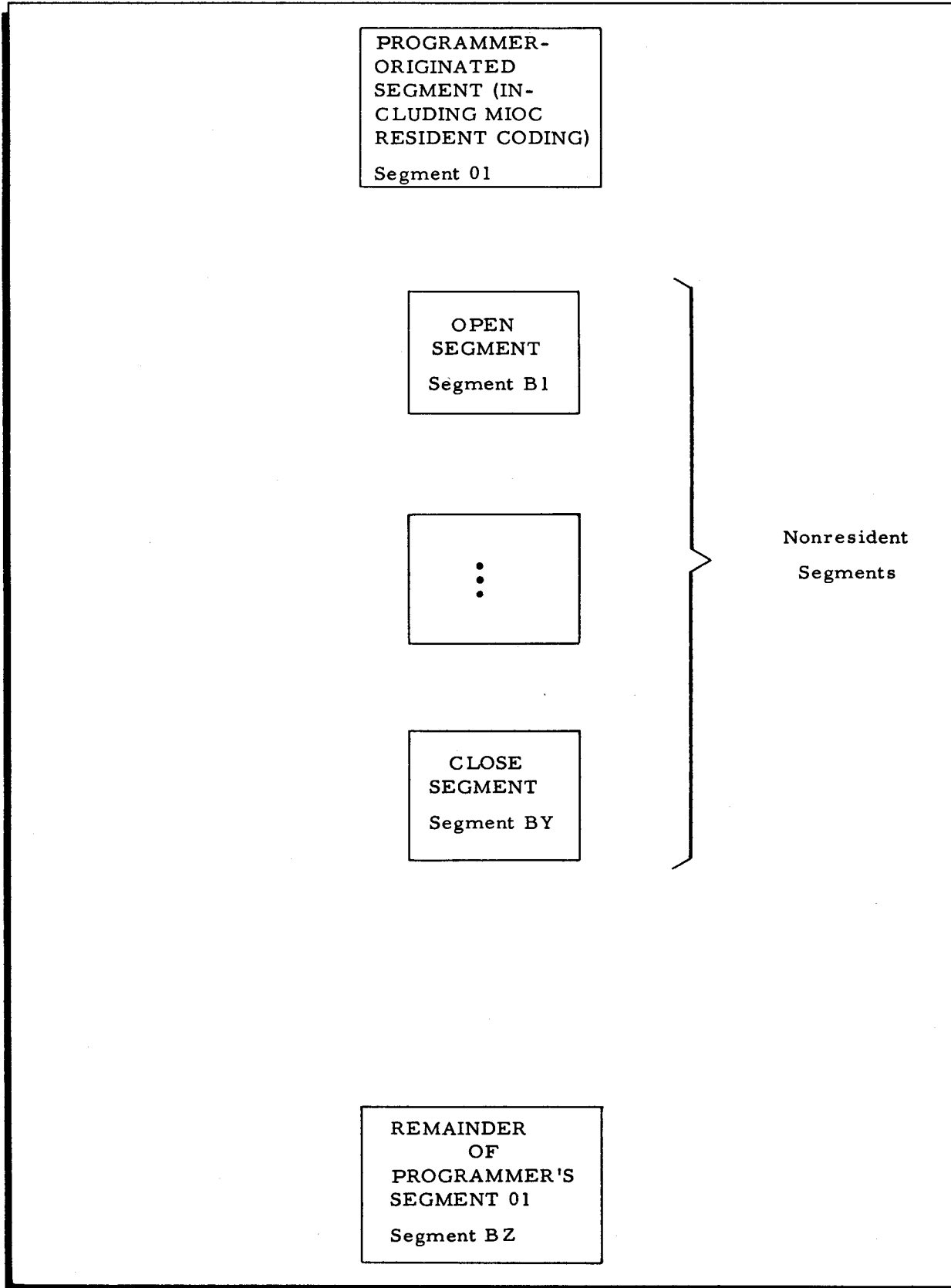


Figure 3-3. Program Segment Loading

1. Search mode field: The assumption is made that the search mode field (location 157 octal) contains a value other than 01 (visibility and relative position).
2. Program name field: Any time a nonresident function is requested, Logical I/O C assumes that the program name field (locations 104 through 111 octal) contains the program name that contains the current MIOC macro call.

PHYSICAL I/O C RELATIONSHIPS WITH MIOC

MIOC does not issue PDT or PCB instructions. Rather, it interfaces with the Physical I/O C program (MPIOC) which does issue such instructions. Normally, the programmer requests that MIOC call and utilize MPIOC. This request is made through parameter 50 of the MIOC macro call. When the programmer wants MIOC to call MPIOC, he must specify, via parameters 51 through 54 of MIOC, the specialization of MPIOC that he wants. In some cases, however, the programmer may want to call MPIOC himself. In this case, he assigns the value "PRESENT" to parameter 50 of MIOC; he must also specify parameters 51 through 54.

When parameter 52 is equal to or less than 07, a 56 is generated. When parameter 52 is greater than 07, a 76 is generated. This ensures that all channels for the appropriate I/O sector are used. Note that when a Type 257A Control is used, a 53 is generated.

PHYSICAL I/O C RELATIONSHIPS WITH MCA

The programmer is required to have one MCA for every file he intends to process in a given program. Each MCA macro automatically generates a Physical I/O C communication area macro call (MPCA). The programmer may desire to interrogate some of the fields in the MPCA; he does this by writing an MUCA macro call. Because the MCA macro routine uses the MPCA exclusively, the programmer should never attempt to alter the contents of any of its fields (other than those listed in Table 3-7.)

Address Mode

The address mode for all Logical I/O C macros must be the same. Also, each time the programmer enters Logical I/O C through a macro call or Logical I/O C returns to the programmer (normally through an exit) from a macro routine, the address mode must be the same as that of the macro calls. Furthermore, the address mode of an MPIOC that has been called by the user must be the same as that of any Logical I/O macros associated with that MPIOC.

Index Registers

MIOC, together with MPIOC, uses and restores index registers X3, X4, X5, and X6. These registers are restored to their original values whenever a return from Logical I/O C is

made to the user's coding. It does not matter whether the coding is in the main line of the program or in an exit routine. Index registers X3 and X4 are restored at the last possible moment before the return is made. Hence, they should not be used as a linkage parameter (parameters 13 and 14 of MCA) to MCA. Index registers X5 and X6 can be used as linkage parameters, however, since they are restored earlier.

Index registers are saved and restored with MCW's. The MCW is performed between respective registers and the DSA fields in MIOC. The length of the DSA fields is consistent with the current addressing mode. MIOC sets its own index register values with LCA instructions. Because of this, the programmer should always punctuate the registers in the normal manner, viz., word marks should be placed in locations 10, 14, 18, and 22 in the 3-character addressing mode and in locations 9, 13, 17, and 21 in the 4-character addressing mode. The permanence of any other punctuation cannot be guaranteed.

Read/Write Channel Utilization

Two data transfer rates are applicable to mass storage devices. When Type 258, 259, or 273 Disk Pack Drives and Type 261 or Type 262 Disk Files are used, data transfer rates accomplished by interlocking at least 1-1/2 channels (such as 1A and 3 or 4A and 6) are required.

When Types 155, 259A, or 259B Disk Pack Drives are used, a single interlocked channel suffices.

In the absence of any other directive, Logical I/O C utilizes channels 2 and 3 or channels 5 and 6 (depending upon the I/O sector) when operating with Type 258, 259, or 273 Disk Pack Drives; alternatively, it utilizes channel 3 or 6 when operating with Types 155, 259A, or 259B Disk Pack Drives.

The user can change this assumption by setting parameter 54 of the MIOC macro call to M and by specifying RWC as the communication area field designator in an MLCA macro call (see Table 3-8). This action should be performed prior to opening the file. The RWC value entered by means of the MLCA macro call must include channel 3 (for I/O sector 0) or channel 6 (for I/O sector 1). Permissible RWC values are shown as follows.

I/O Sector 0	I/O Sector 1
53	73
54	74
55	75
56	76

Direct Access Addressing

Direct access bucket addresses can be relative or actual. A relative bucket address is one in which the address is the same as its ordinal numeric position from the beginning of the file. In

this case, the first bucket in the file is numbered 0000 (in a 4-character binary field) and each following bucket increments this number by a binary one. An actual bucket address is one that is the exact mass storage address of the first record of the bucket. When actual bucket addresses are used in processing a multivolume direct access file, all volumes of the file must be mounted on the same peripheral control unit. When a bucket address is not included in an action macro call, the address of the bucket used in the last action macro call (either explicitly or implicitly) is used again.

The programmer must generate a field in which bucket addresses are stored. Bucket addresses are then delivered to Logical I/O C from this field, whose rightmost location is specified by parameter 02 of the action macro call. This field can have either of the following octal formats.

1. Relative address field: This field must have four character positions, and the leftmost of these must be word-marked. This field contains the exact sequence number of the bucket within the file. The sequence number of the bucket is binary.
2. Actual address field: This field must have eight character positions, and the leftmost of these must be word-marked. This field contains the address of the first record in the desired bucket. The record address is in the form DPCCTRR,

D = device number,
P = 0,
CC = cylinder number,
TT = track number, and
RR = record number.

NOTE: If the actual address is obtained from the RIC field of the MCA communication area following execution of the GET function, the actual address may be invalid if the block is on a substitute track. (See page 3-54.) This results in the normal action taken for an invalid bucket address.

Item Key Specification

For the get macro routine to retrieve an item, the item must contain an identifying key. This key is specified by the programmer. The length and location within the item are specified when the direct access file is allocated. This information is placed in the file description portion (*VOLDESCR*) of the volume directory. When an open function is issued, Logical I/O C retrieves these fields from *VOLDESCR*.

DIRECT ACCESS

The address of the rightmost location of a field that contains the desired key value is specified by parameter 03 of the get action macro call. When items are to be retrieved by searching for the correct item key, parameter 03 of the get action macro call must be specified. The field that contains the key value is set up by the programmer and must contain a word mark in its leftmost location. The corresponding key field within the item in the buffer cannot contain a word mark; yet, if desired, the leftmost character of the item key field may contain a word

mark. The word mark set up by the programmer in the key-value field terminates the operation when the key-value field and the item key field are compared.

INDEXED SEQUENTIAL

In indexed sequential processing, when an item key is specified in an action macro call, the address of the rightmost location is specified by parameter 02. The field that contains the key value set up by the programmer must contain a word mark in its leftmost location. Word marks cannot exist in the buffer at the time an action macro is executed. The word mark set up by the programmer in the key-value field terminates the operation when the key-value field and the item key field are compared.

Exits and Halts

There are four exits associated with MCA. They are summarized in tabular form in Tables 3-11 through 3-14. Each exit pertains to a specific area of Logical I/O C processing. These exits are specified in parameters 40, 41, 43, and 44 of MCA as follows:

1. Parameter 40 - volume directory exit,
2. Parameter 41 - index exit,
3. Parameter 43 - data exit, and
4. Parameter 44 - device exit.

As explained in Note 7 of Table 3-5, four exits are associated with Logical I/O C. Each of these exits relates to a specific area of Logical I/O C processing. Since an exit may be taken for one of a variety of reasons, a code is provided in a single user-provided character one memory location less than the user's entrance point for each exit routine. The user may interrogate this code for equality to a subset of the total number of values possible for a given exit. When an equality does not exist, i.e., when the user has no interest in acting upon the particular situation indicated by the current code, the user may return to Logical I/O C with a request that it handle the situation as it normally would, had the exit not been specified. Namely, it can continue processing in some cases, or it can notify the operator (either through a control panel or console) of the condition and allow him to take appropriate action. When an equality does exist (i.e., a situation exists for which the user has provided a programmed solution), he returns to Logical I/O C with a request that it proceed in a particular direction. The user makes return requests by placing a return code in the same user-provided location, as described above. Sometimes an exit is taken because of a situation which causes Logical I/O C to anticipate no return.

For example, suppose that a programmer wants to specify a device exit (parameter 44 of MCA) only to reattempt to correct read and write errors. The exit code for the read error is 06; the exit code for the write error is 10 (an unsuccessful write verification). The programmer can specify one of three return codes to Logical I/O C. A return code of 21 means that Logical I/O C is to automatically reattempt to correct the error. A return code of 52 means that Logical I/O C is to ignore the error and continue processing, if possible. A return code of 40 means halt. The following coding illustrates the example described above.

SECTION III. LOGICAL I/O C

CARD NUMBER	Y X R	LOCATION	OPERATION CODE	OPERANDS	
				14 15	20 21
1	*	CALL TO MCA			
2	C	FL1	MCA	p01, p02, p03, . . .	
3	L	44	DEXIT,		
4	*	USER EXIT ROUTINE			
5	*				
6	*	WHEN THIS ROUTINE IS ENTERED, THE FOLLOWING DCW WILL			
7	*	CONTAIN THE EXIT CODE.			
8	*				
9	*	WHEN RETURN TO THE I/O IS MADE, THE SAME DCW WILL			
10	*	CONTAIN A CODE SPECIFYING THE DESIRED ACTION.			
11	*				
12	*				
13		DCI	DCW	#1B0.	
14		DEXIT	SCR	MYRT, 70	SAVE RETURN
15			BCE	RDER, DCI, 06	READ ERROR
16			BCE	WTER, DCI, 10	WRITE ERROR
17			MCW	#1C40, DCI	HAVE I/O MESSAGE
18		AMYRT	B	0	
19		RDER	EQU	*	
20		WTER	MCW	#1C21, DCI	REQUEST RE-ATTEMPT
21			B	MYRT-LA	LA IS THE LENGTH OF AN ADDRESS
22			NOP		
23					

Table 3-11. Exit and Return Codes for Volume Directory Exits

Exit Code	Reason for Exit	Return Code	Return Code Meaning
01	The volume directory description (*VOLDESCR*) for the file in the first file volume opened has been read into memory by the open function and can now be interrogated. APD points to the left end of the entry.	10	Continue processing.
		21	Reopen the file.
03 *	The file name cannot be located in *VOLNAMES* by the open function.	40	Halt or typewriter message.
		21	Reopen the file volume.
04 *	The units-of-allocation table set up by the programmer is not large enough to hold all the units of allocation for this file.	40	Halt or typewriter message.
		21	Reopen the file volume.
05 *	A discrepancy exists in *VOLALLOC* for this file.	40	Halt or typewriter message.
		21	Reopen the file volume.
11 *	At the end of file or file-volume processing (after the close function reads *VOLDESCR* in memory and before it writes it back onto mass storage), *VOLDESCR* can be interrogated. APD points to the left end of the entry.	10	Continue processing.
13 *	The open function is attempting to process a file volume whose sequence number is not one greater than the last file volume processed, or one whose sequence number is not zero during the open function for the first file volume of a direct access or	40	Halt or typewriter message.
		21	Reopen the file volume.

SECTION III. LOGICAL I/O C

Table 3-11 (cont). Exit and Return Codes for Volume Directory Exits

Exit Code	Reason for Exit	Return Code	Return Code Meaning
13 * (cont)	indexed sequential file. The open function cannot continue until the volume sequence number is corrected.		
14 *	When this file was allocated, a password was specified. This password provided by this program is not correct.	40	Halt or typewriter message.
		21	Reopen the file volume.
21	The *VOLDESCR* entry for a file volume (other than the first of a multi-volume file) has been read into memory by the open function and can be interrogated. APD points to the left end of the entry.	10	Continue processing.
		21	Reopen the file volume.
		42	Halt or typewriter message; a new file volume is open.
23 *	The open function is attempting to open a file that does not have a legitimate Mod 1 (MSR) file organization.	40	Halt or typewriter message.
		21	Reopen the file volume.
24	When this file was allocated, a password was specified and there is no password check requested in the MCA for this file.	40	Halt or typewriter message.
		21	Reopen the file volume.
33	A file is being opened, and the open function has reached the end of the device table without being able to open all the required file volumes of the file.	40	Halt or typewriter message.
		21	Reopen the file.
34 **	A new file volume for a multivolume file needs to be opened, and no more entries remain in the device address table. The number of additional volumes to be mounted should equal the number of entries in the device address table or those remaining in the file before continuation is requested. Not included in this comparison are those devices reserved for the indexed sequential master/cylinder index and general overflow volumes.	40	Halt or typewriter message.
		11	Continue processing.
43	A sequential file is being opened for input-only or input/output processing, and its sequential number is not zero.	40	Halt or typewriter message.
		21	Reopen the file.
		52	Continue processing if processing was to begin on other than the first volume.
53 *	The open function is processing a file volume as an input-only or input/output file whose data status indicator specifies that there is no data on the file volume.	40	Halt or typewriter message.
		21	Reopen the file volume.
<p>NOTES: * Logical I/O C executes a swap function between file volumes which operates similarly to a close function followed by an open function. This function applies only to sequential processing of all file types. Exit codes shown with "*" contain the phrase "open function" or "close function" to refer to an action resulting from the MSOPEN or MSCLOS macro calls or to the analogous swapping function. The swapping function is internal to Logical I/O C.</p> <p>** Exit codes shown with "**" apply only to the swapping function.</p>			

SECTION III. LOGICAL I/O C

Table 3-12. Exit and Return Codes for Member Index Exits

Exit Code	Reason for Exit	Return Code	Return Code Meaning
03	The set member function (SETM) cannot locate the specified member in the member index.	40	Halt or typewriter message.
		No return	Issue new action, e.g., SETM to another member.
13	The alter member function (MALTER) cannot locate the specified member in this file.	40	Halt or typewriter message.
		No return	Issue new action, e.g., SETM to another member.
04	The set member function (SETM) has been requested to create a new member, but there is no room in the member index for another entry.	40	Halt or typewriter message.
		No return	Issue new action, e.g., SETM to another member.
14	The set member function (SETM) has been requested to set the processing mode of an existing member to the output-only mode, but the status of the member makes it unavailable for output-only processing.	40	Halt or typewriter message.
		No return	Issue new action, e.g., SETM to another member.
24	The alter member function (MALTER) has been requested to delete a member whose status makes it unavailable for output-only processing.	40	Halt or typewriter message.
		No return	Issue new action, e.g., SETM to another member.

Table 3-13. Exit and Return Codes for Data Exits

Exit Code	Reason for Exit	Return Code	Return Code Meaning
01	The MSGET macro call has been issued, and an end of file condition has been detected.	40	Halt or typewriter message.
		No return	Issue new action to continue processing.
11	The MSPUT macro call has been issued, and there is no more room in the file for another item.	40	Halt or typewriter message.
		No return	Issue new action to continue processing.
12	The buffer contains a data block which will be altered during processing by the MSINS macro routine. (This is the first of two exits specifically requested by parameter 46 of MIOC to be taken during processing of each data block of an indexed sequential file insert.)	10	Continue processing.
22	The buffer contains a data block which has been altered by the MSINS routine and will be written back into an indexed sequential file when processing continues. (This is the second of two special exits. See above.)	10	Continue processing.
34	The SETM macro routine has been requested to create a new member, and there is no room in the file (no data blocks remain in the unused area) for a new member.	40	Halt or typewriter message.
		No return	Issue new action to continue processing.

SECTION III. LOGICAL I/O C

Table 3-13 (cont). Exit and Return Codes for Data Exits

Exit Code	Reason for Exit	Return Code	Return Code Meaning
03	The MSGET macro routine cannot locate the specified item key.	40	Halt or typewriter message.
		No return	Issue new action to continue processing.
13	The MSINS macro routine cannot locate an available item position. The insert item has not been placed in the file.	40	Halt or typewriter message.
		No return	Issue new action to continue processing.
04	An invalid bucket address has been specified to the current direct access function.	40	Halt or typewriter message.
		No return	Issue new action to continue processing.
23	An item has been inserted in an indexed sequential file, and all the overflow areas are full. Consequently, the last item previously in the general overflow area has been shifted off the file. APD points to the left end of this item.	40	Halt or typewriter message.
		No return	Issue new action to continue processing.
14	Key verification has failed while replacing an item in an indexed sequential file.	40	Halt or typewriter message.
		No return	Issue new action to continue processing.
24	A duplicate item key has been detected while inserting an item into an indexed sequential file.	40	Halt or typewriter message.
		No return	Issue new action to continue processing.
NOTE: The data exit must be specified, unless Logical I/O C never reaches a situation described as a reason for exit, i. e., a situation in the "Reason for Exit" column above.			

Table 3-14. Exit and Return Codes for Device Exits

Exit Code	Reason for Exit
01	Device inoperable.
02	Protection violation.
03	Device error (after five attempts to reposition the device).
04	Possible device failure.
05	The addressed record cannot be located (after five attempts).
06	Uncorrectable read error. The data, however, has been transferred erroneously after ten attempts.
07	Uncorrectable read error. The data, however, has not been transferred after ten attempts. (The header may contain a read error.)
10	Uncorrectable write error. The last write could not be verified after ten attempts.
11	A track-linking record has been read into memory.
12	The attempt to link to the next track in this file has not been completed after ten attempts.

The following return codes are applicable to all device error exits.

- 21 - Reattempt the operation that caused this error.
- 52 - Ignore the error and continue processing if possible.
- 40 - Halt or typewriter message.

OPERATING PROCEDURES FOR LOGICAL I/O C

To communicate with the operator, Logical I/O C (1) halts with the B-address and A-address registers displaying error information as described below or (2) pauses with messages at the console typewriter.

Control Panel Operating Procedures

At the control panel, whenever Logical I/O C halts, the B-address register contains a code describing generally what problem has occurred. For example, a code of 0401d (where d = the device number) indicates that the problem is related to the open or close function. The operator will often be able to locate and correct the error condition with no need for more information than is contained in the B-address register. However, when he does need more information, the operator can consult the A-address register. This register is set to the beginning of a communication area containing the following fields:

- | | |
|--|---|
| 1. Response field | A 1-character field into which the operator is asked to key a code indicating to Logical I/O C which action it is to take. |
| 2. Specific code field | A 1-character field containing, at the time of the halt, a code indicating the exact nature of the problem for which the halt occurred. Values for this code are found in Table 3-15. |
| 3. File name field | A 10-character field that contains the name of the file being processed when the error occurred. |
| 4. Relative volume number field | A 1-character field containing the relative volume number. The first volume is relative volume zero. |
| 5. Volume name field | A 6-character field containing the name of the next file-volume. |
| 6. Peripheral address assignment field | A 1-character field specifying (in octal) the control unit currently active for the file named in the file name field. |
| 7. Device number field | A 1-character field specifying (in octal) the file's current device number. |
| 8. Pack number field | A 1-character field specifying (in octal) the file's current pack number. |
| 9. Mass storage address | A 6-character field specifying, in binary, the mass storage address (CCTRR), for device error only. |

Specific control panel halt codes for Logical I/O C are listed in Table 3-15.

SECTION III. LOGICAL I/O C

Table 3-15. Halt Codes for Logical I/O C

B-Address Register Value	Specific Code	Condition	Operator Action
0401d		There is a discrepancy in the volume directory which the open function cannot correct; or the Swap function has finished processing a volume or all volumes in a device table.	Inspect device and when possible correct the problem. Usually, the wrong volume will be mounted.
	03	The specified file has not been found.	
	04	There are more units of allocation for the current file than the program provides for in the units of allocation table.	To reopen the file-volume or, in the case of specific codes 21 and 34, to continue processing, enter an octal 27 (G) into the response field (location specified by the A-address register) and press RUN.
	14	A password check, requested by the program, has failed.	To exit to the Supervisor's emergency return address, enter an octal 25 (E) into the response field and press RUN.
	21	A new file volume is being opened. The processing of the previous volume is complete.	
	23	The open function is attempting to open a file that does not have a legitimate Mod 1 (MSR) file organization.	
	24	A password exists on the specified file and password checking has not been requested by the program.	
	05	An uncorrectable condition has arisen in the units of allocation portion of the volume directory.	
	13	The open function has encountered a file volume whose sequence number is not one greater than the last.	
	33	The open function has reached the end of the device address table without opening all of the required file volumes.	
	34	A new file volume is required to be opened, and no entries remain in the device address table.	

SECTION III. LOGICAL I/O C

Table 3-15 (cont). Halt Codes for Logical I/O C

B-Address Register Value	Specific Code	Condition	Operator Action
0401d (cont)	43	The open function is opening a sequential file, and the volume sequence number is not zero.	
	53	The open function is processing a file volume as input-only or as input/output, and the file's data status character indicates that no data exists on the file-volume.	
0410d		An uncorrectable condition has arisen during processing of a member index which precludes any further processing.	No corrective action is possible. To exit to the Supervisor's emergency exit address, enter an octal 25 (E) into the response field (location specified by the A-address register) and press RUN.
	03	The set member function is unable to locate the requested member.	
	13	The alter member function cannot locate the requested member.	
	04	There is no space available for the creating of a new member index entry in the member index.	
	14	The set member function has been requested to process in the output-only mode a member whose status is unavailable for output-only processing.	
	24	The alter member function has been requested to delete a member that is unavailable for output-only processing.	
0430d		An uncorrectable condition has arisen within the data portion of a file or member.	No corrective action is possible. To exit to the Supervisor's emergency exit address, enter an octal 25 (E) into the response field (location specified by the A-address register) and press RUN.
	01	End of data has been reached on an input file.	
	11	There is no space in a sequential output file for another item.	

SECTION III. LOGICAL I/O C

Table 3-15 (cont). Halt Codes for Logical I/O C

B-Address Register Value	Specific Code	Condition	Operator Action
0430d (cont)	34 03 13 04 23 14 24	The set member function has been requested to create a new member. There is no space remaining in the file's unused area. The get function cannot locate an item with a specified key. The insert function is not able to locate an available item position. An invalid bucket address has been specified for a direct access function. Indexed sequential insert has inserted an item and found all overflow areas to be full. Indexed sequential key verification failed while replacing an item. Indexed sequential insert has discovered a duplicate item key while inserting an item.	
0440d	05	An action macro call has been issued for a function whose coding was not requested for this Logical I/O C specialization.	No corrective action is possible. To exit to the Supervisor's emergency exit address, enter an octal 25 (E) into the response field (location specified by the A-address register) and press RUN.
Oppxd	For specific halt codes, see following B-address register values.	An error condition (x) has arisen on device (d) of the mass storage control (pp); pp is less than 40 (octal).	Inspect the device and control unit and, when possible, correct the problem. Choose one of the following corrective actions. To reattempt automatic correction of the problem, enter an octal 27 (G) into the response field (location specified by the A-address register) and press RUN.

SECTION III. LOGICAL I/O C

Table 3-15 (cont). Halt Codes for Logical I/O C

B-Address Register Value	Specific Code	Condition	Operator Action
Oppxd (cont)			Key an octal 21 (A) into the response field and press RUN to ignore the problem and continue processing. This action is not recommended. Key an octal 25 (E) into the response field and press RUN to exit to the Supervisor's emergency return exit (location 213 octal).
Opp0d		The specified device is not available.	Verify that the device is powered up and protection switches on the control unit are set properly.
	01	Device is inoperable.	For possible operator actions, see Oppxd above.
	02	Protection violation.	
Opp1d		An uncorrectable read error has been encountered.	For possible operator actions, see Oppxd above.
	06	The read error is in the data portion of a record. Data transfer has been completed.	
	07	The read error might be in the header portion of the record. Data transfer is not completed.	
	12	The read error is in a track-linking record.	
Opp2d		An uncorrectable write error has occurred.	For possible operator actions, see Oppxd above.
	04	Possible device failure (format write).	
	10	A write verification error has occurred (see Note 1).	

Table 3-15 (cont). Halt Codes for Logical I/O C

B-Address Register Value	Specific Code	Condition	Operator Action
Opp4d	03	A positioning or addressing error has occurred.	For possible operator actions, see Oppxd above.
	05	Device error (unable to position to the requested cylinder). The addressed record cannot be located. Five attempts have been made.	
Opp7d		Miscellaneous condition.	For possible operator actions, see Oppxd above.
	11	Possible device failure.	

¹The following two conditions apply to write errors during the allocate function of File Support C.

- a. If *BADTRACKS and *VOLSPARES files have not been created on the volume and a defective track is encountered (write error), the following message appears on the printer: CYLINDER nnn TRACK nnn ERROR (nnn is a decimal value). The file must be reallocated around the defective track. If *BADTRACKS and *VOLSPARES files are later added to the volume, a substitute track can then be established for the defective track.
- b. If *BADTRACKS and *VOLSPARES files have been created on the volume and an unusable track is encountered (write error), the following message appears on the printer: CYLINDER nnn TRACK nnn UNUSABLE. An unusable track (a very unlikely possibility) has a bad surface and not even one bad-track track-linking record can be read from it. The file must be reallocated around the unusable track. Track substitution is not possible.

Console Typewriter Operating Procedures

When a console typewriter message indicates an error or requests operator action, the operator performs the following steps:

1. Read the typeout. (To repeat the message, press the space bar twice.)
If necessary, consult the manual for possible action.
2. Perform the desired corrective action.
3. Type the appropriate 1-character response (G, E, etc.).
4. If the typein is correct, press the space bar to continue. If incorrect, type any other character and return to step 3.

The first line of messages issued by Logical I/O C is divided into two categories: peripheral device condition messages and file I/O condition messages.

The first line of a file I/O condition message has the following format.

pp d FILE file-name description

The first line of the peripheral device condition is:

pp d description

pp d gives the peripheral control unit (pp) and device number (d) of the peripheral device upon which the condition exists. The value of pp is less than 40 (octal).

file name is the 10-character name of the file upon which the condition occurred.

description is a message describing the error condition (see Table 3-16).

The second line of all messages has the format:

c file-name v volume p d m a

c is a 1-character code indicating the exact nature of the problem (the specific code).

(

(

x

(

SECTION III. LOGICAL I/O C

file-name is the 10-character name of the file containing the error.

v is the relative number of the volume. The first volume is considered relative 0.

volume is the 6-character name of the next file volume.

p is the number of the control unit containing the device upon which the condition exists.

d is the device number of the current file.

m is the number of the current disk pack.

a is the mass storage address (in binary).

The error code and all succeeding information is typed out on the console as a supplementary list if the console typewriter is being used.

Number of Characters	Character-field Location (left)	Explanation
1	A	Response character.
1	A + 1	Error code (applied to mass storage peripheral device or file condition).
10	A + 2	Mass storage file name.
1	A + 12	Relative volume number of the mass storage file.
6	A + 13	Volume name.
1	A + 19	Mass storage peripheral control unit address.
1	A + 20	Mass storage device address.
1	A + 21	Mass storage pack number.
6	A + 22	Mass storage address in binary (CCTTRR), for device error only.

The descriptive messages possible and the specific codes are given in Table 3-16.

Table 3-16. Console Typewriter Pause Codes and Messages for Logical I/O C

Descriptive Message	Specific Code (Alphanumeric)	Condition	Operator Action
FILE NOT FOUND	3	There is a discrepancy in the volume directory which the open function cannot correct, or the swap function has finished processing a volume or all volumes in the device table. The specified file has not been found.	Inspect device and when possible correct the problem. Usually, the wrong volume is mounted. To reopen the file volume or, in the case of specific codes 21 and 34, to continue processing, type G and confirm. To exit to the Supervisor's emergency return address, type E and confirm.

SECTION III. LOGICAL I/O C

Table 3-16 (cont). Console Typewriter Pause Codes and Messages for Logical I/O C

Descriptive Message	Specific Code (Alphanumeric)	Condition	Operator Action
U-A TABLE TOO SMALL	4	There are more units of allocation for the current file than the program provides for in the units of allocation table.	
ERROR IN *VOLALLOC*	5	An uncorrectable condition has arisen in the units of allocation portion of the volume directory.	
VOLUME SEQUENCE NUMBER ERROR	=	The open function has encountered a file-volume whose sequence number is not one greater than the last.	
PASSWORD ERROR	:	A password check, requested by the program, has failed.	
DISMOUNT PREVIOUS VOLUME	A	A new file-volume is being opened. The processing of the previous volume is complete.	
PASSWORD ERROR	D	A password exists on the specified file and password checking has not been requested by the program.	
DEVICE TABLE TOO SMALL	.	The open function has reached the end of the device table without opening all of the required file-volumes.	
MOUNT NEXT VOLUMES)	A new file-volume is required to be opened and no entries remain in the device address table.	
VOLUME SEQUENCE NUMBER ERROR	L	The open function is opening a sequential file and the volume sequence number is not zero.	
NO DATA ON FILE-VOLUME	\$	The open function is processing a file-volume as input-only or as input/output, and the file's data status character indicates that no data exists on the file-volume.	

SECTION III. LOGICAL I/O C

Table 3-16 (cont). Console Typewriter Pause Codes and Messages for Logical I/O C

Descriptive Message	Specific Code (Alphanumeric)	Condition	Operator Action
MEMBER NOT FOUND	3	An uncorrectable condition has arisen during processing of a member index which precludes any further processing. The set member function is unable to locate the requested member.	No corrective action is possible. Type E and confirm to exit to the Supervisor's emergency return address.
MEMBER INDEX FULL	4	There is no space available for the creation of a new member index entry in the member index.	
MEMBER NOT FOUND	=	The alter member function cannot locate the requested member.	
MEMBER CANNOT BE OUTPUT ONLY	:	The set member function has been requested to process in the output-only mode a member whose status is unavailable for output-only processing.	
MEMBER CANNOT BE DELETED	D	The alter member function has been requested to delete a member that is unavailable for output-only processing.	
END FILE (INPUT)	1	An uncorrectable condition has arisen within the data portion of a file or member. End of data has been reached on an input file.	No corrective action is possible. Type E to exit to the Supervisor's emergency return address.
ITEM NOT FOUND	3	The get function cannot locate an item with a specified key.	
INVALID BUCKET	4	An invalid bucket address has been specified for a direct access item.	
END FILE (OUTPUT)	9	There is no space in a sequential output file for another item.	
NO SPACE TO INSERT ITEM	=	The insert item is not able to locate an available item position.	

SECTION III. LOGICAL I/O C

Table 3-16 (cont). Console Typewriter Pause Codes and Messages for Logical I/O C

Descriptive Message	Specific Code (Alphanumeric)	Condition	Operator Action
KEY VERIFICATION FAILURE NO SPACE FOR MOVED ITEM DUPLICATE ITEM NO SPACE FOR NEW MEMBER	: C D)	Indexed sequential key verification failed while replacing an item. Indexed sequential insert has inserted an item and found all overflow areas to be full. Indexed sequential insert has discovered a duplicate item key while inserting an item. The set member function has been requested to create a new member. There is no space remaining in the file's unused area.	
INVALID ACTION INOPERABLE INOPERABLE READ ERROR	5 1 2 6	An action macro call has been issued for a function whose coding was not requested for this Logical I/O C specialization. An error condition has arisen on device (d) of the mass storage control (pp); pp is less than 40 (octal). Device is inoperable. Protection violation. A read error has occurred in the data portion of a record. Data transfer has been completed.	No corrective action is possible. Type E to exit to the Supervisor's emergency return address. Inspect the device and control unit and, when possible, correct the problem. Choose one of the following corrective actions. To reattempt automatic correction of the problem, type G. To continue processing and to ignore the problem type A. This action is not recommended. To exit to the Supervisor's emergency return address, type E.

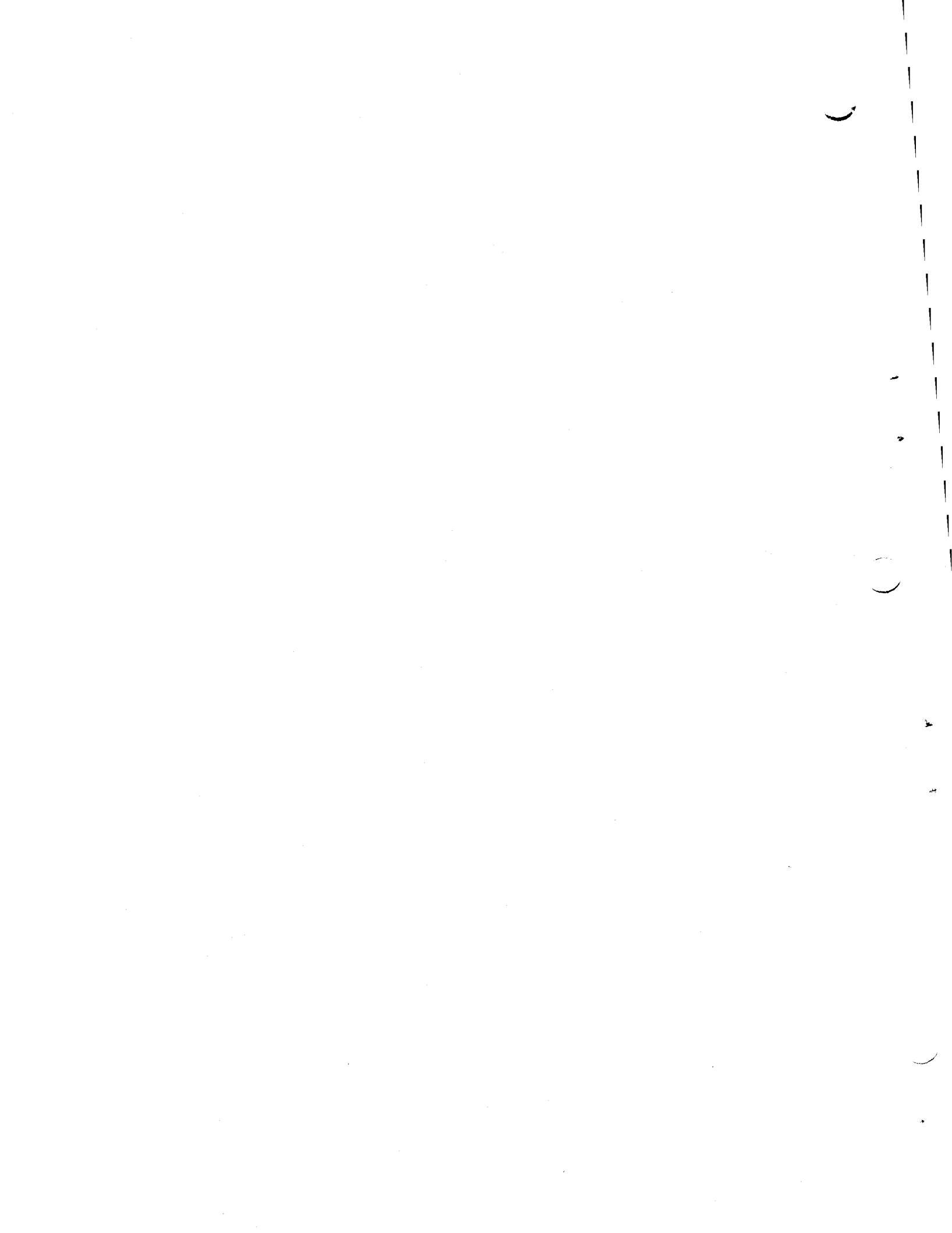
SECTION III. LOGICAL I/O C

Table 3-16 (cont). Console Typewriter Pause Codes and Messages for Logical I/O C

Descriptive Message	Specific Code (Alphanumeric)	Condition	Operator Action
READ ERROR	7	A read error has occurred which is probably in the header portion of a record. Data transfer has not been completed.	
READ ERROR	1	A read error has occurred in a track linking record.	
WRITE ERROR	4	A possible device failure has occurred (format write).	
WRITE ERROR	8	A write verification error has occurred (see Note 1).	
POSITIONING ERROR	3	A device error has occurred (unable to position the requested cylinder).	
POSITIONING ERROR	5	The addressed record cannot be located. Five attempts have been made.	
MISCELLANEOUS	9	Possible device failure.	

¹The following two conditions apply to write errors during the allocate function of File Support C.

- a. If *BADTRACKS and *VOLSPARES files have not been created on the volume and a defective track is encountered (write error), the following message appears on the printer: CYLINDER nnn TRACK nnn ERROR (nnn is a decimal value). The file must be reallocated around the defective track. If *BADTRACKS and *VOLSPARES files are later added to the volume, a substitute track can then be established for the defective track.
- b. If *BADTRACKS and *VOLSPARES files have been created on the volume and an unusable track is encountered (write error), the following message appears on the printer: CYLINDER nnn TRACK nnn UNUSABLE. An unusable track (a very unlikely possibility) has a bad surface and not even one bad-track-linking record can be read from it. The file must be reallocated around the unusable track. Track substitution is not possible.



SECTION IV
FILE SUPPORT C

GENERAL DESCRIPTION OF FILE SUPPORT C

File Support C is a set of routines that perform frequently desired functions on files resident on mass storage. The functions performed by the File Support C routines are:

1. Allocation of files on mass storage volumes,
2. Deallocation of files resident on mass storage volumes,
3. Loading files onto mass storage volumes,
4. Unloading files from mass storage volumes, and
5. Mapping the contents of the volume directory.

The allocate function is used by the programmer to assign a file to specified areas of one or more volumes and to update each volume directory accordingly. This function also formats and initializes a newly allocated file automatically. The deallocate function removes all volume directory entries for a file. This makes all areas used by this file available for future allocation. The load function is used by the programmer to load a mass storage file from cards, tape, or another mass storage file. The unload function is used to unload a file from mass storage onto cards, tape, printer, or another mass storage file. The map function is used by the programmer to obtain printed listings based on the contents of a volume directory. The information can be listed either on an online printer or on a print-image tape.

NOTE: When loading or unloading one mass storage file to another, the files must be of the same organization, with the following exceptions: the input can be a sequential file and the output can be an indexed sequential file, or the input can be a sequential file and the output can be a direct access file.

All File Support C routines are automatically specialized at execution time. The specialization is based on parameters supplied by the programmer in the job control statements. Therefore, it is not necessary for the programmer to perform an assembly operation to specialize these routines.

FOREGROUND/BACKGROUND PROCESSING OF FILE SUPPORT C

All File Support C processing functions can be used as background programs in a multi-programming foreground/background environment, except for the allocate function, which is incompatible with interruption by the foreground program.

FUNCTIONS OF FILE SUPPORT C

Allocate

The allocate function is used to assign a file to one or more specified areas of mass storage. Every file to be stored on mass storage must be allocated before it can be used. The allocate function checks the areas of each volume specified for the file to ensure that no other file occupies any of the specified area. The allocate function also updates the volume directory of each volume being used to include entries for the new file. All tracks for this file are formatted to the requested record size and are initialized according to the requirements of the file organization.

Deallocate

The deallocate function is used to free allocated areas on one or more volumes so that other files can be allocated to these areas. Before a file is deallocated, checks are made on the volume name, the file expiration date, and the password for the file. This is done to ensure that a file which has not expired or which is protected by a password is not removed from its volume(s) inadvertently. Directory item space freed by deallocation will be utilized for subsequent allocation.

Load

The load function is used to load data onto a mass storage file from punched cards, magnetic tape, or another mass storage file. All standard fixed-length card and tape formats can be used with the load function. Multireel magnetic tape files and multivolume mass storage files can be handled.

Unload

The unload function is used to unload data from a mass storage file onto punched cards, magnetic tape, printer, or another mass storage file. Multireel magnetic tape files and multivolume mass storage files can be handled.

Map

The map function is used to extract selected information about the files on a volume. This function can be used to produce a description of all or only specified files on a volume, a description of expired files, or a map of the unassigned tracks on a volume. The information is taken from the contents of the volume directory and is listed on a printer or on a print-image tape. Samples of printer listings produced by the map function are shown below.

MAP DESCRIPTION OF A FILE

A description of a file's structure (and other selected information) can be listed. A description of one or more specified files or of all files on a volume can be produced.

SECTION IV. FILE SUPPORT C

COMPLETE LISTING FOR VOLUME VOLONE SERIAL NUMBER VOLONE

FILE NAME: DIRACC1

FILE TYPE: DIRECT ACCESS	ITEM SIZE: 120	CREATION DATE:
PASSWRD: NO	RECORD SIZE: 250	CREATION NUMBER: 000
CYLINDER OVERFLOW: 1 TRACKS	ITEMS PER BLOCK: 2	MODIFICATION DATE: 00 000
GENERAL OVERFLOW: YES	RECORDS PER BLOCK: 1	MODIFICATION NUMBER: 000
ACTIVE ITEMS: 0	RECORDS PER TRACK: 15	EXPIRATION DATE: 00 000
KEY POSITION: 36	KEY LENGTH: 5	
BADTRACKS: NONE	BLOCKS PER BUCKET: 4	

UNITS OF ALLOCATION

	FROM		TO	
	CYLINDER	TRACK	CYLINDER	TRACK
1ST DATA UNIT	6	0	15	9

FILE CONTINUES ON VOLUME VOLTWO FILE-VOLUME SEQUENCE NUMBER: 0

COMPLETE LISTING FOR VOLUME VOLONE SERIAL NUMBER VOLONE

FILE NAME: SEQFIL2

FILE TYPE: SEQUENTIAL	ITEM SIZE: 250	CREATION DATE:
PASSWRD: NO	RECORD SIZE: 250	CREATION NUMBER: 000
CYLINDER OVERFLOW: 0 TRACKS	ITEMS PER BLOCK: 1	MODIFICATION DATE: 00 000
GENERAL OVERFLOW: NO	RECORDS PER BLOCK: 1	MODIFICATION NUMBER: 000
ACTIVE ITEMS: 0	RECORDS PER TRACK: 15	EXPIRATION DATE: 00 000
BLOCKS IN FILE-VOLUME: 180		
BADTRACKS: NONE		

UNITS OF ALLOCATION

	FROM		TO	
	CYLINDER	TRACK	CYLINDER	TRACK
1ST DATA UNIT	170	0	171	2
2ND DATA UNIT	170	3	171	5

FILE CONTINUES ON VOLUME VOLTWO FILE-VOLUME SEQUENCE NUMBER: 0

SECTION IV. FILE SUPPORT C

COMPLETE LISTING FOR VOLUME VOLONE SERIAL NUMBER VOLONE

FILE NAME: INDSEQ1

FILE TYPE: INDEXED SEQUENTIAL	ITEM SIZE: 100	CREATION DATE:
PASSWCRD: NO	RECORD SIZE: 500	CREATION NUMBER: 000
CYLINDER OVERFLOW: 1 TRACKS	ITEMS PER BLOCK: 10	MODIFICATION DATE: 00 000
GENERAL OVERFLOW: YES	RECORDS PER BLOCK: 2	MODIFICATION NUMBER: 000
ACTIVE ITEMS: 0	RECORDS PER TRACK: 8	EXPIRATION DATE: 00 000
KEY POSITION: 55	KEY LENGTH: 5	BLOCKS IN MASTER INDEX: 1
BADTRACKS: NONE	BLOCKS PER STRING: 2	

UNITS OF ALLOCATION INDEX AREA

	FROM		TO	
	CYLINDER	TRACK	CYLINDER	TRACK
	22	0	23	0
	UNITS OF ALLOCATION GENERAL OVERFLOW AREA			
	24	0	24	9
1ST DATA UNIT	25	0	55	9

FILE CONTINUES ON VOLUME VOLTWO

FILE-VOLUME SEQUENCE NUMBER: 0

COMPLETE LISTING FOR VOLUME VOLONE SERIAL NUMBER VOLONE

FILE NAME: PARTSEQ

FILE TYPE: PARTITIONED SEQUENTIAL	ITEM SIZE: 250	CREATION DATE:
PASSWCRD: NO	RECORD SIZE: 250	CREATION NUMBER: 000
CYLINDER OVERFLOW: 0 TRACKS	ITEMS PER BLOCK: 1	MODIFICATION DATE: 00 000
GENERAL OVERFLOW: NO	RECORDS PER BLOCK: 1	MODIFICATION NUMBER: 000
ACTIVE ITEMS: 0	RECORDS PER TRACK: 15	EXPIRATION DATE: 00 000
BLOCKS IN FILE-VOLUME: 900	BLOCKS IN MEMBER INDEX: 1	
BADTRACKS: NONE		

UNITS OF ALLOCATION

	FROM		TO	
	CYLINDER	TRACK	CYLINDER	TRACK
1ST DATA UNIT	16	0	21	9

SECTION IV. FILE SUPPORT C

UNASSIGNED TRACKS FOR VOLUME: VOLONE

CYLINDERS PER VOLUME: 203

TRACKS PER CYLINDER: 10

	X=USED			TRACKS										O=UNUSED	
	0	C	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	C	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	C	1	1	2	2	3	3	4	4	5	5	6	6	
	0	5	0	5	0	5	0	5	0	5	0	5	0	5	
CYL.															
0000	XXXXXX	CC	0000												
1	XXXXXXXXXX														
2	XXXXXXXXXX														
3	XXXXXXXXXX														
4	XXXXXXXXXX														
0005	XXXXXXXXXX														
6	XXXXXXXXXX														
7	XXXXXXXXXX														
8	XXXXXXXXXX														
9	XXXXXXXXXX														
0010	XXXXXXXXXX														
11	XXXXXXXXXX														
12	XXXXXXXXXX														
13	XXXXXXXXXX														
14	XXXXXXXXXX														
0015	XXXXXXXXXX														
16	XXXXXXXXXX														
17	XXXXXXXXXX														
18	XXXXXXXXXX														
19	XXXXXXXXXX														
0020	XXXXXXXXXX														
21	XXXXXXXXXX														
22	X0000C0000														
23	X0000C0000														
24	XXXXXXXXXX														
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	C	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	1	1	2	2	3	3	4	4	5	5	6	6	
	0	5	0	5	0	5	0	5	0	5	0	5	0	5	

NOTE: The map function produces as many pages as necessary to show unassigned tracks on all the cylinders on the volume. For purposes of brevity, only one such page is reproduced here.

MAP EXPIRED FILES

A description of all files that have expired, as indicated by their expiration dates, can be produced. To do this, the programmer requests a listing of all files whose expiration date is earlier than or the same as the Supervisor's current date or any specified date.

MAP UNUSED AREAS

A map of all unused areas on a mass storage volume can be produced. This map can be used to locate available areas for subsequent allocation of files.

CONSIDERATIONS

There are a number of considerations in using the File Support C routines within the Data Management Subsystem of the Mod 1 (MSR) Operating System. These considerations are discussed in the following paragraphs.

Number of Functions Performed

The number of different functions that can be performed in one execution of File Support C is limited by the amount of memory available. This is because all job control statements for an execution of File Support C are read initially, and the specified parameters are stored in memory. The number of functions which can be performed can be increased if additional main memory is available.

Only one allocate function can be requested in a single execution of File Support C, and only one file can be requested for allocation. Thus, for example, to allocate three files requires three separate executions of File Support C.

Block and Record Sizes Within 12K Memory

The maximum block and record sizes for files being processed in a single function are limited by the amount of main memory available. The following paragraphs are a guide to these maximum values for allocate and load/unload functions.

The figures given below are approximate and subject to change. They are based on the assumptions that only one function is being performed for a single execution of File Support C and that the number of parameters specified in the job control statements is minimal. The main memory size is assumed to be 12,288 characters. Additional memory can be used to increase the values shown below.

The maximum record size that can be formatted by the File Support C allocate function is 1,000 characters. If a partitioned sequential file is being allocated and members are declared, the maximum block size that can be handled is 1,400 characters.

SECTION IV. FILE SUPPORT C

Memory is used by the load/unload function for mass storage blocks, or card or tape records, for item work areas, for item key areas, and for an own-code routine. The steps listed below indicate the block and record sizes that can be handled by the load/unload function.

1. For each item in the following list, add the number of characters specified if the condition is met (each item is independent of the previous item, and the memory required is additive).
 - a. Add one buffer (the size of a block plus three characters) for each file on mass storage.
 - b. Add an additional buffer (the size of one block plus three characters) if the file organization is indexed sequential and the output file is on mass storage.
 - c. Add one buffer (the size of a tape record plus one character) if one file is on magnetic tape.
 - d. If the file is not an output direct access file, add 50 characters; if it is, add an area whose length, in characters, is equal to 8 times the total number of units of allocation in the file.
 - e. If one file is on punched cards and the item size for that file is greater than 80 characters, add one buffer whose size is a multiple of 80 and which is large enough to contain the item.
 - f. If the file organization is direct access, add one buffer (the size of the item of that file).
 - g. Add the size of the own-code routine. An own-code routine is required only if the output file is on mass storage and the file organization is direct access.
 - h. If the file organization is indexed sequential and if the input file is on mass storage, add an area the size of three item keys.

2. If the above total is less than the value shown in Table 4-1, the load/unload operation can be performed in 12K of memory. If the total is greater than the value shown in Table 4-1, the block or record sizes must be reduced, additional main memory must be available, or the size of the own-coding routine must be reduced.

Table 4-1. Available Memory per I/O Media for 12K Configuration

I/O Media	Available Memory	
	Indexed Sequential File Organization	Other File Organization
Magnetic tape to/from mass storage	1100	1200
Punched cards to/from mass storage	1100	1400
Mass storage to mass storage	1000	2100
Mass storage to printer	1800	800
NOTE: If the equipment configuration includes 16K memory with console typewriter, add 1000 characters to each of the above values.		

JOB CONTROL LANGUAGE FOR FILE SUPPORT C

The information below stresses the common job control language and its operation for File Support C. First, the Execute statement is described. Next, a single operation and a sequence of operations are described. A complete description of the job control language for each function of File Support C is included in the appropriate subsequent paragraph of this section.

NOTE: All numeric values are decimal unless otherwise noted.

Execute Statement

To perform a disk-resident File Support C function, the programmer must submit an Execute statement. This statement must be the first statement in the job control file for the particular File Support C run. The format of this statement is illustrated in Figure 4-1. Rules for loading from tape can be found in this section under "Operating Procedures for File Support C."

CARD NUMBER	MARK FIELD	LOCATION	OPERATION CODE	OPERANDS
1 2 3 4 5 6 7 8	14 15	20 21	42 43	80
			EX	*FILESUP,

Figure 4-1. Format of File Support C Execute Statement

When a File Support C job is to be performed, the parameter of the Execute statement is *FILESUP. This is the program segment name used for File Support C. Like all parameters in the operating system's job control language, this parameter must have a terminating comma. The Execute statement is followed by one or more Function statements in the job control file which name the functions to be performed.

Job Control for a Single Operation

To request a single operation (or function), the programmer must submit at least two job control statements. Job control statements are submitted through the job control file, which must be in a card reader.

The job control statements which the programmer must supply are the Execute statement (just described), a Function statement, and any other statement required for that function. The Function statement names the operation to be performed.

There must be a single indication of the end of job control statements per execution of File Support C. This indication is an E in the mark field (column 7 of the Easycoder Coding Form), and it must be on or immediately following the last line that contains job control statement information.

SECTION IV. FILE SUPPORT C

Suppose the desired operation is to obtain a description based on the contents of the volume directory for all the files stored on the volume. The job control statements required for operation are coded as shown in the following example. In this example, it is assumed that the residence file is on the volume being mapped (pcu 04, device 0).

CARD NUMBER	Y	M	D	C	R	LOCATION	OPERATION CODE	OPERANDS		
1	2	3	4	5	6	7	8	9-15	16-21	22-31
1							EX	*FILESUP,		
2						E	FUNCT	MAP, DESCR,		

Job Control for a Sequence of Operations

A sequence of operations can be executed by submitting Execute and Function statements for each operation desired, as just described. However, a sequence of operations can also be performed by submitting a single Execute statement and several Function statements. The operations are performed in the sequence in which the Function statements are submitted and, for certain functions, other job control statements are required.

The indication of the end of job control statements (the E in column 7) for a sequence of operations must be on or immediately follow the last line containing job control statement information. If the E is on any line preceding the last line, then the sequence of operations stops at that line.

Suppose the desired sequence of operations is to allocate a new sequential file and to describe all files on that volume. The minimum job control statements required for this sequence of operations are coded as illustrated in the following example.

PROBLEM						PROGRAMMER						DATE						PAGE						OF					
CARD NUMBER	Y	M	D	C	R	LOCATION	OPERATION CODE	OPERANDS																					
1	2	3	4	5	6	7	8	9-15	16-21	22-31																			
1							EX	*FILESUP,																					
2							FUNCT	ALLOCATE,		Operation 1																			
3							FILE	NAME=FILEAA,ORG=SEQ,																					
4							UNITS	NAME=VOLA,																					
5								FROM=(10,0),TO=(19,9),																					
6						E	FUNCT	MAP,DESCR,		Operation 2																			
7																													
8																													

ALLOCATE FUNCTION

The allocate function is used to assign a file to specified areas on a mass storage volume or volumes. Every file to be stored on mass storage must be allocated before it can be referred to by any program. The allocate function checks the volume directory of each volume specified for the file to ensure that the file name is unique and that no conflicting units of allocation exist on the volume; it also updates the volume directory of each volume to include information about the new file. All tracks of the new file are formatted to the specified record size. If a track is

SECTION IV. FILE SUPPORT C

encountered which cannot be successfully formatted, its cylinder and track address is listed on the printer, prior to a halt or console message.

All volumes of a multivolume file must belong to the same device class, and their devices must have the same data transfer rate.

To allocate a file, the programmer must supply, as a minimum, the name of the file, the file's organization, the volume name, and the units of allocation for each volume of the file.

Job Control Language for Allocate Function

The allocate function is requested by a Function statement whose first parameter is ALLOCATE. The Function statement is followed by File, Size, Units, Member, File list, and Day statements. When used, these statements must be submitted in that order after the Function statement. Figure 4-2 shows all the job control statements that can be used in the allocation of

CARD NUMBER	LOCATOR	LOCATION	OPERATION CODE	OPERANDS										
						1	2	3	4	5	6	7	8	14
1			EX	*FILESUP.	Required.									
2			FUNCT	ALLOCATE.	Required.									
3			FILE	NAME=file-name,	Required.									
4				ORG={SEQ}	Required.									
5				{PART}										
6				{DIR}										
7				{IND}										
8				GENOV={NO}	Optional. Direct									
9				{YES}	Access only.									
10				KEY=(position,length),	Req. for Direct Access & Indexed Seq.									
11				PW=password,	Optional.									
12				EXP=yyddd,	Optional.									
13				PROT={B}	Optional.									
14				{NO}										
15				DEVADD=(pcu,drive),...	Optional. (See note.)									
16			SIZE	REC=record-length,	The Size statement									
17				ITEM=item-length,	is optional.									
18				BLOCK=items-per-block,										
19				BUCKET=blocks-per-bucket,	Direct Access only.									
20				INDEX=blocks-in-index,	Partitioned Seq. only.									
21				CYLOV=number-of-tracks,	D.A. and I.S. only.									
22				STRING=blocks-per-string.	Indexed Seq. only.									
23			UNITS	NAME=volume-name,	Required									
24				MCINDEX=(FROM=(c,t),TO=(c,t)),	MCINDEX and OVERFLOW									
25				OVERFLOW=(FROM=(c,t),TO=(c,t)),	required for I.S.									
26				FROM=(c,t),TO=(c,t),...	At least One pair required									
27			MEMBER	NAME=member-name,	Optional statement. Partitioned									
28				LENGTH=number-of-blocks.	Sequential only.									
29			FILE	LIST, DEVADD=(pcu),	Optional statement.									
30			DAY	yyddd,	Optional statement.									
1					Required.									

NOTE: Ellipses on lines 17 and 28 (. . .) indicate that the keyword and its values can be specified more than once.

Figure 4-2. Job Control Statements for Allocation of Files

SECTION IV. FILE SUPPORT C

files. Each statement and its parameters are described in subsequent paragraphs. Note that no single file requires all the statements shown. Default values should be studied before deciding if a parameter can be omitted. A review of the newly allocated file using the description option of the map function is advisable prior to loading that file.

If the file organization is direct access or indexed sequential, the position and length of the item key must be specified. The number of characters per record, characters per item, and items per block may be specified, although default values of 250 characters per record, 250 characters per item, and 1 item per block are used if these parameters are omitted. (The Size statement itself can be omitted if default values can be assumed for all parameters.)

The volume name and unit(s) of allocation for each volume of the file must be specified. For an indexed sequential file, the units of allocation for the master/cylinder index and for the general overflow area must be specified, in that order. For all file organizations, at least one data unit of allocation and one volume name must be specified. The FROM parameter specifies the mass storage address of the beginning of the unit of allocation, and the TO parameter specifies the mass storage address of the end of the unit of allocation. The *c* stands for a cylinder number and the *t* stands for a track number.

Additional parameters can be used to specify such information as the size of the cylinder overflow area (direct access or indexed sequential files), password protection, file protection, expiration date, record size, bucket size (direct access files), string size (indexed sequential files), member index length (partitioned sequential files), additional Units statements (for multi-volume files), additional data units of allocation, members to be named and allocated within a partitioned sequential file, the creation date, and the general overflow option for direct access files.

EXECUTE STATEMENT

The Execute statement with the program segment name *FILESUP directs the Supervisor to load File Support C. The format of the Execute statement follows.

CARD NUMBER	Y	X	LOCATION	OPERATION CODE	OPERANDS
1 2 3 4 5 6 7 8	14 15	20 21	26 27	32 33	38
1				EX	*FILESUP,
2					

FUNCTION STATEMENT

The Function statement specifies to File Support C what function to perform. This statement is required. Only one allocate function is permitted per execution of File Support C. To perform the allocate function, the statement must be coded in the following format.

CARD NUMBER		OPERATION CODE	OPERANDS
1	2		
1	2	14	20
1	2	14	20
1	2	14	20

FILE STATEMENT

The file which is being allocated is identified by the File statement. This statement is required. Only one File statement is permitted per allocate function.

File-Name Parameter

The file-name parameter specifies the name of the file being allocated. The format of this parameter is as follows.

NAME=file-name,

This parameter is required. The file name specified as the value of this parameter can be up to 10 characters long. When it is less than 10 characters, trailing spaces are automatically added. A file name can consist of the letters A through Z, the digits 0 through 9, and space; the space cannot be the first character of the name. The special character * (asterisk) is used as the first character of the names of systems files only.

File-Organization Parameter

The file-organization parameter specifies the type of file organization of the file being allocated. The format of this parameter is as follows.

ORG = $\left\{ \begin{array}{l} \text{SEQ} \\ \text{PART} \\ \text{DIR} \\ \text{IND} \end{array} \right\},$

The parameter is required. The value chosen for this parameter will have the significance described below.

- SEQ = A sequential file is being allocated.
- PART = A partitioned sequential file is being allocated.
- DIR = A direct access file is being allocated
- IND = An indexed sequential file is being allocated.

General Overflow Parameter

The general overflow parameter specifies whether or not the direct access file being allocated will have a general overflow area. The format of this parameter is as follows.

GENOV= { NO } { YES }

This parameter is optional and applies only to direct access files. When a value for this parameter is not specified, the allocate function assumes a value of YES (the file will have a general overflow area). The value chosen for this parameter has the significance described below.

NO = The file will not contain a general overflow area.

YES = The file will contain a general overflow area consisting of all the assigned tracks of the last cylinder of each unit of allocation.

Item Key Parameter

The item key parameter is used to specify the length and position of the item key. The format of this parameter is as follows.

KEY=(position, length),

This parameter is relevant only for direct access and indexed sequential files, and it is required for such files. The position element of the parameter indicates the position in each item of the left end of the key field. The leftmost character of the item is position one. The length element of the parameter indicates the length in characters of the key field.

Password Parameter

The password parameter specifies the password required for all subsequent access to the file being allocated. The format of this parameter is as follows.

PW=password,

This parameter is optional. When a password is not specified for the file being allocated, no password protection is assigned to the file. The password specified for the file can be up to eight characters long. If the password is not eight characters, trailing spaces are added automatically. The first character of the password cannot be space (Δ).

File Expiration Date Parameter

The file expiration date parameter specifies the year and day the user expects the file being allocated to expire. The format of this parameter is as follows.

EXP=yyddd,

This parameter is optional. When an expiration date for the file is not assigned, the allocate function uses the date 00000 in the volume directory entry for the file. The yy portion of the date represents the last two digits of the year of expiration and the ddd portion represents the number of the day. For example, the date 15 December 1967 is represented as 67349 because the year is 1967 and the fifteenth of December is the three hundred and forty-ninth day of the year.

Protection Status Parameter

The protection status parameter specifies the type of write protection to be assigned to the file being allocated. The format of this parameter is as follows.

PROT = { B }
 { NO }

This parameter is optional. When this parameter is omitted from the File statement, the allocate function assumes a value of NO (write protection is not being assigned to the file being allocated). The value chosen for this parameter has the significance described below.

- B = The file is assigned B-file write protection.
- NO = Write protection is not assigned to the file.

NOTE: A-file write protection cannot be specified during allocation. It can be specified, however, for load and unload functions, since it may be necessary to load and unload system files (e. g. , *BADTRACKS).

Appendix F of this manual contains a complete description of write protection.

Device-Address Parameter

The device-address parameter is used to specify the peripheral address assignment of the control unit and the drive number of the device(s) containing the volume(s) on which the file is being allocated. This parameter can be specified as many times as required for a multivolume file. Each Units statement specifies one volume of the file. When this parameter is specified, it must follow all other parameters of the File statement. The format of this parameter is as follows.

```
DEVADD=(pcu, drive), . . . ,
```

When specifying values for this parameter, the peripheral address assignment is written as two octal digits. The high-order bit is not significant. The drive number is written as one octal digit. This parameter is optional. When a device address is not specified, the allocate function assumes that the peripheral address assignment is 04 and that the drive number is 0.

When specifying a multivolume file for allocation, multiple device address parameters can be used. When more volumes than device addresses are specified, the addresses are used cyclically. (See example on page 4-23.)

To summarize the File statement, the file name and file organization parameters are required. The remaining parameters are optional except that the item key parameter is required for direct access and indexed sequential files. If the programmer omits the optional parameters, the allocate function assumes that no password is assigned to the file, the expiration protection is not required, write protection is not required, the device address is 04, 0 for all volumes of the file, and that there is to be a general overflow cylinder for each unit of allocation if the file is direct access.

SIZE STATEMENT

The Size statement is used by the programmer to specify the sizes of various parameters of the file being allocated.

Record Length Parameter

The record length parameter specifies the number of characters in each record of the file being allocated. The format of this parameter is as follows.

```
REC = record-length,
```

When this parameter is not specified, the allocate function assumes that the size of each record in the file is 250 characters. The number of characters must not exceed 4095.

Item Length Parameter

The item length parameter specifies the number of characters in each item. The format of this parameter is as follows.

```
ITEM = item-length,
```

When this parameter is not specified, the allocate function assumes that the size of each item in the file is the same as the record size. A status character is added onto each item in direct access and indexed sequential files and must be included when specifying the value of this parameter. This parameter value must not exceed 4095.

Block Size Parameter

The block size parameter specifies the number of items in each block in the file being allocated. The format of this parameter is as follows.

BLOCK = items-per-block,

When this parameter is not specified, the allocate function assumes that the block size is equal to the number of whole items that will fit into one record. If the record size is smaller than the item size, then the block size is one item per block (and two or more records compose a single block).

NOTE: The record length is the prime determinant of assumed values of item and/or block size. Appendix C can be useful in determining proper record and block sizes.

Bucket-Size Parameter

The bucket size parameter applies only to direct access files and is used to specify the number of blocks per bucket. When this parameter is not specified, the allocate function assumes that there is one block per bucket in the file being allocated. The format of this parameter is as follows.

BUCKET=blocks-per-bucket,

Index-Size Parameter

The index-size parameter is used to specify the number of blocks in the member index of a partitioned sequential file. When this parameter is not specified, the allocate function assumes that there is one block in the member index. The format of this parameter is as follows, and its value establishes the maximum number of active member names that can be contained in the index (see Appendix B).

INDEX=blocks-in-index,

Cylinder Overflow Size Parameter

The cylinder overflow size parameter is used to specify the number of tracks in the cylinder overflow area for direct access or indexed sequential files. When this parameter is not specified,

the allocate function assumes that the file being allocated has no cylinder overflow area. The format of this parameter is as follows.

CYLOV=number-of-tracks,

String-Size Parameter

The string-size parameter is used to specify the number of data blocks per string for an indexed sequential file. It is not relevant for other file organizations.

STRING=blocks-per-string,

If this parameter is not specified, the assumed value is one.

UNITS STATEMENT

One Units statement specifies the units of allocation for one volume of the file. There must be exactly one Units statement for each volume of the file. There may be up to eight volumes.

Each Units statement specifies the units of allocation on that volume. There must be exactly one pair of from and to parameters for each unit of allocation. There may be up to 6 units of allocation on each volume and up to 16 total units of allocation for a multivolume file.

For a sequential or direct access file, there must be at least one Units statement with at least one pair of FROM and TO parameters. For an indexed sequential file, there must be at least one Units statement and at least three units of allocation; these are the index unit, the overflow unit, and at least one data unit. For a partitioned sequential file, there must not be more than one Units statement.

Volume-Name Parameter

The volume-name parameter must be the first parameter; it is used to specify the name of the volume to which the Units statement applies. The format of this parameter is as follows.

NAME=volume-name,

The volume name specified is checked against the volume name of the volume mounted at the relevant device address. The volume-name parameter must be specified for each volume on which the file is to be allocated.

Master/Cylinder Index Parameter

The master/cylinder index parameter specifies the unit of allocation for the index area of an indexed sequential file. It is not relevant for any other file organization. The index area includes both the master index and the cylinder index.

The first unit of allocation specified for the indexed sequential file must be the index area, as specified by the master/cylinder index parameter. The format of this parameter is as follows.

`MCINDEX=(FROM=(c, t), TO=(c, t)),`

The FROM parameter specifies the low cylinder (c) and track (t) addresses of the unit of allocation. It must be followed immediately by a TO parameter, which specifies the high cylinder and track addresses of the same unit of allocation. Both cylinders and tracks are numbered starting at zero. The cylinder address specified by the FROM parameter must be less than or equal to the cylinder address of the corresponding TO parameter. The track address of the FROM parameter must be less than or equal to the track address of the corresponding TO parameter.

The TO parameter specifies the high cylinder and track addresses of the unit of allocation. It is paired with the immediately preceding FROM parameter.

Overflow Parameter

The second unit of allocation specified for an indexed sequential file must be the general overflow area, specified by the overflow parameter.

The overflow parameter specifies the unit of allocation for the general overflow area of an indexed sequential file. It is not relevant to any other file organization. The format of this parameter is as follows.

`OVERFLOW=(FROM=(c, t), TO=(c, t)),`

The FROM and TO parameters for the overflow area are specified as described for the master/cylinder index parameter.

NOTE: The units of allocation for the master/cylinder index and general overflow can be any width (tracks per cylinder), but no unit of allocation may begin on the cylinder on which the previous unit of allocation for that file ended.

Data Unit of Allocation

A data unit of allocation is specified by a pair of FROM and TO parameters. At least one data unit of allocation must be specified for all file organizations. If a file is assigned more than one unit of allocation, the number of tracks per cylinder in all data units of allocation for the file must be the same.

FROM Parameter

The FROM parameter is used to specify the low cylinder and track numbers of the unit of allocation. The format of this parameter is as follows.

FROM=(c, t),

The FROM parameter must be followed immediately by a TO parameter which specifies the high cylinder and track numbers of the unit of allocation. The cylinder number of the FROM parameter must not exceed the cylinder number of the TO parameter. The same is true for the track numbers.

TO Parameter

The TO parameter is paired with the immediately preceding FROM parameter and is used to specify the high cylinder and track numbers of the unit of allocation. The format of this parameter is as follows.

TO=(c, t),

All cylinder and track values specified must be consistent with the device type of the current mass storage volume.

MEMBER STATEMENT

The Member statement is used to reserve space in a partitioned sequential file for a specific member. When space is being reserved for members, there must be one Member statement for each member to be entered. The use of this statement is optional. When used, the member name and member length parameters are required.

NOTE: A member can be created by use of Logical I/O C, the load function, or the Member statement in the allocate function.

Member Name Parameter

The member-name parameter is used to specify the name of the member for which space is being reserved. The value of this parameter can be up to 14 characters and can consist of the letters A through Z, digits 0 through 9, and blanks; the blank cannot be the first character

of the name. If duplicate member names are requested, only the first is allocated. The format of this parameter is as follows.

NAME=member-name,

Member-Length Parameter

The member-length parameter is used to specify the number of blocks to be reserved for this member. When data is subsequently loaded into the member, it may occupy all or any part of the reserved space. The format of this parameter is as follows.

LENGTH=number-of-blocks,

The value of this parameter must not exceed 4,095. If it is expected that the data area of a member will occupy more than 4,095 blocks of its file, that member should not be requested during allocation. That member can be created at the time its data enters the file by means of the load function.

FILE STATEMENT FOR THE LIST FILE

The peripheral device address of the list file may be specified by a File statement whose first parameter is LIST. If this statement is omitted, the list file is produced on a printer with the device address of (02).

CARD NUMBER	V 1	Y 2	E 3	R 4	K 5	L 6	C 7	I 8	LOCATION	OPERATION CODE	OPERANDS			
1	2	3	4	5	6	7	8	14	15	20	21	62	63	80
1											FILE LIST,			
2														
3														
4														
5														

The list file is used only to print the cylinder and track address of any track which cannot be successfully formatted during allocation of a file. If all tracks are successfully formatted, no list file is produced. The message is produced in the following form:

CYL nnn TR nnn ERROR, (nnn denotes decimal values.)

Device-Address Parameter

The device-address parameter allows changes to be made in the standard assignment for the peripheral device used for the list file.

DEVADD=(pcu),

The peripheral address is written as two octal digits. All bits including sector bits must be specified. The default assumption is device address (02).

DAY STATEMENT

A Day statement is used to specify the value to be placed in the creation date field of the volume directory entry for the file being allocated. When a Day statement is not specified, i. e., when this statement is omitted from the allocate function job control statements, the allocate function places the contents of the Supervisor's current date field in the creation date field. The format of the Day statement is as follows.

CARD NUMBER	Y	M	D	LOCATION	OPERATION CODE	OPERANDS	
						14 15	20 21
1	2	3	4	5	6	7	8
					DAY	yyddd,	
2							
3							
4							

The parameter of the Day statement is used to specify the actual creation date of the file. The yy portion of the parameter specifies the last two digits of the year, and the ddd portion the number of the day. For example, if the creation date of a file is 15 December, 1969, then the value of the parameter is 69349.

Job Control Language Example for Allocate Function

The following job control statements request the allocation of a sequential file named FILEAA. This file's item length is specified as 100 characters, and each of its blocks contains six items. This means that the block length is 600 characters. The record size is specified as 600 characters. Since values are not supplied for the password, expiration date, and protection parameters, the allocate function does not assign password protection, expiration date protection, or write protection.

In this example for a Type 259 Disk Pack Drive, one unit of allocation consisting of all the tracks on cylinders 05 through 09 inclusive is requested. The volume name is VOLA. The allocate function assumes that the volume's peripheral address assignment (pcu) is 04, drive number 0, since a device address is not supplied.

A Day statement is not submitted, so the allocate function assigns the value of the Supervisor's current date field as the creation date of the file.

SECTION IV. FILE SUPPORT C

CARD NUMBER	Y	M	A	R	LOCATION	OPERATION CODE	OPERANDS	
							1415	2021
1						EX	#FILESUP,	
2						FUNCT	ALLOCATE,	
3						FILE	NAME=FILEAA,ORG=SEQ,	
4						SIZE	ITEM=100,BLOCK=6,REC=600,	
5						UNITS	NAME=VOLA,	
6					E		FROM=(05,0),TO=(09,9),	

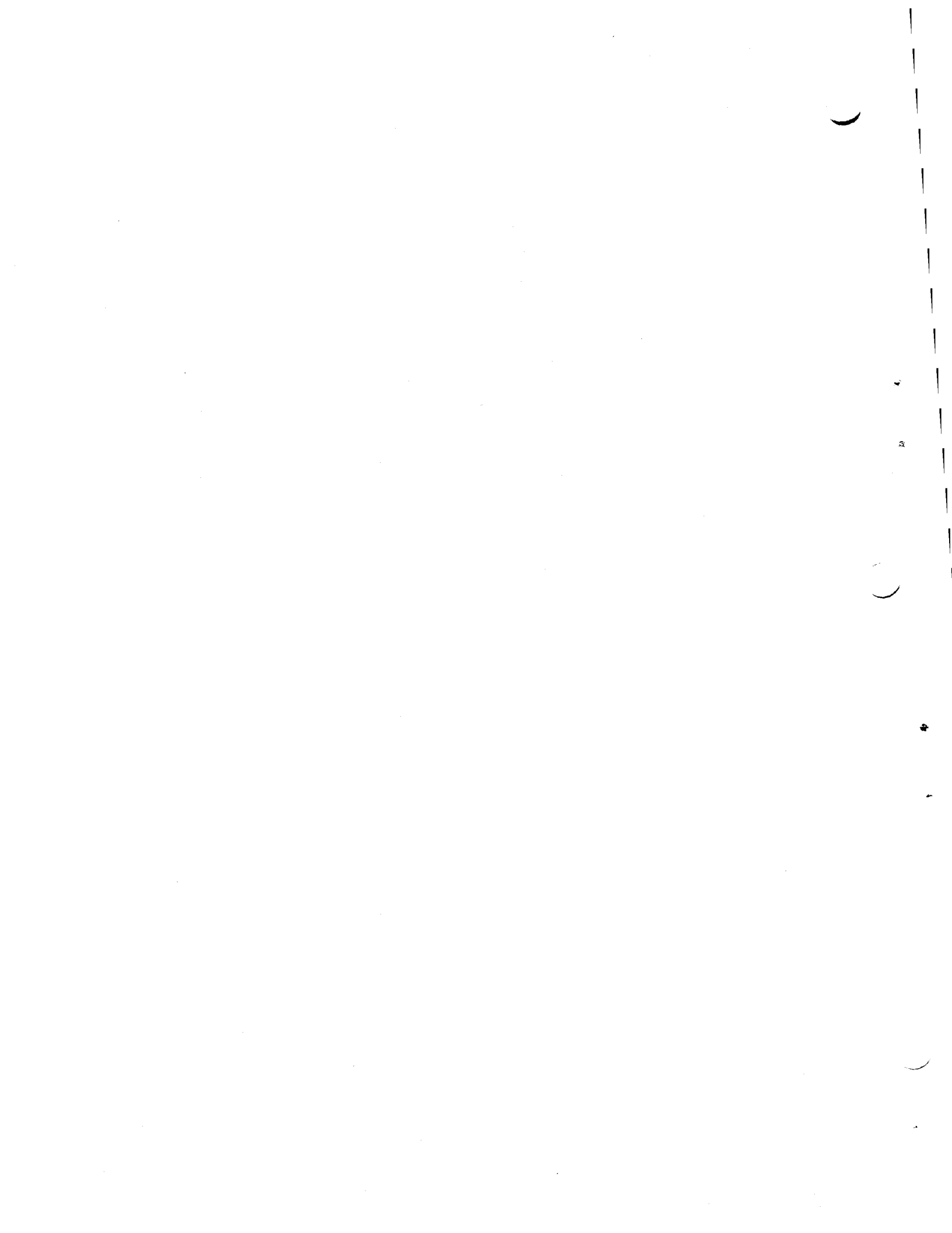
The following example, for a Type 273 Disk Pack Drive, requests the allocation of a sequential file ROBYN. The volume name is RESVOL and the device address is assumed to be pcu 04, drive 0. The item, record, and block sizes are specified. In this example, the unit of allocation is placed on the volume on cylinders 07 through 14 inclusive.

CARD NUMBER	Y	M	A	R	LOCATION	OPERATION CODE	OPERANDS	
							1415	2021
1						EX	#FILESUP,	
2						FUNCT	ALLOCATE,	
3						FILE	NAME=ROBYN,ORG=SEQ,	
4						SIZE	ITEM=200,BLOCK=5,REC=1000,	
5						UNITS	NAME=RESVOL,	
6					E		FROM=(07,0),TO=(14,19),	

The following example requests the allocation of a partitioned sequential file FILEBB. The item, record, and block sizes are specified. Note that two records form one block of nine items, which gives a larger data capacity on each track of the Type 259 Disk Pack Drive than would be achieved using a record size of 900 characters (refer to Appendix C). Because the member index size is not specified, one block is reserved. Two members are reserved during allocation in this example.

CARD NUMBER	Y	M	A	R	LOCATION	OPERATION CODE	OPERANDS	
							1415	2021
1						EX	#FILESUP,	
2						FUNCT	ALLOCATE,	
3						FILE	NAME=FILEBB,ORG=PART,	
4						SIZE	ITEM=100,REC=450,BLOCK=9,	
5						UNITS	NAME=VOLA,	
6							FROM=(10,0),TO=(15,9),	269 blocks for data.
7						MEMBERNAME=MEMBERA,LENGTH=90,		179 blocks unassigned.
8					E	MEMBERNAME=MEMBERB,LENGTH=50,		129 blocks unassigned.

The following example requests the allocation of a sequential file which extends over three volumes. The mass storage device on which each of the three volumes is to be mounted is specified through two device address parameters in the File statement. As many device address parameters are written as there are devices to be used. In this example, the unit of allocation for the first volume (specified in the first Units statement) is placed on the volume at address 04, drive 1; for the second volume at address 04, drive 2; and that for the third volume at address 04, drive 1 (which is thus used again). The number of volumes on which the file is allocated (this is equal to the number of Units statements) does not have to equal the number of devices used (this is obtained from the device address parameters). Before the allocate function allocates the third volume of the file, the volume originally mounted on drive 1 (VOLA) must be replaced by the volume named VOLC, which is to be the third volume for the file.



EASYCODER

CODING FORM

PROBLEM _____										PROGRAMMER _____										DATE _____										PAGE _____ OF _____									
CARD NUMBER		LOCATION		OPERATION CODE		OPERANDS																																	
1	2	3	4	5	6	7	8	14	15	20	21															42	43	50											
1												EX *FILESUP,																											
2												FUNCT ALLOCATE,																											
3												FILE NAME=FILEB,																											
4												ORG=SEQ,																											
5												DEVADD=(04,1),DEVADD=(04,2),																Cyclical addressing											
6												SIZE ITEM=125,REC=500,																											
7												UNITS NAME=VOLA,																											
8												FROM=(14,0),TO=(50,9),																											
9												UNITS NAME=VOLB,																											
10												FROM=(15,0),TO=(44,9),																											
11												FROM=(100,0),TO=(140,9),																											
12												UNITS NAME=VOLC,																											
13												FROM=(60,0),TO=(130,9),																											
14												E																											

The following example requests the allocation of an indexed sequential file on three volumes, each on a Type 259 Disk Pack Drive. The master/cylinder index and overflow units are assigned to the volume named VOLEE; the data units of allocation are assigned to the two volumes VOLED and VOLEF in that sequence. All three volumes are to be mounted at address 04, drive 2; volume VOLED is mounted after volume VOLEE has been processed, and volume VOLEF is mounted after volume VOLED has been processed.

EASYCODER

CODING FORM

PROBLEM _____										PROGRAMMER _____										DATE _____										PAGE _____ OF _____									
CARD NUMBER		LOCATION		OPERATION CODE		OPERANDS																																	
1	2	3	4	5	6	7	8	14	15	20	21															42	43	50											
1												EX *FILESUP,																											
2												FUNCT ALLOCATE,																											
3												FILE NAME=FILE2,																											
4												ORG=IND,KEY=(6,10),																											
5												DEVADD=(04,2),																											
6												SIZE STRING=4,ITEM=50,																											
7												UNITS NAME=VOLEE,																											
8												MCINDEX=(FROM=(11,0),TO=(11,0)),																One track only.											
9												OVERFLOW=(FROM=(10,1),TO=(11,6)),																Twelve tracks only.											
10												UNITS NAME=VOLED,																First data unit											
11												FROM=(10,0),TO=(50,9),																forty-one cylinders.											
12												UNITS NAME=VOLEF,																Second data unit											
13												FROM=(8,0),TO=(42,9),																thirty-five cylinders.											
14												E FILE LIST,DEVADD=(03),																Printer on pcc 03.											
15																																							
16																																							
17																																							

The default assumptions for this file are as follows:

1. Record size = 250 characters,
2. Block size = 5 items,
3. Strings per cylinder = 36 strings,¹
4. String index = 5 blocks,¹
5. Items per string = 20 items,² and
6. No cylinder overflow.

Summary of Job Control Statements for Allocate Function

Table 4-2 contains a complete summary of the job control statements for the allocate function.

¹ Calculated by allocate function from record size, string size, and track width.

² Calculated by allocate function from item size, block size, and string size.

Table 4-2. Summary of Job Control Statements for Allocate Function

Statement	Command (Operation Code)	Parameter Name	Parameter Value (Operands Field)	Function	Comments
Execute	EX	Program Segment Name	*FILESUP,	Directs Supervisor to load File Support C.	Required when running under control of mass storage Supervisor C.
Function	FUNCT	Function Name	ALLOCATE,	The allocate function is to be performed.	Required.
File	FILE			Defines the file being allocated.	Required.
		File Name	NAME = file-name,	Names the file being allocated.	Required.
		File Organization	ORG = $\left\{ \begin{array}{l} \text{SEQ} \\ \text{PART} \\ \text{DIR} \\ \text{IND} \end{array} \right\}$,	Defines the file's organization.	
		General Overflow	GENOV = $\left\{ \begin{array}{l} \text{NO} \\ \text{YES} \end{array} \right\}$,	Specifies whether file is to contain a general overflow area.	Optional; applies to direct access files only. Assumed value = YES.
		Item Key	KEY = (position, length),	Specifies leftmost position and length of keyfield.	Required for direct access and indexed sequential files; does not apply to other file organizations.
		Password	PW = password,	Specifies the password to be assigned to the file.	Optional. Assumed value = no password.
		Expiration Date	EXP = yyddd,	Gives date of file's expiration.	Optional. Assumed value = no expiration date protection.
		Write Protection	PROT = $\left\{ \begin{array}{l} \text{B} \\ \text{NO} \end{array} \right\}$,	Specifies the type of write protection to be assigned.	Optional. Assumed value = NO.
	Device Address	DEVADD = (pcu, drive),	Gives the device address of the volume. More than one can be specified for multivolume files.	Optional. Assumed value = (04, 0),. If more than one value, use is cyclical.	

Table 4-2 (cont). Summary of Job Control Statements for Allocate Function

Statement	Command (Operation Code)	Parameter Name	Parameter Value (Operands Field)	Function	Comments
Size	SIZE				The Size statement is normally present. It may be omitted only if the assumed values for all its parameters are satisfactory.
		Record Size	REC = record-length,	Gives the number of characters in each record.	
		Item Size	ITEM = item-length,	Gives the number of characters in each item including status character when applicable.	
		Block Size	BLOCK = items-per-block,	Gives the number of items in each block.	Optional. Meaningful only for direct access files. Assumed value = 1.
		Bucket Size	BUCKET = blocks-per-bucket,	Gives the number of blocks per bucket.	
		Index Size	INDEX = blocks-in-index,	Gives the number of blocks in member index.	Optional. Meaningful only for partitioned sequential files. Assumed value = 1.
		Cylinder Overflow Size	CYLOV = number-of-tracks,	Gives the number of tracks in the cylinder overflow area.	Optional. Meaningful only for direct access and indexed sequential files. Assumed value = 0.
	String Size	STRING = blocks-per-string,	Specifies the number of blocks per string for an indexed sequential file.	Optional. Meaningful only for indexed sequential files. Assumed value = 1.	
Units	UNITS			Specifies the units of allocation for the file.	One required for each volume of the file.
		Volume Name	NAME = volume-name,	Names the volume to be used for this Units statement.	Required for each volume. Must be first parameter.
		Master/Cylinder Index	MCINDEX = (FROM=(c, t), TO=(c, t)),	Specifies the unit of allocation for the master/cylinder index.	Required. Allowable only for indexed sequential files. Must be first unit.

4-26

#5-618

SECTION IV. FILE SUPPORT C

Table 4-2 (cont). Summary of Job Control Statements for Allocate Function

Statement	Command (Operation Code)	Parameter Name	Parameter Value (Operands Field)	Function	Comments
Units	UNITS (cont)	Overflow	OVERFLOW= (FROM=(c, t), TO=(c, t)),	Specifies the unit of allocation for the general overflow area.	Required. Allowable only for indexed-sequential files. Must be second unit. At least one pair required. Additional pairs must have same track width.
		FROM	FROM = (c, t),	Gives the low cylinder and track addresses of the data unit of allocation.	
		TO	TO = (c, t),	Gives the high cylinder and track addresses of the data unit of allocation.	
Member	MEMBER			Reserves space for a member of a partitioned sequential file.	Optional. Meaningful only for partitioned sequential files. May appear more than once.
		Member Name	NAME = member-name,	Gives the name of the member that space is reserved for.	
		Member Length	LENGTH = number-of-blocks,	Gives the number of blocks in the member.	
File	FILE	File	LIST,	Specifies the list file device address.	Optional. When omitted, listing is produced on a printer with peripheral control address of (02).
		Device Address	DEVADD=(pcu),	Allows changes in the standard device assignment for the list file.	
Day	DAY	Date	yyddd,	Gives the year and day the file is created.	Optional. Assumed value = Supervisor's current date field.

4-27

#5-618

SECTION IV. FILE SUPPORT C

DEALLOCATE FUNCTION

The deallocate function is used to delete files from a mass storage volume or volumes. File deallocation is the only means by which allocated areas on a volume can be freed for the allocation of new files. Before a file is deallocated, checks are made on the volume name, the file expiration date, and the password. This is done to avoid inadvertent removal of a file which has not expired or which is protected by a password.

To deallocate a file, the programmer must supply the volume name of the first volume containing the file, the name of the file, and its password if the file is protected by a password. The deallocate function is requested by a Function statement whose first parameter is DEALLOCATE. This statement must be followed by a Volume statement and at least one File statement. A Day statement may follow the last File statement. The Volume, File, and Day statements must be submitted in that order after the Function statement.

Job Control Language for Deallocate Function

Figure 4-3 shows all the job control statements that are required for, or that can be used in, the deallocation of files from mass storage volumes. Each statement and its parameters are described in subsequent paragraphs.

EASYCODER
CODING FORM

PROBLEM _____ PROGRAMMER _____ DATE _____ PAGE _____ OF _____

CARD NUMBER	Y	M	A	R	LOCATION	OPERATION CODE	OPERANDS	
1	2	3	4	5	6	7	8	9
						EX	*FILESUP,	Req. Statement.
						FUNCT	DEALLOCATE,	Req. Statement.
						VOLUME		Req. Statement.
							NAME=volume-name,	Req. Parameter.
						FILE		Req. Statement.
							NAME=file-name,	Req. Parameter.
							EXP={NO }	Optional Parameter.
							{YES}	
							PW=password,	Req. for file with password protection.
							DEVADD=(pcu,drive),	Optional Parameter.
						DAY	yyddd,	Optional Statement.

Figure 4-3. Job Control Statements for Deallocate Function

EXECUTE STATEMENT

The Execute statement with the parameter *FILESUP directs the Supervisor to load File Support C. The format of the Execute statement is shown in Figure 4-1.

SECTION IV. FILE SUPPORT C

FUNCTION STATEMENT

The Function statement specifies to File Support C what function to perform. This statement is required, and to perform the deallocate function the statement must be coded as follows.

CARD NUMBER	LOCATION	OPERATION CODE	OPERANDS
1 2 3 4 5 6 7 8	14 15	20 21	62 63 80
1		FUNCT	DEALLOCATE,
2			

VOLUME STATEMENT

The Volume statement is used to specify parameters that pertain to the first or only volume containing the files to be deallocated. There can be only one Volume statement per deallocate function; i. e., all files to be deallocated with this execution of the deallocate function must begin on the same volume.

Volume Name Parameter

The volume name parameter is used to specify the name of the first or only volume on which a file to be deallocated resides. The format of this parameter is shown below.

NAME=volume-name,

FILE STATEMENT

Each file to be deallocated is named by a File statement whose first parameter contains the name of the file. This statement is required. To deallocate more than one file with a single Function statement, there must be a File statement naming each file to be deallocated.

File Name Parameter

The file name parameter is required and is used to specify the name of the file to be deallocated. The format of this parameter is as follows.

NAME=file-name,

Expiration Date Check Parameter

The expiration date check parameter is used to specify whether the expiration date of the file to be deallocated is to be checked by the deallocate function. This parameter is optional. When it is not specified by the programmer, the deallocate function automatically checks the expiration date of the file being deallocated against the date specified by the user (see the Day statement). The format of this parameter is as follows:

EXP= { NO } { YES }

NO = The expiration date will not be checked, and

YES = The expiration date will be checked.

Password Parameter

The password parameter is used to permit only authorized deallocation of a file. The format of this parameter is as follows.

PW=password,

The password parameter must be specified if the file being deallocated is protected by a password. When the password specified in this parameter is not the same as that assigned to the file, the file is not deallocated.

Device Address Parameter

The device address parameter is used to specify the peripheral address assignment of the volume or volumes containing the file to be deallocated. The format of this statement is as follows.

DEVADD=(pcu, drive),...

When specifying values for this parameter, the peripheral address assignment (pcu) is written as two octal digits. The high-order bit is not significant. The drive number is written as one octal digit. As many of these parameters as are required for multivolume files may be specified. This parameter, however, is optional.

If the device address parameter is omitted, the default assumption depends on whether or not this is the first file being deallocated by this deallocate function. If it is the first file, the default assumption is that only one device is to be used (04, 0). If it is not the first file, the default assumption is that the devices used for the preceding file specified with this deallocate function are to be used again.

SECTION IV. FILE SUPPORT C

DAY STATEMENT

The Day statement is used to specify the date against which the expiration date for the file is to be checked. This statement is optional. When a Day statement is not specified, the deallocate function compares the contents of the Supervisor's current date field with the expiration date assigned to the file. This check is not made if the value of the expiration date check parameter is NO. The format of the Day statement is as follows.

CARD NUMBER				LOCATION				OPERATION CODE				OPERANDS															
1	2	3	4	5	6	7	8	14	15	20	21																
												DAY yyddd,															

The parameter of the Day statement is yyddd and is used to specify the last two digits of the year (yy) and the day of the year (ddd). Thus, if the date submitted was 15 December, 1969, it would be coded as 69349.

Only one Day statement may appear for each request for the deallocate function. All files named in the File statements following the Function statement have their expiration date checked against this same date.

Job Control Language Example for Deallocate Function

The following job control statements request the deallocation of two files on the volume named A00000. If the two files being deallocated are multivolume files, the deallocate function assumes that both begin on volume A00000. Because only one device address is specified for the files, the deallocate function assumes that the peripheral address assignment (pcu) is 04 and that the drive number is 1 for both files. The first file to be deallocated is named FILEEE. Its expiration date is automatically checked against the current date field of the Supervisor. The file's password is specified as DEPT.100. The second file to be deallocated is named FILECC. Its expiration date is not checked. A password parameter is not specified for the file, but the deallocate function automatically checks to ensure that the file is not protected by a password.

CARD NUMBER				LOCATION				OPERATION CODE				OPERANDS															
1	2	3	4	5	6	7	8	14	15	20	21																
												*FILESUP,															
												FUNCT DEALLOCATE,															
												VOLUME NAME=A00000,															
												FILE NAME=FILEEE, PW=DEPT.100, DEVADD=(04,1),															
												FILE NAME=FILECC,															
												EXP=NO,															

SECTION IV. FILE SUPPORT C

The following example illustrates a request to deallocate several files, all beginning on the same volume. If the files did not all start on the same volume, their deallocation would have to be done through separate requests for the deallocate function. Three files are deallocated in this example. FILEA is a multivolume file and three device addresses are specified for it; its volumes are mounted on drives 1, 2, 3, 1, and so forth as necessary. Since device addresses are not specified for FILEB, its volumes are assumed to be mounted on the same set of devices as FILEA, with the first volume of FILEB mounted on drive 1. FILEC is specified to begin on drive 1. If it is a multivolume file, all subsequent volumes also will be mounted on drive 1, in sequence, each volume replacing the previous volume. All three files must begin on the volume named VOLA; this volume name will be checked for each file. The name of each subsequent volume (not required in the job control file) is taken from the directory of the current file volume.

EASYCODER
CODING FORM

PROBLEM _____ PROGRAMMER _____ DATE _____ PAGE _____ OF _____

CARD NUMBER	Y	M	R	LOCATION	OPERATION CODE	OPERANDS	
						14 15	20 21
1 2 3 4 5 6 7 8	9	10	11	12	13	16 17	22 23
1					EX	*FILESUP,	
2					FUNCT	DEALLOCATE,	
3					VOLUME	NAME=VOLA,	
4					FILE	NAME=FILEA,	
5						DEVADD=(04,1), DEVADD=(04,2), DEVADD=(04,3),	
6					FILE	NAME=FILEB,	
7					FILE	NAME=FILEC,	
8				E		DEVADD=(04,1),	
9							
10							

The following example illustrates a request to deallocate three multivolume files, each of which starts on a different volume. File FILER begins on volume A (VOLA) and continues through VOLB, VOLC, and VOLD. File FILEX begins on VOLC and continues on to VOLA. File FILET begins on VOLD and continues on to VOLA. The volumes reside on the following device addresses (all using pcu 04).

- VOLA - drive 1
- VOLB - drive 2
- VOLC - drive 3
- VOLD - drive 4

Disk changing is not required during the deallocation of these three files. Three Function statements are required for this operation.

EASYCODER

CODING FORM

PROBLEM _____ PROGRAMMER _____ DATE _____ PAGE _____ OF _____

CARD NUMBER	LOCATION	OPERATION CODE	OPERANDS											
				1	2	3	4	5	6	7	8	14	15	20
1		EX	*FILESUP,											
2		FUNCT	DEALLOCATE,											
3		VOLUME	NAME=VOLA,											
4		FILE	NAME=FILER, EXP=NO, PW=DEADFILE,											
5			DEVADD=(04,1), DEVADD=(04,2), DEVADD=(04,3),											
6			DEVADD=(04,4),											
7														
8		FUNCT	DEALLOCATE,											
9		VOLUME	NAME=VOLC,											
10		FILE	NAME=FILEX, EXP=YES,											
11			DEVADD=(04,3), DEVADD=(04,1),											
12		DAY	67200,											
13														
14		FUNCT	DEALLOCATE,											
15		VOLUME	NAME=VOLD,											
16		FILE	NAME=FILET,											
17			DEVADD=(04,4), DEVADD=(04,1),											
18														
19														
20														
21														

Summary of Job Control Statements for Deallocate Function

Table 4-3 contains a complete summary of the job control statements for the deallocate function.

LOAD AND UNLOAD FUNCTIONS

The load function is used to load a file onto a mass storage volume from punched cards, magnetic tape, or another mass storage file of the same organization. The unload function is used to unload a file from a mass storage volume onto punched cards, magnetic tape, printer, or another mass storage file. However, at least one of the files in the load/unload function must be on mass storage. When loading or unloading one mass storage file to another, the files must be of the same organization, with the following exceptions: the input file can be sequential and the output file can be indexed sequential, or, the input file can be sequential and the output file can be a direct access file. All standard fixed-length card and tape formats can be used with the load and unload functions.

A load or unload function is requested by a Function statement whose first parameter is either LOAD or UNLOAD. (The File statements specify whether the operation is to or from mass storage.) The Function statement is followed by two File statements, one for the input file and one for the output file. When a File statement is for a mass storage partitioned sequential file, it may be followed by one or more Member statements. The Member statements associated with a File statement must follow immediately after that statement. There may also be one Exits statement per load or unload function. This is explained under "Exits Statement" later in this section.

Table 4-3. Summary of Job Control Statements for Deallocate Function

Statement	Command (Operation Code)	Parameter Name	Parameter Value (Operands Field)	Function	Comments
Execute	EX	Program Segment Name	*FILESUP,	Directs the Supervisor to load File Support C.	Required when running under control of mass storage Supervisor.
Function	FUNCT	Function Name	DEALLOCATE,		Required.
Volume	VOLUME				Required.
		Volume Name	NAME = volume-name,	Gives the name of the first volume containing the file to be deallocated.	Required.
File	FILE				Required.
		File Name	NAME = file-name,	Names the file to be deallocated.	Required.
		Expiration Date	EXP = { NO } { YES } ,	Specifies whether the file's expiration date is checked.	Optional. Assumed value = YES.
		Password	PW = password,	Gives the password for the file.	Required if file is protected by a password.
		Device Address	DEVADD = (pcu, drive),	Gives the physical device address of a volume. May be repeated for multivolume files.	Optional. Assumed value = (04, 0), for all volumes of a file.
Day	DAY	Expiration Date	yyddd,	Specifies year and day against which the file's expiration date is to be checked.	Optional. Assumed value = Supervisor's current date field.

4-34

#5-618

SECTION IV. FILE SUPPORT C

SECTION IV. FILE SUPPORT C

Job Control Language for Load and Unload Functions

The job control language is similar for load and unload functions. Figure 4-4 shows all the job control statements that are required for, or that can be used in, the loading and unloading of files. Each statement and its parameters are described in following paragraphs. Many of these parameters can be omitted for any given load or unload function.

EASYCODER
CODING FORM

PROBLEM								PROGRAMMER								DATE								PAGE OF							
CARD NUMBER	V	W	R	LOCATION	OPERATION CODE	OPERANDS																									
						1	2	3	4	5	6	7	8	14	15	20	21	26	27	32	33	38	39	44	45	50	51	56	57	62	63
1					EX	*FILESUP,																								Required.	
2					FUNCT	{LOAD }																								Required.	
3						{UNLOAD}																									
4					FILE	{IN }																								Required. One for	
5						{OUT}																								each file.	
6						NAME=file-name,																								Required if mass storage.	
7						DEVTYPE=device-type,																								See page 4-37.	
8						DEVADD=(pcu,drive),...																								Optional.	
9						ITEM=item-length,																								Optional. Tape or Card only.	
10						REC=record-length,																								Optional. Tape only.	
11						BAN={YES																								Optional. Tape only.	
12						{NO																									
13						{banner																									
14						PAR={ODD }																								Optional. Tape only.	
15						{EVEN}																									
16						PAD=padding,																								Optional. Tape only.	
17						MODE={SPEC }																								Optional. Card	
18						{STAND }																								file only.	
19						PW=password,																								Req. for file with	
20																														password.	
21						BUCKET={REL }																								Optional. Direct access	
22						{ABS }																								output only.	
23						PROT={A }																								Specified only for	
24						{B }																								mass storage	
25						{AB }																								output file.	
26						{NO }																									
27						IMBED=unused-items-per-string,																								Optional. Indexed	
28																														sequential output only.	
29						RELEASE={YES }																								Optional. Partitioned sequentia	
30						{NO }																								output file only.	
1						REPORT=report-number,...																								Optional. Unload to printer only.	
2					MEMBER	NAME=member-name,																								Optional. Partitioned	
3																														sequential only.	
4					EXITS	PROG=program-name,																								Required for direct	
5						LMA=low-memory-address,																								access output file.	
6																														Optional for all other files.	
7				E																											
8																															
9																															

Figure 4-4. Job Control Statements for Loading and Unloading Files

EXECUTE STATEMENT

The Execute statement with the parameter *FILESUP directs Supervisor C to load File Support C. The format of the Execute statement is shown in Figure 4-1.

FUNCTION STATEMENT

The Function statement specifies to File Support C what function to perform. This statement is required; and to perform either the load or the unload function, it must be coded as shown in the following example.

CARD NUMBER	Y	W	LOCATION	OPERATION CODE	OPERANDS
1 2 3 4 5 6 7 8	14 15	20 21	62 63	60	
				FUNCT	(LOAD)
					(UNLOAD)

The value chosen for this parameter may be either load or unload. However, the input and output File statements specify the direction of data flow.

FILE STATEMENTS

Both the input and the output File statements are described here, since they are essentially the same in form. The input file for a load or an unload function is identified by a File statement whose first parameter is IN; the output file is identified by a File statement whose first parameter is OUT.

The file name parameter is required to specify the name of the mass storage file. This parameter can be omitted for other device types, in which case label checking is omitted also. For printer output, the file name is always taken from the input mass storage file. The device type parameter specifies the storage medium used to contain the file and describes the type of device used to access the file. The device address parameter allows the physical device address of the file to be altered from the assumed value. When operating with multivolume mass storage files, more than one device address parameter may be specified. Both the input and output files may be assigned to devices connected to the first, second, or both I/O sectors.

The remaining parameters apply to specific storage media. For a mass storage file, size parameters are obtained from the volume directory. The item size, record size, banner, padding, and parity parameters apply to a magnetic tape file. Item size and mode parameters apply to a punched card file. The password parameter applies to a mass storage file. The bucket parameter applies only to an output direct access file on mass storage. The protect parameter applies to an output mass storage file. The imbed parameter applies to an output, indexed sequential mass storage file. The release parameter applies to an output, partitioned sequential mass storage file. The report parameter applies to print-image files.

In/Out Parameter

The in/out parameter is required to specify whether the File statement applies to the input file or to the output file. This parameter must be the first parameter of the File statement. The format of this parameter is as follows.

{ IN } { OUT }

The value chosen for this parameter will have the significance shown below.

IN = The input file is being described by this File statement.

OUT = The output file is being described by this File statement.

File Name Parameter

The file name parameter is used to specify the name of the file being used as input or output. This parameter is required for mass storage files and optional for files stored on other device types. This parameter does not apply to an output printer file. The format of this parameter is as follows.

NAME=file-name,

The name of the input or output file specified here can be up to ten characters long. Trailing spaces are automatically added by the load or unload function. When this parameter is omitted for a non-mass storage file, the file name is not checked on that storage medium by the load or unload function.

Device Type Parameter

The device type parameter is used to specify the storage medium used for the file, as well as the type of peripheral device used to access the file. The format of this parameter is as follows.

DEVTYPE=device-type,

If this parameter is absent, the assumed value is mass storage. When present, this parameter specifies the device type on which the file resides; the parameter value must be chosen from the following list.

Device-Type	Description
227	Card reader or card punch
224-1	Card reader/punch
223	Card reader
224-2	Card reader/punch
214-1	Card punch

Device Type	Description
214-2	Card reader/punch
204B	1/2-inch magnetic tape
155	Mass storage
258	Mass storage
259	Mass storage (259, 259A, 259B)
261	Mass storage
262	Mass storage
273	Mass storage
222	Printer
206	Printer

Device Address Parameter

The device address parameter allows the programmer to change the peripheral address assignment used to access the file. The format of this parameter is as follows.

Card or Printer

DEVADD = (pcu),

Tape

DEVADD = (pcu, drive),

Mass Storage

DEVADD = (pcu, drive), ...,

This parameter is optional. When it is used, the peripheral address assignment (pcu) must be specified as two octal digits. The high-order (I/O) bit is ignored, but all other bits, including the sector bits, must be specified. Also, the drive number must be specified as one octal digit. When this parameter is not used, the load or unload function assumes that one of the following standard peripheral addresses (depending on the device type) is to be used.

I/O Media	Peripheral Address
Punched card input	pcu 41
Punched card output	pcu 01
Magnetic tape input	pcu 40, drive 1
Magnetic tape output	pcu 00, drive 1
Mass storage input	pcu 44, drive 0
Mass storage output	pcu 04, drive 0
Printer output	pcu 02

If the file is on magnetic tape, it can be stored on multiple reels. They all must use the same device address.

If the file is on mass storage, more than one device address parameter can be submitted; the specified devices are used cyclically. For a multivolume mass storage file, two or more

device address parameters can be specified to avoid volume changing. For some operations, multiple mass storage device addresses are required. The load/unload function uses these addresses in the order in which they appear on the File statement.

Table 4-4 illustrates the minimum device requirements for each file organization.

Table 4-4. Minimum Device Requirements for Mass Storage File Organizations

Mass Storage File Organization	File Usage	Minimum Number of Devices Required
Sequential or Partitioned Sequential	Input or Output	One.
Direct Access	Input	One.
	Output	One for each file volume.
Indexed Sequential	Input	One for MCINDEX; one for OVERFLOW if not on same volume as MCINDEX; and one for current data unit if not on same volume as MCINDEX and OVERFLOW.
	Output	One for each file volume.

Item Length Parameter

The item length parameter is used to specify the length in characters of each item in a non-mass-storage file. This parameter is optional and does not apply to mass-storage files or output printer files. Item size must not exceed 4,095 characters. The format of this parameter is as follows.

ITEM = item-length,

The default assumptions for this parameter are as follows.

Mass storage _____ Obtained from the volume directory.
 Magnetic tape _____ Equal to the item size of the mass storage file.
 Card input _____ 80 characters.
 Card output _____ Equal to the item size of the mass storage file.
 Printer _____ Obtained from the volume directory.

Record Length Parameter

The record length parameter applies only to a file stored on tape and is used to specify the number of characters in each record in the file, including the banner character, if present. The format of this parameter is as follows.

REC = record-length,

This parameter is optional. When it is omitted, the load or unload function assumes that the record length of a tape file is equal to the mass storage file block length plus one character if the magnetic tape file is bannered. Record length has no meaning in a printer or card file. If the file is a mass storage file, this parameter is ignored since the record length is obtained from the volume directory.

Banner Character Parameter

The banner character parameter is used to specify whether a magnetic tape file is bannered. The format of this parameter is as follows.

$\text{BAN} = \left\{ \begin{array}{l} \text{banner} \\ \text{NO} \\ \text{YES} \end{array} \right\},$
--

This parameter applies only to magnetic tape files and is optional. When this parameter is not specified, the load or unload function assumes that the file is unbannered. The value chosen for this parameter will have the significance shown below.

- banner = The file is bannered and the parameter specifies the banner character written as any two octal digits.
- NO = The file is unbannered.
- YES = The file is bannered and File Support C assigns 41 (octal) as the banner character for an output file.

Note that for an input file, the banner character parameter specifies only the presence or absence of the banner character; the actual value is irrelevant. For an output file, the value specified is written as the first character of the output data records. To specify record size, refer to "Record Length Parameter" above.

Parity Parameter

The parity parameter is used to specify the parity of recording for magnetic tape files. The format of this parameter is as follows.

$\text{PAR} = \left\{ \begin{array}{l} \text{ODD} \\ \text{EVEN} \end{array} \right\},$

This parameter applies only to magnetic tape files and is optional. When this parameter is not specified, the function assumes that the parity is odd. The value chosen for this parameter has the significance shown in the following.

- ODD = The parity of recording is odd.
- EVEN = The parity of recording is even.

NOTE: A Honeywell file containing the octal character 12 should be handled in odd parity. If this is not done, 12₈ is unloaded as 00₈ and loaded as 00₈.

Padding Character Parameter

The padding character parameter is used to specify the padding character to be used with magnetic tape files. The padding character that is specified must be two octal digits. The format of this parameter is as follows.

PAD=padding,

This parameter applies only to magnetic tape files and is optional. When this parameter is not specified and the file is an odd parity file, the load or unload function assumes the padding character is 77 (octal). When the file is an even parity file, the function assumes the padding character is 11 (octal). If the first character of an item on the input tape is equal to the padding character, the tape input/output routines assume that this is not a valid item, bypass it, and advance to the next item.

Mode Parameter

The mode parameter is used to specify the reading or punching mode to be used for card files. The format of this parameter is as follows.

MODE= { STAND }
{ SPEC } ,

This parameter applies only to card files and is optional. When it is not specified, the function assumes that the "special" punching or reading mode is used. The value chosen for this parameter has the significance shown below.

STAND = The standard card mode is used.

SPEC = The special card mode is used.

NOTE: Refer to the appropriate programmer reference and card reader/punch manuals for a description of the differences between the standard and special modes.

Password Parameter

The password parameter is used to specify the password value to be checked against the password assigned to the mass storage file. The format of this parameter is as follows.

PW=password,

This parameter applies only to mass storage files and is required if the file has password protection. When specified, the password can be up to eight characters; trailing spaces are

added automatically. If the file has password protection, the password check is made regardless of the presence or absence of the password parameter. If the file does not have password protection, a request for password checking must not be made.

Bucket Addressing Parameter

The bucket addressing parameter is used to specify the type of bucket addressing used for an output direct access file. The format of this parameter is as follows.

$$\text{BUCKET} = \left\{ \begin{array}{l} \text{REL} \\ \text{ABS} \end{array} \right\},$$

This parameter is optional and applies to output, direct access files only. When this parameter is not specified, the function assumes that absolute (actual) bucket addresses are being used. When this parameter is specified, the value chosen has the significance shown below.

REL = Relative bucket addresses are supplied by a user-written routine.

ABS = Actual bucket addresses are supplied by a user-written routine.

Parameters describing the user-written routine are supplied in the Exits statement (described under "Exits Statement" which follows).

Protection Status Parameter

The protection status parameter is used to indicate the write protection assigned to an output mass storage file when the file was allocated. The format of this parameter is as follows.

$$\text{PROT} = \left\{ \begin{array}{l} \text{A} \\ \text{B} \\ \text{AB} \\ \text{NO} \end{array} \right\},$$

This parameter is required for an output mass storage file which has write protection. When this parameter is not specified, the load or unload function assumes that the file has no write protection. When a file has been allocated with a protection parameter containing a value other than NO (described under "Protection Status Parameter" earlier in this section), the same value that was used during the allocation must be used when describing the output mass storage file. When this parameter is specified, the value chosen has the significance shown in the following.

A = The file was assigned A-file write protection.

B = The file was assigned B-file write protection.

AB = The file was assigned both A- and B-file write protection.

NO = The file was not assigned write protection.

Imbed Parameter

The imbed parameter specifies the number of item positions that are set aside as unused at the end of each data string of a file when the file is loaded. This parameter applies only to an output indexed sequential file on mass storage.

If these unused item positions (identified by 41 octal) are left within the data strings of an indexed sequential file when it is loaded, items can subsequently be inserted in those files without having to use the cylinder overflow or general overflow areas until all unused item positions are used.

```
IMBED=unused-items-per-string,
```

The default assumption is that item positions are not set aside. The value of the imbed parameter must be less than the number of items per string (as specified by the block and string parameters when the file is allocated).

Release Parameter

The release parameter specifies whether or not the contents of an output, partitioned sequential file are to be released before data is loaded. (Releasing a file consists of removing all member names from the member index so that the entire data area of the file is available as unused space.) This parameter applies only to a partitioned sequential output file on mass storage.

```
RELEASE= { YES }  
          { NO  } ,
```

YES - The file is to be released before loading.

NO - The file is not to be released before loading.

The default assumption is that the file is not to be released before loading. Thus, data from input members is either added to the file as new members or it overlays existing members of the same name.

Report Number Parameter

When a sequential print-image file containing one or more reports is being unloaded onto the printer, the report number parameter specifies the number of the report to be printed. This parameter has the following format.

```
REPORT = report number,...
```

SECTION IV. FILE SUPPORT C

The report number given is the decimal number which is contained (in binary) in locations 2 and 3 of each item of the mass storage file. This parameter is specified when unloading a mass storage print-image file to the printer. It is optional and can be repeated for as many reports as the user wishes to print. When this parameter is used, character position 56 of the *VOLDESCR* entry for the file must contain 42. Refer to "Unloading Mass Storage Files to the Printer" in this section and to Appendix G.

MEMBER STATEMENTS

The input File statement and the output File statement may be followed by member statements if that input or output file is a partitioned sequential file on mass storage. Each Member statement used specifies the name of one member to be loaded or unloaded. There must be one Member statement for each member to be loaded or unloaded. However, if the programmer desires, he can omit all Member statements, and all members of the input file will be loaded onto the output file. Refer to page 4-58 for more detailed information.

CARD NUMBER	OPERATION CODE	LOCATION	OPERANDS
1 2 3 4 5 6 7 8	14 15 20 21		62 63 60
			MEMBERNAME=member-name,

Member Name Parameter

The member name parameter is used to specify the name of the member to be loaded or unloaded. When used with an input file, each named member is extracted for output to any media; when used with an output file, each incoming member is renamed by the corresponding member statement prior to entry in the mass storage file. The format of this parameter is shown in the preceding example. The member name specified in this parameter can be up to 14 characters long; trailing spaces are added automatically.

The status of a member loaded by File Support C allows unrestricted processing.

EXITS STATEMENT

The Exits statement enables the load or unload function to exit to a user-supplied routine just after retrieving the input item and just prior to writing the output item. This statement is required for loading a direct access mass storage file, because bucket addresses must be supplied by the user. It is optional for all other load/unload operations. The Exits statement is used to describe the user-supplied routine.

Program-Segment-Name Parameter

The program-segment-name parameter is used to supply the name of a single-segment user-written routine. The format of this parameter is as follows.

```
PROG=program-segment-name,
```

The first six characters of this parameter are the program name; the last two are the segment name. Imbedded spaces are significant; trailing spaces are added.

Low-Memory-Address Parameter

The low-memory address parameter specifies in decimal the lowest memory address used by the user-supplied routine. The value of this parameter must indicate a memory location below 32K. The format of this parameter is as follows.

```
LMA=low-memory-address,
```

The highest location occupied by the own-code routine cannot exceed the location specified in the Supervisor communication area as the highest memory location available.

Job Control Language Examples for Load and Unload Functions

The following job control statements request that a file be loaded onto mass storage. It is specified that the input file is a bannered magnetic tape file named FILEXX. The output file is on mass storage and is also named FILEXX. Own-coding is employed to modify items while loading the file. (See next page for coding example.)

For this example, it is assumed that the mass storage file FILEXX was allocated as a sequential file with the following values.

No password protection	
No write protection	
Item length	150 character
Record length	450 characters
Block length	900 characters (6 items per block)

The following figure is the assumed value for the mass storage file.

Device address	(04,0)
----------------	--------

The following assumed values will be assigned to the tape input file.

Item length	150 characters
Record length	901 characters (same as mass storage block size plus one character for banner)
Device address	(40,1)
Parity	Odd
Padding	77 (octal)

SECTION IV. FILE SUPPORT C

CARD NUMBER	OPERATION CODE	LOCATION	OPERANDS
1	EX		*FILESUP,
2	FUNCT		LOAD,
3	FILE		IN,
4			NAME=FILEXX,
5			DEVTYPE=204B,
6			BAN=YES,
7	FILE		OUT,
8			NAME=FILEXX, DEVTYPE=259,
9	E		EXITS PROG=XXCODE01, LMA=15900,

In the following example, job control statements request that two reports in a print-image file on mass storage be unloaded to the printer. The input mass storage file is named FILEAP.

In the example below, the following assumptions have been made:

1. For the mass storage file FILEAP:
 - a. That it was allocated as a sequential file without password protection.
 - b. That, at a time prior to unloading, character 56 of the *VOLDESCR* item for this file was set to 42 octal and that character 57 of the *VOLDESCR* item was set to indicate the number of control characters in each item of the file (in binary). These characters must have been set by a means other than File Support C. See Appendix G for description of items of a print-image file. See Table 3-11 for information on own-coding Exit 01.
 - c. The device address is (04, 0).
2. For the on-line printer file:

The device address is (02).

CARD NUMBER	OPERATION CODE	LOCATION	OPERANDS
1	EX		*FILESUP,
2	FUNCT		UNLOAD,
3	FILE		IN,
4			NAME=FILEAP,
5			REPORT=9, REPORT=12,
6	FILE		OUT,
7	E		DEVTYPE=222,

Parameters for file name, item size, and record size are irrelevant to the output File statement when unloading onto the printer.

SECTION IV. FILE SUPPORT C

In the following example selected members of a partitioned sequential file are unloaded onto the printer.

CARD NUMBER	OPERATION CODE	LOCATION	OPERANDS
1 2 3 4 5 6 7 8	14 15 20 21		62 63 80
1	EX		*FILESUP,
2	FUNCT		UNLOAD,
3	FILE		IN, NAME=FI PART0001, DEVTYPE=259,
4			PW=SESAME, DEVADD=(64, 1),
5	MEMBER		NAME=FI PARTMEMBERAH,
6	MEMBER		NAME=FI PARTMEMBERXX,
7	MEMBER		NAME=FI PARTMEMBERAA,
8	FILE		OUT, DEVTYPE=222,
9		E	
10			
11			

In the above example, the three named members are unloaded onto the printer in the order named. The file is on device 1, second sector. All permit switches on the control unit can be in PROTECT position.

In the following example, a tape file is loaded onto a mass storage partitioned sequential file.

CARD NUMBER	OPERATION CODE	LOCATION	OPERANDS
1 2 3 4 5 6 7 8	14 15 20 21		62 63 80
1	EX		*FILESUP,
2	FUNCT		LOAD,
3	FILE		IN, NAME=TAPEFL004A, DEVTYPE=204B,
4			PAR=ODD, PAD=63, BAN=YES, REC=801,
5			ITEM=80, DEVADD=(40, 2),
6	FILE		OUT, NAME=FI PART001, DEVTYPE=259,
7			PW=OPENDOOR, DEVADD=(04, 1),
8	MEMBER		NAME=PARTMEMBER004A,
9	EXIT		TS, PROG=TAPEPT01, LMA=17000,
10		E	
11			

In the above example, the tape file (single file on one or more reels) enters as a single member (renamed PARTMEMBER004A). If the member does not already exist in the mass storage member index, a new member is created in as many mass storage blocks as are necessary to contain the tape file. The own-code program will perform user-desired modifications of the data. The mass storage item, record, and block sizes are as previously determined by the allocate function.

In the following example, the statements request that a file on magnetic tape be loaded onto mass storage. Assume that the mass storage file named FILEC was allocated as an indexed sequential file on three volumes with an item size of 252 and a record and block size of 1,512 (6 items). Also assume that the index area, the general overflow area, and the first data unit

SECTION IV. FILE SUPPORT C

were allocated on the first volume and that additional units were allocated on the second and third volumes. Three device addresses must be specified. The first volume of the file, containing the index and general overflow areas (and the first data unit of allocation), is mounted on drive 1. The second volume of the file is mounted on drive 2. The third volume is mounted on drive 3. The imbed parameter requests that three item positions be left unused in each string of the file's data area when items are loaded from the input file. This value of the imbed parameter implies that the file was allocated with a data string that will hold at least four items. (String size equals one or more blocks, each with a capacity of six items.)

CARD NUMBER	Y	N	LOCATION	OPERATION CODE	OPERANDS																								
						1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
1				EX	#FILESUP,																								
2				FUNCT	LOAD,																								
3				FILE	IN,																								
4					DEVTYPE=204B,																								
5					ITEM=250,																								
6					REC=501,																								
7					BAN=YES,																								
8				FILE	OUT,																								
9					NAME=FILEC,																								
10					IMBED=3,																								
11					DEVADD=(04,1), DEVADD=(04,2),																								
12			E		DEVADD=(04,3).																								

Summary of Job Control Statements for Load and Unload Functions

Table 4-5 contains a complete summary of the job control statements for the load and unload functions.

Table 4-5. Summary of Job Control Statements for Load/Unload Functions

Statement	Command (Operation Code)	Parameter Name	Parameter Value (Operands Field)	Function	Comments
Execute	EX	Program Segment Name	*FILESUP,	Directs Supervisor to load File Support C.	Required when running under control of mass storage Supervisor.
Function	FUNCT	Function Name	{ LOAD UNLOAD } ,	Specifies the function to be performed.	Required.
File	FILE	File Function	{ IN OUT } ,	Identifies the file as input or output.	Two required; one for input file and one for output file.
		File Name	NAME = file-name,	Names the file for the function.	Required for mass storage files only. Optional for other media.
		Device Type	DEVTYPE = device-type,	Gives the type of device used for the file. See list on pages 4-37, 4-38.	Optional for mass storage. Required for other media. Assumed value is 259.
		Device Address	DEVADD = (pcu, drive),	Allows changing the assignment of the peripheral device.	Optional. See device-type list for assumed values, page 4-38.
		Item Size	ITEM = item-length,	Gives the number of characters in each item (card or tape).	Optional. See item length parameter, page 4-39.
		Record Size	REC = record-length,	Gives the number of characters in each tape record including banner character, if any.	Tape only. Optional. Assumed value=mass storage block size (adjusted for banner character).
		Banner Character	BAN= { banner NO YES } ,	For tape files only. Gives the banner character to be used, or the function assigns 41 (octal) as the output banner character.	Optional. Assumed value = NO.
		Parity	PAR= { ODD EVEN } ,	For tape files only.	Optional. Assumed value = ODD.
		Padding Character	PAD = padding,	For tape files only. Gives the padding character to be used.	Optional. When omitted, 77 (octal) is used for odd parity file; 11 (octal) is used for even parity file.
		Mode	MODE= { SPEC STAND } ,	For card files only. Gives the card reading or punching mode.	Optional. Assumed value = SPEC.

4-49

#5-618

Table 4-5.(cont). Summary of Job Control Statements for Load/Unload Functions

Statement	Command (Operation Code)	Parameter Name	Parameter Value (Operands Field)	Function	Comments
File (cont)		Password	PW = password,	For mass storage files. Gives the password for the file.	Required if file is protected by a password.
		Bucket Addressing Mode	BUCKET= {REL}, {ABS},	Gives the bucket addressing mode for direct access output files on mass storage.	Direct access files only. Optional. Assumed value = ABS.
		Protection Status	PROT= {A}, {B}, {AB}, {NO}	Gives the type of write protection assigned to the file when it was allocated.	Required if file was assigned write protection when it was allocated. Assumed value = NO.
		Imbedded Overflow	IMBED= Unused-item-positions-per-string,	Specifies the number of item positions that are set inactive in each data string of a file when the file is loaded.	Optional. Applies only to indexed sequential files. Assumed value = 0
		Release	RELEASE= {YES}, {NO}	Specifies whether the contents of the file are to be released before data is loaded.	Optional. Applies only to output partitioned sequential files. Assumed value = NO.
		Report	REPORT = report-number,	Specifies the report number of the report(s) to be printed.	Optional. Applies only to sequential files unloaded to the printer in print-image format.
Member	MEMBER	Member Name	NAME = member-name,	Each member statement names a member to be processed.	Optional. When omitted, all members are processed. Applies only to partitioned sequential files.
Exits	EXITS	Program and Segment Names	PROG = program-segment-name,	Names the single-segment own-code routine.	Required for direct access mass storage output file; optional otherwise.
		Low Memory Address	LMA = low-memory address,	Gives the lowest main memory location used by the own-code routine, in decimal.	

4-50

#5-618

MAP FUNCTION

The map function is used to extract selected information about the files on one volume. This function can perform three actions: it can produce a description of all files or specified files on the volume, produce a description of expired files, or list unassigned tracks on the volume. The information produced is taken from the contents of the volume directory and listed by means of a printer or print-image tape.

The map function is requested by a Function statement whose first parameter is MAP. The second parameter of the Function statement gives the type of mapping desired. In most cases, these two parameters are sufficient. The Function statement may be followed by a Volume statement, one or more File statements, and a Day statement. The Volume, File, and Day statements must be submitted in that order after the Function statement. They serve to restrict map information, as required. If a File List statement is specified, it must follow all other File statements.

Job Control Language for Map Function

The following list shows all the job control statements that are required for, or that can be used in, the mapping of a volume. Each statement and its parameters are described in the following paragraphs.

EASYCODER
CODING FORM

PROBLEM _____										PROGRAMMER _____										DATE _____										PAGE _____ OF _____									
CARD NUMBER				LOCATION				OPERATION CODE				OPERANDS																											
1	2	3	4	5	6	7	8	14	15	20	21																												
1								EX				*FILESUP,														Required.													
2								FUNCT				MAP, (DESCR														Required.													
3												{ EXPIRED }																											
4												{ UNUSED }																											
5								VOLUME				NAME=volume-name,														Optional.													
6												DEVADD=(pew,drive),																											
7								FILE				NAME=file-name,														Optional.													
8								DAY				yyddd,														Optional.													
9								FILE				LIST,														Optional.													
10												DEVTYPE=device-type,																											
11												DEVADD=(pew,drive),																											
12																																							

EXECUTE STATEMENT

The Execute statement with the program segment name *FILESUP directs the Supervisor to load File Support C.

FUNCTION STATEMENT

The Function statement specifies to File Support C what function to perform. This statement is required. Its format is shown in the preceding example. The format of the Function statement's first two parameters is shown in the following example.

MAP, {	DESCR	}
	EXPIRED	}
	UNUSED	}

Both of these parameters are required. The value assigned to the second parameter has the significance shown in the following list.

- DESCR = A description based on the volume directory information for selected files, or for the whole directory, is produced.
- EXPIRED = A description is produced based on volume directory information for all files whose expiration date is less than or equal to the date specified by the Day statement or the current Supervisor date.
- UNUSED = A listing of all unassigned tracks on the mass storage volume is produced.

VOLUME STATEMENT

The name of the volume to be mapped and its device address are specified by the Volume statement. This statement is optional. When this statement is omitted, the volume name is not checked and the map function assumes that the peripheral address assignment (pcu) is 04, drive number 0.

Volume Name Parameter

The volume name parameter is used to specify the name of the volume to be mapped. The format of this parameter is as follows.

NAME=volume-name,

This parameter is optional. When a volume name is not specified, the map function does not check the volume name in the volume label.

Device Address Parameter

The device address parameter is used to specify the peripheral address assignment (pcu) and the drive number of the mass storage volume. The format of this parameter is as follows.

DEVADD=(pcu, drive),

This parameter is optional. When a device address is specified, the peripheral address assignment (pcu) is given in two octal digits. The high-order bit is not significant. The drive number is given in one octal digit. When a device address is not specified, the map function assumes the peripheral address is 04, drive number 0.

SECTION IV. FILE SUPPORT C

FILE STATEMENT

If a listing of the volume directory information for selected files is desired, one or more File statements must be submitted. The format of the File statement is as follows.

CARD NUMBER	Y	W	PER	LOCATION	OPERATION CODE	OPERANDS	
1 2 3 4 5 6 7 8	14	15	20	21	62	63	80
1					FILE	NAME=file-name,	
2							

The File statement is optional. If a File statement is not submitted, the volume directory information for all files on the volume is listed. This statement can be used only with the DESC option. When the volume directory information for selected files is desired, a File statement for each desired file must be submitted. Files are listed in the order in which they appear in the volume directory. The names of files not found in the directory appear at the end of the printed listing.

DAY STATEMENT

The Day statement is used to specify the date against which each file's expiration date will be checked when the function specified is the mapping of expired files. The format of the Day statement is as follows.

CARD NUMBER	Y	W	PER	LOCATION	OPERATION CODE	OPERANDS	
1 2 3 4 5 6 7 8	14	15	20	21	62	63	80
1					DAY	yyddd,	
2							

This statement is optional. When a Day statement is submitted, the yy portion of the parameter specifies the last two digits of the year, and the ddd portion specifies the number of the day in the year. For example if the day is the first of January and the year is 1975, the parameter is coded 75001. When a Day statement is not submitted, the file's expiration date is checked against the current date field of the Supervisor.

FILE STATEMENT FOR THE LIST FILE

The device type and peripheral device address of the device to be used for the map listing may be specified by a File statement whose first parameter is LIST. If no such File statement appears, the listing is produced on a printer with the device address of (02).

CARD NUMBER	Y	W	PER	LOCATION	OPERATION CODE	OPERANDS	
1 2 3 4 5 6 7 8	14	15	20	21	62	63	80
1					FILE	LIST,	
2							

Device Type Parameter

The device type parameter specifies the storage medium and the peripheral device on which the list file will be created.

DEVTYPE=device-type,

The device type number may be one of the following values.

- 206 - Type 206 Printer,
- 222 - Type 222 Printer, or
- 204B - Type 204B Magnetic Tape Unit

The default assumption is a Type 222 Printer.

Device Address Parameter

The device address parameter allows changes to be made in the standard assignment for the peripheral device used for the list file.

Tape

DEVADD=(pcu, drive),

Printer

DEVADD=(pcu),

The peripheral address assignment is written as two octal digits. All bits including the sector bits must be specified. The drive number is written as one octal digit.

If the device address parameter is omitted, the default assumption depends on the device type as follows.

Device	Assumed Address
Type 222 Printer	(02)
Type 206 Printer	(02)
Type 204B Magnetic Tape Unit	(00, 3)

Job Control Language Examples for Map Function

In the following example, the job control statements request a listing of the volume directory information for files named FILEFF and FILEGG. The volume name is not checked and the map function assumes that the device address is 04, drive number 0. In this case, File Support programs are resident on the volume being mapped.

CARD NUMBER	Y	M	A	P	R	LOCATION	OPERATION CODE	OPERANDS						
1	2	3	4	5	6	7	8	14	15	20	21	42	43	80
1								EX		*FILESUP,				
2								FUNCT		MAP, DESCR,				
3								FILE		NAME=FILEFF,				
4								FILE		NAME=FILEGG,				

In the following example, the job control statements request a listing of the unused tracks on volume PTMS04. The volume is mounted on device 3. A print-image tape (for offline listing) will be created on tape unit 5.

EASYCODER
CODING FORM

PROBLEM _____ PROGRAMMER _____ DATE _____ PAGE ____ OF ____

CARD NUMBER	Y T M P R	LOCATION	OPERATION CODE	OPERANDS	
				14 15 20 21	62 63 80
1			EX	#FILESUP.	
2			FUNCT MAP, UNUSED.		
3			VOLUME NAME=PTMS04, DEVADD=(04, 3).		
4			FILE LIST, DEVTYPE=2048, DEVADD=(00, 5).		
5		E			
6					
7					
8					

The following example illustrates a request to map three volumes for descriptive and unassigned information. In the descriptive operation, assumed values for the volume statements have not been used. Three disk drives are online, so all three volumes are mapped without changing disk packs. All output is on a printer whose peripheral address is 02.

EASYCODER
CODING FORM

PROBLEM _____ PROGRAMMER _____ DATE _____ PAGE ____ OF ____

CARD NUMBER	Y T M P R	LOCATION	OPERATION CODE	OPERANDS	
				14 15 20 21	62 63 80
1			EX	#FILESUP.	
2			FUNCT MAP, DESCR.		
3			VOLUME NAME=PTMS01, DEVADD=(04, 0).		
4			FUNCT MAP, DESCR.		
5			VOLUME NAME=PTMS02, DEVADD=(04, 1).		
6			FUNCT MAP, DESCR.		
7			VOLUME NAME=PTMS03, DEVADD=(04, 2).		
8		E			
9			EX	#FILESUP.	
10			FUNCT MAP, UNUSED.	DEV 0 VOLUME MAPPED. NO NAME CHECK.	
11			FUNCT MAP, UNUSED.	DEV 1 VOLUME MAPPED. NO NAME CHECK.	
12			VOLUME DEVADD=(04, 1).		
13			FUNCT MAP, UNUSED.	DEV 2 VOLUME MUST BE PTMS03.	
14			VOLUME NAME=PTMS03, DEVADD=(04, 2).		
15		E			
16					

When a file is multivolume, each map listing will give the relative volume number (0-7) of each file volume and name the subsequent volume on which the file continues. Combining map listings in the relative volume order appropriate to each file produces a complete description of a multivolume file.

Summary of Job Control Statements for Map Function

Table 4-6 contains a complete summary of the job control statements for the map function.

Table 4-6. Summary of Job Control Statements for Map Function

Statement	Command (Operation Code)	Parameter Name	Parameter Value (Operands Field)	Function	Comments
Execute	EX	Program Segment Name	*FILESUP,	Directs Supervisor to load File Support C.	Required when run under mass storage Supervisor.
Function	FUNCT	Function Name and Listing Type	MAP, { DESCR EXPIRED } , UNUSED	Names the function and specifies the type of listing to be produced.	Required.
Volume	VOLUME				Optional.
		Volume Name	NAME = volume-name,	Names the volume to be mapped.	Optional. When omitted, the volume name is not checked.
		Device Address	DEVADD = (pcu, drive),	Gives the physical device address of the volume to be mapped.	Optional. Assumed value = (04, 0).
File	FILE	File Name	NAME = file-name,	Names each file to be mapped.	Optional (any number). When omitted, all files on the volume are mapped. Applies only to MAP, DESCR, .
Day	DAY	Date	yyddd,	All files whose expiration date is less than or equal to this value are listed if the expired option is requested.	Optional. Assumed value = Supervisor's current date. Applies only to MAP, EXPIRED.
File	FILE	File	LIST,	Specifies the output listing medium.	Optional. When omitted, listing is produced on a printer with peripheral control address of (02).
		Device Type	DEVTYPE= device type,	Specifies storage medium and peripheral device on which file will be created.	Optional. See device type parameter for possible values. Assumed value= Type 222 Printer.
		Device Address	DEVADD= (pcu, drive),	Allows changes to the standard device assignment for the list file.	Optional. See device address parameter for assumed values.

4-56

#5-618

SECTION IV. FILE SUPPORT C

PROGRAMMER'S PREPARATION INFORMATION FOR FILE SUPPORT CFile Considerations**DIRECT ACCESS FILES**

When allocating a direct access file, the item length is interpreted as including the status character (rightmost character of the item). The allocate function sets this character to "inactive" for all items in the file. For any item loaded, the value is set to "active" during the load process. The possible values (in octal) of this character after loading are as follows.

Last Block of File Volume	All Other Blocks	Meaning
76	77	Inactive item.
00	01	Active item.
40	41	Deleted item. ¹

¹ Deleted items can appear in a direct access file only if some interim processing has occurred between successive loads.

Unloading a Direct Access File

Direct access files are unloaded in a sequential manner in the physical order in which the active items are encountered on the file. Only active items are unloaded. The programmer is never requested to supply a bucket address; but he may, however, specify an own-code routine to modify, omit from the output file, or examine the item being unloaded.

Loading a Direct Access File

When loading a direct access file onto mass storage, the Exits statement must always be specified, since the programmer must supply the bucket address (in binary) for each item in the file via an own-code routine.

A direct access file is loaded in a cumulative manner; thus, all items being loaded are added to the data already in the file. A direct access file containing previously loaded data can be initialized by deallocating the file and allocating it again.

SEQUENTIAL FILES

A sequential file is always loaded and unloaded in a sequential manner. An own-code routine may be used as described for unloading a direct access file.

PARTITIONED SEQUENTIAL FILES

Each member of a partitioned sequential file is processed individually. During loading, the entire output file can be made available for new members by the use of the release parameter. Within each member, the items are processed in a sequential manner in the physical order in which they are encountered.

Unloading a Partitioned Sequential File

UNLOADING BY FILE: To unload an entire partitioned sequential file, no member names are specified in the job control file; only the file name is specified. All active members of the partitioned sequential file are unloaded in the order in which their names appear in the member index for the file.

UNLOADING SELECTED MEMBERS: To unload selected members of a partitioned sequential file, the desired member names are specified in the job control file after the file statement for the input file. These are unloaded in the order in which the names appear in the job control file.

Loading a Partitioned Sequential File

LOADING BY FILE: The programmer may load an entire partitioned sequential file by either of the following means.

1. Specify no member names in the job control file. In this case, the member names are taken from the input file.
2. Specify in the job control file after the File statement for the output file, the member names of all members which enter the output mass storage file. Input members are renamed in the order of encounter with the specified output member names.

LOADING SELECTED MEMBERS: The programmer may load selected members of an output mass storage partitioned sequential file by specifying the desired member names in the job control file after the File statement for the output file. Input members are again renamed in the order of encounter with the specified output member names.

Processing a Partitioned Sequential File by Member Names

When loading an output mass storage file, the load function takes the output member names from the job control file if specified or from the input file if not specified. When mass storage is input, names are found in the input member index. When card or tape is input, columns 51 through 64 of each 1HDRA record contain each member name.

If the name under which the member is to be loaded already exists in the member index of the output mass storage file and if the member can be processed in the output-only mode, the input data replaces the member's data on the output mass storage file. This is true whether loading by file or member name. If the member name does not already exist in the output file's member index, the input member and its data are added as a new member to the output mass storage file.

When member names are specified and the output member names in the job control file are exhausted before all indicated input members have been processed, loading is terminated with an appropriate halt (see Table 4-10). Member names are specified in the job control file only for a file which is on mass storage, not for card or tape files.

Loading from Mass Storage to Mass Storage

When a partitioned sequential file is being loaded or unloaded from mass storage to mass storage, both input member names and output member names may be specified. If one set of member names is exhausted before the other, an appropriate halt occurs.

INDEXED SEQUENTIAL FILES

Allocating an Indexed Sequential File

When allocating an indexed sequential file, the item length, as expressed by the item length parameter, must include the item status character (rightmost character of the item).

It is recommended that the master/cylinder index unit of allocation and the general overflow unit of allocation be placed on the same volume, with data units on other volumes. This will result in efficient use of mass storage drives during unloading and sequential processing. If sufficient drives are available, processing may be speeded up by placing the master/cylinder index and the general overflow area on separate volumes with data units on still other volumes.

Loading an Indexed Sequential File

The load function is the only method in the Mod 1 (MSR) Operating System by which an indexed sequential file can be created. All indexes are created during the load operation. A subsequent load operation will recreate all indexes while loading.

The load function reads the input file from cards, magnetic tape, a sequential or an indexed sequential file on mass storage. An input file must be ordered in ascending binary sequence by item key as specified in the file allocation. Duplicate keys are not allowed.

The load function performs the following actions.

1. It creates the master index and the cylinder index.
2. It creates the string indexes.
3. It loads the prime data area of the file with items from the input file. The item-status character of any item loaded is set to active. No active items are loaded into any of the overflow areas.
4. If requested by the user, the load function can create imbedded inactive items in each string processed. The item-status character of each imbedded item is set to deleted.

5. It initializes the cylinder overflow area of each cylinder containing a loaded item. The item status character of each item in the cylinder overflow area is set to inactive.
6. It initializes the general overflow area. Each item status character is set to inactive.
7. For the string containing the last item loaded, the load function sets the high key in the string index to all 1 bits, so that an item having a key higher than that of the last item loaded may be inserted in the file.
8. For the cylinder containing the last item loaded, the load function sets the key in that cylinder index item to all 1 bits. Prime data areas and cylinder overflow areas beyond the last string containing a loaded item are not processed. These areas are not available for data. An own-code routine may be used to examine, modify, or omit from the output file any processed item. The contents of the item key field must not be disturbed.

When a multivolume indexed sequential file is loaded, all volumes must be on-line.

Unloading an Indexed Sequential File

An indexed sequential file is unloaded in ascending sequence by item key field. Only active items are unloaded.

When a multivolume indexed sequential file is unloaded, the volume or volumes containing the master/cylinder index and the general overflow area must remain mounted throughout the unloading. The other volumes of the file are processed one at a time sequentially.

MIXED FILE ORGANIZATIONS

Loading or Unloading

When loading or unloading one mass storage file to another, the files must be of the same organization, with the following exceptions: the input can be a sequential file and the output can be an indexed sequential file, or the input can be a sequential file and the output can be a direct access file. The items of the sequential file must contain a key field acceptable to the indexed sequential file.

Own-Coding Considerations

During a load or unload function, the user may execute an own-coding routine for further item processing. In the case of direct access files which are being loaded onto mass storage, an own-code routine is required. In all other cases this own-coding routine is optional. The user may examine, modify, or omit items at this time. File Support C branches to an own-coding routine for each active item. Whenever a difference in item size exists, the item is moved to the larger of the two storage areas before the branch to own-coding. When the item is returned to File Support C, only the original punctuation (a leftmost word mark) should be present.

NOTE: During loading of an indexed sequential file, the own-code routine must not modify the value of the key. If it should accidentally change the value, one of the following actions would occur.

1. If the key of the current item is changed to a value less than or equal to the key of the preceding item, the change is not detected and loading of the file continues until completion. This file cannot be successfully processed by Logical I/O C or the File Support C unload routine since it contains an out-of-sequence key.
2. If the key of the current item is changed to a value greater than or equal to the key of the next item to be read in, the current item is read in successfully and the key-out-of-sequence exit occurs during processing of the next item.

STRUCTURE OF OWN-CODING ROUTINE

The own-coding routine must be written and assembled as a single-segment program. This program should have its origin located at a point in memory such that the program occupies the memory area immediately below the floating portion of the Supervisor or below location 32,768, whichever is lower. File Support C loads the own-coding routine only from the same storage medium (and the same executable program file) as File Support C itself.

When the user wishes to load an own-coding routine by means of the load/unload function, the following rules must be observed.

1. If the console typewriter is used during the File Support C run, the highest memory location which can be used by the own-coding routine is 3,000 locations below:
 - a. The address specified in the communication area of the Supervisor as the highest available memory location, or
 - b. Location 32,767,whichever is lower.
2. If the console typewriter is not used during the File Support C run, the own-coding routine should occupy memory immediately below the Supervisor or below location 32,767, whichever is lower.

OWN CODING CONSIDERATIONS FOR TAPE-RESIDENT OPERATION

When File Support C is tape resident, the own-coding routine must be placed on the binary run tape (BRT) beyond the load/unload routine with which it is being used. Otherwise, a halt or console timeout occurs when Floating Tape Loader-Monitor C searches for the next segment of the load/unload routine on the BRT. In this case, a response must be made to Floating Tape Loader-Monitor C to reverse direction and search again. When the segment is found, normal operation continues.

OWN-CODING COMMUNICATION WITH LOAD/UNLOAD FUNCTION

In the Exits statement of the load/unload function, the user is required to specify the lowest memory address (LMA) of the own-coding routine. One word-marked character should be reserved at that address for communication with File Support C. When File Support C gives a new item to the user, the communication character is set to zero. More detailed information on the use of this character is given in subsequent paragraphs. The branch to the own-coding routine occurs at the next character location (LMA+1). This location must contain an instruction to store the contents of the B-address register for return to File Support C.

Address communication is made through index registers 1 and 5. Index register 1 is set by File Support C to the leftmost character of the current item before branching to the own-coding routine. Index register 5 is set by the own-coding routine to the rightmost character of a user-supplied field into which the user places his binary bucket address when loading a direct access file. The field is four characters if a relative bucket address was specified, and eight characters (in the form DPCCTRR) if an actual bucket address was specified. The leftmost character of the field must contain a word mark; no other punctuation may appear in this field. If the own-coding routine modifies index registers other than X5, it must save their contents and restore them prior to returning to File Support C.

Omitting Items from the Output File

The communication character is set to zero (00) when the item is given to the user. If the item is to be written onto the output file, the communication character must remain zero. If the user desires to omit the item, he sets the communication character to one (01) prior to return to File Support C.

Invalid Bucket Addresses (Direct Access Files)

If the branch to the own-coding routine shows a communication character of one (01), then the last bucket address supplied to File Support C for a direct access file was an invalid address. When this is the case, the user may do either of the following:

1. Return to File Support C with a communication character of zero to have that item bypassed, or
2. Return to File Support C with a communication character of one to terminate the loading of this file. In this case, processing proceeds to the next File Support C function, if any.

Insufficient Space (Direct Access Files)

If the branch to the own-coding routine shows a communication character of two (02) there was no room left in the bucket or overflow area(s) of a direct access file for the last item given to the load function. In this case, the user may do either of the following:

SECTION IV. FILE SUPPORT C

1. Return to File Support C with a communication character of zero to have the item bypassed, or
2. Return to File Support C with a communication character of one to terminate the loading. In this case, processing proceeds to the next File Support C function.

Entrance to General Overflow (Direct Access Files)

If the branch to the own-coding routine shows a communication character of three (03), this indicates that the item which was last loaded into the direct access file was placed in the general overflow area. The user may:

1. Return to File Support C with a communication character of zero to continue loading the file, i. e., process the next item from the input file, or
2. Return to File Support C with a communication character of three to terminate loading of the file. In this case, processing proceeds to the next function, if any.

NOTE: At this exit the user can only examine the item which overflowed. It has been added to the file and remains there. For purposes of information, index register 1 still points to that item in memory.

Key Out of Sequence (Indexed Sequential Files)

If the branch to the own-coding routine shows a communication character of one (01) while loading (creating) an indexed sequential file, this indicates that the key of the current item is not greater than that of the last item processed. The user may:

1. Return to File Support C with a communication character of zero to have the current item bypassed and continue processing, or
2. Return to File Support C with a communication character of one to terminate loading of the file; the index areas of the file will be completed and control will be transferred to the next File Support C function, if any.

Tape and Card File Considerations

1/2-INCH TAPE FORMATS

Header Label

The tape header label is 80 characters in length and must be the first record of a file. It consists of the following fields:

Field	Characters	Contents
1	1-5	IHDRA
2	6-10	Tape serial number
3	11-15	File serial number
4	16	Minus (-)
5	17-19	Reel sequence number
6	20	Blank
7	21-30	File name

SECTION IV. FILE SUPPORT C

Field	Characters	Contents
8	31-35	Creation data
9	36	Minus (-)
10	37-39	Retention cycle
11	40	Blank
12	41-50	Reserved
13	51-64	Member name (partitioned sequential only)
14	65-80	Reserved

File Support C uses only fields 1, 2, 5, and 7, except for a partitioned sequential file, which uses field 13 also. The LEOFA record terminates loading of all files except partitioned sequential, which may consist of multiple members. The IERIA record terminates loading of a partitioned sequential file.

When a partitioned sequential file exists on tape, each member is one file of a multifile reel or reels. To identify the member on tape, the header includes an additional field in characters 51 through 64 giving the member name. The load/unload function assumes that tapes are properly positioned. No searching for the file name or member name is performed.

The load/unload function operates according to the following rules.

1. When using the load function to load a partitioned sequential file and the output member names are specified in the job control file, the member name field is not required in the header. The output member name is taken from the job control file and the tape file currently positioned is loaded as that member. If the output member names in the job control file are exhausted before all input files have been processed, processing of the load function is terminated without halting and the next function is executed.
2. When loading a partitioned sequential file and the output member names are not specified in the job control file, the member name field in each header is required. All files on that reel of tape are loaded as members until a IERIA record is encountered on that tape. The member name is assumed to be correct and is entered into the member index of the mass storage file. Note that multifile reels are processed only for a single volume partitioned sequential file.
3. When unloading a partitioned sequential file, the member name field in the file header is always filled in by File Support C.
4. When member names are specified, only those members will be unloaded. When no member names are specified, all active members on the file will be unloaded.

SECTION IV. FILE SUPPORT C

When loading a mass storage file from tape with no name specified on the job control file for the input tape file, the file name on the tape is not checked. If a name was specified for the input tape file, the name in the tape header label is checked for equality with the name specified in the job control file. In either case, the header label of the input tape is checked for the presence of 1HDRA in positions 1 through 5, and the value of the reel sequence number in positions 17 through 19 is also checked.

When unloading a mass storage file to tape with no name specified in the job control file for the output tape file, the file name field in the tape header label is not checked and the mass storage file name is written onto the output tape header label. If a name was specified for the output tape file, its header label is read and the file name field is checked for equality against the specified name; if it is equal, the specified file name is then written onto the tape header label. In either case, the output tape header label is read and checked for the presence of 1HDRA in positions 1 through 5.

Whenever any of the preceding equality checks fails, a program halt or console message will occur, at which point the operator must enter the proper response. See Table 4-10 or 4-13.

When an out-of-sequence input tape has been accepted by the operator, the sequence number of the next reel will be tested for a value one greater than the sequence number of the accepted reel.

Data Records

Data records processed by File Support C must be fixed in length (blocked or unblocked) but may use any combination of parity and bannering. The following combinations exist.

Parity	Odd	Odd	Even	Even
Banner*	Yes	No	Yes	No
*When banner is specified, one additional character should be provided in the REC = parameter.				

The four types of data record blocking and bannering can be illustrated as follows.

1. Unblocked, Unbannered



2. Blocked, Unbannered



SECTION IV. FILE SUPPORT C

3. Unblocked, Bannered

B A N	1 ITEM
-------------	--------

4. Blocked, Bannered

B A N	ITEM 1	ITEM 2	ITEM 3	ITEM 4
-------------	--------	--------	--------	--------

For those installations trying to decide which type of file to use, the odd parity, bannered file is the Honeywell recommended standard.

PADDING ITEMS: On an input tape file, File Support C examines the first character of each item for equality to the specified (or assumed) padding character. In the case of equality, the item is bypassed.

Trailer Label

The trailer label is 80 characters in length and must be the first record following the last data record of a file. Only two fields of that record are used by File Support C.

Field	Character Positions	Contents
1	1-5	Must be 1EOFΔ or 1EORΔ.
2	6-10	Is not checked on input; on output (UNLOAD), the tape record count (decimal) is entered.

In the normal situation, this record is followed by two 80-character records containing 1ERIΔ (end of recorded information) in the first five characters. However, in partitioned sequential files, each 1EOF record is followed by the next 1HDR record until all members are accounted for. Only the last 1EOF record is followed by the two 1ERI records. The trailer label containing 1EORΔ (end of reel) signifies a reel to be an intermediate reel of a multireel file.

Tape Marks

Tape marks on an input tape are ignored. On output files, tape marks are not created.

CARD FILE FORMATS

Header Label

Each card file must have a label card with the 1HDRA in columns 1 through 5 and (optionally) the file name in columns 21 through 30. Partitioned sequential files are handled in exactly the same way as in the 1/2-inch tape files previously described, except that only one 1ERI card terminates the file.

Data Items

Card format is always unblocked and unbannered. The item consists of the minimum number of cards which can handle one item. Any character positions left over are ignored. Each item is assumed to start in column 1. For load/unload operations, only the item length parameter is used to describe the card item size. Thus, if the item length parameter is specified as

ITEM=120,

two cards will contain one item.

Trailer Label

Trailer labels for cards are the same as for 1/2-inch tape, as previously described, except that field two (item count) is not used.

Unloading Mass Storage Files onto Printer

Mass storage files may be unloaded onto the printer. Partitioned sequential and sequential files may be either print-image files or non-print-image (data) files; direct access and indexed sequential files are always unloaded as data files. In addition, for sequential print-image files, it is possible to print only selected portions of the file by using the report number option.

The last item in each sequential file or in each member of a partitioned sequential file must have the standard end-of-data configuration (□ EODϕ) in positions 1 through 5. All partitioned sequential and sequential print-image files must contain at least one, and may contain more, control characters as the first (leftmost) character(s) in each item. If more than one character is present, partitioned sequential files use only the first and possibly the fourth of these characters; sequential files use the first four. (Specification of the format of a print-image file is described in Appendix G.)

The first control character always acts as the C3 variant of a PDT instruction to a Type 222 Printer. (For details, see the Programmers' Reference Manual.) The second and third characters, if present, can divide a sequential file into one or more reports. Each report consists of all information to be printed as a single unit. Reports may be unloaded in a specific order, as indicated by the job control file REPORT parameter(s) or in sequential order by report number. In partitioned sequential print-image files, each member is considered a report, regardless of report numbers. When the report number is present, the fourth control character indicates one of the following actions.

1. If the fourth bit from the left is a 0, the item is printed under control of the first control character and the next item is read from the input mass storage file.

2. If the fourth bit from the left is a 1, the item is printed as in 1 above, and a 5465 halt or a console typeout occurs to allow the printer form to be adjusted.

A = go on to the next item of the file,
G = reprint this same item and halt again,
F = go to the next File Support C function, or
E = go to the Supervisor.

The control characters must be located at the beginning of each item.

When a data file is unloaded, the contents of each item are printed in alphanumeric format; each item includes as many 120-character print lines as are required to accommodate it. Each item is preceded by an item header line which gives the item number within the file. The file name (and member name if the file is partitioned) appears at the top of each page of the listing, along with a page count. (See Figure 4-5.)

Print-image files are unloaded in the format in which they appear on mass storage. In a partitioned sequential file, each member is treated as one report. A sequential file may contain interspersed reports with the report number in characters 2 and 3 of each item. If character 56 of the *VOLDESCR* entry for a sequential file is 42 and if the output file device type is an online printer, reports are unloaded according to the numbers specified on the job control file. If report numbers were not specified, the unload function attempts to unload reports by generated number, beginning with report number 00 and continuing in binary increments of 01 until a nonexistent report number is sought on the file. At this point the function comes to a normal end-of-file exit. If the generated report number 00 or a requested report number of any value cannot be found on the mass storage file, a normal file support halt or console message occurs awaiting an operator response.

OPERATING PROCEDURES FOR FILE SUPPORT C

Loading File Support C

The File Support C program may be loaded in either of the following ways.

1. From mass storage by the Supervisor of the Mod 1 (MSR) Operating System or
2. From magnetic tape by the Floating Tape Loader-Monitor C program of the Mod 1 (TR) Operating System. (Loading from mass storage is significantly faster, especially when programs are frequently used.)

MOD 1 (MSR) OPERATING SYSTEM

When loading from mass storage, an Execute statement with the program and segment name *FILESUP should be submitted in the job control file and followed by the desired statements for File Support C. Mass storage device 0 must contain a resident program file (*DRS1RES) and the bootstrap records (cylinder 0, track 0).

FILENAME TEST FILE		PAGE
		1
ITEM NO	1	
DCW :EVEN: B	MICON	68
DCW :EVEN: B		
ITEM NO	2	
L DCW =1C01 B	DIAGR IF MATCHED	67
L DCW =1C01 B		
ITEM NO	3	
DCW :ODD: B	MTNMI	66
DCW :ODD: B		
ITEM NO	4	
DCW :PAR: B	LKVAL	65
DCW :PAR: B		
ITEM NO	5	
L DCW =1C02 C	DIAGR IF MATCHED	64
L DCW =1C02 C		
ITEM NO	6	
DCW :STAND: C	MICON	63
DCW :STAND: C		
ITEM NO	7	
L DCW =1C01 C	DIAGR IF MATCHED	62
L DCW =1C01 C		
ITEM NO	8	
DCW :SPEC: C	MICON	61
DCW :SPEC: C		
ITEM NO	9	
DCW :MODE: C	LKVAL	60
DCW :MODE: C		
ITEM NO	10	

Figure 4-5. Listing of Sample Unload-to-Printer Function

MOD 1 (TR) OPERATING SYSTEM

Loading from magnetic tape is the same as for mass storage except that a console call card with the program and segment name *FILESUP is used instead of an Execute statement.

Protection of Mass Storage During Execution of File Support C

Protection switches on the mass storage control must be set as shown in the following.

PROTECTION DURING ALLOCATE

Format write — PERMIT
Data write — PERMIT
B-file — PERMIT, if specified for the file being allocated

PROTECTION DURING DEALLOCATE

Data write -- PERMIT

PROTECTION DURING LOAD/UNLOAD

If the function includes an output file on mass storage:

Data write — PERMIT
A-file — PERMIT, if the file being loaded is a system file with A protection
B-file — PERMIT, if specified at allocation for the file being loaded

If the function includes no output files on mass storage, all protection can be set to PROTECT.

PROTECTION DURING MAP

All protection switches can be set to PROTECT.

Operator Control and Messages for File Support C

This section describes the messages to the operator and the responses he may make. This information comprises the operator control file.

OPERATOR CONTROL WITH CONTROL PANEL

When the operator control device is a control panel, information is conveyed to the operator through a Halt instruction. The B-address register indicates the meaning of the halt by means of a coded value. The A-address register usually contains the address of a response field. The user must enter a response character into this character location as indicated under

the specific halt. In the case of most halts, additional information is provided in memory locations immediately higher than the response character as described on page 4-69.

Peripheral Conditions

If the contents of the B-address register are in the range 0000 to 3777, the condition relates to a peripheral device. The general form for the B-address register value is ppxd,

- pp = peripheral control address;
 x = code indicating type of condition:
 0 = device not operable,
 1 = uncorrectable read error,
 2 = uncorrectable write error,
 3 = end of storage medium,
 4 = positioning or address error, or
 7 = miscellaneous condition; and
 d = device number.

The operator should determine the peripheral control unit involved and take appropriate action. If the control unit addressed is mass storage, the meanings of the B-address register values can be found in Table 3-15. If the control unit addressed is for some other device, the meanings of the B-address register values are listed in Table 4-7.

NOTE: If a peripheral condition occurs on mass storage during allocation, in conjunction with a cylinder and track message on the printer, see paragraph entitled "Failure During Allocation and Deallocation," in this section.

Table 4-7. Conditions Related to Non-Mass Storage Files

B-Address Register Value	Condition	Function	Operator Action (see note)
ppld (If pp = magnetic tape control unit, the A-address register contains the response location.)	Uncorrectable read error.	Load All	<u>Tape File</u> To attempt rereading, enter G into response location. Press RUN. To bypass record and go on to next record, enter A into response location. Press RUN. <u>Card File</u> Correct card in error if possible and refeed, starting with that card. Press RUN.

Table 4-7 (cont). Conditions Related to Non-Mass Storage Files

B-Address Register Value	Condition	Function	Operator Action (see note)
pp2d	Uncorrectable write error.	Unload and Map Unload Map and Unload	<u>Tape File</u> To attempt rewriting, enter G into response location. Press RUN. <u>Card File</u> Remove erroneously punched card and press RUN to repunch. <u>Printer</u> An erroneous line has been printed. To ignore error and go on to next line, press RUN.
pp3d	End of storage medium	Load/Unload and Map	<u>Tape File</u> When next reel is ready, enter G into the response location. Press RUN.

NOTE: A = 21₈, G = 27₈

File-Related Conditions

If the value of the B-address register is in the range 4000 to 4777, the condition is related to logical operations with mass storage files. See Section III for a description of these halts.

Job Control File Conditions

If the value of the B-address register is in the range 5000 to 5777, the condition is related to errors in the job control file statements. The A-address register contains the address of a list of information in the following format:

R	F	I	S
---	---	---	---

R is a response field.

F indicates the function being performed (response field plus 1);

- 01 = Deallocate
- 02 = Allocate
- 03 = Load/Unload, or
- 05 = Map.

I is an indicator (response field plus 2). This field is significant when a 5040 halt occurs. Its values and their meanings are listed in Table 4-9.

S Indicates the type of statement in which the error has been detected (response field plus 3). This field is always 00 at a 5040 halt.

- 00 = Statement is irrelevant,
- 01 = File statement,
- 02 = Volume statement,
- 03 = Units statement,
- 04 = Exits statement,
- 05 = Size statement,
- 06 = Member statement, or
- 07 = Day statement.

The operator decides what action to take and enters a character into the response location; the punctuation of this character is irrelevant. Each specific condition allows certain possible responses, as shown in Table 4-8. The general meanings of the various response characters are:

- G (27_g) = Attempt to perform the operation again. The operator corrects the erroneous statement (if possible), refeeds job control statements beginning with the Function statement for the function containing the statement in error, enters a G, and presses RUN. The program searches for the next Function statement in the job control file.
- F (26_g) = Go on to next function. If the operator cannot correct the erroneous statement, he may skip to the next function by entering an F and pressing RUN. The program searches for the next Function statement within the File Support C job control file.
- E (25_g) = Emergency exit to the Supervisor. If the entire file support run must be discontinued, the operator enters an E and presses RUN. The program exits to the Supervisor.

Table 4-8. Job Control Halt Codes

B-Address Register Value	Meaning	Possible Operator Responses
5000	Syntactic error.	G F E
5001	Invalid command field.	G F E
5002	Invalid positional parameter.	G F E
5003	Invalid keyword.	G F E
5004	Required parameter missing.	G F E
5005	Invalid keyword parameter value.	G F E

Table 4-8 (cont). Job Control Halt Codes

B-Address Register Value	Meaning	Possible Operator Responses
5010	Invalid combination or sequence of parameters.	G F E
5040	Same as 5010. See below.	F E
5077	Job control file too large for available memory.	E F
NOTE: G=27 ₈ , F=26 ₈ , and E=25 ₈		

When any of the above halts occur with the exception of the 5040 and 5077 halts, the erroneous card can be punched and the entire set of statements, starting with the Function statement in which it is located, can be entered by means of the card reader.

When a 5077 halt occurs, available memory has been exhausted for the storage of the parameters in the job control file. The job control file can be broken into smaller units and File Support C rerun, following an operator response of E. If a response of F is entered, the Function statement read just prior to the halt, and all subsequent Function statements, will be bypassed. They must be included in a later run of File Support C.

When a 5040 halt occurs, the four fields starting at the response character can be displayed to determine the nature of the error. At this halt, no corrective action is possible and the function in which the error has occurred must be bypassed.

The F-field (at response plus 1) indicates the File Support C function within whose parameters error has been detected.

The I-field (at response plus 2) contains a value which indicates the specific error condition that has been detected. The possible values of the I-field are listed in Table 4-9.

Table 4-9. File Support Diagnostics for 5040 Halt

I - Field		Meaning
Console (Alpha)	Control Panel (Octal)	Allocate Diagnostics (allocation is not performed)
A	21	Allocation has been requested for a file whose name starts with an illegal character.
B	22	No file organization was specified.

SECTION IV. FILE SUPPORT C

Table 4-9 (cont). File Support Diagnostics for 5040 Halt

I - Field		Meaning
Console (Alpha)	Control Panel (Octal)	Allocate Diagnostics (allocation is not performed)
C	23	<ol style="list-style-type: none"> 1. Key length was unspecified for a direct access or indexed sequential file. 2. Key position was unspecified for a direct access or indexed sequential file. 3. Specified key position and size are incompatible with item size.
E	25	The number of tracks in the cylinder overflow area is not less than the number of tracks requested per cylinder for a direct access or indexed sequential file.
F	26	Records-per-block is greater than records-per-cylinder.
G	27	The total number of blocks requested for members exceeds the total data blocks in the file.
H	30	<ol style="list-style-type: none"> 1. No Units statement appeared for this file. 2. The to cylinder in a unit of allocation is smaller than the from cylinder. 3. A cylinder has been specified in a Units statement which exceeds the allowable maximum (Type 258 Disk Pack Drive: cylinder 103; Types 155, 259, or 273 Disk Pack Drive: cylinder 202; Type 261 or Type 262 Disk File: cylinder 127). 4. The to track in a unit of allocation is smaller than the from track. 5. A track has been specified in a Units statement which exceeds the allowable maximum (Type 155 Disk Pack Drive: track 1; Type 258 or Type 259 Disk Pack Drive: track 9; Type 273 Disk Pack Drive: track 19; Type 261 or Type 262 Disk File: track 127). 6. Less than two cylinders for a unit of allocation for a direct access file have been specified and general overflow has been requested. 7. The number of tracks per cylinder for each data unit of allocation is not the same. 8. The second of two adjacent units of allocation on the same volume begins on the same cylinder on which the first one ended.
I	31	A volume name was not specified as the first parameter of this Units statement.
.	33	<p>In requesting the allocation of the *BADTRACKS file, one of the following events has occurred.</p> <ol style="list-style-type: none"> 1. File organization was not specified as sequential. 2. A parameter of the Size statement was specified. 3. A password was specified. 4. More than one unit of allocation was specified. 5. More than one Units statement was specified.
%	35	Indexed sequential file: number of records per block exceeds records per track.

SECTION IV. FILE SUPPORT C

Table 4-9 (cont). File Support Diagnostics for 5040 Halt

I - Field		Meaning
Console (Alpha)	Control Panel (Octal)	Allocate Diagnostics (allocation is not performed)
■	36	A requested unit of allocation includes either the bootstrap track or the volume label track.
?	37	<ol style="list-style-type: none"> 1. The requested member index is too small (contains less than 75 characters). 2. The requested member index is too small to contain the names of all the requested members.
J	41	The number of records per bucket or string exceeds the number of records in the data area of a cylinder.
L	43	Either name or length was omitted for a member in a partitioned file.
M	44	Indexed sequential file - <ol style="list-style-type: none"> 1. String index item length is greater than a block size. 2. MCINDEX unit of allocation area is too small for the file's data area size. 3. Block length exceeds one track.
N	45	The MCINDEX and/or OVERFLOW units of allocation were specified in the wrong sequence or were omitted.
O	46	The requested track width per cylinder of an indexed sequential file data unit is not large enough to contain a string plus a string index block.
Deallocate Diagnostics (deallocation is not performed)		
R	51	No file name was specified.
#	52	No volume name was specified.
Load/Unload Diagnostics (load/unload is not performed)		
/	61	No file name was specified for a mass storage file.
S	62	No mass storage file was specified. (Both input and output media are non-mass-storage).
T	63	The input file, the output file, or both files were not specified.
U	64	Either the program segment name or the low memory address was not specified in an Exits statement.
V	65	A memory location greater than 32,767 was specified for the low memory address in an Exits statement.
W	66	The *VOLSPARES file has been specified as a file to be loaded or unloaded.
General Diagnostics		
C _R	75	Insufficient memory for this call of the function indicated in the F character. (A-address plus one.)

SECTION IV. FILE SUPPORT C

Conditions Specific To File Support C

If the B-address register contains a value in the range 5400 to 5477, the halt condition is specific to File Support C processing. The A-address register is the address of a list of information which is presented in the format shown in the following.

The error code and all succeeding information is typed out on the console as a supplementary list if the console typewriter is being used.

Number of Characters	Character-field Location (left)	Explanation
1	A	Response character.
1	A + 1	Error code (applied to mass storage peripheral device or file condition).
10	A + 2	Mass storage file name.
1	A + 12	Relative volume number of the mass storage file.
6	A + 13	Volume name.
1	A + 19	Mass storage peripheral control unit address.
1	A + 20	Mass storage device address.
1	A + 21	Mass storage pack number.
6	A + 22	Mass storage address in binary (CCTRR), for device error only.

If a mass storage partitioned sequential file is being loaded or unloaded, the member name is placed in a 14-character field to the right of the preceding list of information (A+28). If a mass storage print-image file with report numbers is unloaded, the report number appears in a 2-character field to the right of the above list (A+28).

The operator decides what action to take and enters a character into the response location. Each specific halt allows certain responses, as indicated in Table 4-10. The general meanings of the various response characters are:

- A(21₈) Accept the last operation as correct and continue,
- G(27₈) Attempt to perform the operation again,
- S(62₈) Skip to next logically permissible operation,
- F(26₈) Go to next function, and
- E(25₈) Emergency exit to the Supervisor.

SECTION IV. FILE SUPPORT C

Table 4-10. File Support C Halts

B-Address Register Value	Function	Condition	Operator Action (see note 1)
5400	any	Volume name check failed.	Enter G, F, or E.
5401	A, D	Volume name check failed on second or subsequent volume of the file (see note 2).	Enter G or E.
5402	A	Directory has no space for new file (see note 3).	Enter F or E.
5403	D L/U	Mass storage file name not found (see notes 4 and 5).	Enter G, F, or E. For deallocate, can also enter A to accept condition and continue to next volume of file; enter S to go to next file.
5404	L/U	Mass storage file name not specified.	Enter F or E.
5405	A	Duplicate file name (see note 3).	Enter S, F, or E.
5406	L/U	File name check failed on card or tape input or tape output.	Enter A to accept the file (tape only), G, F, or E.
5407	D	Error detected in units of allocation portion of volume directory, due to incomplete deallocation on a prior run (see note 5).	Enter A or E.
5410	D L/U	Wrong password supplied.	Enter G, F, or E. For deallocate, can also enter S to skip file.
5411	D L/U	No password was supplied, but file has a password.	Enter S (for deallocate), F, or E.
5412	D	Expiration date check failed.	Enter S, F, or E.
5413	A	Error in units of allocation. There is a conflicting unit of allocation already on the mass storage volume (see notes 3 and 6).	Enter F or E.
5414	D L/U	Volume sequence check failed (volume mounted out of sequence).	Enter G or E.

SECTION IV. FILE SUPPORT C

Table 4-10 (cont). File Support C Halts

B-Address Register Value	Function	Condition	Operator Action (see note 1)
5415	A	An illegal combination of device classes has been detected while allocating a multivolume file (see note 3).	Enter G or E.
5416	A	During allocation of a multivolume file, a request for a unit of allocation in which a cylinder or track exceeds the allowable maximum for the device being addressed has been encountered (see note 3).	Enter G or E.
5417	L/U	Reel sequence check failed on input tape (reel mounted out of sequence).	A = accept the current reel; G = re-open the tape reel; F = go to next function; E = go to Supervisor.
5420	L/U	Member name not found in member index of input mass storage file.	Enter S to skip member; go to next member; F to close file; go to next function; E to close file; go to Supervisor.
5421	L/U	Member unavailable for output processing.	See halt 5420.
5422	L/U	No space in member index for new member.	See halt 5420.
5423	L/U	No member name in input file header (non-mass storage).	Enter G to re-open file; F to close file; go to next function; or E to close file; go to Supervisor.
5424	L/U	No data space to create new member.	See halt 5420.
5425	L/U	Member names of output mass storage file exhausted before those of input mass storage file.	Enter F to close file; go to next function; or E to close file; go to Supervisor.

SECTION IV. FILE SUPPORT C

Table 4-10 (cont). File Support C Halts

B-Address Register Value	Function	Condition	Operator Action (see note 1)
5426	L/U	No member names in member index of input mass storage file.	See halt 5425.
5427	M	An item with invalid contents has been detected in the volume directory.	Enter S, F, or E.
5430	L/U	No data space to add next item to output mass storage file.	Enter A to close member and/or file; go to next function; S to close member; go to next member; or E to close the file and/or member; go to Supervisor.
5431	L/U	No more input members, but not all specified output members have been processed.	Enter A to close file; go to next function; G to re-open input file; or E to close file; go to Supervisor.
5432	L/U	End of file between two cards of an input item.	Enter A to close member and/or file; go to next function. S to skip member; go to next member. E to close member and/or file; go to Supervisor.
5433	L/U	File header is missing from input card or tape file or output tape file.	Enter A to accept the file (tape output only), G, F, or E.
5440	L/U	Invalid device type specified (see list on pages 4-37 and 4-38).	Enter F or E.
5441	L/U	Invalid mass storage file organization.	Enter F or E.
5442	L/U	Invalid combination of file organizations specified for mass storage to mass storage operation.	Enter F or E.
5443	L/U	No own-coding program name specified when a direct access file is to be loaded.	Enter F or E.

SECTION IV. FILE SUPPORT C

Table 4-10 (cont). File Support C Halts

B-Address Register Value	Function	Condition	Operator Action (see note 1)
5444	L/U	An attempt is being made to load data into the *BADTRACKS file.	Enter A to continue function and to load the file. Enter F to skip to next function. Enter E.
5445	L/U	Specified or assumed tape item size is greater than tape record size.	Enter F or E.
5450	L/U	Indexed sequential load-input data item key not higher than prior key. X1 = lefthand end of item.	Enter S to skip out-of-sequence item and continue processing. F to close file; go to next function; or E to close file; go to Supervisor.
5451	L/U	Insufficient device address parameters have been supplied for a multivolume file (see DEVADD, Table 4-4).	Enter F or E.
5452	L/U	Indexed sequential load-requested number of imbedded overflow items per string is not less than number of data items per string.	Enter A (file will be loaded with one active item per data string), F, or E.
5453	D	An attempt is being made to deallocate the *BADTRACKS file.	Enter A, S, F, or E. Enter A to continue function and to deallocate the file. Enter S, F, or E.
5454	A	An attempt is being made to allocate the *BADTRACKS file, but no *VOLSPARES file exists on the volume.	Enter F or E.
5455	A	A bad track has been detected in the data area of a file being allocated, and the structure of the file does not permit it to contain bad tracks.	Enter E.
5456	A	There are no unused substitute track addresses in the *BADTRACKS file. A substitute cannot be established for the latest bad track detected or declared. (see note 2).	Enter E.

SECTION IV. FILE SUPPORT C

Table 4-10 (cont). File Support C Halts

B-Address Register Value	Function	Condition	Operator Action (see note 1)
5465	L/U	(Unload to printer.) Printer form may require adjustment.	Enter A to print the next item from the mass storage file, G to print the same item and reexecute this halt, F, or E.
5466	L/U	Unload to printer- print-image items in a partitioned file being unloaded to printer contain a report number.	Enter A to ignore all report numbers and print each member as a separate report, F, or E.
5467	L/U	Unload to printer- report number is not found in a sequential print-image mass storage file.	Enter A or S to go on to next report number, F, or E.
5475	A, L/U	Allocate - insufficient memory to declare members. Load/Unload: Insufficient memory to build table of all input member names.	Enter A to allocate file without members, F or E. Enter A to unload those members in the existing table, F or E.
5476	Any	Insufficient memory for buffers.	Enter F or E.

NOTES:

- Operator response codes are interpreted as shown below, unless otherwise indicated.

<u>Enter</u>	<u>Explanation</u>
A (21g)	Accept the condition or file. Continue.
G (27g)	Reopen volume or file.
S (62g)	Skip file; go to next file.
F (26g)	Skip function; go to next function.
E (25g)	Skip function; go to Supervisor.

- This halt may occur during allocation or deallocation. If the allocation or deallocation is not completed, see "Failure During Allocation and Deallocation," below.
- These halts may occur during allocation. If the allocation is not completed, the file must be deallocated. See "Failure During Allocation and Deallocation," below.
- This halt may occur during deallocation. If the deallocation is not completed, the file must be deallocated promptly. See "Failure During Allocation and Deallocation," below.
- These halts may occur during a deallocation that is correcting a prior allocation or deallocation that was incomplete. See "Failure During Allocation and Deallocation," below.
- This halt may occur if allocation of a file was not completed and the required deallocation was not done or was not completed. (It may also occur due to an error in specifying the units of allocation.) See "Failure During Allocation and Deallocation," below.

OPERATOR CONTROL WITH CONSOLE TYPEWRITER

When a console typewriter message indicates an error or requests operator action, the operator performs the following steps.

1. Read the typeout. (To repeat the message, press the space bar twice.)
If necessary, consult the manual for possible action.
2. Perform the desired corrective action.
3. Type the appropriate 1-character response (G, E, etc.).
4. If the typein is correct, press the space bar to continue. If incorrect, type any other character and return to step 3.

Peripheral Conditions

When a peripheral condition causes a console typeout, a 1- or 2-line message is given. With all files, the first line below is given.

pp d description

pp d gives the peripheral control unit (pp) and device number (d) of the peripheral device containing the file in error.

description is a message describing the condition. (See Table 4-11).

With mass storage files, the second line is given which identifies the disk device containing the error.

The operator should determine the peripheral control unit involved and take appropriate action. If the control unit is mass storage, the possible description messages in line 1 and the contents of the second line are described in "Console Typewriter Operating Procedures" in Section III. For control units for some other device, the message, the error condition, the function which may issue the message, and any possible operator actions are given in Table 4-11.

Table 4-11. Typewriter Messages for Conditions Related to Non-Mass Storage Files

Message	Condition	Function	Operator Action
READ ERROR	Uncorrectable read error.	Load	<u>Tape File</u> To attempt rereading, type G. To bypass the record and go on to the next, type A.
		All	<u>Card File</u> Correct card in error if possible and refeed, starting with that card, Type G.
WRITE ERROR	Uncorrectable write error.	Unload and Map	<u>Tape File</u> To attempt rewriting, type G.
		Unload	<u>Card File</u> To repunch, remove the erroneously punched card, and type G.

Table 4-11 (cont). Typewriter Messages for Conditions Related to Non-Mass Storage Files

Message	Condition	Function	Operator Action
WRITE ERROR	Uncorrectable write error (cont).	Map and Unload	<u>Printer</u> An erroneous line has been printed. To ignore the error and go to the next line, type G.
END VOLUME	End-of-storage medium.	Load/Unload and Map	<u>Tape File</u> When next reel ready, type G.

File-Related Conditions

When using File Support C, messages in the format

```
pp d FILE filename description
c filename v volume p d m a
```

may be given. These are related to logical operations with files and are discussed in Section III.

Job Control File Conditions

All console typewriter messages that deal with the job control file begin with the words

JOB CONTROL FILE ERROR,

Certain messages, as indicated in Table 4-12, are preceded by the words

CANNOT REFEED,

Following these words, there is a general message to describe the condition and three fields of characters that further isolate the condition and indicate the function being performed. These fields have the format: F I S

where F indicates the function being performed:

- 1 = Deallocate
- 2 = Allocate
- 3 = Load/Unload
- 5 = Map

I is an indicator. This is significant when a PARAMETER COMBINATION message is given, preceded by the indicator CANNOT REFEED. The possible values I may have and their meanings are given in Table 4-9.

S indicates the type of statement in which an error has been detected. This field has the value 0 if I is greater than 20 (octal). The possible values of the S field are:

- 0 = Statement is irrelevant,
- 1 = File statement,
- 2 = Volume statement,
- 3 = Units statement,
- 4 = Exits statement,
- 5 = Size statement,
- 6 = Member statement, or
- 7 = Day statement.

SECTION IV. FILE SUPPORT C

The operator decides what action to take and types the appropriate response character. Each condition allows certain responses as shown in Table 4-12. The general meanings of the various response characters are as follows.

- G Attempt to perform the operation again. The operator corrects the erroneous statement (if possible), refeeds job control statements beginning with the Function statement for the function containing the statement in error, and types G. The program searches for the next Function statement in the job control file.
- F Go on to next function. If the operator cannot correct an erroneous statement, he may skip to the next function by typing F. The program searches for the next Function statement within the File Support C job control file.
- E Emergency exit to the Supervisor. If the entire file support run must be discontinued, the operator types an E. The program exits to the Supervisor's emergency return address.

Table 4-12. Job Control File Console Typewriter Messages

Message	Meaning	Possible Operator Responses
SYNTAX	Syntactic error.	G F E
COMMAND FIELD	Invalid command field.	G F E
POSITIONAL PARAMETER	Invalid positional parameter.	G F E
KEYWORD PARAMETER	Invalid keyword.	G F E
MISSING PARAMETER	Required parameter missing.	G F E
PARAMETER VALUE	Invalid keyword parameter value.	G F E
PARAMETER COMBINATION	Invalid combination or sequence of parameters.	G F E
CANNOT REFEED, PARAMETER COMBINATION	Invalid combination or sequence of parameters (used in conjunction with I-field).	F E
CANNOT REFEED, JOB CONTROL FILE TOO LONG	Job control file too large for available memory.	E F

SECTION IV. FILE SUPPORT C

When any of the messages indicated above appear, with the exception of those preceded by the message CANNOT REFEED, the erroneous card can be repunched and the entire set of statements, starting with the Function statement in which it is located, can be entered by means of the card reader.

When a PARAMETER COMBINATION message occurs, preceded by CANNOT REFEED, the value of the I-field following it should be checked. The possible values this field may have are given in Table 4-9.

When the message JOB CONTROL FILE TOO LONG is given, available memory has been used for storage of the parameters in the job control file. The job control file should be broken into smaller units and File Support C rerun.

Typewriter Messages Specific to File Support C

The messages given in Table 4-13 pertain to error conditions that are specific to File Support C processing. The operator decides what action to take and types the appropriate response character. Each error allows certain responses, as indicated in Table 4-13. The general meanings of the various response characters are:

- A Accept the last operation as correct and continue,
- G Attempt to perform the operation again,
- S Skip to next logically permissible operation,
- F Go to next function, and
- E Emergency exit to the Supervisor.

Table 4-13. Typewriter Messages Specific to File Support C

Message	Function	Condition	Operator Action
FIRST VOLUME NAME WRONG	Any	Volume name check failed.	G = Reopen volume. F = Skip function; go to next function. E = Skip function; go to Supervisor.
SUBSEQUENT VOLUME NAME WRONG	A, D	Volume name check failed on second or subsequent volume of the file. (See note 5.)	G = Reopen the volume. E = Skip function, go to Supervisor.
VOLUME DIRECTORY FULL	A	Directory has no space for new file. (See note 1.)	F = Do not allocate; go to next function. E = Do not allocate; go to Supervisor.

SECTION IV. FILE SUPPORT C

Table 4-13 (cont). Typewriter Messages Specific to File Support C

Message	Function	Condition	Operator Action
FILE NOT FOUND	D	Mass storage file name not found. (See notes 2 and 3.)	A = Accept the condition. Continue to the next volume of the file. (Deallocate) G = Reopen volume. S = Skip file; go to next file. (Deallocate) F = Discontinue function; go to next function. E = Discontinue function; go to Supervisor.
FILENAME UNSPECIFIED	L/U	Mass storage name not specified.	F = Discontinue function; go to next function. E = Discontinue function; go to Supervisor.
DUPLICATE FILENAME	A	Duplicate file name. (See note 1.)	S = Skip file; go to next file. F = Discontinue function; go to next function. E = Discontinue function; go to Supervisor.
FILENAME CHECK FAILED	L/U	File-name check failed on card or tape output.	A = Accept the file. (Tape only.) G = Reopen card or tape file. F = Discontinue function; go to next function. E = Discontinue function; go to Supervisor.
ERROR IN *VOLALLOC*	D	Error detected in units of allocation portion of volume directory due to incomplete deallocation on a prior run. (See note 3.)	A = Accept condition; continue deallocation. E = Discontinue deallocation; go to Supervisor.
WRONG PASSWORD	D L/U	Wrong password supplied.	G = Reopen volume. S = Skip file; go to next file. (Deallocate) F = Discontinue function; go to next function. E = Discontinue function; go to Supervisor.
MISSING PASSWORD	D L/U	No password was supplied; file has password.	S = Skip file; go to next file. (Deallocate) F = Discontinue function; go to next function. E = Discontinue function; go to Supervisor.

SECTION IV. FILE SUPPORT C

Table 4-13 (cont). Typewriter Messages Specific to File Support C

Message	Function	Condition	Operator Action
EXPIRATION DATE ERROR	D	Expiration date check failed.	S = Skip file; go to next file. F = Discontinue function; go to next function. E = Discontinue function; go to Supervisor.
CONFLICTING UNITS OF ALLOCATION	A	Error in units of allocation. There is a conflicting unit of allocation already on the mass storage volume. (See notes 1 and 4.)	S = Skip file; go to next file. F = Discontinue function; go to next function. E = Discontinue function; go to Supervisor.
VOLUME SEQUENCE NUMBER ERROR	D L/U	Volume sequence check failed (volume mounted out of sequence).	G = Reopen volume. E = Discontinue function; go to Supervisor. F = Discontinue function; go to next function.
ILLEGAL DEVICE COMBINATION	A	An illegal combination of device classes has been detected while allocating a multivolume file. (See note 3.)	G = Attempt to continue the function on the correct volume. E = Exit to Supervisor.
MAXIMUM CYLINDER OR TRACK ILLEGAL	A	During allocation of a multivolume file, a request for a unit of allocation in which the cylinder or track exceeds the allowable maximum for the device being addressed has been encountered. (See note 3.)	G = Attempt to continue the function on the correct volume. E = Exit to Supervisor.
REEL SEQUENCE CHECK FAILED	L/U	Input tape reel mounted out of sequence.	A = Accept the current reel. G = Mount the correct reel and retry. F = Go to next function. E = Go to Supervisor.
MEMBER NOT FOUND	L/U	Member name not found in member index of input mass storage file.	S = Skip member; go to next member. F = Close file; go to next function. E = Close file; go to Supervisor.

SECTION IV. FILE SUPPORT C

Table 4-13 (cont). Typewriter Messages Specific to File Support C

Message	Function	Condition	Operator Action
MEMBER CANNOT BE OUTPUT ONLY	L/U	Member unavailable for output processing.	S = Skip member; go to next member. F = Close file; go to next function. E = Close file; go to Supervisor.
MEMBER INDEX FULL	L/U	No space in member index for new member.	S = Skip member; go to next member. F = Close file; go to next function. E = Close file; go to Supervisor.
MEMBER NAME MISSING	L/U	No member name in input file header (non-mass-storage).	G = Reopen file. F = Close file; go to next function. E = Close file; go to Supervisor.
NO SPACE FOR NEW MEMBER	L/U	No data space to create new member.	S = Skip member; go to next member. F = Close file; go to next function. E = Close file; go to Supervisor.
UNEQUAL NUMBER OF MEMBERS	L/U	Member names of output mass storage file exhausted before those of the input mass storage file.	F = Close file; go to next function. E = Close file; go to Supervisor.
NO MEMBERS IN INPUT FILE	L/U	No member names in member index of input mass storage file.	F = Close file; go to next function. E = Close file; go to Supervisor.
ERROR IN VOLUME DIRECTORY ITEM	M	An item with invalid contents has been detected in the volume directory.	Enter S, F, or E.
NO MORE SPACE IN OUTPUT FILE	L/U	No data space to add next item to output mass storage file.	A = Close member and/or file; go to next function. S = Close member; go to next member. E = Close the file and/or member; go to Supervisor.

SECTION IV. FILE SUPPORT C

Table 4-13 (cont). Typewriter Messages Specific to File Support C

Message	Function	Condition	Operator Action
TOO FEW INPUT MEMBERS	L/U	No more input members, but not all specified output members have been processed.	A = Close file; go to next function. G = Reopen input file. E = Close file; go to Supervisor.
ERRONEOUS END OF CARDS	L/U	End of file between two cards of an input item.	A = Close member and/or file; go to next function. S = Skip member; go to next member. E = Close member and/or file; go to Supervisor.
FILE HEADER MISSING	L/U	File header is missing from input card or tape file or output tape file.	A = Accept the file (tape output only). G = Reopen file. F = Close file; go to next function. E = Close file; go to Supervisor.
INVALID DEVICE TYPE	L/U	Invalid device type specified.	F = Discontinue function; go to next function. E = Discontinue function; go to Supervisor.
INVALID FILE ORGANIZATION	L/U	Invalid mass storage file organization.	F = Discontinue function; go to next function. E = Discontinue function; go to Supervisor.
CONFLICTING FILE ORGANIZATIONS	L/U	Invalid combination of file organizations specified for mass storage to mass storage operation.	F = Discontinue function; go to next function. E = Discontinue function; go to Supervisor.
NO OWN-CODE PROGRAM	L/U	No own-coding program name specified when a direct access file is to be loaded.	F = Discontinue function; go to next function. E = Discontinue function; go to Supervisor.
WILL LOAD INTO *BADTRACKS FILE	L/U	An attempt is being made to load data into *BADTRACKS file.	A = Accept condition; continue function and load. F = Discontinue function; go to next function. E = Discontinue function; go to Supervisor.
INVALID ITEM SIZE	L/U	Specified or assumed tape-item size is greater than tape-record size.	F = Discontinue function; go to next function. E = Discontinue function; go to Supervisor.

SECTION IV. FILE SUPPORT C

Table 4-13 (cont). Typewriter Messages Specific to File Support C

Message	Function	Condition	Operator Action
KEY OUT OF SEQUENCE	L/U	Indexed sequential load-input data item key not higher than prior key.	S = Skip out-of-sequence item; process next item. F = Close file; go to next function. E = Close file; go to Supervisor.
NOT ENOUGH DEVICES	L/U	Insufficient device-address parameters have been specified for a multivolume file (see Table 4-4).	F = Discontinue function; go to next function. E = Discontinue function; go to Supervisor.
TOO MANY IM-BEDDED ITEMS	L/U	The value of the imbedded parameter is not less than the number of items per string.	A = Continue; file is loaded with one active item per string. F = Discontinue function; go to next function. E = Discontinue function; go to Supervisor.
WILL DEALLOCATE *BADTRACKS FILE	D	An attempt is being made to deallocate *BADTRACKS file.	A = Accept condition; continue function and deallocate. S = Skip file; go to next file. (Deallocate) F = Discontinue function; go to next function. E = Discontinue function; go to Supervisor.
VOLUME DOES NOT CONTAIN *VOLSPARES FILE	A	An attempt is being made to allocate *BADTRACKS file, but no *VOLSPARES file exists on the volume.	F = Discontinue function; go to next function. E = Discontinue function; go to Supervisor.
FILE CANNOT CONTAIN BAD TRACKS	A	A bad track is detected in data area of a file being allocated; file structure does not permit it to contain badtracks.	E = Discontinue function; go to Supervisor.
NO MORE SPACE IN *BADTRACKS FILE	A	There are no more available tracks in the *BADTRACKS file. (See note 2.)	E = Discontinue function; go to Supervisor.
PRINTER FORM ADJUSTMENT	L/U	(Unload to printer.) Printer form may require adjustment.	A = Print the next item from the mass storage file. G = Print the same item again and reexecute this halt. F = Discontinue function; go to next function. E = Discontinue function; go to Supervisor.

SECTION IV. FILE SUPPORT C

Table 4-13 (cont). Typewriter Messages Specific to File Support C

Message	Function	Condition	Operator Action
REPORT NUMBERS PRESENT	L/U	(Unload to printer.) Print-image items in a partitioned sequential file that is being unloaded to the printer contain a report number.	A = Ignore all report numbers; print each member as a separate report. F = Discontinue function; go to next function. E = Discontinue function; go to Supervisor.
REPORT NUMBER NOT FOUND	L/U	(Unload to printer.) Report number not found in sequential print-image mass storage file.	A) = Accept; go on to next S) report number. F = Discontinue function; go to next function. E = Discontinue function; go to Supervisor.
INSUFFICIENT MEMORY FOR MEMBERS	A L/U	Allocate: Insufficient memory to declare members. Load/Unload: Insufficient memory to build table of all input member names.	A = Allocate file without members (Allocate). Unload those members in the existing table (Load/Unload). S = Skip file; go to next file. (Allocate) F = Discontinue function; go to next function. E = Discontinue function; go to Supervisor.
INSUFFICIENT MEMORY FOR BUFFERS	A L/U	Insufficient memory for buffers.	F = Discontinue function; go to next function. E = Discontinue function; go to Supervisor.
<p>NOTES: 1. These messages may occur during allocation. If the allocation is not completed, the file must be deallocated. Refer to the following paragraphs.</p> <p>2. This message may occur during deallocation. If the deallocation is not completed, the file must be deallocated promptly. Refer to the following paragraphs.</p> <p>3. These messages may occur during a deallocation that is correcting a prior allocation or deallocation that was incomplete. Refer to the following paragraphs.</p> <p>4. This message may occur if allocation of a file was not completed and the required deallocation was not done or was not completed. (It may also occur due to an error in specifying the units of allocation.) Refer to the following paragraphs.</p> <p>5. This message may occur during allocation or deallocation. If the allocation or deallocation is not completed, refer to the following paragraphs.</p>			

FAILURE DURING ALLOCATION AND DEALLOCATION

This paragraph outlines the procedures to be used if there is a failure during allocation or deallocation. These failures may be due to:

1. Errors, such as attempting to allocate to areas already assigned to another file, or incorrect mounting of volumes, or
2. Equipment malfunctions, evidenced as device errors.

If allocation encounters a track which cannot be formatted successfully, a response to the halt or console message will continue the allocation so that additional cylinder and track messages (if any) will be produced on the printer. Permissible responses are as follows:

- A = continue allocation
- G = reattempt to format the track
- E = exit to Supervisor.

The procedures described here should be followed so that additional problems will not arise at a later time when they will be more difficult to analyze. Reference should be made to Table 4-10 or 4-13 for the appropriate operator action.

Failure During Allocation

If an allocation fails, perform the following operations.

1. Do not use the volumes on which the file was to be allocated until that file is deallocated.
2. Deallocate the file. Use the volume name check option. Mount the volumes for the file in the proper order, starting with the first volume of the file.
3. The deallocation will proceed normally unless a halt 5403 or message FILE NOT FOUND occurs. Note the volume name and file name for future reference. Skip the file and go to the next operation.
4. The deallocation may have failed to remove from the volume noted in step 3 all of the units of allocation assigned to the file. Failure to perform the following steps may lead to a 5413 halt or message (conflicting units of allocation) at a later allocation operation of this volume.
5. Map the volume. Both the descriptions of all files on the volume (MAP, DESCR) and the listing of unassigned tracks (MAP, UNUSED) are needed.
6. Compare the two listings to determine the tracks that are not assigned to any of the files listed. The listings of unassigned tracks will show as "used" any tracks still belonging to the file. If there are no such tracks, the volume is completely usable and step 7 is omitted.
7. This step consists of one of the following:
 - a. Accept the unavailability of the tracks noted in step 6.
 - b. If a recent backup of the volume exists, it may be suitable for restoration by use of the Utility program Disk/Tape Copy. Update activity performed after the date of the backup must be repeated to make the restored volume current.

- c. If neither of the above two possibilities seems desirable, unload all the files on the volume, perform volume preparation, reallocate all the files, and reload all the files.

Failure During Deallocation

If a deallocation fails, perform the following operations.

1. Do not use the volumes from which the file was to be deallocated until the deallocation of that file is completed. It is absolutely essential that the deallocation be completed before any other file is allocated on the volume which was being processed when the deallocation failed. Otherwise, two or more files may attempt to use the same units of allocation.
2. Repeat the deallocation run. Use the volume name check option. Mount the volumes for the file in the proper order, starting with the first volume of the file.
3. The deallocation may come to halt 5403 or message FILE NOT FOUND for the first several volumes of the file. These are the volumes from which the file was completely or partly deallocated in the prior run. Enter A (accept the condition). The deallocation proceeds to the next volume of the file.
4. The deallocation may come to halt 5407 or message ERROR IN *VOLALLOC* portion of the volume directory. The volume name and file should be noted for future reference. Accept the condition and continue deallocation.
5. If the 5407 halt was encountered or message ERROR IN *VOLLALOC* (see step 4), the deallocation may have failed to remove from the volume noted in step 4 all of the units of allocation assigned to the file. Failure to perform the following steps may lead to a 5413 halt or message (CONFLICTING UNITS OF ALLOCATION) at a later allocation operation on this volume.
6. Map the volume. Both the descriptions of all files on the volume (MAP, DESCR) and the listing of unassigned tracks (MAP, UNUSED) are needed.
7. Compare the two listings to determine the tracks that are not assigned to any of the files listed. The listing of unassigned tracks will show as "used" any tracks still belonging to the file. If there are no such tracks, the volume is completely usable and step 8 is omitted.
8. This step consists of one of the following:
 - a. Accept the unavailability of the tracks noted in step 7.
 - b. If a recent backup of the volume exists, it may be suitable for restoration by use of the Utility program, Disk/Tape Copy. Update activity performed after the date of the backup must be repeated to make the restored volume current.
 - c. If neither of the above two possibilities seems desirable, unload all files on the volume, perform volume preparation, reallocate all the files, and reload all the files.

APPENDIX A
VOLUME LABEL AND VOLUME DIRECTORY

Both the volume label and volume directory are created by Volume Preparation C. Table A-1 describes the volume label, and Table A-2 describes the volume directory.

The volume label is the unique identification of the volume. This record is 250 characters long and is recorded as the first record (record 00) on the second track (cylinder 00, track 01) of each volume.

The volume directory is a list of all files that are stored on the volume. The directory begins on the third track (cylinder 00, track 02, record 00) of each volume except for the Type 155 Disk Pack Drive, where it begins on cylinder 00, track 01, record 1. Three sequential files make up the volume directory:

1. File name index (*VOLNAMES*),
2. File description index (*VOLDESCR*), and
3. File allocation index (*VOLALLOC*).

The first file (*VOLNAMES*) is an index of file names and refers to the other two files for additional information. This index contains the names of all files allocated on this volume and the addresses of the associated entries in the file description index and the file allocation index. The item size of the file name index is 30 characters. This file begins on cylinder 00, track 02, record 00 and never exceeds one track except for the Type 155 Disk Pack Drive, where it occupies records 1 through 14 in cylinder 00, track 01. It can accommodate up to 26 file names for a Type 155 Disk Pack Drive, up to 86 names for a Type 258, 259, or 273 Disk Pack Drive, and up to 158 file names for a Type 261 or Type 262 Disk File.

The second file (*VOLDESCR*) is a complete description of each file, including general information, labeling information, and information pertinent to the particular organization and structure of the file. The item size of the file description index is 100 characters. Except for the Type 155 Disk Pack Drive, this file begins on cylinder 00, track 03, and record 00; its length may be one, two, or three tracks, depending on the maximum-number-of-files parameter specified to the Volume Preparation C program. For the Type 155 Disk Pack Drive it begins on cylinder 01, track 00, record 00 and has a maximum length of one track.

APPENDIX A. VOLUME LABEL AND VOLUME DIRECTORY

Table A-1. Volume Label

Field	Position	Name and Length	Description
1	1-5	ID (five characters)	IVOLA
2	6-11	Volume name (six characters)	The unique name assigned to the volume.
3	12	Device type (one character)	11 (octal) = Type 258 12 (octal) = Type 259, 259A, 259B 13 (octal) = Type 273 21 (octal) = Type 155 31 (octal) = Type 261 32 (octal) = Type 262 33 (octal) = Type 261L 34 (octal) = Type 262L
4	13-18	Volume serial number (six characters)	Permanently assigned identification of the physical volume (volume name is its logical identification).
5	19	Operating system flag (one character)	00(octal) = Mod 1 (MSR) 02(octal) = Mod 2 04(octal) = Mod 4 10(octal) = Mod 8
6	20	Status of *VOLNAMES* address field (one character)	B, A bits indicate status as follows: 00 = not present; *VOLNAMES* begins on C0T2. 01 = reserved for future use. 10 = present; *VOLNAMES* begins on CCTT specified in positions 24 through 27. 11 = see 00.
7	21-23	Reserved (three characters)	Reserved for future use.
8	24-25	Cylinder where *VOLNAMES* begins (two characters)	Specifies the cylinder on which *VOLNAMES* file begins.
9	26-27	Track where *VOLNAMES* begins (two characters)	Specifies the track on which *VOLNAMES* file begins.
10	28-31	Reserved (four characters)	Reserved for future use.
11	32-250	Reserved (219 characters)	Reserved for future use.

The third file (*VOLALLOC*) is a list of the mass storage areas allocated to each file stored on the volume. Each unit of allocation is one item. The item size of the file allocation index is 20 characters. Except for the Type 155 Disk Pack Drive, this file begins on cylinder 00, track 04, 05, or 06, depending on the length of the file description index. For the Type 155 Disk Pack Drive it begins on cylinder 01, track 01. The file allocation index is the same number of tracks in length as the file description index.

Table A-2. Volume Directory

FILE NAME INDEX (*VOLNAMES*) ITEM FORMAT			
Field	Position	Name and Length	Description
1	1-10	FILE NAME (ten characters)	The unique name assigned to the file. Pos. 1: 77g = unused.
2	11	VOLUME SEQUENCE NUMBER (one character)	The relative number of this volume in the file, in binary. This field is zero for the first volume in the file.
3	12	SYSTEM OF ALLOCA- TION (one character)	An indication of which operating system allo- cated this file. 00 = Mod 1 04 = Mod 4 02 = Mod 2 10 = Mod 8
4	13-14	RESERVED (two characters)	Reserved for future use.
5	15-22	FILE DESCRIPTION AD- DRESS (eight characters)	Address (in the binary format CCTTRRII) of the entry in the file description index describing the file named in "1" above.
6	23-30	ALLOCATION AD- DRESS (eight characters)	Address (in the binary format CCTTRRII) of the first entry in the file allocation index for the file named in "1" above.
FILE DESCRIPTION INDEX (*VOLDESCR*)			
1	1	FILE ORGANIZATION (one character)	01 = Sequential 02 = Direct access 03 = Indexed sequential 11 = Partitioned sequential 70 = Nonstandard; not processed by Mod 1 (MSR) 77 = Unused item of *VOLDESCR*
2	2-3	ITEM SIZE (two characters)	Number of characters in item, in binary.
3	4-5	RECORD SIZE (two characters)	Number of characters in record, in binary.
4	6-7	BLOCKING FACTOR (two characters)	Number of items per block, in binary.
5	8-9	RECORDS PER BLOCK (two characters)	Number of records per block, in binary.
6	10-11	RECORDS PER TRACK (two characters)	Number of records per track, ex- cluding track linking record, in binary.
7	12	CYLINDER OVERFLOW (one character) (direct access or indexed se- quential files only)	Number of tracks per cylinder as- signed for overflow, in binary.

APPENDIX A. VOLUME LABEL AND VOLUME DIRECTORY

Table A-2 (cont). Volume Directory

FILE DESCRIPTION INDEX (*VOLDESCR*) (cont)			
Field	Position	Name and Length	Description
8	13	GENERAL OVERFLOW (one character)	General overflow indicator for direct access files: 00 = No general overflow 77 = The last cylinder of each unit of allocation is used for general overflow.
9	14	OPEN/CLOSE INDICATOR (one character)	Not used by Mod 1 (MSR). Included for compatibility with other operating systems.
10	15	BAD TRACK INDICATOR (one character)	1. Bit 1 (leftmost bit) = 0; the file is not allowed to have bad tracks. = 1; the file is allowed to have bad tracks. 2. Bit 2 = 0 the file does not contain any bad tracks. = 1; the file contains one or more bad tracks for which substitutes have been established.
11	16-21	RESERVED (six characters)	Reserved for future use.
12	22-26	CREATION DATE (five characters)	Date file was last created, in the form yyddd.
13	27-29	CREATION NO. (three characters)	Number of times this file has been reorganized, in decimal.
14	30-34	MODIFICATION DATE (five characters)	Date this file was last modified (i. e., opened for output-only or input/output only processing), in the form yyddd.
15	35-37	MODIFICATION NO. (three characters)	Number of times this creation of the file has been modified, in decimal.
16	38-42	EXPIRATION DATE (five characters)	The date on which this file is expected to expire, in the form yyddd.
17	43-50	PASSWORD (eight characters)	User-supplied code to permit access to the file. If omitted, no password protection exists for the file.
18	51-54	ITEM COUNT (four characters)	Partitioned Sequential: Inactive. Sequential: Active only for last <u>active</u> volume of the file. Direct Access and Indexed Sequential: Active for last volume of the file only. See also the Note at end of this table.
19	55	DATA STATUS (one character) NOTE: Always = 00 for direct access and partitioned sequential files.	02 - No data has been written on this file volume. 01 - Data has been written on this file volume, and it is not the last file volume with data. 00 = Data has been written on this file volume, and it is the last file volume with data.

APPENDIX A. VOLUME LABEL AND VOLUME DIRECTORY

Table A-2 (cont.) Volume Directory

Field	Position	Name and Length	Description
20	56	FILE DATA TYPE (one character) (sequential and partitioned sequential files only)	40 = Print-image file without report numbers or form adjustment. 42 = Print-image file with report numbers and form adjustment. 41 = Card-image file. (Mod 8 Operating System use) other = standard data file.
21	57	NUMBER OF CONTROL CHARACTERS (one character)	Number of control characters, in binary, for first n positions of each item of a print-image file.
22	58	TERMINAL FILE CONTROL FIELD (one character)	Not used by Mod 1 (MSR); included for compatibility with other operating systems.
23	59-60	MISCELLANEOUS INFORMATION (two characters)	Not used by Mod 1 (MSR); included for compatibility with other operating systems.
24	61-63	RESERVED (three characters)	Reserved for future use.
File Definition Information - Sequential Organization			
25	64-65	INDEX LENGTH (two characters)	Number of blocks in the member index, in binary, for a partitioned sequential file.
26	66-68	BLOCKS IN FILE VOLUME (three characters)	Total number of data blocks available to this file, in binary.
27	69	MEMBER INDEX ITEM LENGTH (one character)	Length of member index items, in binary. Must equal 31 (octal) to be processed by Mod 1 (MSR).
28	70-100	RESERVED (31 characters)	Reserved for future use.
File Definition Information - Direct Access Organization			
25	64-65	KEY LENGTH (two characters)	Number of characters in the key, in binary.
26	66-68	KEY DISPLACEMENT (three characters)	Number of positions from the left end of the item to the rightmost character of the key, in binary. Thus, if the key is the fourth to twelfth characters, this field is 11 (octal 13).
27	69-70	BLOCKS/BUCKET (two characters)	Number of blocks in a bucket, in binary.
28	71-100	RESERVED (30 characters)	Reserved for future use.
File Definition Information - Indexed Sequential Organization			
25	64-65	KEY LENGTH (two characters)	Number of characters in the key, in binary.
26	66-68	KEY DISPLACEMENT (three characters)	Number of positions from the left end of the item to the rightmost character of the key, in binary.
27	69-70	BLOCKS PER STRING (two characters)	Number of blocks in a string, in binary.
28	71-72	BLOCKS IN STRING INDEX (two characters)	Number of blocks in each string index, in binary.

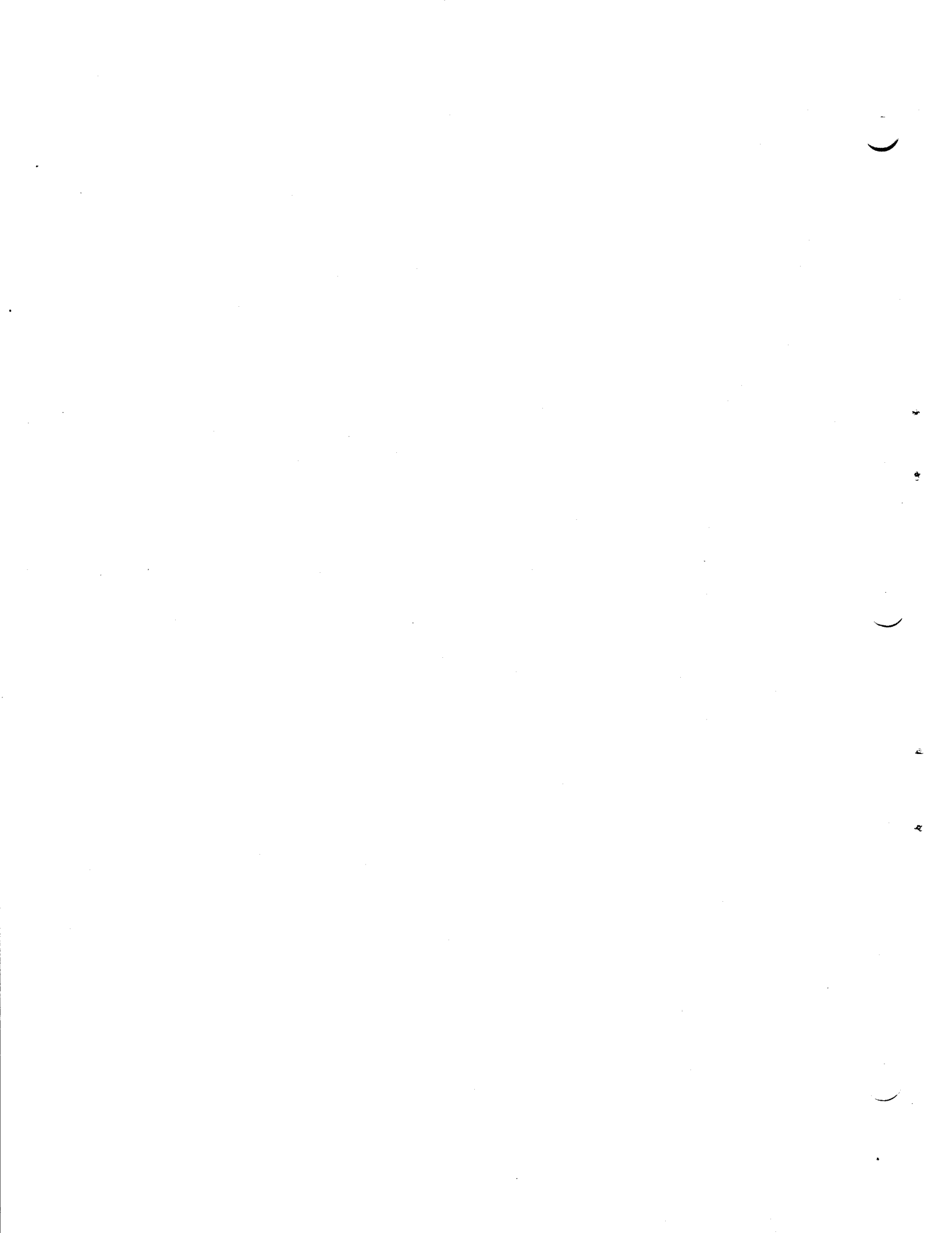
Table A-2 (cont). Volume Directory

FILE DESCRIPTION INDEX (*VOLDESCR*) (cont)			
Field	Position	Name and Length	Description
File Definition Information - Indexed Sequential Organization (cont)			
29	73-74	BLOCKS IN MASTER INDEX (two characters)	Number of blocks in the master index, in binary.
30	75-77	BLOCKS IN FILE VOLUME (three characters)	Total number of blocks in this file volume, in binary. The total includes index and general overflow.
31	78-100	RESERVED (23 characters)	Reserved for future use.
FILE ALLOCATION INDEX (*VOLALLOC*)			
1	1	STATUS (one character)	Status indication for this item: 77 ₈ = Unused or deleted item. 40 ₈ = Last data unit for this file. 60 ₈ = More data units follow on this volume. 20 ₈ = More data units of allocation follow on the next volume. 62 ₈ = Master/cylinder index unit; the general overflow unit follows on this volume. 22 ₈ = Master/cylinder index unit; the general overflow unit follows on the next volume. 61 ₈ = General overflow unit; the first data unit follows on this volume.
1		STATUS (cont)	21 ₈ = General overflow unit; the first data unit follows on the next volume.
2	2-4	RESERVED (three characters)	Reserved for future use.
3	5-12	ALLOCATION UNIT (eight characters)	Boundaries of this unit of allocation (in the binary form CCTTCCTT).

APPENDIX A. VOLUME LABEL AND VOLUME DIRECTORY

Table A-2 (cont). Volume Directory

FILE ALLOCATION INDEX (*VOLALLOC*) (cont)			
Field	Position	Name and Length	Description
FILE ALLOCATION INDEX (*VOLALLOC*) (cont)			
4	13-20	NEXT UNIT ADDRESS (eight characters)	If field 1 = 60, 61, or 62, field 4 = 00000000, where the next unit of allocation is the next item in the current block. Otherwise, field 4 is the address of the item in this file containing the next unit of allocation (in the form CCTTRRII). When field 1 = 20, 21, or 22, field 4 contains the volume name of the next volume in the file as the rightmost six characters. If field 1 = 40 or 77, the contents of field 4 are not specified.
<p>NOTE: If the file is processed using LIMVOL and if processing was terminated prior to the last file volume, the item count field (positions 51-54) in the last file volume processed is updated with the net change in item count during processing. Thus, when the LIMVOL option is used, it may be necessary to add the values in this field in all file volumes to obtain the true item count for this file.</p>			



APPENDIX B
PARTITIONING A SEQUENTIAL FILE

When the partitioning option is used, there are several additional advantages to sequential file organization. With this option, the sequential file is broken into any number of subfiles (members), which can vary in length. The partitioning option may be used for print files, storing various types of tables, or files which are segregated by state, wherein each state may be processed separately.

Each member of a partitioned sequential file must have identical properties (e.g., item size, record size, etc.). A member index is maintained to enable direct access to the beginning of any member. The number of blocks required to store the member index is specified at allocation time by the user (see "NOTE," page B-3). The member index begins with the first block in the file and continues through the number of blocks specified. The record size and block size of the member index are identical to those of the data area of the file. The item of the member index contains the name of the member, its address, the number of blocks in the member, and the status of the member; its size is 25 characters. An index can be examined by using Mass Storage Edit C to edit the first track(s) of the file (see Mod 1 (MSR) Utility Routines manual).

The name of the member identifies the member. A member name is 14 characters in length. The address of the member is the address of the first record in the member. The address is of the form Δ CCTTRR. This identifies the cylinder, track, and record of the first item of the member. The block count simply records the number of blocks in the member. The status of a member may be one of the following:

1. Deleted,
2. Able to be processed as input-only, or input/output, and
3. Able to be processed as input/output, input-only, or output-only. Members created by File Support C are assigned this status to allow unrestricted processing.

All member index items, except the first and the last, are composed of the four fields listed in Table B-1.

APPENDIX B. PARTITIONING A SEQUENTIAL FILE

Table B-1. Fields of Member Index Items

Field	Position	Name and Length	Description
1	1-14	Member name (14 characters)	A field which identifies a member. A member name must be composed from letters, digits, and spaces.
2	15-21	Address (7 characters)	The address of the first record of the member in the form ΔCCTRR, in binary.
3	22-24	Block count (3 characters)	A binary count of the number of blocks in a member.
4	25	Status (1 character)	20 (octal) = This member can be processed as input, output, or input/output. 00 (octal) = This member can be processed as input-only or input/output. 40 (octal) = Deleted member.

The first item in a member index is composed of the following four fields, as listed in Table B-2.

Table B-2. Fields of First Item in Member Index

Field	Position	Name and Length	Description
1	1-14	*UNUSED* ΔΔΔΔΔΔ (14 characters)	
2	15-21	Address (7 characters)	The address of the first record in the unused area of this file in the form ΔCCTRR, in binary.
3	22-24	Block count (3 characters)	A binary count of the number of blocks remaining in the unused area of this file.
4	25	Status (1 character)	10 (octal) = Item pointing to unused area.

The last item in a member index is composed of the following four fields, as listed in Table B-3.

Table B-3. Fields of Last Item in Member Index

Field	Position	Name and Length	Description
1	1-14	*ENDINDEX* ΔΔΔΔ (14 Characters)	
2	15-21	Address (7 characters)	The address of the first data record for this file in the form ΔCCTTRR, in binary.
3	22-24	Block count (3 characters)	Total number of data blocks for this file, in binary.
4	25	Status (1 character)	01 (octal) = End-of-index item.

The first item in the index always contains the address of the first record in the file that is available for the addition of a new member. When the partitioned sequential file is allocated and before data is entered, the member index contains at least two items: one indicating the unused area, and the other indicating the end of index. In addition, at allocation time, block space may have been reserved for one or more members. When a member is created after allocation its block count is computed after the data is placed. When a member is deleted, its data area is not reusable until the file has been reorganized. (The Program Development Subsystem, however, does re-use space in the library and residence files.) Figure B-1 shows a sequentially organized file using the partitioning option.

NOTE: The number of blocks which should be assigned to the member index is determined as follows.

$$I_B = \frac{(\text{item}) (B_I)}{25}$$

Ignore any remainder.

$$N = \frac{M_u + 2}{I_B}$$

I_B = Number of index items per block,

N = Number of index blocks, expressed as next higher integer,
(e.g., 19.2 = 20),

M_u = User members, maximum active at one time,

Item = Item size of the file (per allocate), and

B_I = Items per block of the file (per allocate).

APPENDIX B. PARTITIONING A SEQUENTIAL FILE

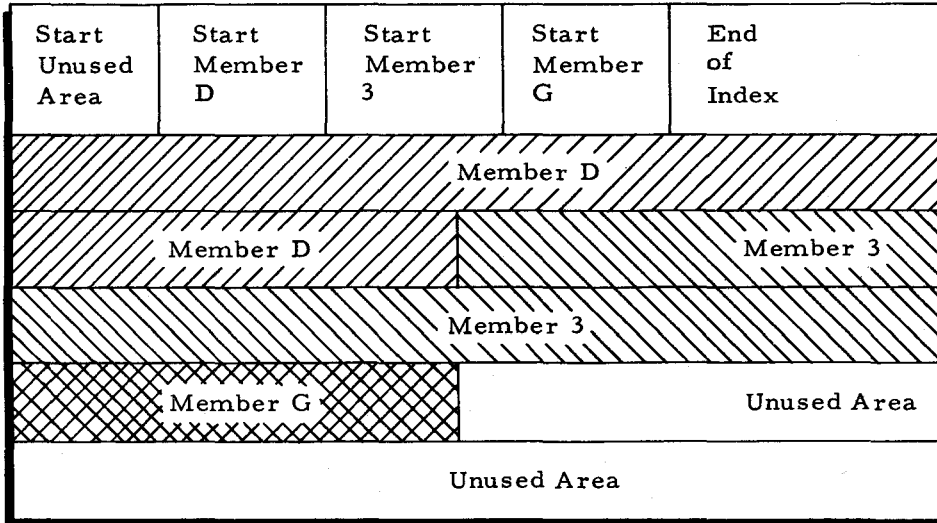


Figure B-1. Sequential File Using Partitioning Option

APPENDIX C

FILE DESIGN AND ALLOCATION

In setting up data files for mass storage, sufficient time should be devoted to the planning process. Too often mass storage files are treated as if they were magnetic tape files. Considerations for both media are frequently quite different. For example, updating of magnetic tape files normally involves a copy of all data, changed or unchanged, from one tape to another. With mass storage, applying the changes directly to the file and updating only affected items is often more efficient. This is especially true when the percentage of the file being changed is relatively low. Careful planning of mass storage record and block lengths is necessary for maximum data storage capacity.

FILE DESIGN CRITERIA

The following paragraphs describe certain considerations that should be taken into account by the user before deciding what file organization to use.

Application Considerations

The most elementary question that must be asked about any file is: "What types of operations are to be performed on the file?" Other pertinent questions follow.

FILE ADDITIONS

Are the programs processing this file going to add items to the file? How many items are to be added to the file? Are the items added in a random manner?

Provision for new items is made in both direct access files and indexed sequential files. The sequential file organization can handle additions only if the organization is treated as magnetic tape, i. e., each time items are added, the file is copied.

The direct access file and the indexed sequential file provide certain overflow capabilities for handling additions. These capabilities are discussed in the sections of the manual describing specific file organizations. Depending upon their frequency and distribution, a specific file organization may be preferred for handling a particular application.

FILE INQUIRIES

Is this file primarily used as a reference to be interrogated? Some file organizations are designed to provide quick and easy access to any given item. The direct access organization provides the fastest access to any given item. However, careful planning of string size, block

length, and placement of the master/cylinder index can provide very rapid access to the indexed sequential organization without requiring any randomizing of a key.

RANDOM VERSUS SEQUENTIAL FILES

Is the activity in this file random or are transactions sorted? What would be the effect of doing it differently?

The answers to these questions will help to determine the design of the system and the files within the system. In many cases, the application being developed is part of an existing system. In such a case, the constraints are usually quite rigid. When designing a new system, all of the assumptions should be closely examined. For example, a decision to handle a set of transactions randomly may at first indicate the use of a direct access or indexed sequential file. However, closer examination might reveal that the volume of transactions is so high that a sequential operation might be faster.

RANDOM PLUS SEQUENTIAL FILES

Is it important to be able to process this file sequentially and directly? Would it be useful to process portions of the file sequentially?

The only file organization offering the full capability of both sequential and direct access processing is the indexed sequential file. In a system in which a file is most frequently processed directly and least frequently processed sequentially, it is important to consider the advantages and disadvantages of using either an indexed sequential file or a direct access file. For instance, although sorting may not be required for an indexed sequential file, this file frequently requires a longer processing time. However, a direct access file which may be faster to process requires sorting prior to any sequential operation.

There are many applications in which it is desirable to process only portions of a file sequentially. A file organized by some multileveled key would lend itself well to this type of operation. For example, if the first three digits of the key of a master file were a branch or area code, it would be possible (using an indexed sequential organization) to go directly to any branch in the file and then process only that branch.

General File Design Considerations

BLOCK LENGTH

In general, the most significant factor in achieving a high rate of throughput is block length. The allocation of the largest possible block to the file, consistent with the requirements of all of the programs processing the file, almost always results in optimum efficiency. The reason

for this is the latency time of mass storage devices. For instance, on the Type 259 Disk Pack Drive, waiting one revolution (25 milliseconds) following a read or write before the next block on that track can be processed is normal. Thus, using small blocks can demand considerable time. For example, the time required to process a track with fifteen 250-character records would be $15 \times 25 = 375$ milliseconds, whereas the time to process a track of five 880-character records would be $5 \times 25 = 125$ milliseconds.

ASSIGNMENTS OF UNITS OF ALLOCATION

On a Type 258 or Type 259 Disk Pack Drive, the allocation of a full ten tracks per cylinder is generally most efficient; on a Type 273 Disk Pack Drive, however, the allocation of 20 tracks per cylinder is most efficient. Occasional exceptions exist for very small files, but usually ten or twenty tracks per cylinder is most efficient, since it reduces the number of cylinders for the file and, hence, the number of seeks to process the file.

NOTE: Only two tracks are available on the Type 155 Disk Pack Drive.

As few units of allocation as possible, preferably one, should be used. However, when handling many files on a single disk pack, it may be economical to use any units available at the time. If the file organization is sequential, the location of these units has little influence on the processing time. If the file is to be processed directly (direct access or indexed sequential), file processing time increases in proportion to the distance (in terms of cylinders) between the units assigned to one volume.

Additional considerations for the placement of units of allocation for an indexed sequential file are discussed in Sections II and IV.

MULTIVOLUME FILE PROCESSING

If a multivolume sequential file is to be processed sequentially from its beginning, only one file volume need be mounted at any given time. If a multivolume indexed sequential file is to be processed sequentially from its beginning, at least one file volume which contains data units of allocation and the volume(s) which contain(s) the master/cylinder index and the general overflow area must be mounted at any given time. (When processing an indexed sequential file in any mode, the master/cylinder index and the general overflow area must always be on line.) Any other type of processing done with a direct access or indexed sequential file requires that all volumes be mounted and available concurrently. When extra disk drives are available, however, delays in changing volumes can be minimized by having the necessary volumes mounted.

Normally, in order to make best use of space, a file should be contained within a minimum number of volumes. However, if speed of direct access is critical, a file may be split over a

larger number of volumes. For example, a file consisting of 160 cylinders of information requires an average seek time of 76 milliseconds if allocated as 160 consecutive cylinders on one disk pack. Splitting the file into four file-volumes, each with a 40-cylinder unit of allocation, results in an average seek time of 47 milliseconds.

ASSIGNMENT OF FILES TO BE PROCESSED CONCURRENTLY

Many programs need to process more than one file. Occasionally, two or more of these files will be on mass storage, possibly reducing processing efficiency. Planning should be done with great care.

When two or more files share the same volume and at least one of the files is to be processed sequentially (regardless of file organization), efficiency is sacrificed. The file(s) being processed sequentially in this application should be placed either on a separate volume or on an entirely different device.

Ordinarily, in processing a mass storage file sequentially, minimum seek time is assumed; i. e., the only seeking required is from the end of one cylinder to the beginning of the next sequential cylinder. If another file on the same volume is being processed, there may be as much as one seek (on a Type 259 Disk Pack Drive this means 30 to 165 ms.) for each block. In such a case, the decrease in efficiency is considerable. When communicating files can be placed on different drives, processing time is improved by reducing head travel.

SEQUENTIAL FILE CONSIDERATIONS

Allocation

The unit of allocation is of the form $C_1 T_1 C_2 T_2$. To determine how much space is required for a given sequential file, the following process is used.

1. The following values must be known; they are represented symbolically as:

BL = Block (or buffer) length,
I = Total number of items in the file,
T = Tracks per cylinder,* and
IB = Items per block.

2. Using Table C-1 or C-2, locate the correct values for number of records per track (RT) and number of records per block (RB). This is accomplished by scanning the leftmost column to locate the block length (BL) and then taking the corresponding values for records per track (RT) and records per block (RB).

* Normally, tracks per cylinder (T) is 2 for the Type 155 Disk Pack Drive, 10 for the Type 258 or Type 259 Disk Pack Drive and 20 for the Type 273 Disk Pack Drive. The user may, however, use any smaller number of tracks.

APPENDIX C. FILE DESIGN AND ALLOCATION

3. Compute blocks per cylinder (BC) as follows:

$$BC = \frac{RT \times T}{RB} \text{ (ignore any remainder).}$$

4. Compute items per cylinder (IC) as follows:

$$IC = BC \times IB.$$

5. Compute the number of cylinders (C) required for this file as follows:

$$C = \frac{I}{IC} \text{ (round up to the next higher integer).}$$

Example 1 - Determining Space for Sequential File

An example of computing the space required on a Type 259 Disk Pack Drive using the process previously described follows:

Assuming that: Block length (BL) = 1,218 (characters per block),
 Total items (I) = 5,600 (approximate value),
 Tracks per cylinder (T) = 10, and
 Items per block (IB) = 6 (item size = 203 characters).

There is to be one unit of allocation, starting on cylinder 20.

1. From Table C-1, records per track (RT) = 7, and records per block (RB) = 2.
2. Blocks per cylinder (BC) = $\frac{10 \times 7}{2} = 35$ (remainder dropped, if any).
3. Items per cylinder (IC) = $35 \times 6 = 210$
4. Cylinders for the file (C) = $\frac{5,600}{210} = 26$, plus a remainder of 140; the result is rounded up to 27.
5. Therefore, the unit of allocation for this file, in the form $C_1 T_1 C_2 T_2$, would be: 20-0-46-9. Its maximum capacity is 5,669 items (210 items per cylinder x 27) - 1 = 5,669.

Table C-1. Optimum Record Size - Types 155, 258, 259, 273, 259A, and 259B Disk Pack Drives

Characters per Block (BL)	Record Size	Number of Records per Block (RB)	Number of Records per Track (RT)	Number of Data Characters per Track
80-81	Same as block	1	32	2560-2592
82-86	Same as block	1	31	2542-2666
87-91	Same as block	1	30	2610-2730
92-96	Same as block	1	29	2668-2784
97-102	Same as block	1	28	2716-2856
103-109	Same as block	1	27	2791-2943
110-115	Same as block	1	26	2860-2990
116-123	Same as block	1	25	2900-3075

APPENDIX C. FILE DESIGN AND ALLOCATION

Table C-1 (cont). Optimum Record Size - Types 155, 258, 259, 273, 259A, and 259B Disk Pack Drives

Characters per Block (BL)	Record Size	Number of Records per Block (RB)	Number of Records per Track (RT)	Number of Data Characters per Track
124-131	Same as block	1	24	2976-3144
132-139	Same as block	1	23	3036-3197
140-149	Same as block	1	22	3080-3278
150-159	Same as block	1	21	3150-3339
160-170	Same as block	1	20	3200-3400
171-183	Same as block	1	19	3249-3477
184-197	Same as block	1	18	3312-3546
198-213	Same as block	1	17	3366-3621
214-230	Same as block	1	16	3404-3680
231-250	Same as block	1	15	3465-3750
251-271	Same as block	1	14	3514-3794
272-297	Same as block	1	13	3536-3861
298-328	Same as block	1	12	3576-3936
329-364	Same as block	1	11	3619-4004
365-407	Same as block	1	10	3650-4070
408-460	Same as block	1	9	3672-4140
461-524	Same as block	1	8	3688-4192
525-609	Same as block	1	7	3675-4263
610-723	Same as block	1	6	3660-4338
724-728	Block/2	2	11	3982-4004
729-880	Same as block	1	5	3645-4400
881-920	Block/2	2	9	3960-4140
921-1117	Same as block	1	4	3684-4468
1118-1218	Block/2	2	7	3913-4263
1219-1221	Block/3	3	10	4060-4070
1222-1512	Same as block	1	3	3666-4536
1513-1572	Block/3	3	8	4032-4192
1573-1760	Block/2	2	5	3930-4400
1761-1827	Block/3	3	7	4109-4263
1828-1840	Block/4	4	9	3888-4140
1841-2301*	Same as block	1	2	3682-4602
2302-2436	Block/4	4	7	4025-4263
2437-2640	Block/3	3	5	4060-4400
2641-3024	Block/2	2	3	3960-4536

APPENDIX C. FILE DESIGN AND ALLOCATION

Table C-1 (cont). Optimum Record Size - Types 155, 258, 259, 273, 259A, and 259B Disk Pack Drives

Characters per Block (BL)	Record Size	Number of Records per Block (RB)	Number of Records per Track (RT)	Number of Data Characters per Track
3025-3045	Block/5	5	7	4235-4263
3046-3351	Block/3	3	4	4060-4468
3352-3520	Block/4	4	5	4190-4400
3521-3615	Block/5	5	6	4224-4338
3616-3654	Block/6	6	7	4214-4263
3655-3668	Block/7	7	8	4176-4192
3669-3680	Block/8	8	9	4122-4140
3681-4602	Block/2	2	2	3680-4602

NOTE: Where the division of block length leaves a fraction, the record size should be expressed as the next higher integer.

*Capacity is maximum when record size is 2,301 characters.

Table C-2. Optimum Record Size - Type 261 or Type 262 Disk Files

Characters per Block (BL)	Record Size	Number of Records Block (RB)	Number of Records per Track (RT)	Number of Data Characters per Track
125-128	Same as block	1	50	6250-6400
129-132	Same as block	1	49	6321-6468
133-136	Same as block	1	48	6384-6528
137-141	Same as block	1	47	6439-6627
142-145	Same as block	1	46	6532-6670
146-150	Same as block	1	45	6570-6750
151-155	Same as block	1	44	6654-6820
156-160	Same as block	1	43	6708-6880
161-165	Same as block	1	42	6762-6930
166-171	Same as block	1	41	6806-7011
172-177	Same as block	1	40	6880-7080
178-183	Same as block	1	39	6942-7137
184-189	Same as block	1	38	6992-7182
190-196	Same as block	1	37	7030-7252
197-203	Same as block	1	36	7092-7308
204-211	Same as block	1	35	7140-7385
212-219	Same as block	1	34	7208-7446
220-228	Same as block	1	33	7260-7524

APPENDIX C. FILE DESIGN AND ALLOCATION

Table C-2 (cont). Optimum Record Size - Type 261 or Type 262 Disk Files

Characters per Block (BL)	Record Size	Number of Records per Block (RB)	Number of Records per Track (RT)	Number of Data Characters per Track
229-237	Same as block	1	32	7328-7584
238-247	Same as block	1	31	7378-7657
248-256	Same as block	1	30	7440-7680
257-266	Same as block	1	29	7453-7714
267-278	Same as block	1	28	7476-7784
279-291	Same as block	1	27	7533-7857
292-304	Same as block	1	26	7592-7904
305-319	Same as block	1	25	7625-7975
320-335	Same as block	1	24	7680-8040
336-353	Same as block	1	23	7728-8119
354-372	Same as block	1	22	7788-8184
373-393	Same as block	1	21	7833-8253
394-416	Same as block	1	20	7880-8320
417-441	Same as block	1	19	7923-8379
442-469	Same as block	1	18	7956-8442
470-501	Same as block	1	17	7990-8517
502-534	Same as block	1	16	8032-8544
535-575	Same as block	1	15	8025-8625
576-620	Same as block	1	14	8064-8680
621-673	Same as block	1	13	8073-8749
674-735	Same as block	1	12	8088-8820
736-806	Same as block	1	11	8096-8866
807-894	Same as block	1	10	8070-8940
895-1001	Same as block	1	9	8055-9009
1002-1002	Block/2	2	17	8517-8517
1003-1133	Same as block	1	8	8024-9064
1134-1150	Block/2	2	15	8505-8625
1151-1303	Same as block	1	7	8057-9121
1304-1346	Block/2	2	13	8476-8749
1347-1533	Same as block	1	6	8082-9198
1534-1612	Block/2	2	11	8437-8866
1613-1851	Same as block	1	5	8065-9255
1852-1860	Block/3	3	14	8638-8680
1861-2002	Block/2	2	9	8370-9009
2003-2019	Block/3	3	13	8671-8749
2020-2329	Same as block	1	4	8080-9316

APPENDIX C. FILE DESIGN AND ALLOCATION

Table C-2 (cont). Optimum Record Size - Type 261 or Type 262 Disk Files

Characters per Block (BL)	Record Size	Number of Records Block (RB)	Number of Records per Track (RT)	Number of Data Characters per Track
2330-2418	Block/3	3	11	8536-8866
2419-2606	Block/2	2	7	8463-9121
2607-2682	Block/3	3	10	8690-8940
2683-2692	Block/4	4	13	8710-8749
2693-3127*	Same as block	1	3	8079-9381
3128-3224	Block/4	4	11	8602-8866
3225-3399	Block/3	3	8	8600-9064
3400-3702	Block/2	2	5	8500-9255
3703-3909	Block/3	3	7	8638-9121
3910-4030	Block/5	5	11	8602-8866
4031-4038	Block/6	6	13	8723-8749
4039-4095	Same as block	1	2	8078-8190
4096-4509	Block/9	9	17	7735-8517
4510-4600	Block/8	8	15	8445-8625
4601-4711	Block/7	7	13	8541-8749
4712-4836	Block/6	6	11	8635-8866
4837-5005	Block/5	5	9	8703-9009
5006-5212	Block/4	4	7	8757-9129
5213-5553	Block/3	3	5	8685-9255
5554-5665	Block/5	5	8	8880-9064
5666-6254	Block/2	2	3	8499-9381
6255-6258	Block/7	7	10	8930-8940
6259-6515	Block/5	5	7	8757-9121
6516-6987	Block/3	3	4	8688-9316
6988-7007	Block/7	7	9	8982-9009
7008-7404	Block/4	4	5	8760-9255
7405-7665	Block/5	5	6	8886-9198
7666-7818	Block/6	6	7	8939-9121
7819-7931	Block/7	7	8	8936-9064
7932-8008	Block/8	8	9	8919-9009
8009-8046	Block/9	9	10	8890-8940
8047-8060	Block/10	10	11	8844-8866
8061-8085	Block/11	11	12	8784-8820
8086-9381	Block/3	3	3	8085-9381

*Capacity is maximum when record size is 3, 127 characters.

DIRECT ACCESS FILE CONSIDERATIONS

To properly use a direct access file requires careful planning. There are two essentials that the user himself must calculate: (1) proper allocation of storage space and (2) addresses for every item, assigned in such a way that items are dispersed as evenly as possible throughout the space allocated to the file. The following paragraphs provide some general guidelines for allocating storage space and assigning addresses in direct access files.

Bucket Size and Overflow

Any method of calculation used to translate item keys into addresses generally produces a number of synonyms (duplicate addresses). (Refer to Appendix E for a description of randomizing techniques.) These synonyms are handled in two ways: (1) buckets may be made large enough to hold all synonyms for a given address, and (2) overflow areas may be specified that hold items which overflow a bucket due to uneven distribution. If there were no variation in the number of synonyms generated for each address, there would be no overflow. But, since some buckets normally contain more synonyms and some less, some items will overflow.

A more even distribution of items can be obtained from a randomizing routine by making the bucket size larger (i. e., generating more synonyms having fewer addresses). The validity of this statement is seen if the extreme cases are used as examples. First, if it is assumed that relative bucket addressing is used and that the whole file contains one bucket, then all items of the file would have 0 as their address. Of course, all items would be synonyms. Since every bucket has an equal chance of having its address generated (because there is only one bucket), there is even distribution of the items over the allocated space. Second, viewing the other extreme, if every item were a bucket, then it would be much more difficult to get an even distribution, since it is difficult to ensure that every item space has an equal chance of being used. Thus, it can be seen from these two examples that it is not the average number of synonyms for a bucket that determines the efficiency of a randomizing routine; rather, it is the amount of deviation from this average or, in other words, the evenness of the distribution of the items over the allocated space that determines the efficiency.

The amount of overflow that occurs is directly related to two factors: (1) bucket size and (2) storage density. The larger the bucket (i. e., the more synonyms for any address), the lower the probability for any item that the bucket will overflow. Storage density also affects bucket overflow. A file with space for 1,000 items will have more bucket overflow when it contains 800 items (storage density = 0.8) than when it contains 500 items (storage density = 0.5). Thus, overflow may be thought of as an extension of the bucket that accommodates the uneven distribution of items in buckets. As the bucket size becomes larger, the distribution becomes

APPENDIX C. FILE DESIGN AND ALLOCATION

more even and there is less need for overflow areas. However, when designing a direct access file, increasing the size of the bucket increases the average time required for access of an item. Thus, when determining the bucket size, the probability of overflow should be weighed against the desired speed of retrieving an item.

Table C-3 summarizes the overflow probabilities (i. e., the probability that an item will overflow), assuming an even distribution.

Table C-3. Overflow Probabilities

Bucket Size (Items/Bucket)	Number of Items/Allocated Space									
	Storage Density									
	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
1	4.84	9.37	13.61	17.58	21.31	24.80	28.08	31.17	34.06	36.79
2	0.60	2.19	4.49	7.27	10.36	13.65	17.03	20.43	23.79	27.07
3	0.09	0.63	1.80	3.61	5.99	8.82	11.99	15.37	18.87	22.40
4	0.02	0.20	0.79	1.96	3.76	6.15	9.05	12.32	15.86	19.54
5	0.00	0.07	0.37	1.12	2.48	4.49	7.11	10.26	13.78	17.55
6	0.00	0.02	0.18	0.67	1.69	3.38	5.75	8.75	12.24	16.06
7	0.00	0.01	0.09	0.41	1.18	2.60	4.74	7.60	11.04	14.00
8	0.00	0.00	0.05	0.25	0.84	2.03	3.97	6.68	10.07	13.96
9	0.00	0.00	0.02	0.16	0.61	1.61	3.36	5.94	9.27	13.18
10	0.00	0.00	0.01	0.10	0.44	1.29	2.88	5.32	8.59	12.51
11	0.00	0.00	0.01	0.07	0.33	1.04	2.48	4.80	8.04	11.94
12	0.00	0.00	0.00	0.04	0.24	0.85	2.15	4.36	7.51	11.44
14	0.00	0.00	0.00	0.02	0.14	0.57	1.65	3.64	6.67	10.60
16	0.00	0.00	0.00	0.01	0.08	0.39	1.28	3.09	6.00	9.92
18	0.00	0.00	0.00	0.00	0.05	0.28	1.01	2.65	5.45	9.36
20	0.00	0.00	0.00	0.00	0.03	0.20	0.81	2.30	4.99	8.88
25	0.00	0.00	0.00	0.00	0.01	0.09	0.48	1.65	4.10	7.95
30	0.00	0.00	0.00	0.00	0.00	0.04	0.29	1.23	3.47	7.26
35	0.00	0.00	0.00	0.00	0.00	0.02	0.18	0.94	2.98	6.73
40	0.00	0.00	0.00	0.00	0.00	0.01	0.12	0.73	2.60	6.29
50	0.00	0.00	0.00	0.00	0.00	0.00	0.05	0.45	2.01	5.63
60	0.00	0.00	0.00	0.00	0.00	0.00	0.02	0.30	1.65	5.14
70	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.20	1.37	4.76
80	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.13	1.14	4.46
90	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.09	0.97	4.20
100	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.06	0.83	3.99

NOTES: 1. These probabilities are given as percentages.
 2. This table assumes an even distribution. In actual practice, perfectly even distribution is seldom, if ever, obtained. The actual probability of overflow, therefore, will usually be higher.

Allocation

The unit of allocation is of the form $C_1T_1C_2T_2$. To determine how much space is required for a given direct access file, the following process is used.

1. The following figures must be known and are represented symbolically as:

BL = Block (or input/output buffer) length,

I = Total number of items in the file,

T = Tracks per cylinder,

IB = Items per block,

SD = Storage density, and

BB = Blocks per bucket.

2. Using Table C-1, locate the correct values for number of records per track (RT) and number of records per block (RB).
3. Compute items per bucket (IK) as follows:
IK = IB x BB.
4. Using Table C-3, determine the probability (P) of overflow by using items per bucket on the vertical axis and storage density on the horizontal axis.
5. Using Table C-4, find the number of overflow tracks whose percentage of data area brackets P. If the lower percentage is only slightly less than P, the corresponding tracks of overflow can be used along with general overflow, or the tracks of overflow corresponding to the higher percentage can be used without general overflow. If the higher percentage is only slightly greater than P, the corresponding tracks of overflow should be used in addition to general overflow. Otherwise, choose one of these bracketing percentages to obtain the number of cylinder overflow tracks required (OT).
6. Compute the item space (IS) required as follows:

$$IS = \frac{I}{SD} \text{ (round up to next higher integer).}$$

7. Compute the buckets (B) required as follows:

$$B = \frac{IS}{IK} \text{ (round up to next higher integer).}$$

8. Compute the buckets per cylinder (BC) as follows:

$$BC = \frac{RT \times (T - OT)}{RB \times BB} \text{ (ignore any remainder).}$$

9. Compute the cylinders (C) required for this file as follows:

$$C = \frac{B}{BC} \text{ (round up to next higher integer).}$$

Table C-4. Cylinder Overflow as Percentage of Data Area

Percentage of Data Area		Number of Cylinder Overflow Tracks Required	Number of Data Tracks Remaining	
Type 259	Types 261/262		Type 259	Types 261/262
0.0	0.00	0	10	128
11.1	0.79	1	9	127
25.0	1.59	2	8	126
42.9	2.40	3	7	125
66.7	3.23	4	6	124
100.0	4.06	5	5	123
150.0	4.92	6	4	122
...

To illustrate the procedure for allocation of a direct access file, two examples are shown: one for optimizing speed and the other for optimizing storage density. For these examples, the following values are assumed.

Total number of items in file (I)	10,000 items
Items per block (IB)	4 (item size = 200 characters)
Block length (BL)	800
Storage density (SD)	0.8
Tracks per cylinder (T)	10

Example 1 - Optimizing Speed

For this example, the bucket is equal to one block ($BB = 1$). Thus, each bucket has a capacity of four items. Using the probabilities chart (Table C-3), it can be seen that the likelihood of any one item overflowing is 12.3 percent. If one track for cylinder overflow is allowed, 1/9th or 11.1 percent of the data area is set aside for overflow. If two tracks per cylinder are allowed, 2/8ths or 25 percent of the data area is set aside for overflow.

Since the 12.3 percent from Table C-3 is bracketed by the 11.1 percent (one overflow track) and the 25 percent (two overflow tracks), it means that with even distribution the overflow could almost be accommodated with one cylinder overflow track (the rest would go in general overflow). Excess overflow space would be available if two tracks for cylinder overflow were specified.

If the important consideration for this file is average access time, one track per cylinder for overflow would probably be sufficient (along with general overflow). However, if it is important that no access exceed a certain time limit, two tracks could be used to gain the 25 percent overflow provision so that general overflow would rarely, if ever, be accessed.

APPENDIX C. FILE DESIGN AND ALLOCATION

Using one track for overflow (OT = 1), the cylinders required for allocation would be computed as follows:

$$\text{Item space (IS) required} = \frac{10,000 (I)}{.80 (SD)} = 12,500 \text{ and}$$

$$\text{Buckets (B) required} = \frac{12,500 \text{ items (IS)}}{4 \text{ items/bucket (IK)}} = 3,125 \text{ buckets.}$$

From Table C-1, it can be seen that we should have five records per track, one record per block.

$$\text{Buckets per cylinder (BC)} = \frac{5 \text{ records/track (RT)} \times [10 \text{ tracks (T)/cylinder} - 1 \text{ track (OT)/cylinder}]}{1 \text{ record/block (RB)} \times 1 \text{ block/bucket (BB)}} = 45 \text{ buckets/cylinder, and}$$

$$\text{Cylinders (C) per file} = \frac{3,125 \text{ buckets (B)}}{45 \text{ buckets/cylinder (BC)}} = 69.4 \text{ or } 70.0,$$

plus one cylinder per unit of allocation for general overflow. Assuming that there is one unit of allocation, 71 cylinders would be required for this file.

If two tracks were to be used for overflow, the above calculations change to:

$$\text{Buckets per cylinder (BC)} = \frac{5 \text{ records/track (RT)} \times [10 \text{ tracks (T)/cylinder} - 2 \text{ tracks (OT)/cylinder}]}{1 \text{ record/block (RB)} \times 1 \text{ block/bucket (BB)}} = 40 \text{ buckets/cylinder, and}$$

$$\text{Cylinders (C) per file} = \frac{3,125 \text{ buckets (B)}}{40 \text{ buckets/cylinder (BC)}} = 78.1 \text{ or } 79.$$

Again, if general overflow is desired, it is added accordingly.

Example 2 - Optimizing Density

In the second example, an attempt is made to make more efficient use of storage (thus sacrificing some speed). It is planned to have 25 blocks per bucket (BB = 25). Looking at the probability chart, it can be seen that the likelihood of overflow in this case is about 0.1 percent. This percentage is so small that it would be sufficient to have no cylinder overflow and use only general overflow. In this case, all ten tracks are used for the data area (OT = 0). To compute the number of cylinders required for this file, the following computations are performed:

$$\text{Item space required (IS)} = \frac{10,000 (I)}{0.80 (SD)} = 12,500, \text{ and}$$

$$\text{Buckets per file (B)} = \frac{12,500 (IS)}{100 (IK)} = 125 \text{ buckets.}$$

$$\text{Buckets per cylinder (BC)} = \frac{5 \text{ records/track (RT)} \times 10 \text{ tracks (T)/cylinder}}{1 \text{ record/block (RB)} \times 25 \text{ blocks/bucket (BB)}} = 2, \text{ and}$$

$$\text{Cylinders per file (C)} = \frac{125 (B)}{2 (BC)} = 62.5 \text{ or } 63 \text{ cylinders.}$$

One cylinder per unit of allocation must be added for general overflow. Assuming that there is one unit of allocation, 64 cylinders would be required for this file. In the first case, if relative addressing were being used, addresses distributed between 0 and 3,124 would be required. In the second case, addresses between 0 and 124 would be required.

Example 3 - Optimizing Capacity of a Direct Access File on a Type 261 Disk File

Assume that 120,000 items are to be placed with a storage density of 0.88. Item size will be 208 characters. Maximum buffer size is 6,448 characters. One block per bucket is required.

An examination of Table C-2 establishes the following choices.

Block Length	Record Size	Records Per Track	Order of Preference	Data Characters Per Track
6448	Block/5 = 1290	7	C ₂	9030
6240	Block/2 = 3120	3	A ₂	9360
6032	Block/2 = 3016	3	C ₁	9048
5824	Block/2 = 2912	3	E ₂	8736
5616	Block/5 = 1124	8	D ₂	8992
5408	Block/3 = 1803	5	C ₃	9015
5200	Block/4 = 1300	7	B	9100
4992	Block/5 = 999	9	D ₁	8991
4784	Block/6 = 798	11	E ₁	8778
4576	Block/8 = 572	15	F	8580
4368	Block/9 = 486	17	G	8262
3952	Block/5 = 791	11	E ₃	8701
3120	Same as Block = 3120	3	A ₁	9360

In the above example, "Order of Preference" indicates relative maximization of storage capacity, from A (which provides the maximum value of characters per track) descending through G (which provides the lowest value in the example).

Since a buffer size up to 6,448 characters is permissible, a block length of 6,240 characters (case A₂) offers maximum track capacity (9,360) and a large buffer. The next choice of block length which yields equivalent track capacity is a block of only 3,120 characters (case A₁). A choice of a 4,368-character block (case G) would result in an 11 percent loss of capacity compared with case A₁ or A₂.

If we now assume that the direct access file will have a track width of 60 tracks, we establish the following.

1. BL = Block length = 6,240.
 I = Items in file = 120,000.
 T = Tracks per cylinder = 60.
 IB = Items per block = $\frac{6240}{208} = 30$.
 SD = Storage density = 0.88.
 BB = Blocks per bucket = 1.
2. Referring to Table C-2:
 RT = Records per track = 3 and
 RB = Records per block = 2.
3. IK = Items per bucket, computed by:
 IK = IB x BB = 30 x 1 = 30
4. P = Probability of overflow, extrapolated from Table C-3:
 $1.23 + \frac{8}{10} \times (3.47 - 1.23) = 1.23 + 1.79 = 3.02\%$
5. Since 2 tracks of overflow will provide:
 $\frac{60 - 58}{58} \times 100 = 3.45\%$ overflow (cylinder level),
 and 3 tracks will provide:
 $\frac{60 - 57}{57} \times 100 = 5.27\%$ overflow (cylinder level).
 The user can therefore elect no general overflow and three tracks of cylinder overflow with a reasonable assurance of safety.
6. Compute item space (IS) as follows:
 $IS = \frac{I}{SD} = \frac{120,000}{0.88} = 136,363$
7. Compute the required number of buckets as follows:
 $B = \frac{IS}{IK} = \frac{136,363}{30} = 4,546$ buckets (next higher integer).
8. Compute the buckets per cylinder as follows:
 $BC = \frac{RT \times (T - OT)}{RB \times BB} = \frac{3 \times (60 - 3)}{2 \times 1} = 85$ buckets (remainder dropped).
9. Compute the cylinders required as follows:
 $C = \frac{B}{BC} = \frac{4546}{85} = 54$ cylinders (next higher integer).

Thus, a decision might be made to put 18 cylinders on each of three separate Type 261 Disk Files; the average head motion would then be five cylinders (allowing for the paired cylinder concept of the Type 261 Disk File).

If prime number randomizing of relative bucket address is planned, the suitable prime would be 4,547 (from Table E-1).

The actual number of buckets available, however, ($85 \times 54 = 4,590$) is 44 more than the number required (4,546). Use of a larger prime, such as 4,583, would be permissible, if it was desired to lower the storage density of the file slightly.

INDEXED SEQUENTIAL FILE CONSIDERATIONS

Indexed sequential file organization provides flexibility and can be used with many types of applications. This flexibility introduces a wide range of considerations that must be examined before an indexed sequential file is defined and introduced into an application.

Design Considerations

The following paragraphs describe design considerations which should be taken into account by the user when organizing an indexed sequential file.

ITEM SEQUENCE

Items in an indexed sequential file are ordered in ascending binary collating sequence according to some item key field. Active items in the file cannot have duplicate keys.

DISTRIBUTION AND VOLATILITY

The structure of an indexed sequential file is fixed when the file is loaded. Thus, all indexes are generated at this time, and physical boundaries cannot be changed until the file is reorganized. If reorganizations, such as unload and reload, are to be infrequent, and if numerous changes to the file are anticipated, the user should consider the type and distribution of these changes. If the changes are expected to add a large number of items to the file or to alter the distribution because of a larger number of insertions in a relatively few strings of the file, careful attention must be given to overflow provisions. If changes are to be primarily updating of items, overflow need not be a major consideration.

TYPES OF OVERFLOW

Three types of overflow are provided, and each offers advantages in particular situations. Only general overflow is required.

Imbedded overflow is judiciously used when a relative uniform distribution of additions occurs. However, an uneven distribution of additions may cause inefficient use of disk area.

Cylinder overflow is used to provide a fixed number of tracks for items overflowing any of the strings on that cylinder. Use of the overflow area by all of the strings on a cylinder lessens the effect of a large number of additions occurring at some point on the cylinder. However, processing time may be increased. For example, if the cylinder overflow is three times as long as a string, the time required to directly retrieve an item in the cylinder overflow area will be three times as long (on the average) as the time required to retrieve an item in a string when cylinder overflow is full.

General overflow is used primarily as a safety valve. Processing time is commonly lengthened, but if few additions to the file are anticipated, and if the time required to retrieve items is not critical, general overflow may suffice. Whenever general overflow is entered, Logical I/O C sets indicators in the file's communication area for possible interrogation by the user.

Optimization

An indexed sequential file can be designed to optimize access time, to optimize use of the disk area, or to achieve a compromise between these two considerations.

OPTIMIZING ACCESS TIME

An indexed sequential file can be accessed both sequentially and directly. The user may choose to optimize operations using one of these types of access and to ignore the other. Normally, however, both sequential access time and direct access time can be optimized simultaneously.

Increasing block length is the primary means of reducing access time. Files should be allocated with the largest possible block length consistent with the memory requirements of all programs using the file. Both sequential and direct access times are reduced because master, cylinder, and string indexes require fewer blocks.

Careful determination of the relative sizes of the string index and the string is vital. The sum of the number of blocks in the string index and the number of blocks per string should be as small as possible.

Both direct access and sequential access times can be optimized by the following procedure. Values which must be known are identified by the following symbols.

K = characters per key
BL = block length
T = tracks per cylinder
OT = overflow tracks per cylinder

1. Refer to Table C-1 or Table C-2 to determine optimum values for number of records per track (RT) and number of records per block (RB). Recall that block length cannot exceed one track, i. e., $RB \leq RT$.

2. Compute the number of blocks per cylinder (BC) as follows.

$$BC = \frac{RT \times (T - OT)}{RB} \quad \text{Ignore any remainder.}$$

3. Compute the number of items per block in the string index (ISI) as follows.

$$ISI = \frac{BL}{2K + 8} \quad \text{Ignore any remainder.}$$

4. Compute a tentative value for the optimum number of blocks per string (BS) as follows.

$$BS = \sqrt{\frac{BC}{ISI}} \quad \begin{array}{l} \text{Retain any remainder.} \\ \text{Minimum value of BS is 1.} \end{array}$$

5. Choose the two integers bracketing this value of BS and perform the following computations for each integer value of BS.

- a. Compute the number of strings per cylinder (SC) as follows.

$$SC = \frac{BC}{BS + 1/ISI} \quad \text{Ignore any remainder.}$$

- b. Compute the direct access time criterion (DATC) as follows.

$$DATC = \frac{SC}{ISI} + BS + 8 \quad \text{Retain any remainder.}$$

6. Choose the integer value for BS which has resulted in the smaller value for DATC. This is the optimum value of blocks per string (BS). For later computations, retain the value of SC corresponding to this value of BS.

Example — Type 258 or 259 Disk Pack Drives:

Assume the following values:

IL = 121 characters per item
 BL = 605 characters per block
 K = 16 characters per key
 T = 10 per cylinder
 OT = 0 tracks per cylinder overflow

To compute the string size, locate the values $RT = 7$ and $RB = 1$ from Table C-1. Then,

$$BC = \frac{7 \times (10 - 0)}{1} = 70$$

$$ISI = \frac{605}{(2 \times 16) + 8} = 15 \text{ (remainder dropped)}$$

$$BS = \sqrt{\frac{70}{15}} = 2.14 \text{ (tentative value)}$$

The two possible integer values for BS are, therefore, 2 and 3.

Compute the trial values of SC and DATC.

If BS = 2, then

$$SC = \frac{70}{2 + 1/15} = 33 \text{ (remainder dropped)}$$

$$DATC = \frac{33}{15} + 2 + 8 = 12.20$$

If BS = 3, then

$$SC = \frac{70}{3 + 1/15} = 22 \text{ (remainder dropped)}$$

$$DATC = \frac{22}{15} + 3 + 8 = 12.5$$

Two blocks per string (BS = 2) yields the smaller value for DATC, although the difference is small (2.17%).

OPTIMIZING STORAGE CAPACITY

If the user desires to maximize use of the disk area, he may find it necessary to accept a block length (BL) that is smaller than the maximum block length that can be accommodated by the application and a string size (BS) that is larger than the optimum value for direct access time. Preferable values increase the number of blocks per cylinder and reduce the amount of unused space at the end of the string index and the amount of unused space at the end of a cylinder. These preferable values are a complex function of the record length that is best for a given block length and the relationship between key length and block length.

Assume that the item size is fixed and that there is a known maximum block length that can be handled. The following procedure can be used to determine the optimum block length (BL) and string size (BS) for maximum use of disk area. Values which must be known are identified by the following symbols.

- IL = item length
- K = characters per key
- MBL = maximum block length
- T = tracks per cylinder
- OT = overflow tracks per cylinder
- IBD = number of item positions per string to be imbedded while loading the file

1. Find the values of ISI and BC that correspond to the possible values for the blocking factor (IB) by performing the computations described in steps a, b, and c, below.

To find the possible values for IB, first calculate the maximum possible items per block (MIB) as follows:

$$MIB = \frac{MBL}{IL} \quad \text{Ignore any remainder.}$$

Then IB can assume the values MIB, MIB-1, MIB-2, etc. It may be necessary to find values of ISI and BC that correspond to small blocking factors.

- a. Refer to Table C-1 or C-2 to determine optimum values for number of records per track (RT) and number of records per block (RB).
- b. Compute the number of blocks per cylinder (BC) as follows:

$$BC = \frac{RT \times (T - OT)}{RB} \quad \text{Ignore any remainder.}$$

- c. Compute the number of items per block in the string index (ISI) as follows:

$$BL = IB \times IL$$

$$ISI = \frac{BL}{2K + 8} \quad \text{Ignore any remainder.}$$

2. For each of the preceding values of IB, BC, and ISI, choose integral values for BS starting with one. Perform the following calculations for each value of BS. It may be necessary to perform this calculation with quite large values of BS, as illustrated in the examples below.

- a. Compute the number of strings per cylinder (SC) as follows:

$$SC = \frac{BC}{BS + 1/ISI} \quad \text{Ignore any remainder.}$$

- b. Compute the number of items per loaded string (IS) as follows:

$$IS = (IB \times BS) - IBD$$

- c. Compute the number of items per loaded cylinder (IC) as follows:

$$IC = IS \times SC$$

3. Choose the pair of values for IB and BS that results in the largest value of IC.

Example 1 — Type 258 or 259 Disk Pack Drives:

Assume the following values:

IL = 121 characters per item
 K = 21 characters per key
 MBL = 605 characters per block (maximum)
 T = 10 tracks per cylinder
 OT = 0 tracks for cylinder overflow
 IBD = 0 item positions imbedded

The optimum point for storage capacity is as follows.

IB = 5
 BS = 23
 IC = 345

Note that in this example, the optimum storage capacity occurs at the maximum blocking factor, thus optimizing sequential access time. However, direct access time is more than twice as long when these values are accepted as when the following values are accepted:

IB = 5 BS = 4 IC = 340

Note that the corresponding loss in storage capacity is only 1.5 percent.

Example 2 — Type 258 or 259 Disk Pack Drives:

Assume the following values:

IL = 50 characters per item
 K = 21 characters per key
 MBL = 550 characters per block (maximum)
 T = 10 tracks per cylinder
 OT = 0 tracks for cylinder overflow
 IBD = 0 item positions imbedded

The optimum point for storage capacity as determined from Table C-5 is as follows.

IB = 9
 BS = 89
 IC = 801

However, direct access time is extremely long.

COMPROMISING BETWEEN ACCESS TIME AND STORAGE CAPACITY

An indexed sequential file can be designed to achieve a compromise between optimum access time and optimum use of storage capacity. The relative importance of these two factors must be determined for the particular application. The following procedures are applicable.

1. For many pairs of values of IB and BS, where IB varies from 1 through MIB and BS varies from 1 to BC - 1, compute the following quantities:
 BL = block length, which is a measure of sequential access time;
 DATC = direct access time criterion; and
 IC = items per cylinder, a measure of space utilization.
2. Choose the pair of values of IB and BS that is the preferable compromise among the three quantities BL, DATC, and IC.

Example — Type 258 or 259 Disk Pack Drive:

Assume the following values:

IL = 50 characters per item
 K = 21 characters per key
 MBL = 550 characters per block (maximum)
 T = 10 tracks per cylinder
 OT = 0 overflow tracks per cylinder
 IBD = 0 imbedded items per cylinder

Table C-5 lists values of BL, DATC, and IC for selected pairs of values of IB and BS. The number of different points that may have to be considered in determining the preferable compromise is indicated.

The optimum point for direct access is Case A; this is also an optimum point for sequential access. The optimum point for storage capacity is Case K; however, this point gives a direct access time that is seven times as long as Case A.

Table C-5. Example - Optimization for an Indexed Sequential File

Case	IB	BS	BL	DATC	IC
A	11	2	550	13.04	726
B	11	3	550	13.06	726
C	11	4	550	13.55	748
D	10	2	500	13.80	760
E	10	3	500	13.50	750
F	10	6	500	15.30	780
G	10	79	500	87.10	790
H	9	3	450	14.11	756
I	9	5	450	14.89	765
J	9	8	450	17.22	792
K	9	89	450	97.11	801
L	8	3	400	15.00	768
M	8	4	400	15.00	768
N	8	7	400	16.75	784
P	8	33	400	41.38	792
R	7	3	350	16.00	735
S	7	4	350	15.71	728
T	7	5	350	16.00	735
U	7	9	350	18.71	756
V	7	109	350	117.14	763

Certain points in Table C-5 are less desirable in all three values than some other point. For example, Case B as compared to Case A; Case R as compared to Case C; etc. Also, Cases K and G are characterized by access times that are much too long.

When these points are eliminated, the list is reduced to the cases shown in Table C-6. The optimum points show a direct tradeoff between direct access time and storage capacity: the shorter the direct access time, the smaller the storage capacity.

Table C-6. Example - Summary of Optimum Points

Case	IB	BS	BL	DATC	IC
J	9	8	450	17.22	792
N	8	7	400	16.75	784
F	10	6	500	15.30	780
L	8	3	400	15.00	768
M	8	4	400	15.00	768
I	9	5	450	14.89	765
D	10	2	500	13.80	760
E	10	3	500	13.50	750
A	11	2	550	13.04	726

Allocation

At least three units of allocation are required for an indexed sequential file. Proper placement of these units optimizes the performance achieved in processing the file. Placement of the master/cylinder index, the general overflow, and the data on separate volumes is preferable but frequently infeasible. (See method 1 on page 2-10 under "Allocation.") If only one volume more than those required for data is available, the master/cylinder index and general overflow should be placed together on that volume, assigned to relative volume 0. (See method 4 on page 2-10.) If all units must be placed on one volume they should be kept close together to minimize head travel. A unit of allocation cannot begin on the cylinder on which the preceding unit ends. Procedures for calculating the number of cylinders required for data and the number of tracks required for the master/cylinder index are described in the following paragraphs.

DATA CYLINDERS REQUIRED

To compute the number of data cylinders required for an indexed sequential file, values for blocking factor (IB) and string size (BS) must be determined. Values which must be known are identified by the symbols that follow.

IL = characters per item
 K = characters per key
 MBL = maximum block length
 T = tracks per cylinder
 OT = tracks per cylinder overflow
 IBD = number of imbedded item positions per loaded string.
 I = total number of items to be loaded onto the file (i. e., active items to be presented to the load function).

1. Choose a pair of values for IB and BS according to the type of optimization desired.
2. Compute block length (BL) as follows:

$$BL = IB \times IL$$
3. Refer to Table C-1 or Table C-2 to determine optimum values for number of records per track (RT) and number of records per block (RB).
4. Compute the number of blocks per cylinder (BC) as follows:

$$BC = \frac{RT \times (T - OT)}{RB} \quad \text{Ignore any remainder.}$$
5. Compute the number of items per block in the string index (ISI) as follows:

$$ISI = \frac{BL}{2K + 8} \quad \text{Ignore any remainder.}$$
6. Compute the number of strings per cylinder (SC) as follows:

$$SC = \frac{BC}{BS + 1/ISI} \quad \text{Ignore any remainder.}$$

7. Compute the number of items per loaded string (IS) as follows:

$$IS = (IB \times BS) - IBD$$

8. Compute the number of items per loaded cylinder (IC) as follows:

$$IC = IS \times SC$$

9. Compute the number of cylinders required for the prime data area (C) as follows:

$$C = \frac{I}{IC} \quad \text{Round the quotient up to the next higher integer.}$$

TRACKS REQUIRED FOR MASTER/CYLINDER INDEX

The values computed as described in the preceding paragraph can be used to compute the number of tracks required for the master/cylinder index. The following procedures are applicable.

1. Compute the number of items per block in the cylinder index (IBC) as follows:

$$IBC = \frac{BL}{K + 8} \quad \text{Ignore any remainder.}$$

2. Compute the number of blocks required for the cylinder index (BCI) as follows:

$$BCI = \frac{C}{IBC} \quad \text{Round up to the next higher integer.}$$

3. Compute the number of blocks required for the master index (BMI) as follows:

$$BMI = \frac{BCI}{IBC} \quad \text{Round up to the next higher integer.}$$

4. Compute the number of tracks required for the master/cylinder index (TMC) as follows:

$$TMC = \frac{(BMI + BCI) RB}{RT} \quad \text{Round up to the next higher integer.}$$

Example - Computing Units of Allocation

Computation of the units of allocation for the master/cylinder index, general overflow, and one data unit for an indexed sequential file on a Type 258 or 259 Disk Pack Drive follows.

Assume the following values.

- IL = 121 characters per item
- I = 10,000 items
- K = 16 characters per key
- MBL = 605 characters per block (maximum)
- T = 10 tracks per cylinder
- OT = 0 tracks for cylinder overflow

Data Cylinders Required (C)

1. Assume that the following values are chosen to optimize direct access time:
 - IB = 5 items per block
 - BS = 2 blocks per string
2. $BL = (5) (121) = 605$
3. The following values can be obtained from Table C-1:
 - RT = 7
 - RB = 1
4. $BC = \frac{7 \times (10 - 0)}{1} = 70$
5. $ISI = \frac{605}{2(16) + 8} = 15$ (remainder dropped)
6. $SC = \frac{70}{2 + 1/15} = 33$
7. Assume that approximately 10 percent of the data area is to be reserved via imbedded overflow. Then $IBD = 1$, since the number of item positions per string is $(BS) (IB) = 2 \times 5 = 10$. Thus, $IS = (5 \times 2) - 1 = 9$.
8. $IC = 9 \times 33 = 297$
9. $C = \frac{10,000}{297} = 34$ cylinders. (33.67 expressed as next higher integer)

Tracks Required for Master/Cylinder Index

1. $IBC = \frac{605}{24} = 25$ (remainder dropped)
2. $BCI = \frac{34}{25} = 2$ (1.36 expressed as next higher integer)
3. $BMI = \frac{2}{25} = 1$ (0.08 expressed as next higher integer)
4. $TMC = \frac{(1 + 2) 1}{7} = 1$ track. (0.43 expressed as next higher integer)

Thus, to allocate this file, a unit of allocation consisting of one track is required for the master/cylinder index; a second unit is required for general overflow; and a third unit of 34 cylinders is required for data. The units of allocation for this file can be specified in the units statement of the allocate function as follows (see pages 4-17 through 4-19).

CARD NUMBER	Y	M	A	P	R	LOCATION	OPERATION CODE	OPERANDS						
1	2	3	4	5	6	7	8	14	15	20	21	60	61	80
1								MC INDEX = (FROM = (0, 9), TO = (0, 9)),						
2								OVERFLOW = (FROM = (1, 0), TO = (1, 9)),						
3								FROM = (2, 0), TO = (35, 9).						
4														

APPENDIX D

PHYSICAL I/O C

The Physical I/O C program serves as the interface between the user program and the mass storage device, eliminating the need for the programmer to refer to the device directly. In most cases, the programmer uses Logical I/O C (described in Section III), which in turn, refers to Physical I/O C whenever access to the mass storage device is required.

Physical I/O C consists of a set of macro routines which provides a simple means of processing data stored on mass storage devices. These routines fall into four categories:

1. Input/output control,
2. Communication area,
3. Communication area service, and
4. Action.

To retrieve an area of the storage device, the programmer issues an action macro call which refers to the relevant communication area and links to the input/output control routine. The input/output control routine, in turn, initiates the required action according to the current contents of the communication area.

NOTE: The use of Physical I/O C in place of Logical I/O C removes the user from the Data Management Subsystem rules and conventions. All such users should be careful to follow the data management conventions if it is desired to use the same volume with Logical I/O C, File Support C, Mass Storage Sort C, etc.

USE OF PHYSICAL I/O C

Physical I/O C provides the programmer with the capability of initiating several basic processing functions for a mass storage device. These functions are: read, write, wait, restore, verify, and seek.

Each time Physical I/O C is entered for one of these functions, it performs all the operations required to initiate the requested function. In addition, each time a read or write function is specified, the previous data transfer is checked for successful completion. If any type of error is recognized, Physical I/O C attempts to correct the error whenever correction is feasible. If any error is uncorrectable, an exit to the programmer's coding occurs with an indication of the type of error.

Read Action

Any type of read instruction¹ may be requested by the programmer, but it is his responsibility to set the limits of the data transfer. He can do this either by specifying a nonextended read or by setting a record mark in main memory.

Write Action

Any type of write instruction¹, except those specifying format writing, may be requested by the programmer. However, it is the programmer's responsibility to limit the data transfer as in the read action above, when necessary.

Wait Action

The wait action is requested when the programmer requires the assurance that the last data transfer initiated for a particular file has been completed successfully. Normal return to the programmer's coding occurs only upon successful completion, but that does not guarantee that the last data transfer for any other file has been checked. An error exit occurs only if an uncorrectable error condition was encountered in the file in question.

Restore Action

The restore action is requested when the programmer desires to restore the device to its initial state. The initial state for a device is defined in the hardware bulletin Disk Pack Drives and Control (Order Number 514).

Verify Action

The verify action is requested when the programmer desires to verify that the area last written is error free.

Seek Action

The seek action is requested when the programmer wishes to position the read/write heads of a disk device to a specified cylinder. When the seek action is initiated, the read/write heads are positioned to the cylinder currently specified in the CYL field of the communication area. When the seek action is requested, the programmer should specify parameters which load the desired information into CYL or other relevant fields of the communication area.

The seek action does not cause the disk control to become busy; therefore, a seek can be performed on a non-busy disk device while the disk control unit for that device is busy with

¹The Extended Multiprogramming and 8-bit Transfer (Feature 1120, 1121, or 1118) is not supported by Physical I/O C. Therefore, the 8-bit transfer capability cannot be used.

activities of another disk device connected to it. The seek action, however, does not provide any error checking of a previously initiated read or write action.

DETAILED DESCRIPTION OF PHYSICAL I/O C MACRO ROUTINES

Four types of macro routines are available to simplify the task of utilizing the mass storage device. Each type is described in the following paragraphs.

Control Macro Routine (MPIOC)

The control macro routine, MPIOC (mass storage physical input/output control), is activated each time there is a request for one of the actions previously described. One MPIOC can control actions on both class A and class C devices used concurrently. The device types are shown below:

Class	Device Type
A	258, 259, 273, 259A, 259B
B	155
C	261, 262

If at any time during processing an error is detected, an error analysis and correction routine is entered. This routine determines the type of error and, if possible, attempts to correct the error. If the error is corrected, processing continues. Otherwise, control is returned to the programmer's coding at a location specified by him, and an indication of the type of error is made available.

Communication Area Macro Routine (MPCA)

The communication area macro routine, MPCA (mass storage physical communication area), provides an area which contains a series of fields in which all information pertinent to a particular file is stored. This area allows MPIOC to specialize itself to accomplish peripheral actions on various devices and files. These fields are available to the programmer and to the control macro routine (MPIOC). The programmer can change or interrogate the values of these fields as required. The control routine uses the current values of these fields in initiating its mass storage instructions. It also maintains values in the communication area of interest to the programmer.

A separate communication area must be used for each set of data being processed concurrently. For example, Logical I/O C requests the generation of a communication area for each separate file specified.

Communication Area Service Macro Calls (MLCA and MUCA)

The macro routines to service the communication area, MLCA and MUCA, are used to

load information into certain fields of the communication area and to interrogate certain fields. Using these macro routines, the programmer can alter the contents of certain fields of the communication area without knowing the structure of the area.

Action Macro Routines

There is a separate action macro routine for each of the actions previously described. Each of these effects an entrance to the proper routine of the control macro routine. The communication area referred to by the action macro routine provides the information necessary to perform the requested function. The read, write, and seek action macro routines also provide certain communication area service functions, as described previously.

LANGUAGE ELEMENTS OF PHYSICAL I/O C

For a program to use Physical I/O C, the programmer need only insert a library call in the program where required for the appropriate macro routine. Control macro calls and communication area macro calls are placed in a subroutine portion of the program. Action macro calls and communication area service macro calls go into the program's main line coding. Each such call causes the related coding to be inserted at that point. The language for calling the various routines is defined in the following paragraphs.

Control Macro Call (MPIOC)

The example below illustrates the method of coding the MPIOC macro call.

CARD NUMBER	TYPE	LOCATION	OPERATION CODE	OPERANDS
1 2 3 4 5 6 7 8 14 15 20 21 62 63 80				
1	C	tag	MPIOC	parameter 01, parameter 02, parameter 03,
2	L			parameter 04, ...

Parameters of MPIOC Macro Call

Table D-1 lists the parameters of the MPIOC macro call. The function of some MPIOC parameters is to include or eliminate certain subroutines or instructions. Thus, a given specialization of MPIOC occupies as little memory as possible.

Table D-1. Parameters of MPIOC Macro Call

Number	Name	Value	Function	Comments
00	Base	Anytag	Tag is equated with the lowest memory location occupied by MPIOC.	

APPENDIX D. PHYSICAL I/O C

Table D-1 (cont). Parameters of MPIOC Macro Call

Number	Name	Value	Function	Comments
01	Unique character	See "Comments" column for valid characters	A single character appended to each tag used by this specialization of MPIOC. Used to achieve tag uniqueness when more than one specialization of MPIOC is being used in a single program and to ensure that a user's tag does not duplicate any tag in MPIOC.	Required for each MPIOC. Valid characters are shown below. Key Punch Print Symbol (+, 8, 5) % (+, 8, 6) ■ (-, 8, 3) \$ (-, 8, 5) " (0, 1) / (0, 8, 5) C _R
02	Peripheral control address	Δ	Honeywell-recommended address assignment for the mass storage control (04g).	This parameter has no meaning if parameter 04 is assigned the value M. When the peripheral address assignment is specified, the I/O bit, (high-order bit) must be zero.
		xx (octal)	Peripheral address assignment to which the mass storage control applicable to this MPIOC is attached.	
03	Write verification	Δ	Automatic verification coding is not included.	The verify action macro call is valid only when V is the specified value of this parameter.
		V	The verify routine is to be included in this MPIOC.	
04	Control of more than one peripheral control	Δ	The peripheral address assignment and RWC configuration are specialized at assembly time as specified by parameter 2 and 5 and cannot be changed without reassembly.	
		M	The peripheral address assignment and RWC configuration are specialized at execution time whenever the control unit number in the current MPCA differs from that used in the last operation performed by this MPIOC.	
05	RWC definition	Δ	Automatic specialization is performed at assembly time, according to the value of parameter 02. When parameter 02 is blank or is equal to or less than 07, 56 is generated. When parameter 02 is greater than 07, 76 is generated.	This parameter has no significance if parameter 04 is assigned the value M.

Table D-1 (cont). Parameters of MPIOC Macro Call

Number	Name	Value	Function	Comments
05 (cont)		xx (octal)	Read/write channel to be used for all data transfers. Cannot be changed without reassembly.	When this entry is used, it must be a variant which includes read/write channel 3 of the I/O sector corresponding to the PUC assignment specified by parameter 02.
06	Seek indicator	Δ	The seek action macro routine is not to be included in this specialization of MPIOC and cannot be called by the program.	
		SEEK	The seek action macro call is to be a valid call for the program.	
07	LOKDEV See "Device Protection" later in this appendix.	Δ	The LOKDEV action is not called.	
		LOKDEV	The LOKDEV action is called.	

Communication Area Macro Call (MPCA)

This macro call sets up a communication area in a specific format which MPIOC refers to as required. If an optional parameter is omitted from the call, an area is still reserved for the corresponding field. These fields may be specialized at execution time when an MLCA macro call is used. See Section III of this manual for a description of the MLCA macro call.

The following example illustrates the method of coding the MPCA macro call.

CARD NUMBER	1	2	3	4	5	6	7	8	14	15	20	21	OPERANDS		
													42	43	
1															
2															
3															

Note: Handwritten entry on card 1: L tag MPCA parameter 01, ..., parameter n.

Parameters of the MPCA Macro Call

Table D-2 lists the parameters of the MPCA macro call.

APPENDIX D. PHYSICAL I/O C

Table D-2. Parameters of MPCA Macro Call

Number	Name	Value	Function	Comments
00	Prefix	1, 2, or 3 characters	Tag prefix for all MPCA entries. All action macro calls refer to this prefix in their calling sequences.	Must be specified.
01	Suffix	x	Same as character specified as parameter 01 of MPIOC.	Must be specified.

)

•

)

B

B

)

—

Table D-2 (cont). Parameters of MPCA Macro Call

Number	Name	Value	Function	Comments
02	Buffer address	Tag	Location of the leftmost character of an area to or from which data is transferred. Buffer must be as long as longest block of data transferred. There must be three available character positions to right of this buffer when input data transfers are executed.	Must be specified.
03	Error exit	Tag	Address of a user-provided routine to which MPIOC branches in case of an uncorrected error condition.	Must be specified.
04	C3 Variant	Δ	The value of C3 is 04 (octal).	This parameter is normally left blank, since the programmer normally alters its field in the communication area with the MLCA macro routine.
		xx (octal)	Octal value defining the type of data transfer to be executed by MPIOC.	<p>Permissible values in octal are:</p> <ul style="list-style-type: none"> 04 = Load/unload address register, 02 = Search and read/write, 22 = Extended search and read/write, 03 = Search and read/write next, 23 = Extended search and read/write next, 00 = Read initial, 20 = Extended read initial, 01 = Read, and 21 = Extended read. <p>NOTE: When verification is desired, set the third bit from the left (in the 6-bit variant) to 1, e.g.,</p> <div style="display: flex; justify-content: space-between; align-items: center;"> <div style="border: 1px solid black; padding: 2px;">0 1 0 0 0 1</div> = Extended read </div> <div style="display: flex; justify-content: space-between; align-items: center; margin-top: 5px;"> <div style="border: 1px solid black; padding: 2px;">0 1 1 0 0 1</div> = Extended read and verify </div>

Table D-2 (cont). Parameters of MPCA Macro Call

Number	Name	Value	Function	Comments
05	Protection bits	Δ	The initial value is 00(octal).	Permissible values in octal are: 00 = Permit no writing, 02 = Permit writing to records that do not have A- or B-protection bits set in the record headers, 06 = Permit A-file writing, 12 = Permit B-file writing, and 16 = Permit A- and B-file writing. NOTE: When any type of writing is permitted, the writing specified by 02 is also permitted.
		xx(octal)	An octal value indicating the protection bits to be loaded into the control unit address register.	
06	RWC configuration	Δ	Automatic specialization is performed at assembly time, according to the value of parameter 07. If parameter 07 is blank or is less than or equal to 07, 56 is generated. Otherwise, 76 is generated.	This parameter is significant only if parameter 04 of the MPIOC macro call is assigned the value M (more than one PCU is specified for MPIOC).
		xx(octal)	Read/write channel to be used for data transfers.	
07	Peripheral control address	Δ	Honeywell-recommended address assignment for mass storage control (04_8).	This parameter has significance only if parameter 04 of the MPIOC macro call is assigned the value M. This single-character field can be modified at execution time whenever another PCU is to be addressed. When the peripheral address assignment is specified, the I/O bit (high-order bit) must be zero.
		xx(octal)	Peripheral address assignment to which the mass storage control to be used for data transfers is attached.	

Communication Area Service Macro Calls (MLCA and MUCA)

The method of coding the MLCA and MUCA macro calls is the same as that illustrated in Section III. The mnemonic designators to be used with MLCA and MUCA for Physical I/O C are not the same as those for Logical I/O C. The mnemonic designators for MLCA and MUCA macro calls when using Physical I/O C are listed in Table D-3.

Table D-3. Mnemonic Designators for MLCA and MUCA

Mnemonic	Description
CYL	This designator refers to a 4-character field containing the device, pack, and cylinder number (in binary) for any future action related to the file table referred to by parameter 01. The leftmost character of this field must contain a word mark.
CAD	This designator refers to an 8-character field containing CYL as its high-order characters. The remaining four characters are the 2-character track and the 2-character record number (in binary). Note that CAD and CYL can overlay each other. The final value of this field is loaded into the control unit address register by MPIOC whenever required. The leftmost character of this field must contain a word mark.
PRT	This designator refers to the right end of a 10-character field whose high-order eight characters are defined by CAD. The character to the right of CAD must be 0. The tenth character corresponds to parameter 05 of MPCA. MPIOC will load the address register of the control unit with the current value of these ten characters. Note that the ten characters include CAD, which in turn, includes CYL.
TRW	This designator refers to a single character corresponding to parameter 04 of MPCA (C3 variant).
AAD	This designator corresponds to the buffer address, as defined for parameter 02 of the MPCA macro call.
EAD	This designator refers to the entrance address to a user's error routine. Refer to parameter 03 of the MPCA macro call.
RWC	This designator refers to the read/write channel(s) being used for data transfer and is significant only if parameter 04 of the MPIOC macro call is M. It must be a variant which includes read/write channel 3 if I/O sector 0 is being used or channel 6 if I/O sector 1 is being used.
CPU	This designator refers to the peripheral address assignment and is significant only if parameter 04 of the MPIOC macro call is M. The I/O bit (high-order bit) must be zero.
LAD*	This designator refers to an 8-character field designating the address of the last record involved in the previous data transfer initiated via the related communication area. Its format is "DPCCTTRR" (device, pack, cylinder, track, and record numbers).
ECT*	This designator refers to a 1-character field containing a binary count of the number of rereads or rewrites executed by MPIOC in attempting to correct read and write errors detected in executions for the designated MPCA table.

Table D-3 (cont). Mnemonic Designators for MLCA and MUCA

Mnemonic	Description
ERI*	This designator refers to a 1-character field containing an indication of the type of uncorrectable error condition which was last encountered in executions for the designated MPCA table.
EDF*	This designator refers to a 14-character field containing the contents of the control unit address register at the time the last error condition was detected in executions for the designated MPCA file table. The rightmost four characters are unspecified.
LRT*	This is a field containing the address of the last return to the user from the initiation of an action referring to the designated MPCA table.
*These designators can be used only with MUCA.	

Action Macro Calls

Action macro calls provide the programmer with the capability of initiating the following actions: read, write, wait, restore, verify, and seek. Each time one of these action macro calls is entered, the corresponding function is initiated or executed by MPIOC. MPIOC refers to the indicated communication area as required for these actions.

READ ACTION MACRO CALL

The call for this macro routine is inserted in the programmer's coding whenever a data transfer from the mass storage device to main memory is required. MLCA functions can also be accomplished by this action. The example below illustrates the method of coding the read action macro call.

CARD NUMBER	Y	M	OPERATION CODE	OPERANDS
1 2 3 4 5 6 7 8	9	10	11 12 13 14 15 16 17 18 19 20 21	22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64
1	C	anyTag	READ	FL1, CHGRWC, RWC,
2	L			MPRT, PRT

In this example, the value of parameter 01 is FL1. This corresponds to the value assigned to parameter 00 of the MPCA macro call. The remaining parameters shown in this example, 02, 03, 04, and 05, correspond to the similarly numbered parameters of the MLCA macro call described in Section III. This macro call can have up to 63 parameters.

WRITE ACTION MACRO CALL

The call for this macro routine is inserted in the programmer's coding whenever a data transfer from main memory to mass storage is required. MLCA functions can also be accomplished by this function. The method of coding the write action macro call is the same as that shown for coding the read action macro call, except that the word "WRITE" is placed in the operation code field.

WAIT ACTION MACRO CALL

The call for this macro routine is inserted in the programmer's coding whenever the programmer desires to wait for the completion of, and check for errors on, the last function for the MPCA specified. If the normal return to the programmer's coding occurs after this macro routine is entered, the programmer is guaranteed that the data transfer is completed successfully. The following example illustrates the coding for the wait action macro call.

CARD NUMBER		TYPE	LOCATION	OPERATION CODE	OPERANDS																									
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	
			L	anytag	WAIT	FL2,																								

In this example, the value of parameter 01, i.e., FL2, is the value assigned parameter 00 of the MPCA macro call.

RESTORE ACTION MACRO CALL

The call for this macro routine is inserted in the programmer's coding whenever he desires to restore the device associated with a specific MPCA to its initial state. The method of coding the restore action macro call is the same as that shown above for the wait action macro call, except that "RESTOR" is written in the operation code field.

VERIFY ACTION MACRO CALL

The call for this macro routine is inserted in the programmer's coding whenever he desires to initiate a read (in the write verify mode) from the last area written onto the mass storage device. The write macro and the verify macro calls may be separated by main line coding if a mass storage device on the same control unit is not referred to in this interval. The method of coding the verify action macro call is the same as that shown above for coding the wait action macro call except that "VERIFY" is written in the operation code field.

SEEK ACTION MACRO CALL

The call for this action macro routine is inserted in the programmer's coding whenever he wishes to position the read/write heads of a disk device to a specified cylinder.

Parameter 01 of the seek action macro call identifies which communication area Physical I/O C is to refer to for execution of this macro routine. The remaining parameters correspond to similarly numbered parameter pairs of an MLCA macro call (02 and 03, 04 and 05, etc.) and are used to load the communication area fields with appropriate values for the seek routine.

An example of seek coding follows:

CARD NUMBER		TYPE	LOCATION	OPERATION CODE	OPERANDS																									
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	
			L	anytag	SEEK	file-tag,	cyl-tag,	cyl,																						

APPENDIX D. PHYSICAL I/O C

In the above example, file-tag corresponds to parameter 00 of the MPCA macro call, and cyl-tag is a user-supplied tag for a 4-character field in memory containing data to be loaded into the CYL field of the MPCA communication area. CYL is one of the MLCA mnemonics listed in Table D-3.

If the programmer prefers to write a separate MLCA call prior to the seek call, or to seek to the cylinder currently specified in MPCA, only parameter 01 of the seek call need be specified. The following two examples, then, perform identical functions:

CARD NUMBER	OPERATION	LOCATION	OPERATION CODE	OPERANDS
1	L	anytag	SEEK	file-tag, cad-tag, CAD
2				
3	*			or
4	L	anytag	MLCA	file-tag, cad-tag, CAD
5	L		SEEK	file-tag
6				
7				
8				
9				
10				

In the two examples above, the full mass storage address for an item to be read or written is loaded into the communication area. CAD is the mnemonic for a communication area field comprising CYL plus four characters which specify track and record. Parameter 02, cad-tag, would therefore be the address of an 8-character field in memory containing the information to be loaded into CAD.

PROGRAMMER'S PREPARATION INFORMATION FOR PHYSICAL I/O C

The following paragraphs describe pertinent programming considerations for Physical I/O C.

Address Mode

The coding generated as a result of a Physical I/O C macro call is in either 3- or 4-character address mode, corresponding to the mode of the user's program.

Read/Write Channel Utilization

Two data transfer rates are applicable to mass storage devices. When a Type 258, 259, or 273 Disk Pack Drive, or a Type 261 or Type 262 Disk File is used, data transfer rates accomplished by interlocking at least 1-1/2 channels (such as 1A and 3 or 4A and 6) are required. When a Type 155, 259A, or 259B Disk Pack Drive is used, a single interlocked channel suffices.

In the absence of any other directive, Logical I/O C utilizes channels 2 and 3 or channels 5 and 6 (depending upon the I/O sector) when operating with Type 258, 259, or 273 Disk Pack Drives; alternatively, it utilizes channel 3 or 6 when operating with Type 155, 259A, or 259B Disk Pack Drives.

The user can change this assumption by setting parameter 04 of the MPIOC macro call to M and by specifying RWC as the communication area field designator of an MLCA macro call (see Table D-3). This assignment can also be made by means of the MPCA macro call.

Special Considerations for Specifying Parameters

USE OF INDEX REGISTERS

Physical I/O C uses index registers X5 and X6 but restores them to their original values before returning to the user's program, regardless of whether the return is in the normal mode or is an uncorrectable error exit. Index registers X5 and X6, therefore, may be employed by the user-written program and can be used in conjunction with the MLCA and MUCA macro functions.

PERIPHERAL ADDRESS ASSIGNMENT AND RWC CONFIGURATION CONSIDERATIONS

Physical I/O C can control one or more peripheral control units as described below.

Fixed Peripheral Address Assignment

When parameter 04 of the MPIOC is blank, the peripheral control unit number and read/write channel configuration are specialized at assembly time and cannot be changed without re-assembly. The peripheral control unit number and read/write channel configuration are specified by parameters 2 and 5 of the MPIOC macro call.

Variable Peripheral Address Assignment

When more than one peripheral control unit is to be controlled, or when the particular unit to be controlled is determined at execution time, parameter 04 of the MPIOC macro call must contain M.

MPIOC is specialized at execution time according to the peripheral control unit number and read/write channel configuration values specified in the MPCA macro call. This specialization occurs only in the peripheral control unit number specified in the current MPCA macro call differs from that specified for the preceding operation. Within one program, all MPCA's containing the same peripheral control unit number must be able to operate with the read/write channel configuration specified for any other MPCA containing that number. Specialization also occurs when MPIOC is initially entered during execution.

The peripheral control unit number and read/write channel configuration can be set at assembly time through use of parameters 6 and 7 of MPCA. The values can be set or changed during execution through use of MLCA. However, the read/write channel configuration can be changed only when the peripheral control unit number specified by the MPCA is also changed. The RWC value must include channel 3 for I/O sector 0 or channel 6 for I/O sector 1. Permissible values are listed below.

I/O Sector 0	I/O Sector 1
53	73
54	74
55	75
56	76

When the peripheral address assignment is variable, the sector bits of the RWC value are updated in the specialized MPIOC instructions to correspond to the I/O sector specified by the peripheral control unit number. Therefore, the read/write channel configuration need not be set or modified to correspond to the sector specified by the peripheral control unit number.

Considerations for MPIOC Parameter Specification

SUFFIX CHARACTER

The suffix character specified in parameter 01 is used as the last character of all tags in MPIOC. For this reason, any tag written in the program by the user should not end with this character. When a program contains more than one MPIOC, each call must be assigned an individual suffix character.

PERIPHERAL ADDRESS ASSIGNMENT

When the peripheral address assignment is specified by the user, it must be expressed as an output assignment (00 through 07 or 20 through 27). When the user intends to change the peripheral address assignment during the execution of the program, parameter 04 of MPIOC must be assigned the value M. For considerations related to variable peripheral address assignments, refer to "Variable Peripheral Address Assignment" in this appendix.

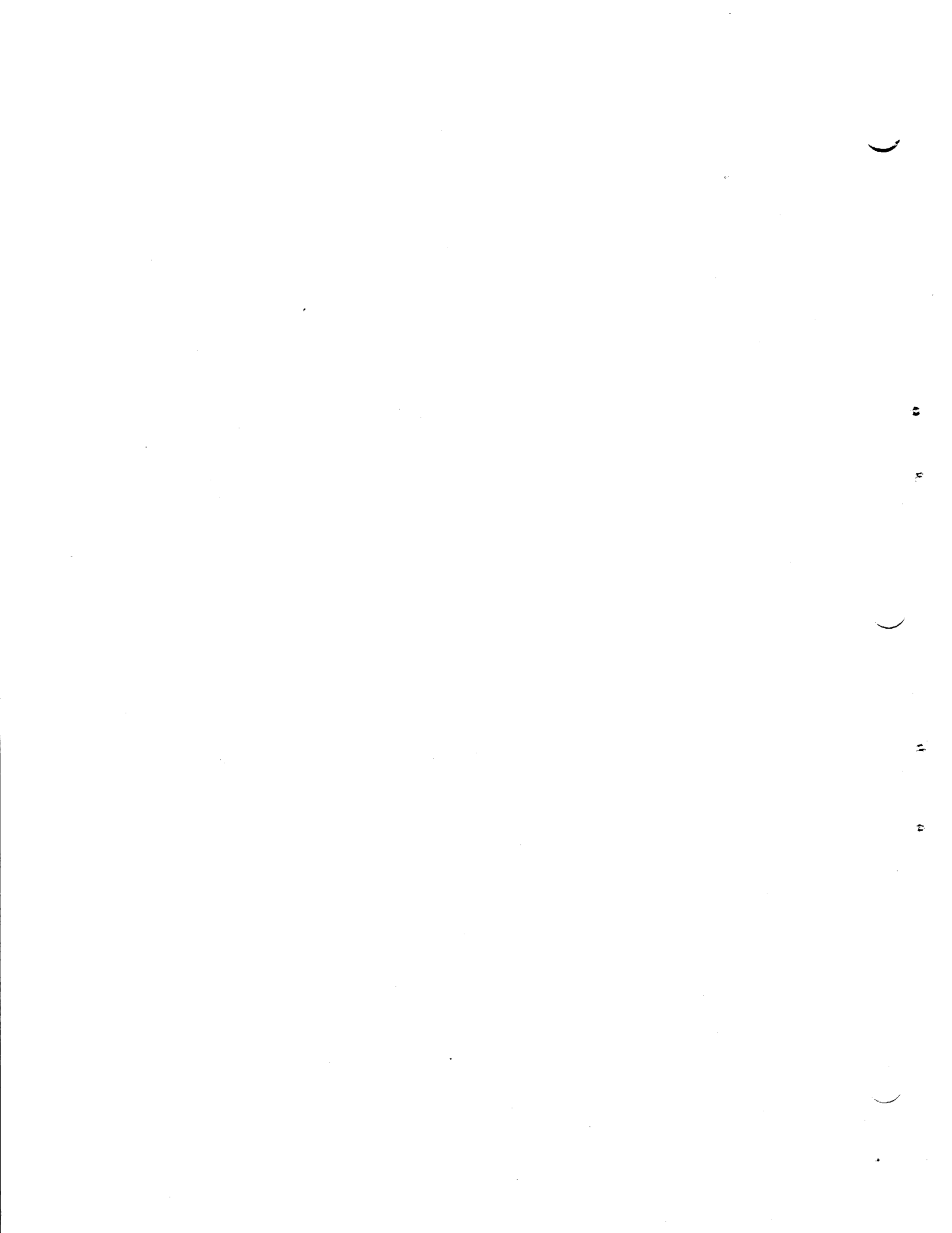
DEVICE PROTECTION

If feature 079 is available to the disk peripheral control unit and parameter 07 is specified as LOKDEV, it is possible to power down a device through MPIOC. This feature permits protection of a volume after the user has finished processing it.

Considerations for MPCA Parameter Specification

An area in memory which is specialized in a fixed format is provided by MPCA for communication. The area is specialized according to the values assigned to the MPCA

parameters. A separate MPCA macro call must be in the program for each set of data (e. g., a file) for which separate communication values are to be established. Each Physical I/O C action macro call is linked by the file prefix parameter of MCA to a specific MPCA and by the MPIOC suffix character to the MPIOC. The MPIOC performs the action required by the Physical I/O C action macro call, obtaining the required values from the related MPCA.



FILE PREFIX

The file prefix is established by assigning one, two, or three characters to parameter 00 of MPCA and is used to identify the tags used by the MPCA from all other tags in the program. When the program contains more than one MPCA, each file prefix value must be different. Also, each character used as a file prefix must be valid in a tag, according to the rules of Mass Storage Easycoder Assembler C.

SUFFIX OF RELATED MPIOC

Because it is possible for a program to contain more than one MPIOC, the value assigned the suffix parameter (MPCA parameter 01) must be the same character assigned as the value of MPIOC parameter 01 to which the MPCA is linked. This ensures that the Physical I/O C action macro calls link to the appropriately specialized MPIOC.

BUFFER ADDRESS (AAD)

An address constant (DSA) is generated by the buffer address parameter (MPCA parameter 02). Except for indexing with index registers X5 and X6, any form of addressing can be used. Also, the value of the address constant may be changed prior to execution of any Physical I/O C action macro routine except the verify macro routine.

USER'S UNCORRECTABLE ERROR ROUTINE ENTRANCE (EAD)

Parameter 03 of MPCA contains the symbolic address (tag) of the user-supplied uncorrectable error routine. Any form of addressing can be used, except for indexed addressing using index registers X5 and X6. This symbolic address can be changed at any time.

TYPE OF READ OR WRITE (TRW)

When a read or a write action is initiated, MPIOC interrogates the value assigned to parameter 04 of MPCA to determine the type of read or write desired. This value is changed frequently during the execution of the program through the read, write, or MLCA macro routine. Frequently, therefore, no value is assigned to parameter 04. The value that can be assigned to parameter 04 is a 2-digit octal character. These characters are listed below, along with the type of read or write action that will be performed.

Value of Parameter 04	Type of Read or Write Performed
Δ or 04	Load/unload address register.
02	Search and read/write.
22	Extended search and read/write.
03	Search and read/write next.
23	Extended search and read/write next.
00	Read initial.
20	Extended read initial.
01	Read.
21	Extended read.

NOTE: The values 02, 22, and 03 may be specified as 12, 33, 13 if a verify operation is desired. The use of these values also requires that a write action is being performed. This is not to be confused with the Physical I/O C verify action macro routine which automatically initiates write verify operations.

CONTROL UNIT CURRENT ADDRESS AND STATUS

In each MPCA, a 10-character field (word-marked at its left end) contains the current peripheral address assignment and the status of the control unit for the actions being issued through that MPCA. The field is not an exact image of the control unit address register, particularly when more than one MPCA is included in the program. The field does, however, indicate the status of one set of operations being issued through that MPCA.

The field is shown in Figure D-1 and the three mnemonics (CYL, CAD, or PRT) can be included in a read, write, or MLCA macro call to change the contents of the applicable portion of the field. The contents of each character in the field are in binary form.

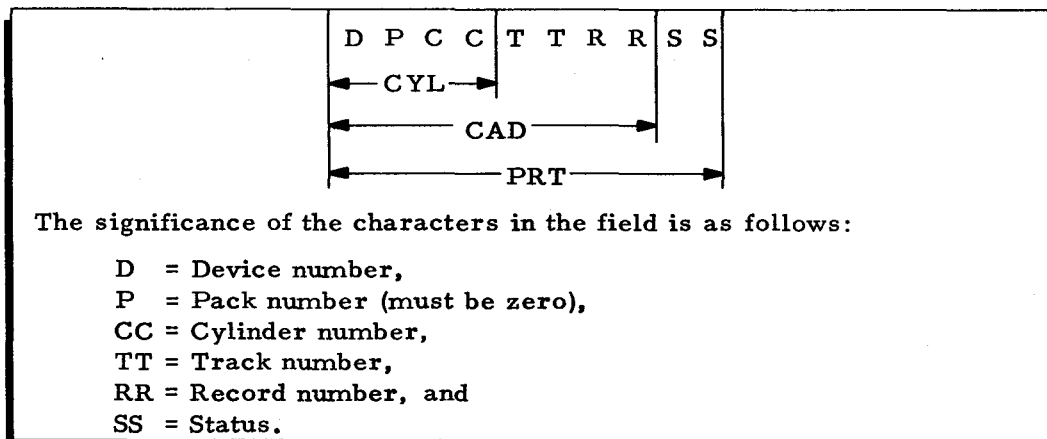


Figure D-1. MPCA Control Unit Current Address and Status Field

The rightmost two characters (SS) of the 10-character field in Figure D-1 represent the following: the leftmost eight bits represent the type of error condition, and the rightmost four bits represent the type of file protection. Whenever the mnemonic BRT is used in an MLCA macro call to load this 10-character field, the leftmost character position of the status portion of the field must be zero. The file protection character can be set to any of the following octal characters:

- 00 = Permit no writing,
- 02 = Permit writing when there is no A- or B-file protection,
- 06 = Permit A-file writing,
- 12 = Permit B-file writing, and
- 16 = Permit A- and B-file writing.

Notice that if a file does not have A- or B-file protection, the 06 or 12 values, respectively, also permit writing. A write operation cannot be performed if the corresponding switch is not set at the control unit. Also, the data write permit bit must be 1 to allow any type of write operation.

Considerations for Action Macro Routines

Normally, return from an active macro routine is to the location following the generated coding. When an uncorrectable error is caused by the action, however, return is made at the address specified by the EAD (parameter 03) field of the associated MPCA. In the action macro call, parameter 00 (written in the location field on the coding form) may be used as a tag referring to the first (high-order) character of the generated coding. Parameter 01 of the action macro call, starting in column 21 on the coding form, must be assigned the same value as the unique prefix specified as parameter 00 of the related MPCA.

READ ACTION MACRO ROUTINE

After ensuring the error-free completion of the previous read or write function, the read macro routine initiates a data transfer from mass storage to main memory and may perform MICA macro functions. The type of read operation performed depends on the TRW field (see page D-15) of the associated MPCA.

WRITE ACTION MACRO ROUTINE

After ensuring error-free completion of the previous read or write function, the write macro routine operates like the read routine, except that the data transfer is in the opposite direction, i. e., from main memory to mass storage. Note, however, that if the verify bit is set in the TRW, no data transfer occurs; only the readability of the data is checked.

VERIFY ACTION MACRO ROUTINE

The verify macro routine is used to read the data recorded by the last write action. There is no data transfer associated with the verify operation, but this is not the same as specifying a read action with the TRW bit set to verify in the MPCA. When desired, the verify macro call must be issued after a write is to be checked and before any other action call is issued.

WAIT ACTION MACRO ROUTINE

Whenever the programmer intends to check the last action initiated by MPIOC (via the appropriate MPCA) for error-free completion, he issues a wait action macro call. If the MPCA indicated in the call was not the last MPCA to be active, there is no guarantee that any other action initiated by MPIOC is completed successfully. If the action is completed successfully, a normal return to the user is made. If the action is not completed successfully, the error detection and correction action is performed automatically. If the error is corrected, a normal return to the user is made at this time; if not, the user's uncorrectable error routine is entered.

RESTORE ACTION MACRO ROUTINE

Whenever the programmer intends to restore a device to its initial state, he issues a restore action macro call. When the MPIOC is entered from a restore action macro call, the last action for this MPCA is checked for error-free completion, a restore operation is initiated, and a normal return to the user's coding is made.

LOKDEV ACTION MACRO ROUTINE

To power down a device, a LOKDEV action macro call is issued. The following events occur when the MPIOC is entered from a LOKDEV action macro call: (1) the last action for this MPCA is checked for error-free completion, (2) a LOKDEV action is initiated, and (3) a normal return to the user's coding is made.

HANDLING TRACK LINKING RECORDS

A read or read initial operation (extended or not extended) or a search and read/write operation which encounters a track linking record (TLR) handles the TLR as a normal data record.

A search and read/write operation (extended and/or next) which encounters a TLR continues the operation on the record specified by the TLR. The search for the record specified by the TLR is done on the current cylinder, and the instruction is not completed if the TLR links to another cylinder. A subsequent READ, WRITE, or VERIFY operation to any file or a WAIT operation to a specific file detects the instruction-incomplete condition. MPIOC then seeks the cylinder specified by the TLR and performs the original read/write operation (starting at the

address specified by the TLR) prior to continuing with the subsequent action macro routine. The above description indicates why a block may not span cylinders.

User's Uncorrectable Error Routine

When MPIOC returns control to the user at the address specified in EAD (see page D-15) of a specific MPCA table, the user's program must direct the actions to be taken for the condition which has occurred. The MPCA involved in the condition contains information which enables the user's error routine to determine which path to follow at this point. The user may return to MPIOC to : (1) try again to execute the instruction sequence which precipitated the error, (2) bypass the offending instruction and continue with the requested action, or (3) issue a new action macro call.

The following paragraphs describe the types of conditions which may occur and the various corrective actions (both programmed and manual) which may be attempted.

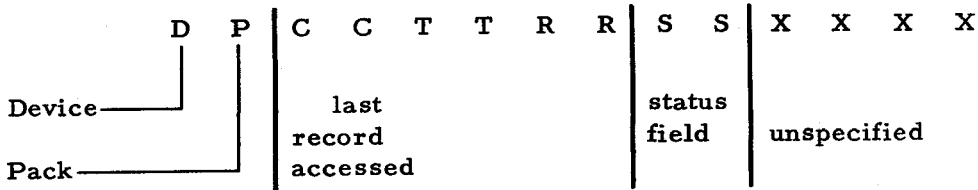
ERROR TYPE INDICATOR (ERI)

The designator "ERI" refers to a single-character field which indicates the type of error last encountered while processing with this MPCA. The possible values and their meanings are listed below.

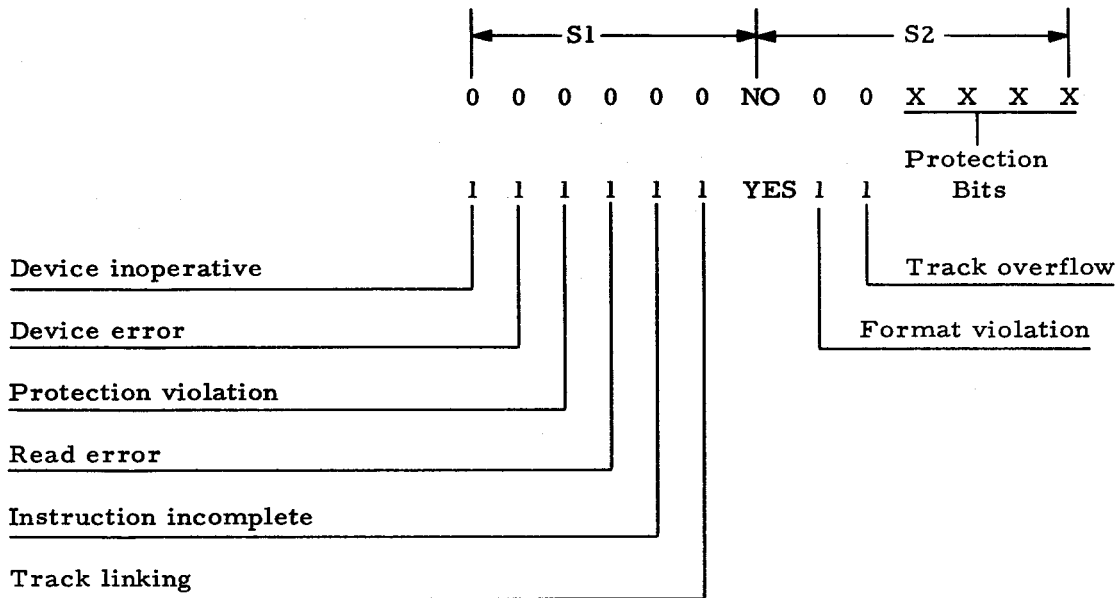
Octal Value	Meaning
00	No errors.
01	Device inoperative.
02	Protection violation.
03	Device error (after five attempts at positioning).
04	Formatting error (format violation or track overflow).
05	Addressed record not located (after five attempts).
06	Uncorrectable read error; data transfer was completed (after ten rereads).
07	Same as 06 above, except that data transfer is not completed (header may contain a read error).
10	Automatic verification failed (after ten reverify attempts).
11	Track-linking record was read into core or rewritten from core (this is not necessarily an error).
12	Read error in track-linking record while attempting to link. Contents of current CCTTRR are invalid (after ten attempts to reread).

ADDRESS REGISTER CONTENTS AT TIME OF ERROR EXIT (EDF)

The designator EDF refers to a 14-character field that reflects the contents of the control unit address register at the time the last error condition was recognized by MPIOC for this MPCA. The format of this field is shown below.



The format of the status field (SS) is shown below.



RE-EXECUTION OF CORRECTION PROCEDURE

At the time of return to the user's error routine, the B-address register contains the address at which MPIOC may be reentered for further re-execution of the instruction sequence in error. This return is especially valuable if ERI contains one of the values 01 through 04, since these types of errors may possibly be corrected by manual action. Suggested manual operations to be performed in these cases are listed in Table D-4.

Table D-4. Corrective Action for User's Error Routine

Value of ERI	Condition	Suggested Cause/Manual Action
01	Device inoperable	1. Device may not be turned on. 2. Device may be cycled down. Stop the device, if necessary, and cycle it up.
02	Protection violation	1. Manual protection switches may be set incorrectly. Set protection switches as directed by operating instructions of program.
03	Device error	Clear the error condition at the device.
04	Format violation or track overflow	Same as "Protection Violation" above.
NOTE: All other conditions, except that represented by an ERI value of octal 11, may possibly be corrected by re-execution. Note that MPIOC has already made a number of attempts at correction.		

BYPASS ERROR CONDITION

If the user desires to accept the last execution as correct, he may re-enter MPIOC to bypass the error condition. This may have some value in certain cases (e.g., a read error) but is dangerous in others (e.g., a seek error).

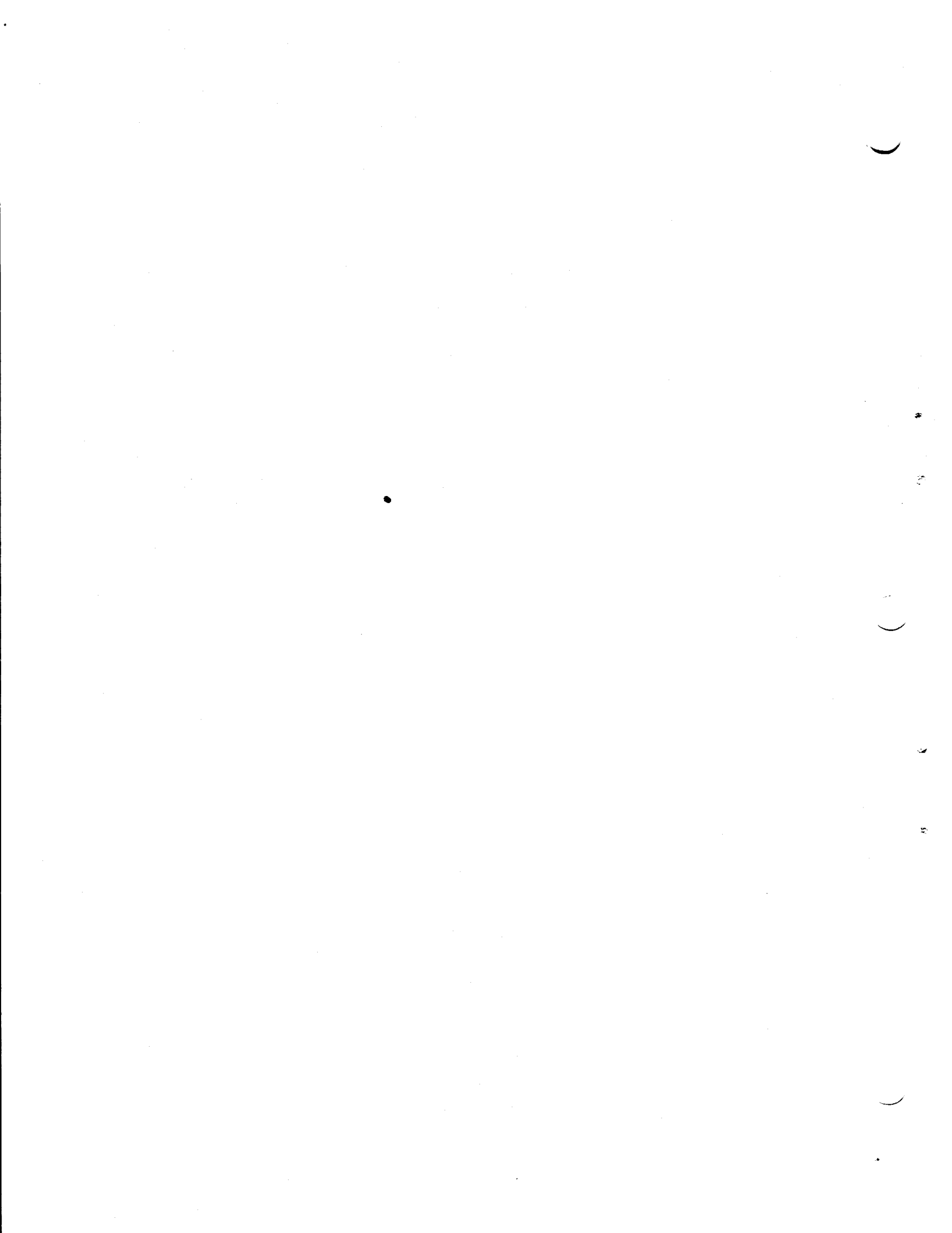
The user must add one (plus the current address mode) to the value in the B-address register at the time of the uncorrectable exit to compute the MPIOC bypass re-entry address. For example, if Physical I/O C is being executed in 3-character addressing mode, four is added to the B-address register value in order to bypass the error condition.

ISSUE NEW ACTION MACRO CALL

If the user decides to discontinue the previous path of processing, he may issue any desired action macro call. Physical I/O C permits the user to issue any action call; however, the type of error encountered may make certain actions inadvisable.

OPERATING PROCEDURES FOR PHYSICAL I/O C

Physical I/O C does not communicate directly with the operator in the case of unusual conditions. Rather, it exits to the user with a code indicating a specific condition. It is the responsibility of the user of Physical I/O C to provide operator notification of conditions which cannot be corrected under program control.



APPENDIX E

RANDOMIZING TECHNIQUES

RANDOMIZING ADDRESSING

Randomizing is the process of transforming the key of an item into a valid address. This usually consists of obtaining a relative bucket address (e.g., the 204th bucket of the file), which is then converted by Logical I/O C into a valid mass storage bucket address consisting of cylinder, track, and record designation). This enables the user to retain his present item numbering system, and yet have full online processing capability.

There are many randomizing methods, each one being somewhat better suited to a particular application than another. All have the same objective — to produce a valid address for each item from its item key (control field) in such a way that the items are evenly distributed. Depending on the randomizing technique employed, storage utilization can reach between 80 and 90 percent efficiency. Generally speaking, a technique that achieves high mass storage utilization generates more synonyms (duplicate bucket address) and thus increases access time. Therefore, the randomizing technique chosen depends on the relative importance of mass storage utilization versus access time. In addition, the input/output buffer size of the file is important and affects the access time. For instance, if several items are blocked together, then synonyms would have a less serious effect on access time.

Once a randomizing technique has been selected for possible use, the technique should be evaluated with a sample selection of actual item keys. This evaluation should provide information on the efficiency of mass storage utilization, the frequency and distribution of synonyms, the processing time required for the calculation, and how evenly the generated addresses are distributed. The results enable the user to select the technique most suited to his particular requirements and data pattern.

The following paragraphs outline a few of the most commonly used key transformation methods. They have the advantage of being economical in processing time and core memory requirements. There are many possible variations of these methods, in addition to far more complicated methods not covered in this manual.

Prime Number Division

Division of the item key field by a prime number (a number divisible only by itself or one) is a widely accepted method of transforming a key into a mass storage address. The prime

number divisor should be slightly less than the number of buckets allocated to the data area of the file, but it should be as large as possible. The larger the prime number divisor, the smaller the chance of generating synonyms.

The prime number division method consists of dividing the item key by the selected prime number divisor, discarding the quotient, and using the remainder as the basis for the address.

Example - Prime Number Division

A file of 5,000 items on a Type 259 Disk Pack Drive stores five items per bucket, one bucket per track, with item keys ranging from 000,000,000 to 999,999,999. Space is allocated to this file for 1,000 buckets. The file is to start on cylinder 50, track 0. The prime number divisor chosen is 997, which leaves three buckets unused in the 1,000 allocated.

Suppose that 777,775,925 is the key of the item to be processed. Then:

$$\frac{777,775,925}{997} = 780,116, \text{ with a remainder of } 268.$$

Thus, this item is to be placed in the 268th bucket from the beginning of the file.

Using this relative bucket address, Logical I/O C then computes that the actual address is 26 cylinders and 8 tracks after the starting location of the file (cylinder 50, track 0), which in this case would be cylinder 76, track 8.

It has been assumed in this example that a unit of allocation is made up of a whole cylinder (ten tracks) and that there are no cylinder overflow tracks. However, if overflow exists, Logical I/O C makes the necessary adjustments.

In cases where purely alphabetic or mixed alphabetic/numeric item keys are concerned, the item key can be treated as a binary field to be binarily divided by the binary form of the prime number. The final calculations are also performed binarily so that the relative address is produced in a usable form.

Table E-1 is a list of prime numbers. It is divided into two parts: part A contains every third prime between 2 and 2,939, and part B contains every fifth prime between 2,953 and 8,039.

Square Enfold and Extract

In this randomizing technique, the item key field is squared, the result is split in half, and the two halves are added together. Then the required number of digits needed for an address is extracted from the middle of the result. Normally, the two low-order characters are ignored and the extraction is made from the third low-order character and above.

APPENDIX E. RANDOMIZING TECHNIQUES

Table E-1. Prime Numbers

A. Primes (Every Third Prime 2-2, 939)													
5	137	307	487	677	883	1093	1303	1543	1753	1999	2239	2447	2707
13	151	317	503	701	911	1109	1321	1559	1783	2017	2267	2473	2719
23	167	347	523	727	937	1129	1367	1579	1801	2039	2281	2531	2741
37	181	359	557	733	953	1163	1399	1601	1831	2069	2297	2549	2767
47	197	379	571	761	977	1187	1427	1613	1867	2087	2333	2579	2791
61	223	397	593	787	997	1213	1439	1627	1877	2111	2347	2609	2803
73	233	419	607	811	1019	1229	1453	1663	1901	2131	2371	2633	2837
89	251	433	619	827	1033	1249	1481	1693	1931	2143	2383	2659	2857
103	269	449	643	853	1051	1279	1489	1709	1951	2179	2399	2677	2887
113	281	463	659	863	1069	1291	1511	1733	1987	2213	2423	2689	2909

B. Additional Primes (Every 5th Prime - 2, 953-8, 039)													
2957	3343	3697	4073	4457	4861	5233	5641	6029	6373	6803	7211	7603	
3001	3373	3733	4111	4507	4909	5281	5659	6067	6427	6841	7243	7649	
3041	3433	3779	4153	4547	4943	5333	5701	6101	6481	6883	7307	7691	
3083	3467	3823	4211	4591	4973	5393	5743	6143	6551	6947	7349	7727	
3137	3517	3863	4241	4639	5009	5419	5801	6199	6577	6971	7417	7789	
3187	3541	3911	4271	4663	5051	5449	5839	6229	6637	7001	7477	7841	
3221	3581	3931	4327	4721	5099	5501	5861	6271	6679	7043	7507	7879	
3259	3617	4001	4363	4759	5147	5527	5897	6311	6709	7109	7541	7927	
3313	3659	4021	4421	4799	5189	5573	5953	6343	6763	7159	7573	7963	

Example 1 - Square Enfold and Extract

The following values are assumed in this example: a file of 10,000 items; item keys of nine digits; ten items per bucket; one bucket per track. Therefore, there are 1,000 buckets. A total of 100 cylinders is required (exclusive of overflow).

Control number: 493, 725, 816

Squared: 243, 765, 181, 384, 865, 856

Enfolded: 243, 765, 181
384, 865, 856

628, 631, 037

Extracted result: 310 relative bucket address.

Logical I/O C computes: $\frac{310}{10}$ = Cylinder 31, track 0 (added to the starting address of the file)

Since the field extracted ranges over some power of 10 (depending on the number of digits extracted), and unless the number of buckets available is some whole multiple of 10, the result of this calculation is not suitable. The extracted number can be compressed by multiplying the result by a percentage. If a 3-digit field is extracted, this gives a range of 1,000 numbers which may be multiplied by 70 percent if there are only 700 buckets available.

Example 2 - Square Enfold and Extract

If the file consisted of 600 buckets instead of 1,000 buckets with the same control number range:

Control number:	569, 183, 582
Squared:	323, 969, 950, 018, 350, 724
Enfolded:	323, 969, 950
	<u>018, 350, 724</u>
	342, <u>320, 674</u>
Extracted result:	206

X206 = 123.60 (.60 discarded)

This gives a relative bucket address of 123.

Radix Conversion

When this method is applied to purely numeric item keys, each decimal digit is interpreted as if it were a radix-11 digit instead of the actual radix-10 and is then converted back to radix-10. When applied to alphabetic or alphanumeric item keys, each character is treated as two octal digits. Each digit is then interpreted as if it were a radix-9 digit instead of the actual radix-8 and is then converted back to radix-8. In this case, the numbers can range only from 0 to 7, whereas in the numeric case, the numbers could range from 0 to 9.

The normal procedure after the radix conversion, is to truncate the result by discarding high-order digits until a field of the desired length is obtained. Note that compression of the resultant number can be done by multiplying it by a percentage, as in the square enfold and extract method.

Example 1 - Radix Conversion (from Radix-11 to Radix-10)

The item key in this example is 301, 283 and is treated as radix-11 as follows:

$$\begin{aligned}
 \text{Radix-11} &= (3 \times 11^5) + (0 \times 11^4) + (1 \times 11^3) + (2 \times 11^2) + (8 \times 11^1) + (3 \times 11^0) \\
 &= 483,153 + 0 + 1,331 + 242 + 88 + 3 \\
 &= 484,817, \text{ leaving } 817 \text{ as the truncated address} \\
 &\quad \text{(cylinder 81, track 7 on Type 259).}
 \end{aligned}$$

NOTE: The arithmetic is done in the radix being converted to, i. e., radix-10.

Radix conversion is a better method than truncation alone, since it tends to disperse troublesome runs of keys differing in the numeric case by some power of 10 (e.g., 02309 and 12309) or in the alphanumeric case by some power of 8 (e.g., 0247₈ and 1247₈). The main advantage of this method is the simplicity of calculation. The conversion from radix-11 to radix-10, or from radix-9 to radix-8 may be accomplished without multiplication. It can be done simply by a series of decimal additions and shifts, or binary additions and shifts. Radix conversion does tend, however, to produce more synonyms than prime number division.

Example 2 - Radix Conversion (from Radix-11 to Radix-10, Using Addition and Shifting)

Using the formula in example 1 above, the item key 301, 283 can be reduced to:

$$(((3 \times 11 + 0) \times 11 + 1) \times 11 + 2) \times 11 + 8) \times 11 + 3)$$

$$\begin{array}{r}
 3 \\
 + 30 \\
 + \underline{0} \\
 33 \\
 + 330 \\
 + \underline{1} \\
 364 \\
 + 3640 \\
 + \underline{2} \\
 4006 \\
 + 40060 \\
 + \underline{8} \\
 \\
 44074 \\
 + 440740 \\
 + \underline{3} \\
 484817 \quad 10
 \end{array}$$

Example 3 - Radix Conversion (from Radix-9 to Radix-8)

The item key 247₈ in this example is treated as radix-9:

$$(2 \times 9 + 4) \times 9 + 7$$

$$\begin{array}{r}
 2 \\
 20 \\
 + \underline{4} \\
 26 \\
 260 \\
 + \underline{7} \\
 315_8
 \end{array}$$

NOTE: The arithmetic in this example is done in radix-8.

Nonnumeric Item Keys

Where item key fields comprise purely alphabetic or special characters or a mixture of alphanumeric characters, one method of randomizing is to treat the field as a binary number

and perform binary arithmetic on it. This has the advantage of retaining zone bits and, therefore, avoiding unnecessary synonyms.

Another method of randomizing is to consider each 6-bit character as two octal digits which are extracted by means of binary addition and extraction to form two decimal digits in the range 0 to 7. The resultant key is then manipulated by decimal arithmetic according to the particular method employed. This method is useful where binary arithmetic is impracticable, but it does result in doubling the length of the item keys.

Example - Nonnumeric Item Keys

<u>Key</u>	<u>Decimalized Octal</u>
810	100100
8246Y2-951-7	100204067002401105014007
8415RST	10040105516263
84X113-177-16 (13 characters)	10046701010340010707400106 (26 digits)

NOTE: One common misconception in converting alphabetic keys is that the zone bits should be dropped before converting. This, however, immediately produces three groups of synonyms:

G, H, I P, Q, R X, Y, Z

Zone suppression (with the consequent advantages of decimal arithmetic) may be an acceptable method, however, for cases where the item keys are largely numeric with only a few nonnumeric characters scattered through them.

Multifield Keys

Up to this point, only item keys with a single field have been considered where the range of key numbers is broadly sequential, no matter whether continuous or not. It is, however, fairly common for item keys to be divided into definite fields where each field has a range which is quite independent of the other fields. To treat such keys as a single field may be wasteful unless each field has a maximum value such that the entire key forms a continuous series, as follows:

00	0	000	00
to	to	to	to
77	9	999	99

Apart from cases like the above example, it is generally desirable to manipulate each field independently. Otherwise, an unduly large number of synonyms would be generated. Unless a weighting factor is applied to the most significant keys, most of the methods previously discussed would generate too many synonymous addresses. One such technique has been developed by Honeywell. It has the advantage of being readily adaptable to other multifield key applications, and it generates no synonyms.

APPENDIX E. RANDOMIZING TECHNIQUES

Suppose, for example, that the file contains 30,000 items and that each item contains 100 characters which are to be blocked six items to a block, one block per bucket, on a Type 259 Disk Pack Drive. Each item has a 6 digit item key comprising three fields:

<u>Division No.</u>	<u>Page No.</u>	<u>Line No.</u>
(1 char.)	(3 chars.)	(2 chars.)
1 to 5	1 to 120	1 to 50

The calculations are as follows:

1. Subtract 1 from division number;
2. Multiply the resulting division number by the sum of the maximum number of pages multiplied by the maximum number of lines, i. e., $120 \times 50 = 6,000$, and place the result in final result X;
3. Subtract 1 from page number;
4. Multiply the resulting page number by the maximum number of lines, i. e., 50, and add the result to final result X;
5. Subtract 1 from line number;
6. Add the resulting line number (1) into final result X; and
7. Divide final result X by the number of items per bucket. The quotient is the relative bucket number.

This method converts each 6-digit key field into a unique number in the range from 1 to 30,000. If the field numbers ranged from zero instead of one, the subtractions in stages 1, 3, and 5 would be omitted, since their only function is to convert each field to a range beginning with zero.

Example - Multifield Keys

In this example, the following values are assumed: division number = 5, page number = 120, line number = 50. The calculations are performed as follows:

1. $5-1 = 4$,
2. $4 \times 120 \times 50 = 24,000$,
3. $120-1 = 119$,
4. $119 \times 50 = 5,950 + 24,000 = 29,950$,
5. $50-1 = 49$,
6. $49 + 29,950 = 29,999$, and
7. $\frac{29,999}{6} = 4,999$, with a remainder of 5. The remainder is discarded, giving a relative bucket address of 4,999.

Frequency Analysis

This method consists of analyzing the keys of all the items in the file to determine the

APPENDIX E. RANDOMIZING TECHNIQUES

frequency that any digit appears in any one position of the item key. For each digit position of the item key, examine all the items to determine the number of times any one digit (0 through 9) appears. For example, if there were 16,045 items in the file, a 0 might occur in the fifth key position for 5,168 different items, a 1 might occur in the fifth key position for 5,138 different items, a 2 might occur in that position for 4,958 items, a 3 might occur for 281 items, and the numbers 4 through 9 might not occur in this position for any item. This count gives the actual distribution of digits occurring in each key position. If the distribution were perfectly even, each of the ten digits would occur the same number of times as any other digit; thus, each digit would occur 1/10th of the time. With 16,045 items, each digit should occur approximately 1,605 times in any one key position.

To determine the deviation from this ideal distribution, take the difference between the actual number of times a digit occurs in the key position and the ideal number of times it should occur (in this case, 1,605). Thus, if 0 actually occurs in the fifth key position of 5,168 different items, the deviation would be 5,168 minus 1,605 = 3,563. This is done for each digit that appears in that key position and then all the results are summed to find the total deviation for that key position. The total deviation could then be expressed as a percentage of the total number of items. The lower the sum is, the more even the distribution is. The pattern of distribution indicates which positions are best to use when truncating or extracting addresses from the item keys.

Example 1 - Frequency Analysis

16,045 items

Variance factor = 1,605, i. e., 10 percent of number of items.

Digit	Key Position Number						
	1	2	3	4	5	6	7
0	16045	0	0	1852	5168	1807	1738
1	0	0	4408	3147	5638	2120	1748
2	0	2198	3792	1174	4958	1745	1743
3	0	576	2231	2724	281	1684	1610
4	0	1195	2459	1194	0	1378	1617
5	0	12076	3155	1267	0	1647	1688
6	0	0	0	1243	0	1560	1606
7	0	0	0	1228	0	1329	1450
8	0	0	0	1227	0	1415	1411
9	0	0	0	989	0	1360	1434
Total Variance	28885	22133	16045	5821	21903	1961	1035
% file	180 Least even distribution	138	100	36	137	12	6 Most even distribution

APPENDIX E. RANDOMIZING TECHNIQUES

A method of utilizing a frequency analysis to obtain an address is to express each digit count in an item key field position as a percentage of the number of file items. A cumulative total is formed for each digit to which is added half of the actual percentage for that digit to give an adjusted constant for each digit in every item key position. The constants for every digit in an item key are accumulated and the total (excluding the whole number carry) is multiplied by the number of storage locations allocated. The whole-number product is then converted to a cylinder and track address in the normal manner.

Example 2 - Method of Using Frequency Analysis

File of 20,000 items. Storage allocated, 25,000 locations.

Key position 1 is illustrated.

Digit	Count	Percentage	Cumulative Total	Adjusted Constant
0	6,400	.32000	.00000	.16000
1	300	.01500	.32000	.32750
2	1,300	.06500	.33500	.36750
3	800	.04000	.40000	.42000
4	1,200	.06000	.44000	.47000
5	0	.00000	-	-
6	0	.00000	-	-
7	4,800	.24000	.50000	.62000
8	1,200	.06000	.74000	.77000
9	4,000	.20000	.80000	.90000

The above process is repeated for every key position, and a table of adjusted constants is built up as follows, illustrating just the constants required for item 13,689:

Digit	Item Key Position					
	1	2	3	4	5	
0						Value from previous table
1	.32750					
2						
3		.39875				
4						
5						
6			.59327			
7						
8				.83125		
9					.96250	

.32750
 .39875
 .59327
 .83125
 .96250
 .11327

$25,000 \times .11327 = 2,831.75000$

02831 = Relative bucket address.

APPENDIX E. RANDOMIZING TECHNIQUES

The table of adjusted constants has to be set up initially, but the actual key transformation can be done quickly. Such a table would have to be recalculated when sufficient changes have occurred to affect materially the frequency distribution. The table itself requires 50 locations for every item key field position, i. e., 250 locations for a 5-digit item.

APPENDIX F

MASS STORAGE FILE PROTECTION

FILE PROTECTION

The introduction of mass storage devices into data processing brings additional considerations into the area of data file protection. In magnetic tape processing, several methods of protection against inadvertent destruction have been in use for some time. With the Type 204B Magnetic Tape Units, a user may put any drive in protect by using a manual switch on that drive. He may also remove the plastic ring on the back of the tape itself. Finally, in common practice, each file is contained on a separate reel of tape. These three methods of protection are generally adequate. In addition, if a particular tape file is confidential, its owner (for example, a payroll department) can keep that reel in its own restricted area of storage. This guarantees that no unauthorized persons have access to this file.

On mass storage, however, it is common for more than one file to exist on a single volume. When this is true, the tape-oriented methods of protection are not adequate. To provide the user with maximum data protection, the Mod 1 (MSR) Operating System offers two types of protection: (1) a hardware/software protection against inadvertent data destruction and (2) a software protection against unauthorized access to a confidential file.

These two features are explained in detail below.

WRITE PROTECTION

There are four classes of write protection offered through a combination of hardware and software features. These classes are:

1. Format write protection,
2. Data write protection,
3. A-file write protection, and
4. B-file write protection.

Corresponding to these four classes of write protection are four hardware switches.

For example, to do any formatting, the FORMAT WRITE PERMIT switch must be ON. In terms of this operating system, formatting would occur during any run of Volume Preparation C or File Support C which is performing allocation.

Any program which is directing any form of writing (e. g. , an update, an assembly, or a sort) requires that the control unit to which the write is being directed must have the DATA WRITE PERMIT switch ON. In addition, if it is a user's program, parameter 31 in MCA of Logical I/O C must be 02 (i. e. , permit data write).

The use of these two switches is not optional. When formatting is in progress, the FORMAT WRITE switch must be in PERMIT. When writing is in progress, the DATA WRITE switch must be in PERMIT.

The use of A-file and B-file protection, however, is optional. For example, if there is a master file which may be written on only by a limited, well-defined number of programs, it may be desirable to give this file further protection. To illustrate, let us suppose that FILE-X is a payroll master file which may be updated by only one program. In addition to the payroll file, however, there may be, from time to time, one or two other files on the same volume. To protect FILE-X from inadvertent destruction, it is decided to give this file B-file protection.

When allocating FILE-X, the parameter PROT = B is used. If the file is being loaded by the File Support C load function, the PROT = B parameter must be used again. In addition, the B-FILE WRITE PERMIT switch and DATA WRITE PERMIT switch must both be ON during this load process.

The program written to update this file must include the value 12 in parameter 31 of Logical I/O C's MCA macro call. (B-FILE WRITE PERMIT switch and the DATA WRITE PERMIT switch must both be ON.)

The possible combinations of file write protection are:

<u>Combination of Protection</u>	<u>Switches in Permit When Writing</u>
NONE.	Data Write.
A-file.	Data Write and A-file.
B-file.	Data Write and B-file.
A-file and B-file.	Data Write, A-file, and B-file.

PASSWORD PROTECTION

In addition to guaranteeing that a file will not be improperly destroyed, it is often important to guarantee that a file is not read by unauthorized personnel. Thus, in the preceding example, it is important to be able to know that the other users of the volume containing FILE-X, the payroll master file, cannot open, read, or write to FILE-X. To effect this type of protection, this operating system provides the use of a password.

Thus, in the above example, a password of PAY66164 might be used. During allocation, the parameter PW = PAY66164 is entered. If using the load function, PW = PAY66164 is used again. Any program processing FILE-X must have as parameter 21 of Logical I/O C's MCA macro call a tag pointing to a field in memory which contains PAY66164.

For example:

C	MCA
C	21	PCWORD
	⋮	
PCWORD	DCW	@PAY66164@.

)

8

9

)

10

11

)

APPENDIX G
TERMINAL FILES

Card-image and print-image files (terminal files) can be placed temporarily on mass storage files to be punched or printed at a later time. These files can be created by means of a user routine executed with Logical I/O C. File Support C can be used to process either card-image or print-image files as a part of the unload function. (See "Unloading Print-Image Files to Printer" in Section IV.)

CREATION OF TERMINAL FILES

To create a terminal file, the user must perform the following actions.

1. Allocate the file as a sequential file or a partitioned sequential file.
2. Specify output processing by means of parameter 02 of the MIOC macro call.
3. Specify parameter 40 of the MCA macro call (volume directory exit).
4. When the volume directory exit is taken (exit code 01), modify character position 56 and possibly character position 57 of the *VOLDESCR* entry for the file to one of the following values:

Col. 56 Value

- | | |
|-----------------|--|
| 40 ₈ | = Print-image file with n control characters per item. Position 57 contains the number of control characters per item. |
| 42 ₈ | = Print-image file with n control characters per item and report numbers with form adjustment. Position 57 contains the number of control characters per item. |
| 41 _g | = For Mod 8 use only. Card-image file with no control characters and any number of characters per item. Column 57 is ignored. |
| other | = Standard data file. |

Characters 56 and 57 of the *VOLDESCR* item cannot be set by using File Support C. See Logical I/O C, Table 3-11, for information on own-coding exit 01.

Field APD of the communication area contains the address of the *VOLDESCR* item. See Table 3-8.

5. Place print or punch items in the file by means of PUT action macro calls.

CARD-IMAGE FILES

Any mass storage file, regardless of file organization, can be treated as a card-image file simply by specifying a card punch as the output device type.

Each item of the file is punched in a minimum number of 80-column cards; control characters are not recognized as such and are treated as part of the data item. For compatibility with the Mod 8 Operating System, a sequential or partitioned sequential file can be identified as a card-image file by placing a value of 41g in character position 56 of *VOLDESCR*.

PRINT-IMAGE FILES

A print-image file is identified by a value of 40g or 42g in field 18 (character position 56) of *VOLDESCR*, depending upon the number and use of control characters in each item. The file may be a sequential or partitioned sequential file.

If character position 56 contains 40g, the number of control character positions in each item (as specified in character position 57 of *VOLDESCR*) can be 01g, 04g, or 10g. The first character of the item must be the C3 variant of a PDT instruction to the Type 222 Printer to be used in printing the item. All items of the file are printed. Report number is not applicable, nor is form adjustment. The total number of characters in the item is given in character positions 2 and 3 of *VOLDESCR*. The formats of print-image items are illustrated below.

Character 57 = 01g		Character 57 = 04g		Character 57 = 10g	
Position	Use	Position	Use	Position	Use
1	Type 222 control	1	Type 222 control	1	Type 222 control
2-133	print image	2-4	unused	2 - 8	unused
		5-137	print image	9-141	print image
<p>NOTES: 1. These values apply when a 132-character printer is being used.</p> <p>2. Any additional characters beyond the indicated maximum values are ignored.</p>					

Character position 57 of *VOLDESCR* contains the number of control characters per item. The total number of characters per item is given in character positions 2 and 3 of *VOLDESCR*.

APPENDIX G. TERMINAL FILES

If character position 56 contains 428, two additional control characters are used for each item to specify a report number. Print form adjustment is also permissible. The format of an item is illustrated below.

Position	Use
1	Type 222 control character.
2-3	Report number (see "Report Number Parameter" in Section IV.)
4	<p>If the bit configuration of this field is xxxlxx, a halt occurs after printing of this item. Use of this option, in conjunction with File Support C, enables the operator to align forms.</p> <p>If the bit configuration of this field is xxx0xx, no halt occurs.</p>
5-8	Reserved (see note).
9-140	Print image.
141-144	Reserved (see note).
<p>NOTE: This field is reserved to provide compatibility with the Mod 8 Operating System. If compatibility with Mod 8 is not desired, this field can be deleted.</p>	

INDEX

AAD
 BUFFER ADDRESS (AAD), D-15

ACCESS
 CLOSING INDEXED SEQUENTIAL AND
 DIRECT ACCESS FILES, 3-10
 COMPRISING BETWEEN ACCESS TIME AND
 STORAGE CAPACITY, C-22
 CUMULATIVE LOADING OF A DIRECT
 ACCESS FILE, 2-25
 DIRECT ACCESS, 3-71
 DIRECT ACCESS ADDRESSING, 3-70
 DIRECT ACCESS FILE
 CONSIDERATIONS, C-10
 DIRECT ACCESS FILE
 ORGANIZATION, 2-20
 DIRECT ACCESS FILES, 5-57
 DIRECT ACCESS FILES AND KEYS, 2-24
 INSERTING ITEMS IN DIRECT ACCESS
 FILES, 3-21
 LOADING A DIRECT ACCESS FILE, 5-57
 OPENING DIRECT ACCESS FILES, 3-9
 OPTIMIZING ACCESS TIME, C-18
 REPLACING ITEMS IN DIRECT ACCESS
 FILES, 3-16
 RETRIEVING ITEMS IN DIRECT ACCESS
 FILES, 3-13
 SUMMARY OF MSGET MACRO FUNCTIONS
 FOR DIRECT ACCESS FILES, 3-15
 UNLOADING A DIRECT ACCESS
 FILE, 5-57

ACTION
 ACTION MACRO CALLS, 3-54, D-10
 ACTION MACRO CALLS FOR EACH FILE
 TYPE IN EACH PROCESSING MODE, 3-6
 ACTION MACRO CALLS (FOR PARTITIONED
 SEQUENTIAL FILES ONLY), 3-17
 ACTION MACRO PROCESSING
 FUNCTIONS, 3-4
 ACTION MACRO ROUTINES, 3-2, D-4
 CONSIDERATIONS FOR ACTION MACRO
 ROUTINES, D-17
 CORRECTIVE ACTION FOR USER'S ERROR
 ROUTINE, D-21
 ISSUE NEW ACTION MACRO CALL, D-21
 LOKDEV ACTION MACRO ROUTINE, D-18
 READ ACTION, D-2
 READ ACTION MACRO CALL, D-10
 READ ACTION MACRO ROUTINE, D-17
 RESTORE ACTION, D-2
 RESTORE ACTION MACRO CALL, D-11
 RESTORE ACTION MACRO ROUTINE, D-18
 SEEK ACTION, D-2
 SEEK ACTION MACRO CALL, D-11
 SUMMARY OF ACTION MACRO CALL
 CODING, 3-63
 VERIFY ACTION, D-2
 VERIFY ACTION MACRO CALL, D-11
 VERIFY ACTION MACRO ROUTINE, D-18
 WAIT ACTION, D-2
 WAIT ACTION MACRO CALL, D-11
 WAIT ACTION MACRO ROUTINE, D-18
 WRITE ACTION, D-2
 WRITE ACTION MACRO CALL, D-10
 WRITE ACTION MACRO ROUTINE, D-17

ADDITIONAL
 ADDITIONAL USABLE EQUIPMENT, 1-7

ADDITIONS
 FILE ADDITIONS, C-1

ADDRESS
 ADDRESS MODE, 3-69, D-12
 ADDRESS REGISTER CONTENTS AT TIME
 OF ERROR EXIT (EDF), D-20
 BUFFER ADDRESS (AAD), D-15
 CONTROL UNIT CURRENT ADDRESS AND
 STATUS, D-16
 FIXED PERIPHERAL ADDRESS
 ASSIGNMENT, D-13

MPCA CONTROL UNIT CURRENT ADDRESS
 AND STATUS FIELD, D-16
 PERIPHERAL ADDRESS
 ASSIGNMENT, D-14
 PERIPHERAL ADDRESS ASSIGNMENT AND
 RWC CONFIGURATION
 CONSIDERATIONS, D-13
 VARIABLE PERIPHERAL ADDRESS
 ASSIGNMENT, D-13

ADDRESSES
 INVALID BUCKET ADDRESSES, 4-62

ADDRESSING
 DIRECT ACCESS ADDRESSING, 3-70
 RANDOMIZING ADDRESSING, E-1

ALLOCATE
 ALLOCATE, 4-2
 ALLOCATE FUNCTION, 4-9
 JOB CONTROL LANGUAGE EXAMPLE FOR
 ALLOCATE FUNCTION, 4-21
 JOB CONTROL LANGUAGE FOR ALLOCATE
 FUNCTION, 4-10
 PROTECTION DURING ALLOCATE, 4-70
 SUMMARY OF JOB CONTROL STATEMENTS
 FOR ALLOCATE FUNCTION, 4-24

ALLOCATING
 ALLOCATING AN INDEXED SEQUENTIAL
 FILE, 4-59

ALLOCATION
 ALLOCATION, 2-9, 2-10, 2-23, C-4,
 C-12, C-24
 ALLOCATION CONVENTIONS, 2-6
 ASSIGNMENT OF UNITS OF
 ALLOCATION, C-3
 DATA UNIT OF ALLOCATION, 4-19
 FAILURE DURING ALLOCATION, 4-93
 FAILURE DURING ALLOCATION AND
 DEALLOCATION, 4-93
 FILE DESIGN AND ALLOCATION, C-1
 ILLUSTRATION OF UNITS OF ALLOCATION
 - TYPE 261 OR TYPE 262 DISK
 FILE, 2-7
 JOB CONTROL STATEMENTS FOR
 ALLOCATION OF FILES, 4-10
 SUMMARY OF JOB CONTROL STATEMENTS
 FOR ALLOCATION FUNCTION, 4-25
 UNITS OF ALLOCATION, 2-6

ALTER
 ALTER MEMBER (ALTER), 3-60
 ALTER STATUS OF MEMBER
 (ALTER), 3-19

ANALYSIS
 FREQUENCY ANALYSIS, E-7

ANC
 COMMUNICATION AREA SERVICE MACRO
 ROUTINES (MLCA ANC MUCA), 3-50

AREA
 COMMUNICATION AREA SERVICE MACRO
 ROUTINES (MLCA ANC MUCA), 3-50
 COMMUNICATION AREA SERVICE MACRO
 ROUTINES (MLCA AND MUCA), 3-2
 COMMUNICATION AREA FIELD
 DESIGNATORS, 3-52
 COMMUNICATION AREA MACRO CALL
 (MPCA), D-6
 COMMUNICATION AREA MACRO ROUTINE
 (MPCA), D-3
 COMMUNICATION AREA SERVICE MACRU
 CALLS (MLCA AND MUCA), D-3, D-9
 CYLINDER OVERFLOW AS PERCENTAGE OF
 DATA AREA, C-13
 DATA AREA, 2-23
 MASS STORAGE LOAD COMMUNICATION
 AREA MACRO CALL (MLCA), 3-51

INDEX

- MASS STORAGE UNLOAD COMMUNICATION AREA MACRO CALL (MUCA), 3-51
- MNEMONIC DESIGNATORS FOR COMMUNICATION AREA FIELDS, 3-52
- PRIME DATA AREA, 2-11
- RELATIONSHIP BETWEEN STRING INDEX ITEMS AND THE DATA AREA OF A CYLINDER, 2-15
- AREAS
 - INDEX AREAS, 2-11
 - MAP UNUSED AREAS, 4-6
 - OVERFLOW AREAS, 2-12, 2-23
- ASSIGNMENT
 - ASSIGNMENT OF FILES TO BE PROCESSED CONCURRENTLY, C-4
 - ASSIGNMENT OF UNITS OF ALLOCATION, C-3
 - FIXED PERIPHERAL ADDRESS ASSIGNMENT, D-13
 - PERIPHERAL ADDRESS ASSIGNMENT, D-14
 - PERIPHERAL ADDRESS ASSIGNMENT AND RWC CONFIGURATION CONSIDERATIONS, D-13
 - VARIABLE PERIPHERAL ADDRESS ASSIGNMENT, D-13
- BACKUP
 - BACKUP PROCEDURES, 2-27
 - LOGICAL BACKUP, 2-27
 - PHYSICAL BACKUP, 2-27
- BANNER-CHARACTER
 - BANNER-CHARACTER PARAMETER, 4-40
- BLOCK
 - BLOCK AND RECORD SIZES WITHIN 12K MEMORY, 4-6
 - BLOCK SIZE, C-2
- BLOCK-SIZE
 - BLOCK-SIZE PARAMETER, 4-16
- BLOCKS
 - RELATIONSHIP BETWEEN ITEMS RECORDS AND BLOCKS, 2-5
 - RELATIONSHIP BETWEEN ITEMS RECORDS BLOCKS AND BUCKETS, 2-24
- BOOTSTRAP
 - BOOTSTRAP RECORDS, 2-3
- BUCKET
 - BUCKET SIZE AND OVERFLOW, C-10
 - INVALID BUCKET ADDRESSES, 4-62
- BUCKET-ADDRESSING
 - BUCKET-ADDRESSING PARAMETER, 4-42
- BUCKET-SIZE
 - BUCKET-SIZE PARAMETER, 4-16
- BUCKETS
 - RELATIONSHIP BETWEEN ITEMS RECORDS BLOCKS AND BUCKETS, 2-24
- BUFFER
 - BUFFER ADDRESS (AAD), D-15
- BYPASS
 - BYPASS ERROR CONDITION, D-21
- CALL
 - COMMUNICATION AREA MACRO CALL (MPCA), D-6
 - CONTROL MACRO CALL (MPIOC), D-4
 - ISSUE NEW ACTION MACRO CALL, D-21
 - MASS STORAGE LOAD COMMUNICATION AREA MACRO CALL (MLCA), 3-51
 - MASS STORAGE UNLOAD COMMUNICATION AREA MACRO CALL (MUCA), 3-51
 - MCA MACRO CALL, 3-38
 - MIOC MACRO CALL, 3-26
 - OMISSION OF CONSECUTIVE PARAMETERS FROM MACRO CALL, 3-25
 - OMISSION OF SINGLE PARAMETER FROM MACRO CALL, 3-25
 - PARAMETERS OF MCA MACRO CALL, 3-39
 - PARAMETERS OF MIOC MACRO CALL, 3-27, 3-36
 - PARAMETERS OF MPCA MACRO CALL, D-6.1
 - PARAMETERS OF MPIOC MACRO CALL, D-4
 - PARAMETERS OF THE MPCA MACRO CALL, D-6
 - READ ACTION MACRO CALL, D-10
 - RESTORE ACTION MACRO CALL, D-11
 - SEEK ACTION MACRO CALL, D-11
 - SUMMARY OF ACTION MACRO CALL CODING, 3-63
 - VERIFY ACTION MACRO CALL, D-11
 - WAIT ACTION MACRO CALL, D-11
 - WRITE ACTION MACRO CALL, D-10
- CALLS
 - ACTION MACRO CALLS, 3-54, D-10
 - ACTION MACRO CALLS FOR EACH FILE TYPE IN EACH PROCESSING MODE, 3-6
 - ACTION MACRO CALLS (FOR PARTITIONED SEQUENTIAL FILES ONLY), 3-17
 - COMMUNICATION AREA SERVICE MACRO CALLS (MLCA AND MUCA), D-3, D-9
- CAPACITY
 - COMPRISING BETWEEN ACCESS TIME AND STORAGE CAPACITY, C-22
 - OPTIMIZING STORAGE CAPACITY, C-20
- CARD
 - CARD FILE FORMATS, 4-66
 - TAPE AND CARD FILE CONSIDERATIONS, 4-63
- CARD-IMAGE
 - CARD-IMAGE FILES, G-2
- CHANNEL
 - READ/WRITE CHANNEL UTILIZATION, 3-70, D-12
- CHARACTER
 - DATA ITEM STATUS CHARACTER, 2-20, 2-25
 - SUFFIX CHARACTER, D-14
- CHECK
 - EXPIRATION-DATE CHECK PARAMETER, 4-29
- CLOSE
 - CLOSE (MSCLOS), 3-56
- CLOSING
 - CLOSING FILES, 3-10
 - CLOSING INDEXED SEQUENTIAL AND DIRECT ACCESS FILES, 3-10
 - CLOSING SEQUENTIAL AND PARTITIONED SEQUENTIAL FILES, 3-10
- CODES
 - CONSOLE TYPEWRITER PAUSE CODES AND MESSAGES FOR LOGICAL I/O C, 3-83
 - EXIT AND RETURN CODES FOR DATA EXITS, 3-75
 - EXIT AND RETURN CODES FOR DEVICE EXITS, 3-76
 - EXIT AND RETURN CODES FOR MEMBER INDEX EXITS, 3-75
 - EXIT AND RETURN CODES FOR VOLUME DIRECTORY EXITS, 3-73
 - HALT CODES FOR LOGICAL I/O C, 3-78

INDEX

JOB CONTROL HALT CODES, 4-73

CODING
SUMMARY OF ACTION MACRO CALL
CODING, 3-63

COMMUNICATION
COMMUNICATION AREA SERVICE MACRO
ROUTINES (MLCA AND MUCA), 3-50
COMMUNICATION AREA SERVICE MACRO
ROUTINES (MLCA AND MUCA), 3-2

COMMUNICATION
COMMUNICATION AREA FIELD
DESIGNATORS, 3-52
COMMUNICATION AREA MACRO CALL
(MPCA), D-6
COMMUNICATION AREA MACRO ROUTINE
(MPCA), D-3
COMMUNICATION AREA SERVICE MACRO
CALLS (MLCA AND MUCA), D-3, D-9
MASS STORAGE LOAD COMMUNICATION
AREA MACRO CALL (MLCA), 3-51
MASS STORAGE UNLOAD COMMUNICATION
AREA MACRO CALL (MUCA), 3-51
MNEMONIC DESIGNATORS FOR
COMMUNICATION AREA FIELDS, 3-52
OWN-CODING COMMUNICATION WITH
LOAD-UNLOAD FUNCTION, 4-62

CONCEPT
DISK PACK CYLINDER CONCEPT - TYPE
259 DISK PACK DRIVES, 2-2

CONCURRENTLY
ASSIGNMENT OF FILES TO BE PROCESSED
CONCURRENTLY, C-4

CONDITION
BYPASS ERROR CONDITION, D-21

CONDITIONS
CONDITIONS RELATED TO NON-MASS
STORAGE FILE, 4-71
CONDITIONS SPECIFIC TO FILE SUPPORT
C, 4-77
FILE RELATED CONDITIONS, 4-72
FILE-RELATED CONDITIONS, 4-84
JOB CONTROL FILE CONDITIONS,
4-72, 4-84
PERIPHERAL CONDITIONS, 4-71, 4-83
TYPEWRITER MESSAGES FOR CONDITIONS
RELATED TO NON-MASS STORAGE
FILES, 4-83

CONFIGURATION
AVAILABLE MEMORY PER I/O MEDIA FOR
12K CONFIGURATION, 4-7
PERIPHERAL ADDRESS ASSIGNMENT AND
RWC CONFIGURATION
CONSIDERATIONS, D-13

CONSECUTIVE
OMISSION OF CONSECUTIVE PARAMETERS
FROM MACRO CALL, 3-25

CONSOLE
CONSOLE TYPEWRITER OPERATING
PROCEDURES, 3-82
CONSOLE TYPEWRITER PAUSE CODES AND
MESSAGES FOR LOGICAL I/O C, 3-83
JOB CONTROL FILE CONSOLE TYPEWRITER
MESSAGES, 4-85
OPERATOR CONTROL WITH CONSOLE
TYPEWRITER, 4-83

CONTROL
CONTROL MACRO CALL (MPIOC), D-4
CONTROL MACRO ROUTINE (MPIOC), U-3
CONTROL PANEL OPERATING
PROCEDURES, 3-77
CONTROL UNIT CURRENT ADDRESS AND

STATUS, D-16
INPUT/OUTPUT CONTROL MACRO ROUTINE
(MIOC), 3-26
JOB CONTROL FILE CONDITIONS,
4-72, 4-84
JOB CONTROL FILE CONSOLE TYPEWRITER
MESSAGES, 4-85
JOB CONTROL FOR A SEQUENCE OF
OPERATIONS, 4-9
JOB CONTROL FOR A SINGLE
OPERATION, 4-8
JOB CONTROL HALT CODES, 4-73
JOB CONTROL LANGUAGE EXAMPLE FOR
ALLOCATE FUNCTION, 4-21
JOB CONTROL LANGUAGE EXAMPLE FOR
DEALLOCATE FUNCTION, 4-31
JOB CONTROL LANGUAGE EXAMPLES FOR
LOAD AND UNLOAD FUNCTIONS, 4-45
JOB CONTROL LANGUAGE EXAMPLES FOR
MAP FUNCTION, 4-54
JOB CONTROL LANGUAGE FOR ALLOCATE
FUNCTION, 4-10
JOB CONTROL LANGUAGE FOR DATA
MANAGEMENT SUBSYSTEM, 1-6
JOB CONTROL LANGUAGE FOR DEALLOCATE
FUNCTION, 4-28
JOB CONTROL LANGUAGE FOR FILE
SUPPORT C, 4-8
JOB CONTROL LANGUAGE FOR LOAD AND
UNLOAD FUNCTIONS, 4-35
JOB CONTROL LANGUAGE FOR MAP
FUNCTION, 4-51
JOB CONTROL STATEMENTS FOR
ALLOCATION OF FILES, 4-10
JOB CONTROL STATEMENTS FOR
DEALLOCATE FUNCTION, 4-28
JOB CONTROL STATEMENTS FOR LOADING
AND UNLOADING FILES, 4-35
MASS STORAGE INPUT/OUTPUT CONTROL
MACRO ROUTINE (MIOC), 3-2
MPCA CONTROL UNIT CURRENT ADDRESS
AND STATUS FIELD, D-16
OPERATOR CONTROL AND MESSAGES FOR
FILE SUPPORT C, 4-70
OPERATOR CONTROL WITH CONSOLE
TYPEWRITER, 4-83
OPERATOR CONTROL WITH CONTROL
PANEL, 4-70
SUMMARY OF JOB CONTROL STATEMENTS
FOR ALLOCATE FUNCTION, 4-24
SUMMARY OF JOB CONTROL STATEMENTS
FOR ALLOCATION FUNCTION, 4-25
SUMMARY OF JOB CONTROL STATEMENTS
FOR DEALLOCATE FUNCTION,
4-33, 4-34
SUMMARY OF JOB CONTROL STATEMENTS
FOR LOAD AND UNLOAD
FUNCTIONS, 4-48
SUMMARY OF JOB CONTROL STATEMENTS
FOR LOAD/UNLOAD FUNCTIONS, 4-49
SUMMARY OF JOB CONTROL STATEMENTS
FOR MAP FUNCTION, 4-55, 4-56

CONVENTIONS
ALLOCATION CONVENTIONS, 2-6
DATA CONVENTIONS, 2-4
DATA MANAGEMENT CONVENTIONS,
1-1, 2-1
FILE ORGANIZATION CONVENTIONS, 2-8
PROCESSING CONVENTIONS, 2-26
VOLUME CONVENTIONS, 2-1

CONVERSION
RADIX CONVERSION, E-4

CORRECTION
RE-EXECUTION OF CORRECTION
PROCEDURE, D-20

CORRECTIVE
CORRECTIVE ACTION FOR USER'S ERROR
ROUTINE, D-21

INDEX

- CREATION
 - CREATION OF TERMINAL FILES, G-1
- CRITERIA
 - FILE DESIGN CRITERIA, C-1
- CUMULATIVE
 - CUMULATIVE LOADING OF A DIRECT ACCESS FILE, 2-25
- CURRENT
 - CONTROL UNIT CURRENT ADDRESS AND STATUS, D-16
 - END PROCESSING OF CURRENT MEMBER (ENDM), 3-18
 - MPCA CONTROL UNIT CURRENT ADDRESS AND STATUS FIELD, D-16
- CYLINDER
 - CYLINDER OVERFLOW AS PERCENTAGE OF DATA AREA, C-13
 - CYLINDER OVERFLOW-SIZE PARAMETER, 4-16
 - DISK PACK CYLINDER CONCEPT - TYPE 259 DISK PACK DRIVES, 2-2
 - RELATIONSHIP BETWEEN ITEMS OF THE MASTER AND CYLINDER INDEX, 2-14
 - RELATIONSHIP BETWEEN STRING INDEX ITEMS AND THE DATA AREA OF A CYLINDER, 2-15
 - SEEKING A DESIRED CYLINDER, 3-22
- CYLINDERS
 - DATA CYLINDERS REQUIRED, C-24
- DATA
 - CYLINDER OVERFLOW AS PERCENTAGE OF DATA AREA, C-13
 - DATA AREA, 2-23
 - DATA CONVENTIONS, 2-4
 - DATA CYLINDERS REQUIRED, C-24
 - DATA ITEM STATUS CHARACTER, 2-20, 2-25
 - DATA ITEMS, 4-67
 - DATA MANAGEMENT CONVENTIONS, 1-1, 2-1
 - DATA RECORDS, 4-65
 - DATA STRUCTURE, 2-9
 - DATA UNIT OF ALLOCATION, 4-19
 - EQUIPMENT REQUIREMENTS FOR DATA MANAGEMENT SUBSYSTEM, 1-7
 - EXIT AND RETURN CODES FOR DATA EXITS, 3-75
 - FILE-EXPIRATION DATA PARAMETER, 4-14
 - JOB CONTROL LANGUAGE FOR DATA MANAGEMENT SUBSYSTEM, 1-6
 - PRIME DATA AREA, 2-11
 - RELATIONSHIP BETWEEN STRING INDEX ITEMS AND THE DATA AREA OF A CYLINDER, 2-15
- DAY
 - DAY STATEMENT, 4-21, 4-31, 4-53
- DEALLOCATE
 - DEALLOCATE, 4-2
 - DEALLOCATE FUNCTION, 4-28
 - JOB CONTROL LANGUAGE EXAMPLE FOR DEALLOCATE FUNCTION, 4-31
 - JOB CONTROL LANGUAGE FOR DEALLOCATE FUNCTION, 4-28
 - JOB CONTROL STATEMENTS FOR DEALLOCATE FUNCTION, 4-28
 - PROTECTION DURING DEALLOCATE, 4-70
 - SUMMARY OF JOB CONTROL STATEMENTS FOR DEALLOCATE FUNCTION, 4-33, 4-34
- DEALLOCATION
 - FAILURE DURING ALLOCATION AND DEALLOCATION, 4-93
 - FAILURE DURING DEALLOCATION, 4-94
- DELETE
 - DELETE (MSDEL), 3-59
- DELETED
 - USING THE ITEM POSITION OF A DELETED ITEM, 2-22
- DELETING
 - DELETING ITEMS FROM FILES, 3-21
- DELETION
 - DELETION OF AN ITEM FROM A STRING, 2-21
- DESIGN
 - DESIGN CONSIDERATIONS, C-17
 - FILE DESIGN AND ALLOCATION, C-1
 - FILE DESIGN CRITERIA, C-1
 - GENERAL FILE DESIGN CONSIDERATIONS, C-2
- DESIGNATORS
 - COMMUNICATION AREA FIELD DESIGNATORS, 3-52
 - MNEMONIC DESIGNATORS FOR COMMUNICATION AREA FIELDS, 3-52
 - MNEMONIC DESIGNATORS FOR MLCA AND MUCA, D-9
- DEVICE
 - DEVICE PROTECTION, D-14
 - EXIT AND RETURN CODES FOR DEVICE EXITS, 3-76
 - MINIMUM DEVICE REQUIREMENTS FOR MASS STORAGE FILE ORGANIZATIONS, 4-39
- DEVICE-ADDRESS
 - DEVICE-ADDRESS PARAMETER, 4-14, 4-30, 4-38, 4-52, 4-54
 - DEVICE-ADDRESS PARAMETERS, 4-20
- DEVICE-TYPE
 - DEVICE-TYPE PARAMETER, 4-37, 4-54
- DIAGNOSTICS
 - FILE SUPPORT DIAGNOSTICS FOR 5040 HALT, 4-74
- DISK
 - DISK PACK CYLINDER CONCEPT - TYPE 259 DISK PACK DRIVES, 2-2
 - ILLUSTRATION OF UNITS OF ALLOCATION - TYPE 261 OR TYPE 262 DISK FILE, 2-7
 - OPTIMUM RECORD SIZE - TYPE 261 OR TYPE 262 DISK FILES, C-7
 - OPTIMUM RECORD SIZE - TYPES 258 259 OR 259A DISK PACK DRIVES, C-5
- DISTRIBUTION
 - DISTRIBUTION AND VOLATILITY, C-17
- DIVISION
 - PRIME NUMBER DIVISION, E-1
- DRIVES
 - DISK PACK CYLINDER CONCEPT - TYPE 259 DISK PACK DRIVES, 2-2
 - OPTIMUM RECORD SIZE - TYPES 258 259 OR 259A DISK PACK DRIVES, C-5
- EAD
 - USER'S UNCORRECTABLE ERROR ROUTINE ENTRANCE (EAD), D-15
- EDF
 - ADDRESS REGISTER CONTENTS AT TIME OF ERROR EXIT (EDF), D-20
- ELEMENTS
 - LANGUAGE ELEMENTS FOR LOGICAL I/O, C, 3-23
 - LANGUAGE ELEMENTS OF PHYSICAL I/O, C, D-4

INDEX

END
 END MEMBER (ENDM), 3-60
 END PROCESSING OF CURRENT MEMBER (ENDM), 3-18

ENDM
 END MEMBER (ENDM), 3-60
 END PROCESSING OF CURRENT MEMBER (ENDM), 3-18

ENFOLD
 SQUARE ENFOLD AND EXTRACT, E-2

EQUIPMENT
 ADDITIONAL USABLE EQUIPMENT, 1-7
 EQUIPMENT REQUIREMENTS FOR DATA MANAGEMENT SUBSYSTEM, 1-7
 REQUIRED EQUIPMENT, 1-7

ERI
 ERROR TYPE INDICATOR (ERI), D-19

ERROR
 ADDRESS REGISTER CONTENTS AT TIME OF ERROR EXIT (EDF), D-20
 BYPASS ERROR CONDITION, D-21
 CORRECTIVE ACTION FOR USER'S ERROR ROUTINE, D-21
 ERROR TYPE INDICATOR (ERI), D-19
 USER'S UNCORRECTABLE ERROR ROUTINE, D-19
 USER'S UNCORRECTABLE ERROR ROUTINE ENTRANCE (EAD), D-15

EXAMPLE
 JOB CONTROL LANGUAGE EXAMPLE FOR ALLOCATE FUNCTION, 4-21
 JOB CONTROL LANGUAGE EXAMPLE FOR DEALLOCATE FUNCTION, 4-31

EXAMPLE-OPTIMIZATION
 EXAMPLE-OPTIMIZATION FOR AN INDEXED SEQUENTIAL FILE, C-23

EXAMPLE-SUMMARY
 EXAMPLE-SUMMARY OF OPTIMUM POINTS, C-23

EXAMPLES
 JOB CONTROL LANGUAGE EXAMPLES FOR LOAD AND UNLOAD FUNCTIONS, 4-45
 JOB CONTROL LANGUAGE EXAMPLES FOR MAP FUNCTION, 4-54

EXECUTE
 EXECUTE STATEMENT, 4-8, 4-11, 4-28, 4-36, 4-51
 FORMAT OF FILE SUPPORT C EXECUTE STATEMENT, 4-8

EXECUTION
 PROTECTION OF MASS STORAGE DURING EXECUTION OF FILE SUPPORT C, 4-70

EXIT
 ADDRESS REGISTER CONTENTS AT TIME OF ERROR EXIT (EDF), D-20
 EXIT AND RETURN CODES FOR DATA EXITS, 3-75
 EXIT AND RETURN CODES FOR DEVICE EXITS, 3-76
 EXIT AND RETURN CODES FOR MEMBER INDEX EXITS, 3-75
 EXIT AND RETURN CODES FOR VOLUME DIRECTORY EXITS, 3-73

EXITS
 EXIT AND RETURN CODES FOR DATA EXITS, 3-75
 EXIT AND RETURN CODES FOR DEVICE EXITS, 3-76
 EXIT AND RETURN CODES FOR MEMBER INDEX EXITS, 3-75

EXIT AND RETURN CODES FOR VOLUME DIRECTORY EXITS, 3-73
 EXITS AND HALTS, 3-72
 EXITS STATEMENT, 4-44

EXPIRATION-DATE
 EXPIRATION-DATE CHECK PARAMETER, 4-29

EXPIRED
 MAP EXPIRED FILES, 4-6

EXTRACT
 SQUARE ENFOLD AND EXTRACT, E-2

FAILURE
 FAILURE DURING ALLOCATION, 4-93
 FAILURE DURING ALLOCATION AND DEALLOCATION, 4-93
 FAILURE DURING DEALLOCATION, 4-94

FIELD
 COMMUNICATION AREA FIELD DESIGNATORS, 3-52
 MPCA CONTROL UNIT CURRENT ADDRESS AND STATUS FIELD, D-16

FIELDS
 FIELDS OF FIRST ITEM IN MEMBER INDEX, B-2
 FIELDS OF LAST ITEM IN MEMBER INDEX, B-3
 FIELDS OF MEMBER INDEX ITEMS, B-2
 MNEMONIC DESIGNATORS FOR COMMUNICATION AREA FIELDS, 3-52

FILE
 ACTION MACRO CALLS FOR EACH FILE TYPE IN EACH PROCESSING MODE, 3-6
 ALLOCATING AN INDEXED SEQUENTIAL FILE, 4-59
 CARD FILE FORMATS, 4-66
 CONDITIONS RELATED TO NON-MASS STORAGE FILE, 4-71
 CONDITIONS SPECIFIC TO FILE SUPPORT C, 4-77
 CUMULATIVE LOADING OF A DIRECT ACCESS FILE, 2-25
 DIRECT ACCESS FILE CONSIDERATIONS, C-10
 DIRECT ACCESS FILE ORGANIZATION, 2-20
 DIRECTLY PROCESSING AN INDEXED SEQUENTIAL FILE, 2-13
 EXAMPLE-OPTIMIZATION FOR AN INDEXED SEQUENTIAL FILE, C-23
 FILE ADDITIONS, C-1
 FILE CONSIDERATIONS, 5-57
 FILE DESCRIPTION MACRO ROUTINE (MCA), 3-2, 3-38
 FILE DESIGN AND ALLOCATION, C-1
 FILE DESIGN CRITERIA, C-1
 FILE INQUIRIES, C-1
 FILE ORGANIZATION, 2-23
 FILE ORGANIZATION CONVENTIONS, 2-8
 FILE PREFIX, D-15
 FILE PROCESSING FUNCTIONS, 2-27
 FILE PROCESSING MODES, 3-4
 FILE PROTECTION, F-1
 FILE RELATED CONDITIONS, 4-72
 FILE STATEMENT, 4-12, 4-29, 4-53
 FILE STATEMENT FOR THE LIST FILE, 4-20, 4-53
 FILE STATEMENTS, 4-36
 FILE STRUCTURE, 2-11
 FILE SUPPORT C, 4-1
 FILE SUPPORT C HALTS, 4-78
 FILE SUPPORT DIAGNOSTICS FOR 5040 HALT, 4-74
 FILE SUPPORT C PROGRAM, 1-5
 FOREGROUND/BACKGROUND PROCESSING OF FILE SUPPORT C, 4-1

- FORMAT OF FILE SUPPORT C EXECUTE STATEMENT, 4-8
 FUNCTIONS OF FILE SUPPORT C, 4-2
 GENERAL DESCRIPTION OF FILE SUPPORT C, 4-1
 GENERAL FILE DESIGN CONSIDERATIONS, C-2
 ILLUSTRATION OF UNITS OF ALLOCATION - TYPE 261 OR TYPE 262 DISK FILE, 2-7
 INDEXED SEQUENTIAL FILE CONSIDERATIONS, C-17
 INDEXED SEQUENTIAL FILE ORGANIZATION, 2-9
 JOB CONTROL FILE CONDITIONS, 4-72, 4-84
 JOB CONTROL FILE CONSOLE TYPEWRITER MESSAGES, 4-85
 JOB CONTROL LANGUAGE FOR FILE SUPPORT C, 4-8
 LOADING A DIRECT ACCESS FILE, 5-57
 LOADING A PARTITIONED SEQUENTIAL FILE, 4-58
 LOADING AN INDEXED SEQUENTIAL FILE, 4-59
 LOADING BY FILE, 4-58
 LOADING FILE SUPPORT C, 4-68
 MAP DESCRIPTION OF A FILE, 4-2
 MASS STORAGE FILE PROTECTION, F-1
 MINIMUM DEVICE REQUIREMENTS FOR MASS STORAGE FILE ORGANIZATIONS, 4-39
 MIXED FILE ORGANIZATIONS, 4-60
 MULTIVOLUME FILE PROCESSING, C-3
 OMITTING ITEMS FROM THE OUTPUT FILE, 4-62
 OPENING AN INDEXED SEQUENTIAL FILE, 3-8
 OPERATING PROCEDURES FOR FILE SUPPORT C, 4-68
 OPERATOR CONTROL AND MESSAGES FOR FILE SUPPORT C, 4-70
 PARTITIONING A SEQUENTIAL FILE, B-1
 PROCESSING A PARTITIONED SEQUENTIAL FILE BY MEMBER NAMES, 4-58
 PROGRAMMER'S PREPARATION INFORMATION FOR FILE SUPPORT C, 5-57
 PROTECTION OF MASS STORAGE DURING EXECUTION OF FILE SUPPORT C, 4-70
 RELEASE COMPLETE FILE TO UNUSED STATE (MSREL), 3-19
 SEQUENTIAL FILE CONSIDERATIONS, C-4
 SEQUENTIAL FILE ORGANIZATION, 2-8
 SEQUENTIAL FILE USING PARTITIONING OPTION, B-4
 TAPE AND CARD FILE CONSIDERATIONS, 4-63
 TYPEWRITER MESSAGES SPECIFIC TO FILE SUPPORT C, 4-86
 UNLOADING A DIRECT ACCESS FILE, 5-57
 UNLOADING A PARTITIONED SEQUENTIAL FILE, 4-58
 UNLOADING AN INDEXED SEQUENTIAL FILE, 4-60
 UNLOADING BY FILE, 4-58
- FILE-EXPIRATION
 FILE-EXPIRATION DATA PARAMETER, 4-14
- FILE-NAME
 FILE-NAME PARAMETER, 4-12, 4-29, 4-37
- FILE-ORGANIZATION
 FILE-ORGANIZATION PARAMETER, 4-12
- FILE-RELATED
 FILE-RELATED CONDITIONS, 4-84
- FILES
 ACTION MACRO CALLS (FOR PARTITIONED SEQUENTIAL FILES ONLY), 3-17
 ASSIGNMENT OF FILES TO BE PROCESSED CONCURRENTLY, C-4
 CARD-IMAGE FILES, G-2
 CLOSING FILES, 3-10
 CLOSING INDEXED SEQUENTIAL AND DIRECT ACCESS FILES, 3-10
 CLOSING SEQUENTIAL AND PARTITIONED SEQUENTIAL FILES, 3-10
 CREATION OF TERMINAL FILES, G-1
 DELETING ITEMS FROM FILES, 3-21
 DIRECT ACCESS FILES, 5-57
 DIRECT ACCESS FILES AND KEYS, 2-24
 INDEXED SEQUENTIAL FILES, 4-59
 INSERTING ITEMS IN DIRECT ACCESS FILES, 3-21
 INSERTING ITEMS IN FILES, 3-20
 INSERTING ITEMS IN INDEXED SEQUENTIAL FILES, 3-20
 JOB CONTROL STATEMENTS FOR ALLOCATION OF FILES, 4-10
 JOB CONTROL STATEMENTS FOR LOADING AND UNLOADING FILES, 4-35
 MAP EXPIRED FILES, 4-6
 OPENING DIRECT ACCESS FILES, 3-9
 OPENING FILES, 3-6
 OPENING PARTITIONED SEQUENTIAL FILES, 3-8
 OPENING SEQUENTIAL FILES, 3-6
 OPTIMUM RECORD SIZE - TYPE 261 OR TYPE 262 DISK FILES, C-7
 PARTITIONED SEQUENTIAL FILES, 4-58
 PRINT-IMAGE FILES, G-2
 PUTTING ITEMS TO SEQUENTIAL AND PARTITIONED SEQUENTIAL FILES, 3-17
 RANDOM PLUS SEQUENTIAL FILES, C-2
 RANDOM VERSUS SEQUENTIAL FILES, C-2
 REPLACING ITEMS IN DIRECT ACCESS FILES, 3-16
 REPLACING ITEMS IN FILES, 3-15
 REPLACING ITEMS IN INDEXED SEQUENTIAL FILES, 3-16
 REPLACING ITEMS IN SEQUENTIAL AND PARTITIONED SEQUENTIAL FILES, 3-16
 RETRIEVING ITEMS IN DIRECT ACCESS FILES, 3-13
 RETRIEVING ITEMS IN FILES, 3-11
 RETRIEVING ITEMS IN INDEXED SEQUENTIAL FILES, 3-12
 RETRIEVING ITEMS IN SEQUENTIAL AND PARTITIONED SEQUENTIAL FILES, 3-11
 SEQUENTIAL FILES, 5-57
 SUMMARY OF MSGET MACRO FUNCTIONS FOR DIRECT ACCESS FILES, 3-15
 TERMINAL FILES, G-1
 TYPEWRITER MESSAGES FOR CONDITIONS RELATED TO NON-MASS STORAGE FILES, 4-83
 UNLOADING MASS STORAGE FILES ONTO PRINTER, 4-67
- FIXED
 FIXED PERIPHERAL ADDRESS ASSIGNMENT, D-13
- FOREGROUND/BACKGROUND
 FOREGROUND/BACKGROUND PROCESSING OF
- FILE SUPPORT C, 4-1

INDEX

FORMAT
 FORMAT OF FILE SUPPORT C EXECUTE STATEMENT, 4-8

FORMATS
 1/2-INCH TAPE FORMATS, 4-63
 CARD FILE FORMATS, 4-66

FORMATTING
 FORMATTING AND VOLUME PREPARATION, 2-3

FUNCTION
 ALLOCATE FUNCTION, 4-9
 DEALLOCATE FUNCTION, 4-28
 FUNCTION STATEMENT, 4-12, 4-29, 4-36, 4-51
 JOB CONTROL LANGUAGE EXAMPLE FOR ALLOCATE FUNCTION, 4-21
 JOB CONTROL LANGUAGE EXAMPLE FOR DEALLOCATE FUNCTION, 4-31
 JOB CONTROL LANGUAGE EXAMPLES FOR MAP FUNCTION, 4-54
 JOB CONTROL LANGUAGE FOR ALLOCATE FUNCTION, 4-10
 JOB CONTROL LANGUAGE FOR DEALLOCATE FUNCTION, 4-28
 JOB CONTROL LANGUAGE FOR MAP FUNCTION, 4-51
 JOB CONTROL STATEMENTS FOR DEALLOCATE FUNCTION, 4-28
 LISTING OF SAMPLE UNLOAD-TO-PRINTER FUNCTION, 4-69
 MAP FUNCTION, 4-51
 OWN-CODING COMMUNICATION WITH LOAD-UNLOAD FUNCTION, 4-62
 SUMMARY OF JOB CONTROL STATEMENTS FOR ALLOCATE FUNCTION, 4-24
 SUMMARY OF JOB CONTROL STATEMENTS FOR ALLOCATION FUNCTION, 4-25
 SUMMARY OF JOB CONTROL STATEMENTS FOR DEALLOCATE FUNCTION, 4-33, 4-34
 SUMMARY OF JOB CONTROL STATEMENTS FOR MAP FUNCTION, 4-55, 4-56

FUNCTIONS
 ACTION MACRO PROCESSING FUNCTIONS, 3-4
 FILE PROCESSING FUNCTIONS, 2-27
 FUNCTIONS OF FILE SUPPORT C, 4-2
 JOB CONTROL LANGUAGE EXAMPLES FOR LOAD AND UNLOAD FUNCTIONS, 4-45
 JOB CONTROL LANGUAGE FOR LOAD AND UNLOAD FUNCTIONS, 4-35
 LOAD AND UNLOAD FUNCTIONS, 4-33
 NUMBER OF FUNCTIONS PERFORMED, 4-6
 SUMMARY OF JOB CONTROL STATEMENTS FOR LOAD AND UNLOAD FUNCTIONS, 4-48
 SUMMARY OF JOB CONTROL STATEMENTS FOR LOAD/UNLOAD FUNCTIONS, 4-49
 SUMMARY OF MSGET MACRO FUNCTIONS FOR DIRECT ACCESS FILES, 3-15

GENERAL
 ENTRANCE TO GENERAL OVERFLOW, 4-63
 GENERAL DESCRIPTION OF FILE SUPPORT C, 4-1
 GENERAL FILE DESIGN CONSIDERATIONS, C-2
 GENERAL OVERFLOW PARAMETER, 4-13

GET
 GET (MSGET), 3-56

HALT
 FILE SUPPORT DIAGNOSTICS FOR 5040 HALT, 4-74
 HALT CODES FOR LOGICAL I/O C, 3-78
 JOB CONTROL HALT CODES, 4-73

HALTS
 EXITS AND HALTS, 3-72
 FILE SUPPORT C HALTS, 4-78

HEADER
 HEADER LABEL, 4-63, 4-66

ILLUSTRATION
 ILLUSTRATION OF UNITS OF ALLOCATION - TYPE 261 OR TYPE 262 DISK FILE, 2-7

IMBED
 IMBED PARAMETER, 4-43

INDEX
 EXIT AND RETURN CODES FOR MEMBER INDEX EXITS, 3-75
 FIELDS OF FIRST ITEM IN MEMBER INDEX, B-2
 FIELDS OF LAST ITEM IN MEMBER INDEX, B-3
 FIELDS OF MEMBER INDEX ITEMS, B-2
 INDEX AREAS, 2-11
 INDEX REGISTERS, 3-69
 MASTER/CYLINDER INDEX PARAMETER, 4-18
 RELATIONSHIP BETWEEN ITEMS OF THE MASTER AND CYLINDER INDEX, 2-14
 RELATIONSHIP BETWEEN STRING INDEX ITEMS AND THE DATA AREA OF A CYLINDER, 2-15
 TRACKS REQUIRED FOR MASTER/CYLINDER INDEX, C-25
 USE OF INDEX REGISTERS, D-13

INDEXED
 ALLOCATING AN INDEXED SEQUENTIAL FILE, 4-59
 CLOSING INDEXED SEQUENTIAL AND DIRECT ACCESS FILES, 3-10
 DIRECTLY PROCESSING AN INDEXED SEQUENTIAL FILE, 2-13
 EXAMPLE-OPTIMIZATION FOR AN INDEXED SEQUENTIAL FILE, C-23
 INDEXED SEQUENTIAL, 3-72
 INDEXED SEQUENTIAL FILE CONSIDERATIONS, C-17
 INDEXED SEQUENTIAL FILE ORGANIZATION, 2-9
 INDEXED SEQUENTIAL FILES, 4-59
 INSERTING ITEMS IN INDEXED SEQUENTIAL FILES, 3-20
 LOADING AN INDEXED SEQUENTIAL FILE, 4-59
 OPENING AN INDEXED SEQUENTIAL FILE, 3-8
 REPLACING ITEMS IN INDEXED SEQUENTIAL FILES, 3-16
 RETRIEVING ITEMS IN INDEXED SEQUENTIAL FILES, 3-12
 UNLOADING AN INDEXED SEQUENTIAL FILE, 4-60

INDEX-SIZE
 INDEX-SIZE PARAMETER, 4-16

INDICATOR
 ERROR TYPE INDICATOR (ERI), D-19

INFORMATION
 PROGRAMMER'S PREPARATION INFORMATION FOR FILE SUPPORT C, 5-57

INFORMATION
 PROGRAMMER'S PREPARATION INFORMATION FOR LOGICAL I/O C, 3-64
 PROGRAMMER'S PREPARATION

INDEX

INFORMATION FOR PHYSICAL I/O
C, D-12

INPUT-ONLY
INPUT-ONLY PROCESSING MODE, 3-4

INPUT/OUTPUT
INPUT/OUTPUT CONTROL MACRO ROUTINE
(MIOC), 3-26
INPUT/OUTPUT PROCESSING MODE, 3-4
MASS STORAGE INPUT/OUTPUT CONTROL
MACRO ROUTINE (MIOC), 3-2

INQUIRIES
FILE INQUIRIES, C-1

IN/OUT
IN/OUT PARAMETER, 4-37

INSERT
INSERT (MSINS), 3-58

INSERTING
INSERTING ITEMS IN DIRECT ACCESS
FILES, 3-21
INSERTING ITEMS IN FILES, 3-20
INSERTING ITEMS IN INDEXED
SEQUENTIAL FILES, 3-20

INSERTION
INSERTION OF ITEMS INTO A
STRING, 2-16

INSUFFICIENT
INSUFFICIENT SPACE, 4-62

INTRODUCTION
INTRODUCTION, 1-1

INVALID
INVALID BUCKET ADDRESSES, 4-62

I/O
AVAILABLE MEMORY PER I/O MEDIA FOR
12K CONFIGURATION, 4-7
CONSOLE TYPEWRITER PAUSE CODES AND
MESSAGES FOR LOGICAL I/O C, 3-83
DETAILED DESCRIPTION OF PHYSICAL
I/O C MACRO ROUTINES, D-3
HALT CODES FOR LOGICAL I/O C, 3-78
LANGUAGE ELEMENTS FOR LOGICAL I/O
C, 3-23
LANGUAGE ELEMENTS OF PHYSICAL I/O
C, D-4
LOGICAL I/O C, 3-1
LOGICAL I/O C MEMORY
REQUIREMENTS, 3-64
LOGICAL I/O C PROGRAM, 1-4
OPERATING PROCEDURES FOR LOGICAL
I/O C, 3-77
OPERATING PROCEDURES FOR PHYSICAL
I/O C, D-21
PHYSICAL I/O C, D-1
PHYSICAL I/O C RELATIONSHIPS WITH
MCA, 3-69
PHYSICAL I/O C RELATIONSHIPS WITH
MIOC, 3-69
PROGRAMMER'S PREPARATION
INFORMATION FOR LOGICAL I/O
C, 3-64
PROGRAMMER'S PREPARATION
INFORMATION FOR PHYSICAL I/O
C, D-12
SUMMARY OF LOGICAL I/O C MACRO
ROUTINES, 3-2, 3-3
USE OF PHYSICAL I/O C, D-1

ISSUE
ISSUE NEW ACTION MACRO CALL, D-21

ITEM
DATA ITEM STATUS CHARACTER,

2-20, 2-25
DELETION OF AN ITEM FROM A
STRING, 2-21
FIELDS OF FIRST ITEM IN MEMBER
INDEX, B-2
FIELDS OF LAST ITEM IN MEMBER
INDEX, B-3
ITEM KEY SPECIFICATION, 3-71
ITEM SEQUENCE, C-17
NONNUMERIC ITEM KEYS, E-5
USING THE ITEM POSITION OF A
DELETED ITEM, 2-22

ITEM-KEY
ITEM-KEY PARAMETER, 4-13

ITEM-LENGTH
ITEM-LENGTH PARAMETER, 4-15, 4-39

ITEMS
DATA ITEMS, 4-67
DELETING ITEMS FROM FILES, 3-21
FIELDS OF MEMBER INDEX ITEMS, B-2
INSERTING ITEMS IN DIRECT ACCESS
FILES, 3-21
INSERTING ITEMS IN FILES, 3-20
INSERTING ITEMS IN INDEXED
SEQUENTIAL FILES, 3-20
INSERTION OF ITEMS INTO A
STRING, 2-16
OMITTING ITEMS FROM THE OUTPUT
FILE, 4-62
PADDING ITEMS, 4-66
PUTTING ITEMS TO SEQUENTIAL AND
PARTITIONED SEQUENTIAL
FILES, 3-17
RELATIONSHIP BETWEEN ITEMS AND
RECORDS, 2-5
RELATIONSHIP BETWEEN ITEMS OF THE
MASTER AND CYLINDER INDEX, 2-14
RELATIONSHIP BETWEEN ITEMS RECORDS
AND BLOCKS, 2-5
RELATIONSHIP BETWEEN ITEMS RECORDS
BLOCKS AND BUCKETS, 2-24
RELATIONSHIP BETWEEN STRING INDEX
ITEMS AND THE DATA AREA OF A
CYLINDER, 2-15
REPLACING ITEMS IN DIRECT ACCESS
FILES, 3-16
REPLACING ITEMS IN FILES, 3-15
REPLACING ITEMS IN INDEXED
SEQUENTIAL FILES, 3-16
REPLACING ITEMS IN SEQUENTIAL AND
PARTITIONED SEQUENTIAL
FILES, 3-16
RETRIEVING ITEMS IN DIRECT ACCESS
FILES, 3-13
RETRIEVING ITEMS IN FILES, 3-11
RETRIEVING ITEMS IN INDEXED
SEQUENTIAL FILES, 3-12
RETRIEVING ITEMS IN SEQUENTIAL AND
PARTITIONED SEQUENTIAL
FILES, 3-11

JOB
JOB CONTROL FILE CONDITIONS,
4-72, 4-84
JOB CONTROL FILE CONSOLE TYPEWRITER
MESSAGES, 4-85
JOB CONTROL FOR A SEQUENCE OF
OPERATIONS, 4-9
JOB CONTROL FOR A SINGLE
OPERATION, 4-8
JOB CONTROL HALT CODES, 4-73
JOB CONTROL LANGUAGE EXAMPLE FOR
ALLOCATE FUNCTION, 4-21
JOB CONTROL LANGUAGE EXAMPLE FOR
DEALLOCATE FUNCTION, 4-31
JOB CONTROL LANGUAGE EXAMPLES FOR
LOAD AND UNLOAD FUNCTIONS, 4-45
JOB CONTROL LANGUAGE EXAMPLES FOR
MAP FUNCTION, 4-54

INDEX

JOB CONTROL LANGUAGE FOR ALLOCATE FUNCTION, 4-10
 JOB CONTROL LANGUAGE FOR DATA MANAGEMENT SUBSYSTEM, 1-6
 JOB CONTROL LANGUAGE FOR DEALLOCATE FUNCTION, 4-28
 JOB CONTROL LANGUAGE FOR FILE SUPPORT C, 4-8
 JOB CONTROL LANGUAGE FOR LOAD AND UNLOAD FUNCTIONS, 4-35
 JOB CONTROL LANGUAGE FOR MAP FUNCTION, 4-51
 JOB CONTROL STATEMENTS FOR ALLOCATION OF FILES, 4-10
 JOB CONTROL STATEMENTS FOR DEALLOCATE FUNCTION, 4-28
 JOB CONTROL STATEMENTS FOR LOADING AND UNLOADING FILES, 4-35
 SUMMARY OF JOB CONTROL STATEMENTS FOR ALLOCATE FUNCTION, 4-24
 SUMMARY OF JOB CONTROL STATEMENTS FOR ALLOCATION FUNCTION, 4-25
 SUMMARY OF JOB CONTROL STATEMENTS FOR DEALLOCATE FUNCTION, 4-33, 4-34
 SUMMARY OF JOB CONTROL STATEMENTS FOR LOAD AND UNLOAD FUNCTIONS, 4-48
 SUMMARY OF JOB CONTROL STATEMENTS FOR LOAD/UNLOAD FUNCTIONS, 4-49
 SUMMARY OF JOB CONTROL STATEMENTS FOR MAP FUNCTION, 4-55, 4-56

KEY
 ITEM KEY SPECIFICATION, 3-71
 KEY OUT OF SEQUENC, 4-63

KEYS
 DIRECT ACCESS FILES AND KEYS, 2-24
 MULTIFIELD KEYS, E-6
 NONNUMERIC ITEM KEYS, E-5

LABEL
 HEADER LABEL, 4-63, 4-66
 TRAILER LABEL, 4-66, 4-67
 VOLUME LABEL, 2-3, A-2
 VOLUME LABEL AND VOLUME DIRECTORY, A-1

LANGUAGE
 JOB CONTROL LANGUAGE EXAMPLE FOR ALLOCATE FUNCTION, 4-21
 JOB CONTROL LANGUAGE EXAMPLE FOR DEALLOCATE FUNCTION, 4-31
 JOB CONTROL LANGUAGE EXAMPLES FOR LOAD AND UNLOAD FUNCTIONS, 4-45
 JOB CONTROL LANGUAGE EXAMPLES FOR MAP FUNCTION, 4-54
 JOB CONTROL LANGUAGE FOR ALLOCATE FUNCTION, 4-10
 JOB CONTROL LANGUAGE FOR DATA MANAGEMENT SUBSYSTEM, 1-6
 JOB CONTROL LANGUAGE FOR DEALLOCATE FUNCTION, 4-28
 JOB CONTROL LANGUAGE FOR FILE SUPPORT C, 4-8
 JOB CONTROL LANGUAGE FOR LOAD AND UNLOAD FUNCTIONS, 4-35
 JOB CONTROL LANGUAGE FOR MAP FUNCTION, 4-51
 LANGUAGE ELEMENTS FOR LOGICAL I/O C, 3-23
 LANGUAGE ELEMENTS OF PHYSICAL I/O C, D-4

LOAD
 LOAD AND UNLOAD FUNCTIONS, 4-33

LINKING
 HANDLING TRACK LINKING RECORDS, D-18

LIST
 FILE STATEMENT FOR THE LIST FILE, 4-20, 4-53

LISTING
 LISTING OF SAMPLE UNLOAD-TO-PRINTER FUNCTION, 4-69

LOAD
 JOB CONTROL LANGUAGE EXAMPLES FOR LOAD AND UNLOAD FUNCTIONS, 4-45
 JOB CONTROL LANGUAGE FOR LOAD AND UNLOAD FUNCTIONS, 4-35
 LOAD, 4-2
 MASS STORAGE LOAD COMMUNICATION AREA MACRO CALL (MLCA), 3-51
 SUMMARY OF JOB CONTROL STATEMENTS FOR LOAD AND UNLOAD FUNCTIONS, 4-48

LOADING
 CUMULATIVE LOADING OF A DIRECT ACCESS FILE, 2-25
 JOB CONTROL STATEMENTS FOR LOADING AND UNLOADING FILES, 4-35
 LOADING A DIRECT ACCESS FILE, 5-57
 LOADING A PARTITIONED SEQUENTIAL FILE, 4-58
 LOADING AN INDEXED SEQUENTIAL FILE, 4-59
 LOADING BY FILE, 4-58
 LOADING FILE SUPPORT C, 4-68
 LOADING FROM MASS STORAGE TO MASS STORAGE, 4-59
 LOADING OR UNLOADING, 4-60
 LOADING SELECTED MEMBERS, 4-58
 PROGRAM SEGMENT LOADING, 3-68

LOAD-UNLOAD
 OWN-CODING COMMUNICATION WITH LOAD-UNLOAD FUNCTION, 4-62
 PROTECTION DURING LOAD-UNLOAD, 4-70

LOAD/UNLOAD
 SUMMARY OF JOB CONTROL STATEMENTS FOR LOAD/UNLOAD FUNCTIONS, 4-49

LOCATION
 SET LOCATION (SETL), 3-62
 SETTING PROCESSING TO A SPECIFIED LOCATION, 3-22

LOGICAL
 CONSOLE TYPEWRITER FAUSE CODES AND MESSAGES FOR LOGICAL I/O C, 3-83
 HALT CODES FOR LOGICAL I/O C, 3-78
 LANGUAGE ELEMENTS FOR LOGICAL I/O C, 3-23
 LOGICAL BACKUP, 2-27
 LOGICAL I/O C, 3-1
 LOGICAL I/O C MEMORY REQUIREMENTS, 3-64
 LOGICAL I/O C PROGRAM, 1-4
 OPERATING PROCEDURES FOR LOGICAL I/O C, 3-77
 PROGRAMMER'S PREPARATION INFORMATION FOR LOGICAL I/O C, 3-64
 SUMMARY OF LOGICAL I/O C MACRO ROUTINES, 3-2, 3-3

LOKDEV
 LOKDEV ACTION MACRO ROUTINE, D-18

LOW-MEMORY-ADDRESS
 LOW-MEMORY-ADDRESS PARAMETER, 4-45

MACRO
 ACTION MACRO CALLS, 3-54, D-10
 ACTION MACRO CALLS FOR EACH FILE TYPE IN EACH PROCESSING MODE, 3-6

INDEX

- ACTION MACRO CALLS (FOR PARTITIONED SEQUENTIAL FILES ONLY), 3-17
- ACTION MACRO PROCESSING FUNCTIONS, 3-4
- ACTION MACRO ROUTINES, 3-2, D-4
- COMMUNICATION AREA SERVICE MACRO ROUTINES (MLCA AND MUCA), 3-50
- COMMUNICATION AREA SERVICE MACRO ROUTINES (MLCA AND MUCA), 3-2
- COMMUNICATION AREA MACRO CALL (MPCA), D-6
- COMMUNICATION AREA MACRO ROUTINE (MPCA), D-3
- COMMUNICATION AREA SERVICE MACRO CALLS (MLCA AND MUCA), D-3, D-9
- CONSIDERATIONS FOR ACTION MACRO ROUTINES, D-17
- CONTROL MACRO CALL (MPIOC), D-4
- CONTROL MACRO ROUTINE (MPIOC), D-3
- DETAILED DESCRIPTION OF PHYSICAL I/O C MACRO ROUTINES, D-3
- FILE DESCRIPTION MACRO ROUTINE (MCA), 3-2, 3-38
- INPUT/OUTPUT CONTROL MACRO ROUTINE (MIOC), 3-26
- ISSUE NEW ACTION MACRO CALL, D-21
- LOKDEV ACTION MACRO ROUTINE, D-18
- MASS STORAGE INPUT/OUTPUT CONTROL MACRO ROUTINE (MIOC), 3-2
- MASS STORAGE LOAD COMMUNICATION AREA MACRO CALL (MLCA), 3-51
- MASS STORAGE UNLOAD COMMUNICATION AREA MACRO CALL (MUCA), 3-51
- MCA MACRO CALL, 3-38
- MIOC MACRO CALL, 3-26
- OMISSION OF CONSECUTIVE PARAMETERS FROM MACRO CALL, 3-25
- OMISSION OF SINGLE PARAMETER FROM MACRO CALL, 3-25
- PARAMETERS OF MCA MACRO CALL, 3-39
- PARAMETERS OF MIOC MACRO CALL, 3-27, 3-36
- PARAMETERS OF MPCA MACRO CALL, D-6.1
- PARAMETERS OF MPIOC MACRO CALL, D-4
- PARAMETERS OF THE MPCA MACRO CALL, D-6
- READ ACTION MACRO CALL, D-10
- READ ACTION MACRO ROUTINE, D-17
- RESTORE ACTION MACRO CALL, D-11
- RESTORE ACTION MACRO ROUTINE, D-18
- SEEK ACTION MACRO CALL, D-11
- SUMMARY OF ACTION MACRO CALL CODING, 3-63
- SUMMARY OF LOGICAL I/O C MACRO ROUTINES, 3-2, 3-3
- SUMMARY OF MSGET MACRO FUNCTIONS FOR DIRECT ACCESS FILES, 3-15
- VERIFY ACTION MACRO CALL, D-11
- VERIFY ACTION MACRO ROUTINE, D-18
- WAIT ACTION MACRO CALL, D-11
- WAIT ACTION MACRO ROUTINE, D-18
- WRITE ACTION MACRO CALL, D-10
- WRITE ACTION MACRO ROUTINE, D-17

- MALTER
 - ALTER MEMBER (MALTER), 3-60
 - ALTER STATUS OF MEMBER (MALTER), 3-19

- MANAGEMENT
 - DATA MANAGEMENT CONVENTIONS, 1-1, 2-1
 - EQUIPMENT REQUIREMENTS FOR DATA MANAGEMENT SUBSYSTEM, 1-7
 - JOB CONTROL LANGUAGE FOR DATA MANAGEMENT SUBSYSTEM, 1-6

- MAP
 - JOB CONTROL LANGUAGE EXAMPLES FOR MAP FUNCTION, 4-54
 - JOB CONTROL LANGUAGE FOR MAP FUNCTION, 4-51
 - MAP, 4-2
 - MAP DESCRIPTION OF A FILE, 4-2
 - MAP EXPIRED FILES, 4-6
 - MAP FUNCTION, 4-51
 - MAP UNUSED AREAS, 4-6
 - PROTECTION DURING MAP, 4-70
 - SUMMARY OF JOB CONTROL STATEMENTS FOR MAP FUNCTION, 4-55, 4-56

- MARKS
 - TAPE MARKS, 4-66

- MASS
 - LOADING FROM MASS STORAGE TO MASS STORAGE, 4-59
 - MASS STORAGE FILE PROTECTION, F-1
 - MASS STORAGE INPUT/OUTPUT CONTROL MACRO ROUTINE (MIOC), 3-2
 - MASS STORAGE LOAD COMMUNICATION AREA MACRO CALL (MLCA), 3-51
 - MASS STORAGE UNLOAD COMMUNICATION AREA MACRO CALL (MUCA), 3-51
 - MINIMUM DEVICE REQUIREMENTS FOR MASS STORAGE FILE ORGANIZATIONS, 4-39
 - PROTECTION OF MASS STORAGE DURING EXECUTION OF FILE SUPPORT C, 4-70
 - UNLOADING MASS STORAGE FILES ONTO PRINTER, 4-67

- MASTER
 - RELATIONSHIP BETWEEN ITEMS OF THE MASTER AND CYLINDER INDEX, 2-14

- MASTER/CYLINDER
 - MASTER/CYLINDER INDEX PARAMETER, 4-18
 - TRACKS REQUIRED FOR MASTER/CYLINDER INDEX, C-25

- MCA
 - FILE DESCRIPTION MACRO ROUTINE (MCA), 3-2, 3-38
 - MCA MACRO CALL, 3-38
 - PARAMETERS OF MCA MACRO CALL, 3-39
 - PHYSICAL I/O C RELATIONSHIPS WITH MCA, 3-69
 - SUMMARY OF MCA PARAMETER VALUES, 3-49

- MEDIA
 - AVAILABLE MEMORY PER I/O MEDIA FOR 12K CONFIGURATION, 4-7

- MEMBER
 - ALTER MEMBER (MALTER), 3-60
 - ALTER STATUS OF MEMBER (MALTER), 3-19
 - END MEMBER (ENDM), 3-60
 - EXIT AND RETURN CODES FOR MEMBER INDEX EXITS, 3-75
 - FIELDS OF FIRST ITEM IN MEMBER INDEX, B-2
 - FIELDS OF LAST ITEM IN MEMBER INDEX, B-3
 - FIELDS OF MEMBER INDEX ITEMS, B-2
 - MEMBER STATEMENT, 4-19
 - MEMBER STATEMENTS, 4-44
 - PROCESSING A PARTITIONED SEQUENTIAL FILE BY MEMBER NAMES, 4-58
 - SET MEMBER (SETM), 3-59
 - SET PROCESSING TO BEGINNING OF SPECIFIED MEMBER (SETM), 3-17

- MEMBER-LENGTH
 - MEMBER-LENGTH PARAMETER, 4-20

- MEMBER-NAME
 - MEMBER-NAME PARAMETER, 4-19, 4-44

INDEX

MEMBERS
 LOADING SELECTED MEMBERS, 4-58
 UNLOADING SELECTED MEMBERS, 4-58

MEMEBER
 END PROCESSING OF CURRENT MEMEBER
 (ENDM), 3-18

MEMORY
 AVAILABLE MEMORY PER I/O MEDIA FOR
 12K CONFIGURATION, 4-7
 BLOCK AND RECORD SIZES WITHIN 12K
 MEMORY, 4-6
 LOGICAL I/O C MEMORY
 REQUIREMENTS, 3-64

MESSAGES
 CONSOLE TYPEWRITER PAUSE CODES AND
 MESSAGES FOR LOGICAL I/O C, 3-83
 JOB CONTROL FILE CONSOLE TYPEWRITER
 MESSAGES, 4-85
 OPERATOR CONTROL AND MESSAGES FOR
 FILE SUPPORT C, 4-70
 TYPEWRITER MESSAGES FOR CONDITIONS
 RELATED TO NON-MASS STORAGE
 FILES, 4-83
 TYPEWRITER MESSAGES SPECIFIC TO
 FILE SUPPORT C, 4-86

MINIMUM
 MINIMUM DEVICE REQUIREMENTS FOR
 MASS STORAGE FILE
 ORGANIZATIONS, 4-39

MIOC
 INPUT/OUTPUT CONTROL MACRO ROUTINE
 (MIOC), 3-26
 MASS STORAGE INPUT/OUTPUT CONTROL
 MACRO ROUTINE (MIOC), 3-2
 MIOC MACRO CALL, 3-26
 MIOC RESTRICTIONS, 3-67
 MIOC SEGMENTATION, 3-65, 3-66
 PARAMETERS OF MIOC MACRO CALL,
 3-27, 3-36
 PHYSICAL I/O C RELATIONSHIPS WITH
 MIOC, 3-69
 SUMMARY OF MIOC PARAMETER
 VALUES, 3-36

MIXED
 MIXED FILE ORGANIZATIONS, 4-60

MLCA
 COMMUNICATION AREA SERVICE MACRO
 ROUTINES (MLCA AND MUCA), 3-50
 COMMUNICATION AREA SERVICE MACRO
 ROUTINES (MLCA AND MUCA), 3-2
 COMMUNICATION AREA SERVICE MACRO
 CALLS (MLCA AND MUCA), D-3, D-9
 MASS STORAGE LOAD COMMUNICATION
 AREA MACRO CALL (MLCA), 3-51
 MNEMONIC DESIGNATORS FOR MLCA AND
 MUCA, D-9

MNEMONIC
 MNEMONIC DESIGNATORS FOR
 COMMUNICATION AREA FIELDS, 3-52
 MNEMONIC DESIGNATORS FOR MLCA AND
 MUCA, D-9

MOD
 MOD 1 (MSR) OPERATING SYSTEM, 4-68
 MOD 1 (TR) OPERATING SYSTEM, 4-70

MODE
 ACTION MACRO CALLS FOR EACH FILE
 TYPE IN EACH PROCESSING MODE, 3-6
 ADDRESS MODE, 3-69, D-12
 INPUT-ONLY PROCESSING MODE, 3-4
 INPUT/OUTPUT PROCESSING MODE, 3-4
 MODE PARAMETER, 4-41
 OUTPUT-ONLY PROCESSING MODE, 3-4

MODES
 FILE PROCESSING MODES, 3-4

MPCA
 COMMUNICATION AREA MACRO CALL
 (MPCA), D-6
 COMMUNICATION AREA MACRO ROUTINE
 (MPCA), D-3
 CONSIDERATIONS FOR MPCA PARAMETER
 SPECIFICATION, D-14
 MPCA CONTROL UNIT CURRENT ADDRESS
 AND STATUS FIELD, D-16
 PARAMETERS OF MPCA MACRO
 CALL, D-6.1
 PARAMETERS OF THE MPCA MACRO
 CALL, D-6

MPIOC
 CONSIDERATIONS FOR MPIOC PARAMETER
 SPECIFICATION, D-14
 CONTROL MACRO CALL (MPIOC), D-4
 CONTROL MACRO ROUTINE (MPIOC), D-3
 PARAMETERS OF MPIOC MACRO
 CALL, D-4
 SUFFIX OF RELATED MPIOC, D-15

MSCLOS
 CLOSE (MSCLOS), 3-56

MSDEL
 DELETE (MSDEL), 3-59

MSEEK
 SEEK (MSEEK), 3-62

MSGET
 GET (MSGET), 3-56
 SUMMARY OF MSGET MACRO FUNCTIONS
 FOR DIRECT ACCESS FILES, 3-15

MSINS
 INSERT (MSINS), 3-58

MSOPEN
 OPEN (MSOPEN), 3-55

MSPUT
 PUT (MSPUT), 3-59

MSR
 MOD 1 (MSR) OPERATING SYSTEM, 4-68

MSREL
 RELEASE COMPLETE FILE TO UNUSED
 STATE (MSREL), 3-19
 RELEASE (MSREL), 3-61

MSREP
 REPLACE (MSREP), 3-58

MUCA
 COMMUNICATION AREA SERVICE MACRO
 ROUTINES (MLCA AND MUCA), 3-50
 COMMUNICATION AREA SERVICE MACRO
 ROUTINES (MLCA AND MUCA), 3-2
 COMMUNICATION AREA SERVICE MACRO
 CALLS (MLCA AND MUCA), D-3, D-9
 MASS STORAGE UNLOAD COMMUNICATION
 AREA MACRO CALL (MUCA), 3-51
 MNEMONIC DESIGNATORS FOR MLCA AND
 MUCA, D-9

MULTIFIELD
 MULTIFIELD KEYS, E-6

MULTIVOLUME
 MULTIVOLUME FILE PROCESSING, C-3

NAMES
 PROCESSING A PARTITIONED SEQUENTIAL
 FILE BY MEMBER NAMES, 4-58

INDEX

NON-MASS
 CONDITIONS RELATED TO NON-MASS
 STORAGE FILE, 4-71
 TYPEWRITER MESSAGES FOR CONDITIONS
 RELATED TO NON-MASS STORAGE
 FILES, 4-83

NONNUMERIC
 NONNUMERIC ITEM KEYS, E-5

NUMBER
 NUMBER OF FUNCTIONS PERFORMED, 4-6
 PRIME NUMBER DIVISION, E-1

OMISSION
 OMISSION OF CONSECUTIVE PARAMETERS
 FROM MACRO CALL, 3-25
 OMISSION OF SINGLE PARAMETER FROM
 MACRO CALL, 3-25

OMITTING
 OMITTING ITEMS FROM THE OUTPUT
 FILE, 4-62

OPEN
 OPEN (MSOPEN), 3-55

OPENING
 OPENING AN INDEXED SEQUENTIAL
 FILE, 3-8
 OPENING DIRECT ACCESS FILES, 3-9
 OPENING FILES, 3-6
 OPENING PARTITIONED SEQUENTIAL
 FILES, 3-8
 OPENING SEQUENTIAL FILES, 3-6

OPERATING
 CONSOLE TYPEWRITER OPERATING
 PROCEDURES, 3-82
 CONTROL PANEL OPERATING
 PROCEDURES, 3-77
 MOD 1 (MSR) OPERATING SYSTEM, 4-68
 MOD 1 (TR) OPERATING SYSTEM, 4-70
 OPERATING PROCEDURES FOR FILE
 SUPPORT C, 4-68
 OPERATING PROCEDURES FOR LOGICAL
 I/O C, 3-77
 OPERATING PROCEDURES FOR PHYSICAL
 I/O C, D-21

OPERATION
 JOB CONTROL FOR A SINGLE
 OPERATION, 4-8
 OWN-CODING CONSIDERATIONS FOR
 TAPE-RESIDENT OPERATION, 4-61

OPERATIONS
 JOB CONTROL FOR A SEQUENCE OF
 OPERATIONS, 4-9

OPERATOR
 OPERATOR CONTROL AND MESSAGES FOR
 FILE SUPPORT C, 4-70
 OPERATOR CONTROL WITH CONSOLE
 TYPEWRITER, 4-83
 OPERATOR CONTROL WITH CONTROL
 PANEL, 4-70

OPTIMIZATION
 OPTIMIZATION, C-18

OPTIMIZING
 OPTIMIZING ACCESS TIME, C-18
 OPTIMIZING STORAGE CAPACITY, C-20

OPTIMUM
 EXAMPLE-SUMMARY OF OPTIMUM
 POINTS, C-23
 OPTIMUM RECORD SIZE - TYPE 261 OR
 TYPE 262 DISK FILES, C-7
 OPTIMUM RECORD SIZE - TYPES 258 259
 OR 259A DISK PACK DRIVES, C-5

OPTION
 SEQUENTIAL FILE USING PARTITIONING
 OPTION, B-4

ORGANIZATION
 DIRECT ACCESS FILE
 ORGANIZATION, 2-20
 FILE ORGANIZATION, 2-23
 FILE ORGANIZATION CONVENTIONS, 2-8
 INDEXED SEQUENTIAL FILE
 ORGANIZATION, 2-9
 PROGRAM ORGANIZATION, 3-22, 3-64
 SEQUENTIAL FILE ORGANIZATION, 2-8

ORGANIZATIONS
 MINIMUM DEVICE REQUIREMENTS FOR
 MASS STORAGE FILE
 ORGANIZATIONS, 4-39
 MIXED FILE ORGANIZATIONS, 4-60

OUTPUT
 OMITTING ITEMS FROM THE OUTPUT
 FILE, 4-62

OUTPUT-ONLY
 OUTPUT-ONLY PROCESSING MODE, 3-4

OVERFLOW
 BUCKET SIZE AND OVERFLOW, C-10
 CYLINDER OVERFLOW AS PERCENTAGE OF
 DATA AREA, C-13
 ENTRANCE TO GENERAL OVERFLOW, 4-63
 GENERAL OVERFLOW PARAMETER, 4-13
 OVERFLOW AREAS, 2-12, 2-23
 OVERFLOW PARAMETER, 4-18
 OVERFLOW PROBABILITIES, C-11
 TYPES OF OVERFLOW, C-17

OVERFLOW-SIZE
 CYLINDER OVERFLOW-SIZE
 PARAMETER, 4-16

OWN-CODING
 OWN-CODING COMMUNICATION WITH
 LOAD-UNLOAD FUNCTION, 4-62
 OWN-CODING CONSIDERATIONS, 4-60
 OWN-CODING CONSIDERATIONS FOR
 TAPE-RESIDENT OPERATION, 4-61
 STRUCTURE OF OWN-CODING
 ROUTINE, 4-61

PACK
 DISK PACK CYLINDER CONCEPT - TYPE
 259 DISK PACK DRIVES, 2-2
 OPTIMUM RECORD SIZE - TYPES 258 259
 OR 259A DISK PACK DRIVES, C-5

PADDING
 PADDING ITEMS, 4-66

PADDING-CHARACTER
 PADDING-CHARACTER PARAMETER, 4-41

PANEL
 CONTROL PANEL OPERATING
 PROCEDURES, 3-77
 OPERATOR CONTROL WITH CONTROL
 PANEL, 4-70

PARAMETER
 BANNER-CHARACTER PARAMETER, 4-40
 BLOCK-SIZE PARAMETER, 4-16
 BUCKET-ADDRESSING PARAMETER, 4-42
 BUCKET-SIZE PARAMETER, 4-16
 CONSIDERATIONS FOR MPCA PARAMETER
 SPECIFICATION, D-14
 CONSIDERATIONS FOR MPIOC PARAMETER
 SPECIFICATION, D-14
 CYLINDER OVERFLOW-SIZE
 PARAMETER, 4-16
 DEVICE-ADDRESS PARAMETER, 4-14,
 4-30, 4-38, 4-52, 4-54

INDEX

DEVICE-TYPE PARAMETER, 4-37, 4-54
 EXPIRATION-DATE CHECK
 PARAMETER, 4-29
 FILE-EXPIRATION DATA
 PARAMETER, 4-14
 FILE-NAME PARAMETER, 4-12,
 4-29, 4-37
 FILE-ORGANIZATION PARAMETER, 4-12
 FROM PARAMETER, 4-19
 GENERAL OVERFLOW PARAMETER, 4-13
 IMBED PARAMETER, 4-43
 INDEX-SIZE PARAMETER, 4-16
 IN/OJT PARAMETER, 4-37
 ITEM-KEY PARAMETER, 4-13
 ITEM-LENGTH PARAMETER, 4-15, 4-39
 LOW-MEMORY-ADDRESS PARAMETER, 4-45
 MASTER/CYLINDER INDEX
 PARAMETER, 4-18
 MEMBER-LENGTH PARAMETER, 4-20
 MEMBER-NAME PARAMETER, 4-19, 4-44
 MODE PARAMETER, 4-41
 OMISSION OF SINGLE PARAMETER FROM
 MACRO CALL, 3-25
 OVERFLOW PARAMETER, 4-18
 PADDING-CHARACTER PARAMETER, 4-41
 PARITY PARAMETER, 4-40
 PASSWORD PARAMETER, 4-13,
 4-30, 4-41
 PROGRAM-SEGMENT-NAME
 PARAMETER, 4-45
 PROTECTION-STATUS PARAMETER,
 4-14, 4-42
 RECORD-LENGTH PARAMETER,
 4-15, 4-39
 RELEASE PARAMETER, 4-43
 REPORT-NUMBER PARAMETER, 4-43
 STRING-SIZE PARAMETER, 4-17
 SUMMARY OF MCA PARAMETER
 VALUES, 3-49
 SUMMARY OF MIOC PARAMETER
 VALUES, 3-36
 TO PARAMETER, 4-19
 VOLUME-NAME PARAMETER, 4-17,
 4-29, 4-52

PARAMETERS
 DEVICE-ADDRESS PARAMETERS, 4-20
 OMISSION OF CONSECUTIVE PARAMETERS
 FROM MACRO CALL, 3-25
 PARAMETERS OF MCA MACRO CALL, 3-39
 PARAMETERS OF MIOC MACRO CALL,
 3-27, 3-36
 PARAMETERS OF MPCA MACRO
 CALL, D-6.1
 PARAMETERS OF MPIOC MACRO
 CALL, D-4
 PARAMETERS OF THE MPCA MACRO
 CALL, D-6
 SPECIAL CONSIDERATIONS FOR
 SPECIFYING PARAMETERS, D-13

PARITY
 PARITY PARAMETER, 4-40

PARTITIONED
 ACTION MACRO CALLS (FOR PARTITIONED
 SEQUENTIAL FILES ONLY), 3-17
 CLOSING SEQUENTIAL AND PARTITIONED
 SEQUENTIAL FILES, 3-10
 LOADING A PARTITIONED SEQUENTIAL
 FILE, 4-58
 OPENING PARTITIONED SEQUENTIAL
 FILES, 3-8
 PARTITIONED SEQUENTIAL FILES, 4-58
 PROCESSING A PARTITIONED SEQUENTIAL
 FILE BY MEMBER NAMES, 4-58
 PUTTING ITEMS TO SEQUENTIAL AND
 PARTITIONED SEQUENTIAL
 FILES, 3-17
 REPLACING ITEMS IN SEQUENTIAL AND
 PARTITIONED SEQUENTIAL
 FILES, 3-16

RETRIEVING ITEMS IN SEQUENTIAL AND
 PARTITIONED SEQUENTIAL
 FILES, 3-11
 UNLOADING A PARTITIONED SEQUENTIAL
 FILE, 4-58

PARTITIONING
 PARTITIONING A SEQUENTIAL
 FILE, B-1
 SEQUENTIAL FILE USING PARTITIONING
 OPTION, B-4

PASSWORD
 PASSWORD PARAMETER, 4-13,
 4-30, 4-41
 PASSWORD PROTECTION, F-2

PAUSE
 CONSOLE TYPEWRITER PAUSE CODES AND
 MESSAGES FOR LOGICAL I/O C, 3-83

PERCENTAGE
 CYLINDER OVERFLOW AS PERCENTAGE OF
 DATA AREA, C-13

PERIPHERAL
 FIXED PERIPHERAL ADDRESS
 ASSIGNMENT, D-13
 PERIPHERAL ADDRESS
 ASSIGNMENT, D-14
 PERIPHERAL ADDRESS ASSIGNMENT AND
 RWC CONFIGURATION
 CONSIDERATIONS, D-13
 PERIPHERAL CONDITIONS, 4-71, 4-83
 VARIABLE PERIPHERAL ADDRESS
 ASSIGNMENT, D-13

PHYSICAL
 DETAILED DESCRIPTION OF PHYSICAL
 I/O C MACRO ROUTINES, D-3
 LANGUAGE ELEMENTS OF PHYSICAL I/O
 C, D-4
 OPERATING PROCEDURES FOR PHYSICAL
 I/O C, D-21
 PHYSICAL BACKUP, 2-27
 PHYSICAL I/O C, D-1
 PHYSICAL I/O C RELATIONSHIPS WITH
 MCA, 3-69
 PHYSICAL I/O C RELATIONSHIPS WITH
 MIOC, 3-69
 PROGRAMMER'S PREPARATION
 INFORMATION FOR PHYSICAL I/O
 C, D-12
 USE OF PHYSICAL I/O C, D-1

PLUS
 RANDOM PLUS SEQUENTIAL FILES, C-2

POINTS
 EXAMPLE-SUMMARY OF OPTIMUM
 POINTS, C-23

POSITION
 USING THE ITEM POSITION OF A
 DELETED ITEM, 2-22

PREFIX
 FILE PREFIX, D-15

PREPARATION
 FORMATTING AND VOLUME
 PREPARATION, 2-3
 PROGRAMMER'S PREPARATION
 INFORMATION FOR FILE SUPPORT
 C, 5-57
 PROGRAMMER'S PREPARATION
 INFORMATION FOR LOGICAL I/O
 C, 3-64
 PROGRAMMER'S PREPARATION
 INFORMATION FOR PHYSICAL I/O
 C, D-12

INDEX

- PRIME
 - PRIME DATA AREA, 2-11
 - PRIME NUMBER DIVISION, E-1
- PRINTER
 - UNLOADING MASS STORAGE FILES ONTO PRINTER, 4-67
- PRINT-IMAGE
 - PRINT-IMAGE FILES, G-2
- PROCEDURE
 - RE-EXECUTION OF CORRECTION PROCEDURE, D-20
- PROCEDURES
 - BACKUP PROCEDURES, 2-27
 - CONSOLE TYPEWRITER OPERATING PROCEDURES, 3-82
 - CONTROL PANEL OPERATING PROCEDURES, 3-77
 - OPERATING PROCEDURES FOR FILE SUPPORT C, 4-68
 - OPERATING PROCEDURES FOR LOGICAL I/O C, 3-77
 - OPERATING PROCEDURES FOR PHYSICAL I/O C, D-21
- PROCESSED
 - ASSIGNMENT OF FILES TO BE PROCESSED CONCURRENTLY, C-4
- PROCESSING
 - ACTION MACRO CALLS FOR EACH FILE TYPE IN EACH PROCESSING MODE, 3-6
 - ACTION MACRO PROCESSING FUNCTIONS, 3-4
 - DIRECTLY PROCESSING AN INDEXED SEQUENTIAL FILE, 2-13
 - END PROCESSING OF CURRENT MEMBER (ENDM), 3-18
 - FILE PROCESSING FUNCTIONS, 2-27
 - FILE PROCESSING MODES, 3-4
 - BACKGROUND/BACKGROUND PROCESSING OF FILE SUPPORT C, 4-1
 - INPUT-ONLY PROCESSING MODE, 3-4
 - INPUT/OUTPUT PROCESSING MODE, 3-4
 - MULTIVOLUME FILE PROCESSING, C-3
 - OUTPUT-ONLY PROCESSING MODE, 3-4
 - PROCESSING A PARTITIONED SEQUENTIAL FILE BY MEMBER NAMES, 4-58
 - PROCESSING CONVENTIONS, 2-26
 - SEQUENTIAL OR DIRECT PROCESSING, 2-26
 - SET PROCESSING TO BEGINNING OF SPECIFIED MEMBER (SETM), 3-17
 - SETTING PROCESSING TO A SPECIFIED LOCATION, 3-22
- PROGRAM
 - FILE SUPPORT C PROGRAM, 1-5
 - LOGICAL I/O C PROGRAM, 1-4
 - PROGRAM ORGANIZATION, 3-22, 3-64
 - PROGRAM SEGMENT LOADING, 3-68
- PROGRAM-SEGMENT-NAME
 - PROGRAM-SEGMENT-NAME PARAMETER, 4-45
- PROGRAMMER'S
 - PROGRAMMER'S PREPARATION INFORMATION FOR FILE SUPPORT C, 5-57
 - PROGRAMMER'S PREPARATION INFORMATION FOR LOGICAL I/O C, 3-64
 - PROGRAMMER'S PREPARATION INFORMATION FOR PHYSICAL I/O C, D-12
- PROTECTION
 - DEVICE PROTECTION, D-14
 - FILE PROTECTION, F-1
 - MASS STORAGE FILE PROTECTION, F-1
 - PASSWORD PROTECTION, F-2
 - PROTECTION DURING ALLOCATE, 4-70
 - PROTECTION DURING DEALLOCATE, 4-70
 - PROTECTION DURING LOAD-UNLOAD, 4-70
 - PROTECTION DURING MAP, 4-70
 - PROTECTION OF MASS STORAGE DURING EXECUTION OF FILE SUPPORT C, 4-70
 - WRITE PROTECTION, F-1
- PROTECTION-STATUS
 - PROTECTION-STATUS PARAMETER, 4-14, 4-42
- RADIX
 - RADIX CONVERSION, E-4
- RANDOM
 - RANDOM PLUS SEQUENTIAL FILES, C-2
 - RANDOM VERSUS SEQUENTIAL FILES, C-2
- RANDOMIZING
 - RANDOMIZING ADDRESSING, E-1
 - RANDOMIZING TECHNIQUES, E-1
- READ
 - READ ACTION, D-2
 - READ ACTION MACRO CALL, D-10
 - READ ACTION MACRO ROUTINE, D-17
 - TYPE OF READ OR WRITE (TRW), D-15
- READ/WRITE
 - READ/WRITE CHANNEL UTILIZATION, 3-70, D-12
- RECORD
 - BLOCK AND RECORD SIZES WITHIN 12K MEMORY, 4-6
 - OPTIMUM RECORD SIZE - TYPE 261 OR TYPE 262 DISK FILES, C-7
 - OPTIMUM RECORD SIZE - TYPES 258 259 OR 259A DISK PACK DRIVES, C-5
- RECORD-LENGTH
 - RECORD-LENGTH PARAMETER, 4-15, 4-39
- RECORDS
 - BOOTSTRAP RECORDS, 2-3
 - DATA RECORDS, 4-65
 - HANDLING TRACK LINKING RECORDS, D-18
 - RELATIONSHIP BETWEEN ITEMS AND RECORDS, 2-5
 - RELATIONSHIP BETWEEN ITEMS RECORDS AND BLOCKS, 2-5
 - RELATIONSHIP BETWEEN ITEMS RECORDS BLOCKS AND BUCKETS, 2-24
 - TRACK-LINKING RECORDS, 2-8
- REGISTER
 - ADDRESS REGISTER CONTENTS AT TIME OF ERROR EXIT (EDF), D-20
- REGISTERS
 - INDEX REGISTERS, 3-69
 - USE OF INDEX REGISTERS, D-13
- RE-EXECUTION
 - RE-EXECUTION OF CORRECTION PROCEDURE, D-20
- RELATED
 - CONDITIONS RELATED TO NON-MASS STORAGE FILE, 4-71
 - FILE RELATED CONDITIONS, 4-72
 - SUFFIX OF RELATED MPIOC, D-15
 - TYPEWRITER MESSAGES FOR CONDITIONS RELATED TO NON-MASS STORAGE

INDEX

FILES, 4-83

RELATIONSHIP
 RELATIONSHIP BETWEEN ITEMS AND RECORDS, 2-5
 RELATIONSHIP BETWEEN ITEMS OF THE MASTER AND CYLINDER INDEX, 2-14
 RELATIONSHIP BETWEEN ITEMS RECORDS AND BLOCKS, 2-5
 RELATIONSHIP BETWEEN ITEMS RECORDS BLOCKS AND BUCKETS, 2-24
 RELATIONSHIP BETWEEN STRING INDEX ITEMS AND THE DATA AREA OF A CYLINDER, 2-15

RELATIONSHIPS
 PHYSICAL I/O C RELATIONSHIPS WITH MCA, 3-69
 PHYSICAL I/O C RELATIONSHIPS WITH MIOC, 3-69

RELEASE
 RELEASE COMPLETE FILE TO UNUSED STATE (MSREL), 3-19
 RELEASE PARAMETER, 4-43
 RELEASE (MSREL), 3-61

REPLACE
 REPLACE (MSREP), 3-58

REPLACING
 REPLACING ITEMS IN DIRECT ACCESS FILES, 3-16
 REPLACING ITEMS IN FILES, 3-15
 REPLACING ITEMS IN INDEXED SEQUENTIAL FILES, 3-16
 REPLACING ITEMS IN SEQUENTIAL AND PARTITIONED SEQUENTIAL FILES, 3-16

REPORT-NUMBER
 REPORT-NUMBER PARAMETER, 4-43

RESTORE
 RESTORE ACTION, D-2
 RESTORE ACTION MACRO CALL, D-11
 RESTORE ACTION MACRO ROUTINE, D-18

RESTRICTIONS
 MIOC RESTRICTIONS, 3-67

RETRIEVING
 RETRIEVING ITEMS IN DIRECT ACCESS FILES, 3-13
 RETRIEVING ITEMS IN FILES, 3-11
 RETRIEVING ITEMS IN INDEXED SEQUENTIAL FILES, 3-12
 RETRIEVING ITEMS IN SEQUENTIAL AND PARTITIONED SEQUENTIAL FILES, 3-11

RETURN
 EXIT AND RETURN CODES FOR DATA EXITS, 3-75
 EXIT AND RETURN CODES FOR DEVICE EXITS, 3-76
 EXIT AND RETURN CODES FOR MEMBER INDEX EXITS, 3-75
 EXIT AND RETURN CODES FOR VOLUME DIRECTORY EXITS, 3-73

ROUTINE
 COMMUNICATION AREA MACRO ROUTINE (MPCA), D-3
 CONTROL MACRO ROUTINE (MPIOC), D-3
 CORRECTIVE ACTION FOR USER'S ERROR ROUTINE, D-21
 FILE DESCRIPTION MACRO ROUTINE (MCA), 3-2, 3-38
 INPUT/OUTPUT CONTROL MACRO ROUTINE (MIOC), 3-26
 LOKDEV ACTION MACRO ROUTINE, D-18

MASS STORAGE INPUT/OUTPUT CONTROL MACRO ROUTINE (MIOC), 3-2
 READ ACTION MACRO ROUTINE, D-17
 RESTORE ACTION MACRO ROUTINE, D-18
 STRUCTURE OF OWN-CODING ROUTINE, 4-61
 USER'S UNCORRECTABLE ERROR ROUTINE, D-19
 USER'S UNCORRECTABLE ERROR ROUTINE ENTRANCE (EAD), D-15
 VERIFY ACTION MACRO ROUTINE, D-18
 WAIT ACTION MACRO ROUTINE, D-18
 WRITE ACTION MACRO ROUTINE, D-17

ROUTINES
 ACTION MACRO ROUTINES, 3-2, D-4
 COMMUNICATION AREA SERVICE MACRO ROUTINES (MLCA AND MUCA), 3-50
 COMMUNICATION AREA SERVICE MACRO ROUTINES (MLCA AND MUCA), 3-2
 CONSIDERATIONS FOR ACTION MACRO ROUTINES, D-17
 DETAILED DESCRIPTION OF PHYSICAL I/O C MACRO ROUTINES, D-3
 SUMMARY OF LOGICAL I/O C MACRO ROUTINES, 3-2, 3-3

RWC
 PERIPHERAL ADDRESS ASSIGNMENT AND RWC CONFIGURATION CONSIDERATIONS, D-13

SAMPLE
 LISTING OF SAMPLE UNLOAD-TO-PRINTER FUNCTION, 4-69

SEEK
 SEEK ACTION, D-2
 SEEK ACTION MACRO CALL, D-11
 SEEK (MSEEK), 3-62

SEEKING
 SEEKING A DESIRED CYLINDER, 3-22

SEGMENT
 PROGRAM SEGMENT LOADING, 3-68

SEGMENTATION
 MIOC SEGMENTATION, 3-65, 3-66

SELECTED
 LOADING SELECTED MEMBERS, 4-58
 UNLOADING SELECTED MEMBERS, 4-58

SEQUENCE
 ITEM SEQUENCE, C-17
 JOB CONTROL FOR A SEQUENCE OF OPERATIONS, 4-9
 KEY OUT OF SEQUENCE, 4-63

SEQUENTIAL
 ACTION MACRO CALLS (FOR PARTITIONED SEQUENTIAL FILES ONLY), 3-17
 ALLOCATING AN INDEXED SEQUENTIAL FILE, 4-59
 CLOSING INDEXED SEQUENTIAL AND DIRECT ACCESS FILES, 3-10
 CLOSING SEQUENTIAL AND PARTITIONED SEQUENTIAL FILES, 3-10
 DIRECTLY PROCESSING AN INDEXED SEQUENTIAL FILE, 2-13
 EXAMPLE-OPTIMIZATION FOR AN INDEXED SEQUENTIAL FILE, C-23
 INDEXED SEQUENTIAL, 3-72
 INDEXED SEQUENTIAL FILE CONSIDERATIONS, C-17
 INDEXED SEQUENTIAL FILE ORGANIZATION, 2-9
 INDEXED SEQUENTIAL FILES, 4-59
 INSERTING ITEMS IN INDEXED SEQUENTIAL FILES, 3-20
 LOADING A PARTITIONED SEQUENTIAL

- FILE, 4-58
 - LOADING AN INDEXED SEQUENTIAL FILE, 4-59
 - OPENING AN INDEXED SEQUENTIAL FILE, 3-8
 - OPENING PARTITIONED SEQUENTIAL FILES, 3-8
 - OPENING SEQUENTIAL FILES, 3-6
 - PARTITIONED SEQUENTIAL FILES, 4-58
 - PARTITIONING A SEQUENTIAL FILE, B-1
 - PROCESSING A PARTITIONED SEQUENTIAL FILE BY MEMBER NAMES, 4-58
 - PUTTING ITEMS TO SEQUENTIAL AND PARTITIONED SEQUENTIAL FILES, 3-17
 - RANDOM PLUS SEQUENTIAL FILES, C-2
 - RANDOM VERSUS SEQUENTIAL FILES, C-2
 - REPLACING ITEMS IN INDEXED SEQUENTIAL FILES, 3-16
 - REPLACING ITEMS IN SEQUENTIAL AND PARTITIONED SEQUENTIAL FILES, 3-16
 - RETRIEVING ITEMS IN INDEXED SEQUENTIAL FILES, 3-12
 - RETRIEVING ITEMS IN SEQUENTIAL AND PARTITIONED SEQUENTIAL FILES, 3-11
 - SEQUENTIAL FILE CONSIDERATIONS, C-4
 - SEQUENTIAL FILE ORGANIZATION, 2-8
 - SEQUENTIAL FILE USING PARTITIONING OPTION, B-4
 - SEQUENTIAL FILES, 5-57
 - SEQUENTIAL OR DIRECT PROCESSING, 2-26
 - UNLOADING A PARTITIONED SEQUENTIAL FILE, 4-58
 - UNLOADING AN INDEXED SEQUENTIAL FILE, 4-60
- SERVICE**
- COMMUNICATION AREA SERVICE MACRO ROUTINES (MLCA AND MUCA), 3-50
 - COMMUNICATION AREA SERVICE MACRO ROUTINES (MLCA AND MUCA), 3-2
 - COMMUNICATION AREA SERVICE MACRO CALLS (MLCA AND MUCA), D-3, D-9
- SET**
- SET LOCATION (SETL), 3-62
 - SET MEMBER (SETM), 3-59
 - SET PROCESSING TO BEGINNING OF SPECIFIED MEMBER (SETM), 3-17
- SETL**
- SET LOCATION (SETL), 3-62
- SETM**
- SET MEMBER (SETM), 3-59
 - SET PROCESSING TO BEGINNING OF SPECIFIED MEMBER (SETM), 3-17
- SETTING**
- SETTING PROCESSING TO A SPECIFIED LOCATION, 3-22
- SIZE**
- BLOCK SIZE, C-2
 - BUCKET SIZE AND OVERFLOW, C-10
 - OPTIMUM RECORD SIZE - TYPE 261 OR TYPE 262 DISK FILES, C-7
 - OPTIMUM RECORD SIZE - TYPES 258 259 OR 259A DISK PACK DRIVES, C-5
 - SIZE STATEMENT, 4-15
- SIZES**
- BLOCK AND RECORD SIZES WITHIN 12K MEMORY, 4-6
- SPACE**
- INSUFFICIENT SPACE, 4-62
- SPECIFIC**
- CONDITIONS SPECIFIC TO FILE SUPPORT C, 4-77
 - TYPEWRITER MESSAGES SPECIFIC TO FILE SUPPORT C, 4-86
- SPECIFICATION**
- CONSIDERATIONS FOR MPCA PARAMETER SPECIFICATION, D-14
 - CONSIDERATIONS FOR MPIOC PARAMETER SPECIFICATION, D-14
 - ITEM KEY SPECIFICATION, 3-71
- SPECIFIED**
- SET PROCESSING TO BEGINNING OF SPECIFIED MEMBER (SETM), 3-17
 - SETTING PROCESSING TO A SPECIFIED LOCATION, 3-22
- SPECIFYING**
- SPECIAL CONSIDERATIONS FOR SPECIFYING PARAMETERS, D-13
- SQUARE**
- SQUARE ENFOLD AND EXTRACT, E-2
- STATUS**
- ALTER STATUS OF MEMBER (MALTER), 3-19
 - CONTROL UNIT CURRENT ADDRESS AND STATUS, D-16
 - DATA ITEM STATUS CHARACTER, 2-20, 2-25
 - MPCA CONTROL UNIT CURRENT ADDRESS AND STATUS FIELD, D-16
- STORAGE**
- COMPRISING BETWEEN ACCESS TIME AND STORAGE CAPACITY, C-22
 - CONDITIONS RELATED TO NON-MASS STORAGE FILE, 4-71
 - LOADING FROM MASS STORAGE TO MASS STORAGE, 4-59
 - MASS STORAGE FILE PROTECTION, F-1
 - MASS STORAGE INPUT/OUTPUT CONTROL MACRO ROUTINE (MIOC), 3-2
 - MASS STORAGE LOAD COMMUNICATION AREA MACRO CALL (MLCA), 3-51
 - MASS STORAGE UNLOAD COMMUNICATION AREA MACRO CALL (MUCA), 3-51
 - MINIMUM DEVICE REQUIREMENTS FOR MASS STORAGE FILE ORGANIZATIONS, 4-39
 - OPTIMIZING STORAGE CAPACITY, C-20
 - PROTECTION OF MASS STORAGE DURING EXECUTION OF FILE SUPPORT C, 4-70
 - TYPEWRITER MESSAGES FOR CONDITIONS RELATED TO NON-MASS STORAGE FILES, 4-83
 - UNLOADING MASS STORAGE FILES ONTO PRINTER, 4-67
- STRING**
- DELETION OF AN ITEM FROM A STRING, 2-21
 - INSERTION OF ITEMS INTO A STRING, 2-16
 - RELATIONSHIP BETWEEN STRING INDEX ITEMS AND THE DATA AREA OF A CYLINDER, 2-15
- STRING-SIZE**
- STRING-SIZE PARAMETER, 4-17
- STRUCTURE**
- DATA STRUCTURE, 2-9
 - FILE STRUCTURE, 2-11
 - STRUCTURE OF OWN-CODING ROUTINE, 4-61

INDEX

SUBSYSTEM
EQUIPMENT REQUIREMENTS FOR DATA
MANAGEMENT SUBSYSTEM, 1-7
JOB CONTROL LANGUAGE FOR DATA
MANAGEMENT SUBSYSTEM, 1-6

SUFFIX
SUFFIX CHARACTER, D-14
SUFFIX OF RELATED MPIOC, D-15

SUMMARY
SUMMARY OF ACTION MACRO CALL
CODING, 3-63
SUMMARY OF JOB CONTROL STATEMENTS
FOR ALLOCATE FUNCTION, 4-24
SUMMARY OF JOB CONTROL STATEMENTS
FOR ALLOCATION FUNCTION, 4-25
SUMMARY OF JOB CONTROL STATEMENTS
FOR DEALLOCATE FUNCTION,
4-33, 4-34
SUMMARY OF JOB CONTROL STATEMENTS
FOR LOAD AND UNLOAD
FUNCTIONS, 4-48
SUMMARY OF JOB CONTROL STATEMENTS
FOR LOAD/UNLOAD FUNCTIONS, 4-49
SUMMARY OF JOB CONTROL STATEMENTS
FOR MAP FUNCTION, 4-55, 4-56
SUMMARY OF LOGICAL I/O C MACRO
ROUTINES, 3-2, 3-3
SUMMARY OF MCA PARAMETER
VALUES, 3-49
SUMMARY OF MIOC PARAMETER
VALUES, 3-36
SUMMARY OF MSGET MACRO FUNCTIONS
FOR DIRECT ACCESS FILES, 3-15

SUPPORT
CONDITIONS SPECIFIC TO FILE SUPPORT
C, 4-77
FILE SUPPORT C, 4-1
FILE SUPPORT C HALTS, 4-78
FILE SUPPORT C DIAGNOSTICS FOR 5040
HALT, 4-74
BACKGROUND/BACKGROUND PROCESSING OF
FILE SUPPORT C, 4-1
FORMAT OF FILE SUPPORT C EXECUTE
STATEMENT, 4-8
FUNCTIONS OF FILE SUPPORT C, 4-2
GENERAL DESCRIPTION OF FILE SUPPORT
C, 4-1
JOB CONTROL LANGUAGE FOR FILE
SUPPORT C, 4-8
LOADING FILE SUPPORT C, 4-68
OPERATING PROCEDURES FOR FILE
SUPPORT C, 4-68
OPERATOR CONTROL AND MESSAGES FOR
FILE SUPPORT C, 4-70
PROGRAMMER'S PREPARATION
INFORMATION FOR FILE SUPPORT
C, 5-57
PROTECTION OF MASS STORAGE DURING
EXECUTION OF FILE SUPPORT C, 4-70
TYPEWRITER MESSAGES SPECIFIC TO
FILE SUPPORT C, 4-86

SUPPORT
FILE SUPPORT C PROGRAM, 1-5

SYSTEM
MOD 1 (MSR) OPERATING SYSTEM, 4-68
MOD 1 (TR) OPERATING SYSTEM, 4-70

TAPE
1/2-INCH TAPE FORMATS, 4-63
TAPE AND CARD FILE
CONSIDERATIONS, 4-63
TAPE MARKS, 4-66

TAPE-RESIDENT
OWN-CODING CONSIDERATIONS FOR
TAPE-RESIDENT OPERATION, 4-61

TECHNIQUES
RANDOMIZING TECHNIQUES, E-1

TERMINAL
CREATION OF TERMINAL FILES, G-1
TERMINAL FILES, G-1

TIME
ADDRESS REGISTER CONTENTS AT TIME
OF ERROR EXIT (EDF), D-20
COMPRISING BETWEEN ACCESS TIME AND
STORAGE CAPACITY, C-22
OPTIMIZING ACCESS TIME, C-18

TR
MOD 1 (TR) OPERATING SYSTEM, 4-70

TRACK
HANDLING TRACK LINKING
RECORDS, D-18

TRACK-LINKING
TRACK-LINKING RECORDS, 2-8

TRACKS
TRACKS REQUIRED FOR MASTER/CYLINDER
INDEX, C-25

TRAILER
TRAILER LABEL, 4-66, 4-67

TRW
TYPE OF READ OR WRITE (TRW), D-15

TYPE
ACTION MACRO CALLS FOR EACH FILE
TYPE IN EACH PROCESSING MODE, 3-6
DISK PACK CYLINDER CONCEPT - TYPE
259 DISK PACK DRIVES, 2-2
ERROR TYPE INDICATOR (ERI), D-19
ILLUSTRATION OF UNITS OF ALLOCATION
- TYPE 261 OR TYPE 262 DISK
FILE, 2-7
OPTIMUM RECORD SIZE - TYPE 261 OR
TYPE 262 DISK FILES, C-7
TYPE OF READ OR WRITE (TRW), D-15

TYPES
OPTIMUM RECORD SIZE - TYPES 258 259
OR 259A DISK PACK DRIVES, C-5
TYPES OF OVERFLOW, C-17

TYPEWRITER
CONSOLE TYPEWRITER OPERATING
PROCEDURES, 3-82
CONSOLE TYPEWRITER PAUSE CODES AND
MESSAGES FOR LOGICAL I/O C, 3-63
JOB CONTROL FILE CONSOLE TYPEWRITER
MESSAGES, 4-85
OPERATOR CONTROL WITH CONSOLE
TYPEWRITER, 4-83
TYPEWRITER MESSAGES FOR CONDITIONS
RELATED TO NON-MASS STORAGE
FILES, 4-83
TYPEWRITER MESSAGES SPECIFIC TO
FILE SUPPORT C, 4-86

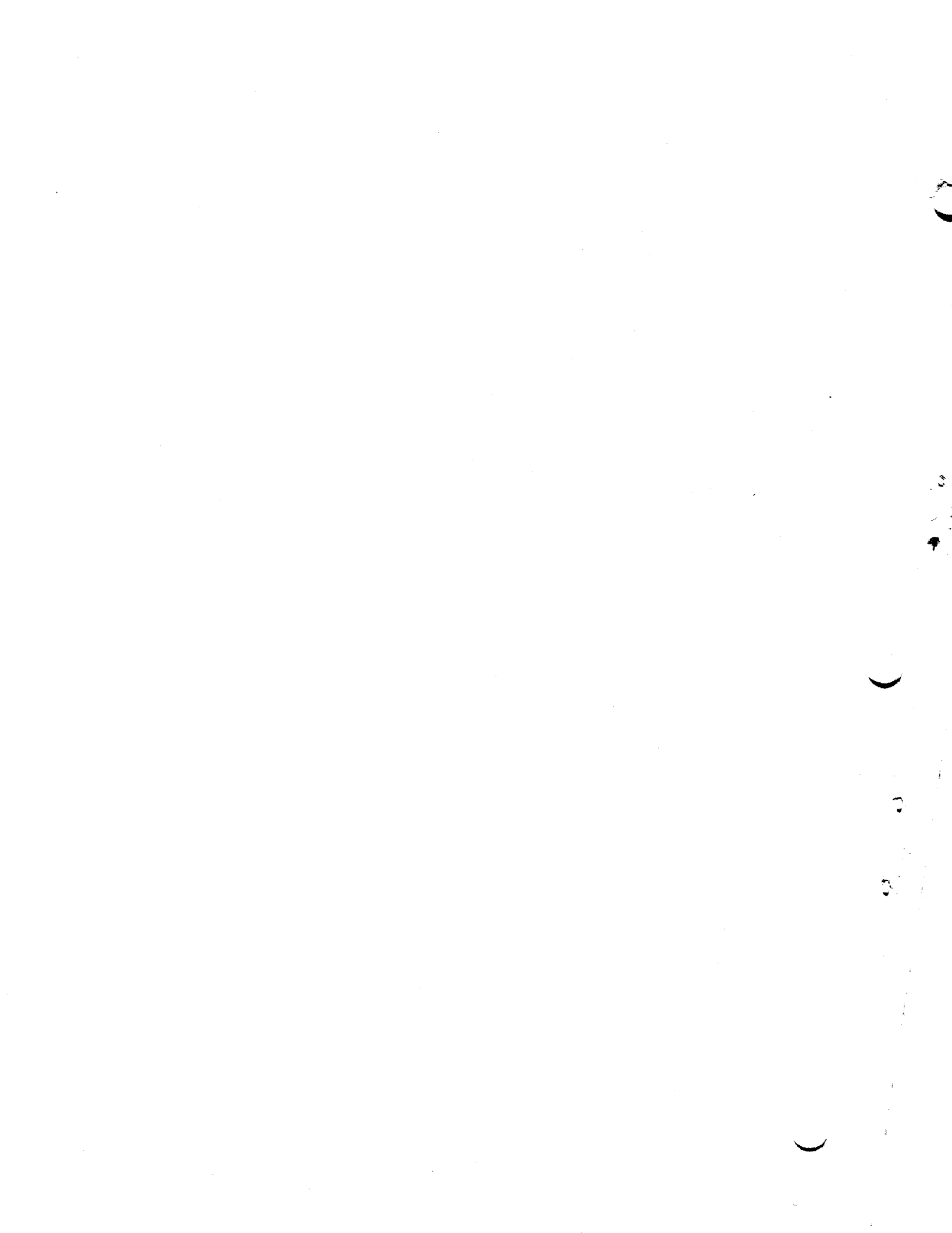
UNCORRECTABLE
USER'S UNCORRECTABLE ERROR
ROUTINE, D-19
USER'S UNCORRECTABLE ERROR ROUTINE
ENTRANCE (EAD), D-15

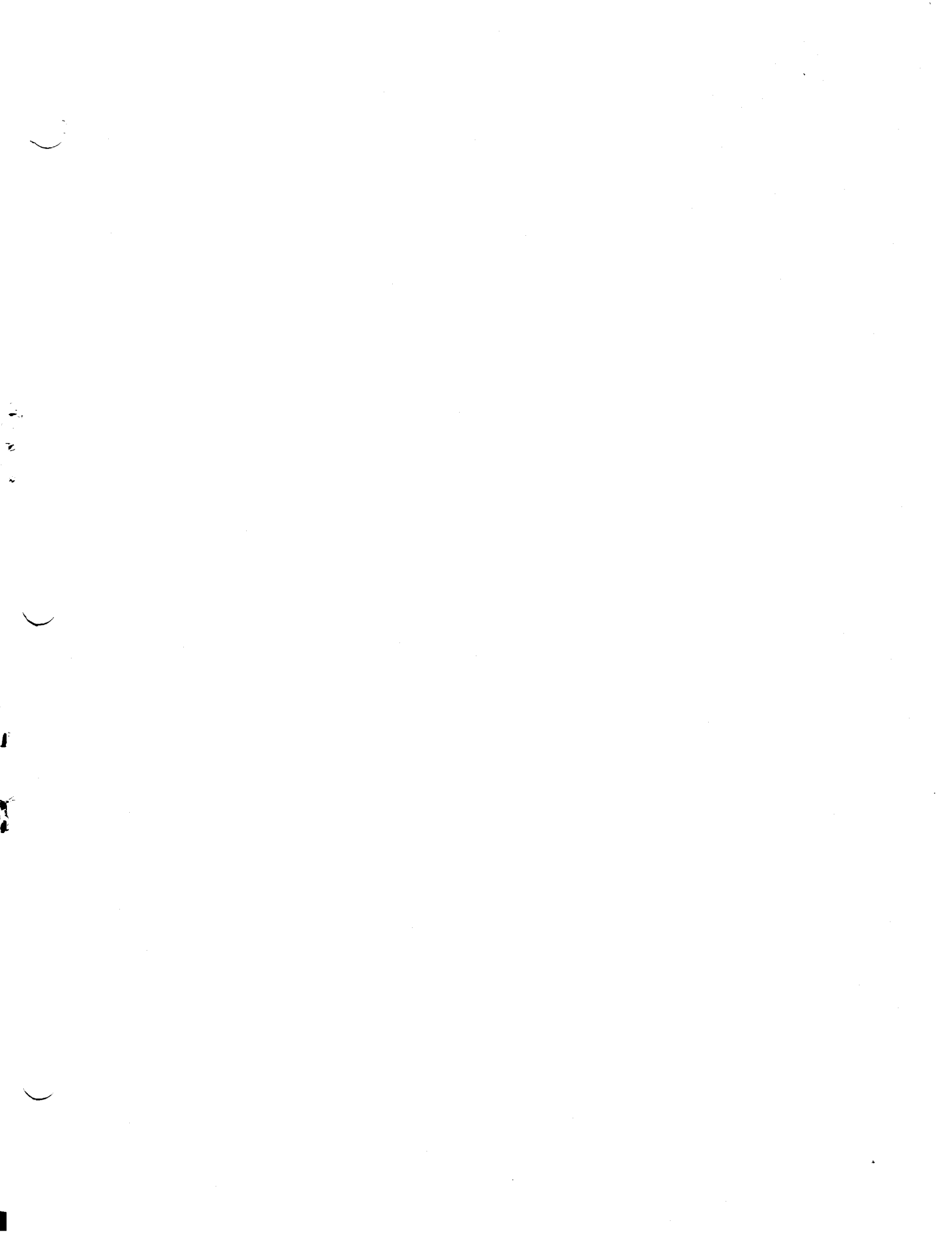
UNIT
CONTROL UNIT CURRENT ADDRESS AND
STATUS, D-16
DATA UNIT OF ALLOCATION, 4-19
MPCA CONTROL UNIT CURRENT ADDRESS
AND STATUS FIELD, D-16

UNITS
ASSIGNMENT OF UNITS OF

INDEX

- ALLOCATION, C-3
- ILLUSTRATION OF UNITS OF ALLOCATION
- TYPE 261 OR TYPE 262 DISK
FILE, 2-7
- UNITS OF ALLOCATION, 2-6
- UNITS STATEMENT, 4-17
- UNLOAD
 - LOAD AND UNLOAD FUNCTIONS, 4-33
- UNLOAD
 - JOB CONTROL LANGUAGE EXAMPLES FOR
LOAD AND UNLOAD FUNCTIONS, 4-45
 - JOB CONTROL LANGUAGE FOR LOAD AND
UNLOAD FUNCTIONS, 4-35
 - MASS STORAGE UNLOAD COMMUNICATION
AREA MACRO CALL (MUCA), 3-51
 - SUMMARY OF JOB CONTROL STATEMENTS
FOR LOAD AND UNLOAD
FUNCTIONS, 4-48
 - UNLOAD, 4-2
- UNLOADING
 - JOB CONTROL STATEMENTS FOR LOADING
AND UNLOADING FILES, 4-35
 - LOADING OR UNLOADING, 4-60
 - UNLOADING A DIRECT ACCESS
FILE, 5-57
 - UNLOADING A PARTITIONED SEQUENTIAL
FILE, 4-58
 - UNLOADING AN INDEXED SEQUENTIAL
FILE, 4-60
 - UNLOADING BY FILE, 4-58
 - UNLOADING MASS STORAGE FILES ONTO
PRINTER, 4-67
 - UNLOADING SELECTED MEMBERS, 4-58
- UNLOAD-TO-PRINTER
 - LISTING OF SAMPLE UNLOAD-TO-PRINTER
FUNCTION, 4-69
- UNUSED
 - MAP UNUSED AREAS, 4-6
 - RELEASE COMPLETE FILE TO UNUSED
STATE (MSREL), 3-19
- USABLE
 - ADDITIONAL USABLE EQUIPMENT, 1-7
- USER'S
 - CORRECTIVE ACTION FOR USER'S ERROR
ROUTINE, D-21
 - USER'S UNCORRECTABLE ERROR
ROUTINE, D-19
- USER'S UNCORRECTABLE ERROR ROUTINE
ENTRANCE (EAD), D-15
- UTILIZATION
 - READ/WRITE CHANNEL UTILIZATION,
3-70, D-12
- VALUES
 - SUMMARY OF MCA PARAMETER
VALUES, 3-49
 - SUMMARY OF MIOC PARAMETER
VALUES, 3-36
- VARIABLE
 - VARIABLE PERIPHERAL ADDRESS
ASSIGNMENT, D-13
- VERIFY
 - VERIFY ACTION, D-2
 - VERIFY ACTION MACRO CALL, D-11
 - VERIFY ACTION MACRO ROUTINE, D-18
- VOLATILITY
 - DISTRIBUTION AND VOLATILITY, C-17
- VOLUME
 - EXIT AND RETURN CODES FOR VOLUME
DIRECTORY EXITS, 3-73
 - FORMATTING AND VOLUME
PREPARATION, 2-3
 - VOLUME CONVENTIONS, 2-1
 - VOLUME DIRECTORY, 2-3, A-3
 - VOLUME LABEL, 2-3, A-2
 - VOLUME LABEL AND VOLUME
DIRECTORY, A-1
 - VOLUME STATEMENT, 4-29, 4-52
- VOLUME-NAME
 - VOLUME-NAME PARAMETER, 4-17,
4-29, 4-52
- WAIT
 - WAIT ACTION, D-2
 - WAIT ACTION MACRO CALL, D-11
 - WAIT ACTION MACRO ROUTINE, D-18
- WRITE
 - TYPE OF READ OR WRITE (TRW), D-15
 - WRITE ACTION, D-2
 - WRITE ACTION MACRO CALL, D-10
 - WRITE ACTION MACRO ROUTINE, D-17
 - WRITE PROTECTION, F-1





Honeywell

HONEYWELL
TECHNICAL PUBLICATIONS REMARKS FORM *

TITLE: MOD 1 (MSR)
DATA MANAGEMENT SUBSYSTEM

DATED: DECEMBER, 1968
FILE NO: 123.6005.141C.5-618

ERRORS NOTED IN PUBLICATION:

Fold

SUGGESTIONS FOR IMPROVEMENT TO PUBLICATION:

Fold

(Please Print)

FROM: NAME _____ DATE _____
COMPANY _____
TITLE _____
ADDRESS _____

* Your comments will be promptly investigated by appropriate technical personnel, action will be taken as required, and you will receive a written reply. If you do not require a written reply, please check here .

Cut Along Line

BUSINESS REPLY MAIL

No postage stamp necessary if mailed in the United States

POSTAGE WILL BE PAID BY

HONEYWELL
151 NEEDHAM STREET
NEWTON HIGHLANDS, MASS. 02161

ATT'N: MARKETING INFORMATION SERVICES, MS 251

FIRST CLASS
PERMIT NO. 39531
NEWTON HIGHLANDS
MASS.



Cut /) Line

Honeywell