

MRX/OS Control Program and Data Management Services

Extended Reference Manual

2200.002

MEMORREX

**Computer System
Products**

December 1972 Edition

Requests for copies of Memorex publications should be made to your Memorex representative or to the Memorex branch office serving your locality.

A reader's comment form is provided at the back of this publication. If the form has been removed, comments may be addressed to the Memorex Corporation, Publications Dept., 8941 - 10th Ave. No. (Golden Valley) Minneapolis, Minnesota 55427.

© 1972, MEMOREX CORPORATION

PREFACE

The MRX/OS Control Program and Data Management Services are discussed in two separate documents, each designed for a specific type of input/output (I/O) level user. The Basic Reference manual contains information at the logical I/O level of processing, where the blocking and deblocking of data is done for the user. This document, the Extended Reference manual, is designed for the block and physical I/O level user. Block I/O level processing recognizes no logical records; therefore, all data is read or written as a data block. The physical I/O level of processing allows the user to do his own processing of data.

Chapter 7 of this document contains all the macros required for block and physical I/O level processing. Appendix B contains the expansions of these macros with the standard system suffixes listed.

Related information may be found in the following documents.

- MRX/OS Control Language Services, Extended Reference
- MRX/OS Control Program and Data Management Services, Basic Reference
- MRX/OS Assembler Reference

TABLE OF CONTENTS

Section		Page
1	INTRODUCTION	1-1
	Environment	1-1
	Description	1-1
	Input/Output Levels	1-1
	General	1-1
	Logical Input/Output	1-1
	Block Input/Output	1-2
	Physical Input/Output	1-2
	Implementation Method	1-2
2	DATA STRUCTURES	2-1
	Files and Records	2-1
	Labels	2-1
	Tape Labels	2-1
	Disc Label	2-2
	Disc Storage Catalogs	2-3
	Tables	2-12
	File Description Table	2-12
	Buffer Description Table	2-12
3	BLOCK INPUT/OUTPUT	3-1
	Introduction	3-1
	General Rules	3-1
	Block Input/Output Coding	3-2
	Block Reading	3-2
	Block Writing	3-2
	Block Positioning	3-3
	Device Control Commands	3-3
	Space Management and File Control	3-3
	Processing Considerations	3-3
	Priority	3-3
	End Conditions	3-3
	Processing Multi-Volume Files	3-4
	Sense Information	3-4
	Request Termination	3-4
	Error Processing	3-4
4	PHYSICAL INPUT/OUTPUT	4-1
	Introduction	4-1
	Physical Input/Output Coding	4-1
	Defining and Opening Devices	4-1
	Performing the Physical I/O Operation	4-1

TABLE OF CONTENTS (Continued)

Section	Page
4 (cont)	
Basic Method	4-1
Sharing a PCB	4-1
Sharing an EXCP	4-3
Overlapped Operation	4-3
Physical I/O Restrictions	4-3
Error Processing	4-3
Example of Physical I/O Program	4-3
5	
CONTROL PROGRAM SERVICES	5-1
Introduction	5-1
Service Request Control	5-1
Inter Step and Control Language Communication	5-1
Finding Partition Size	5-1
Reading Data from //PAR Cards	5-2
Writing to the SYSOUT File	5-2
6	
INTERACTION OF DATA MANAGEMENT AND THE CONTROL LANGUAGE	6-1
7	
MACROS	7-1
Data Management	7-1
Block I/O Level Declarative Macro	7-2
DEFLB – Define File Label	7-2
Block I/O Level and Physical I/O Level Executive Requests	7-2
Space Management Macros	7-2
ALLOC – Allocate Space	7-3
EXPND – Add Mass Storage Space	7-5
PURGE – Release Disc File Space	7-5
File Control	7-6
OPEN – Open File for Data Transmission	7-6
CLOSE – Close File for Data Transmission	7-7
CLOVE – Close Volume	7-8
I/O Service Macro	7-9
LABRTN – Return File Label Information	7-9
Block Input/Output Macros	7-10
Read	7-10
Write	7-11
Magnetic Tape and Disc	7-12
Line Printer	7-12
Card Punch	7-12
POSITN – Change Current Block Number	7-12
CNTRL – Hardware Control Operation	7-13
STATUS – Report of Status	7-15
TYPE – Device and File Type	7-15
RESET – Reset Exception Conditions	7-21
Physical Input/Output Macros	7-22
EXCP – Input/Output Action	7-22

TABLE OF CONTENTS (Continued)

Section	Page
7 (cont) Command	7-23
COMMAND Macro for Basic Data Channel (Unit Record Devices and Magnetic Tape)	7-24
COMMAND Macro for DCABLE Operation	7-24
COMMAND Macro for a DCSEEK Operation	7-24
COMMAND Macro for a DCSRCH Operation	7-27
COMMAND Macro for a DCREAD Operation	7-28
COMMAND Macro for a DCWRIT or DCFWRIT Operation	7-29
COMMAND Macro for a RESTORE Operation	7-30
COMMAND Macro for a DCJUMP Operation	7-30
Control Program Macros	
WAIT – Wait for Service Request Completion	7-30
Delay – Suspend Program Execution	7-31
INFORM – Service Request Completed	7-31
POST – Create Compressed Communication Byte	7-32
RPOST – Expand from Communication Byte	7-32
SETCOM – Transfer to Job Control Table	7-33
GETCOM – Transfer from Job Control Table Communication Area	7-33
ACCEPT – Read //PAR Card	7-34
DISPLAY – Write Message on SYSOUT	7-35
MEMLIM – Identify Partition Limit	7-35
SETIF – Post Code for Control Language Test	7-36
HALT – Terminate Program	7-36
EHALT – Terminate Program	7-37
ABEND – Terminate Program Abnormally	7-37
TIME – Retrieve Time of Day	7-37
SDATE – Retrieve System Data	7-37
JDATE – Retrieve Job Date	7-37
Console Communication Macros	7-37
CONSOLE – Transmit Message to Console and Optionally Receive Reply	7-38
MESSAGE – Set Up Message Format	7-38
Generation of an Output Message	7-38
Generation of a Reply Buffer	7-38
APPENDIX A – PACK CATALOG AND CENTRAL CATALOG FORMATS	A-1
APPENDIX B – SERVICE REQUEST MECHANISM AND MACRO EXPANSIONS	B-1
APPENDIX C – ERROR RECOVERY	C-1
APPENDIX D – GAP SPECIFICATIONS	D-1
APPENDIX E – DISC TRACK FORMAT	E-1
APPENDIX F – INDEX – BLOCK SIZE FOR INDEXED FILES	F-1

LIST OF FIGURES

Figure		Page
2-1	Standard Tape Volume Label	2-2
2-2	Standard Tape File Label	2-2
2-3	Disc Label	2-3
3-1	Block I/O Program to Read Cards and Print	3-5
3-2	Block I/O Program to Read Cards into Disc File	3-6
4-1	Basic Method for Physical I/O	4-2
4-2	Sharing a PCB	4-2
4-3	Sharing an EXCP	4-2
4-4	Physical I/O Program to Read Cards and Print	4-4
7-1	Card Image/Storage Relationship	7-11

LIST OF TABLES

Table		Page
2-1	Name Element Format	2-4
2-2	Attribute Element Format	2-6
2-3	Space Element and Space Element Continuation Format	2-8
2-4	Volume Element and Volume Element Continuation Format	2-10
2-5	File Description Table Format	2-13
2-6	Disc Device Format	2-16
2-7	Magnetic Tape Device Format	2-18
2-8	Unit Record Device Format	2-19
2-9	Buffer Description Table Format for Sequential Files	2-20
2-10	Buffer Description Table Format for Relative Files	2-24
2-11	Buffer Description Table Format for Indexed Files	2-29
3-1	Assumed Block Numbers	3-2
6-1	Summary of Data Management and Control Language Interaction	6-2
7-1	Returned Information Format	7-16
7-2	Status Word for Basic Data Channel Operations	7-19
7-3	Status Word For Disc Channel Operations	7-20
7-4	Bit Significance	7-22
7-5	Peripheral Device Basic Hardware Operation Codes	7-25

1. INTRODUCTION

ENVIRONMENT

The logical, block and physical input-output facilities and control program services described in this manual are all available on the MRX/OS nominal system.* All except for those flagged are also available on the MRX/OS minimal system.† Since MRX/OS is a disc-resident system, a minimum of one disc drive is required for the system disc pack.

DESCRIPTION

The services described in this manual are those provided by the central part of the operating system‡. Access to these services, which are mainly concerned with input/output processing, is through system macro instructions.

*The nominal system has a minimum main storage size of 24K bytes, including a system region of 10K bytes.

†The minimal system has a minimum main storage size of 16K bytes, including a system region of 8K bytes.

‡Other operating system services, such as job control, link editing and library utilities, are described in separate manuals.

INPUT/OUTPUT LEVELS

GENERAL

An Assembly Language programmer may code his input/output implementation at three different levels: the logical, block, and physical levels. The logical level is processed by the Data Management system while the block and physical levels are processed directly by the system's basic input/output routines (upon which Data Management depends).

The logical level, the highest level, is the easiest to use. The block level provides greater flexibility but is still dependent on the standard system data file structures. The physical level is the most involved, regarding coding, but provides the greatest flexibility, including independence from the system's file structures.

LOGICAL INPUT/OUTPUT

Logical input/output coding (described in the **Control Program and Data Management Services, Basic Reference** and in Section 2 of this book) is normally employed in situations where no special handling of a condition such as end-of-file is required and the amount of available storage space is not especially restricted (though careful coding will still enable the programmer to economize on storage space). The chief advantage of logical input/output is the simplicity of coding for most common input/output applications.

BLOCK INPUT/OUTPUT

Block input/output coding (described in Section 3) enables the programmer to create an object program smaller than would result from logical input/output, since the large general-purpose GET/PUT subroutines are not included. When the user's input/output application is limited to a certain particular type of operation, block input/output produces a smaller, more efficient program but involves more work in writing the code. Block input/output is also necessary in the case where some special handling of abnormal conditions is required.

PHYSICAL INPUT/OUTPUT

Physical input/output coding (described in Section 4) is used where the user desires to create and process his own special external data structures for some special application, or where some special input/output device such as a plotter is being used. (The standard system files may also be processed by physical input/output if desired, though this would rarely be done in practice.)

IMPLEMENTATION METHOD

Ultimately, whether logical, block, or physical I/O is being used, all input/output implementation is performed by the system's input/output drivers. The basic mode of communication with the drivers is always the same; it involves supplying them with a "command program", which is a contiguous group of command control blocks, each one calling for the execution of a specific I/O operation.

The physical input/output user must create these command programs himself with physical input/output macros, and issue *do input/output* macros which instruct the system to execute them. The block input/output user, on the other hand, simply codes the appropriate block input/output macro; in this case the system builds the command program and executes it. At the logical input/output level (data management) the system performs certain pre- or post-processing of data connected with the logical data file structure before or after it passes control to the block input/output level for implementation of the actual data transfer.

2. DATA STRUCTURES

The structure of records, labels, and files is discussed in connection with their use. The **Control Program and Data Management Services, Basic Reference** discusses the general data structures. This manual gives the specific description of labels and tables.

FILES AND RECORDS

Data Management supports records that have the common stored data format, which include logical record headers and may include space headers in each block of data. If the block I/O user elects to use the common stored data format, he must specify CSD = YES on the Control Language //DEFINE card to allocate a block size sufficient for the required headers. The control information (record headers and space headers) is managed by Data Management for logical I/O users. However, block I/O level or physical I/O level users must generate or process the control information if the common stored data format is used. The common stored data format is used only for data located in mass storage; thus data stored on unit record devices does not have this format; and this format is not applied to such data by the block I/O level READ function (Section 7, Macros).

Records may be variable length or fixed length limited only by file organization and devices; that is, only fixed-length records can be used for unit record devices.

Three file organizations, which produce a variety of applications, are available for the user: sequential, relative, and indexed. At the logical I/O level, two access methods are provided — sequential and random. At the block I/O level, each access is by specified block (random) or by implied next block (sequential, by adding one to the access count). Sequential access is provided for all three file organizations; however, random access is limited to relative or indexed files.

A more detailed discussion of file and record structures is in the **Control Program and Data Management Services, Basic Reference**.

LABELS

Two types of identification are associated with a disc pack or tape reel: external (such as a sticker) and internal (labels). This section is concerned with the internal identification, namely the labels.

The internal identification serves two purposes.

1. Labels protect files from careless mistakes resulting from the user's disregard for external labeling.
2. Labels specify the location of data on files and store variable file information.

Tapes are identified by both volume and file labels; the disc files have volume labels and central and/or pack catalogs.

TAPE LABELS

The MRX 40 and 50 Systems have two types of tape labels: volume labels and file labels.

The volume label is found at the beginning of a tape reel and the file label at the beginning of a file. Tape marks separate files and their associated labels on a tape reel.

The volume label (Figure 2-1) is identified by the characters, VOL, found in the first three positions. The volume label number is always 1 for compatibility with IBM. The volume serial number occupies positions 4 through 9 and identifies the volume; and a unique owner name and address code identifies the installation.

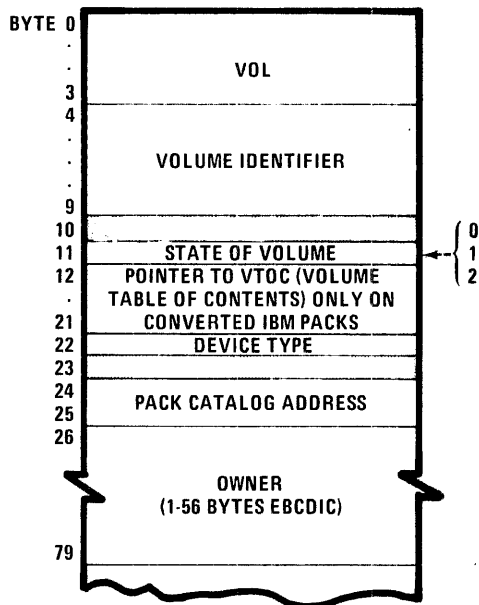


Figure 2-1. Standard Tape Volume Label

The standard tape file label (Figure 2-2) provides information concerning the user's file such as creation and expiration dates, file label number, file name, and sequence number.

The label identification field identifies the type of standard label with a three letter abbreviation. Three types supported by the system are header labels (HDR), end-of-file labels (EOF), and end-of-volume labels (EOV). The file serial number found in positions 22-27 is identical to the volume serial number in the volume label. The volume sequence number identifies the order of the volume of data records in a multi-volume logical file. The block count provides the number of physical records written in a file at creation.

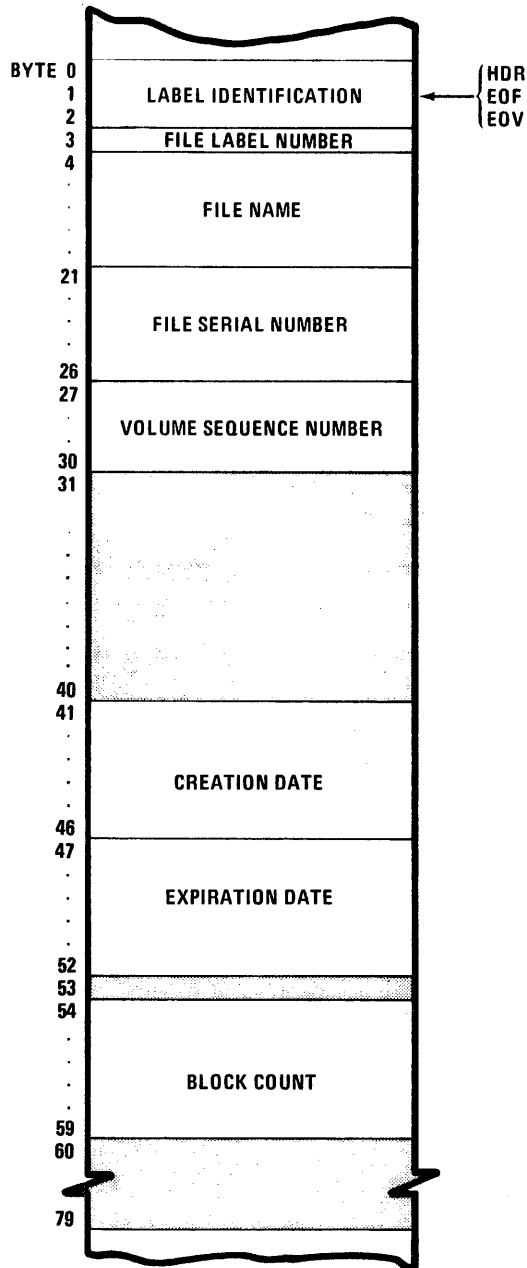


Figure 2-2. Standard Tape File Label

DISC LABEL

Disc files must be labeled with volume and actual file identification (Figure 2-3). The disc pack volume label is written by the Pack Initialize Utility Routine (MRX/OS Utility Programs Reference). The volume label identifies the volume; gives the state of the volume — unrestricted (0), restricted (1), or locked (2), — device type, owner; and gives the starting track address of the pack catalog for this volume. The actual file identification is found in the disc catalogs.

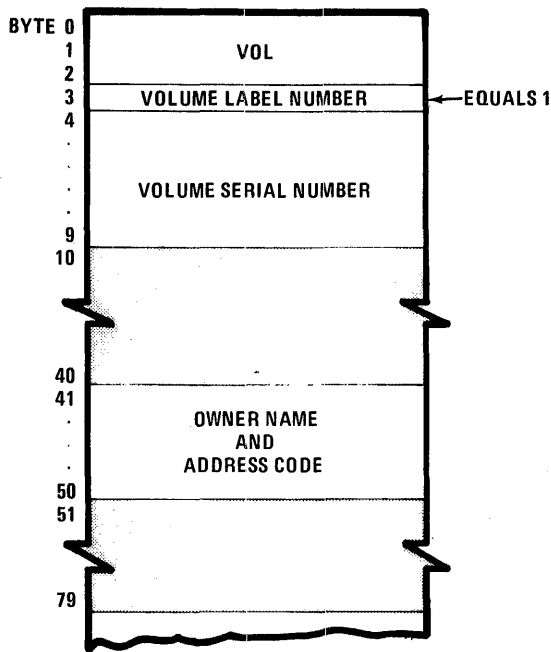


Figure 2-3. Disc Label

Allowance is made in the catalog structure for overflow from a pack catalog space element or central catalog volume element. These elements are known as element continuations: space element continuation and volume element continuation.

Tables 2-1 through 2-4 explain format of the four basic elements and the two element continuations. At the beginning of each table is an illustration of the specific element format. Appendix A contains detailed discussion of the pack catalog and central catalog formats.

DISC STORAGE CATALOGS

The space management routines, in performing their functions, maintain the central catalog and pack catalogs on disc. The pack catalog, existing on each volume, contains an entry for each file occupying space on the volume. The entry identifies the file and describes the space occupied by the file.

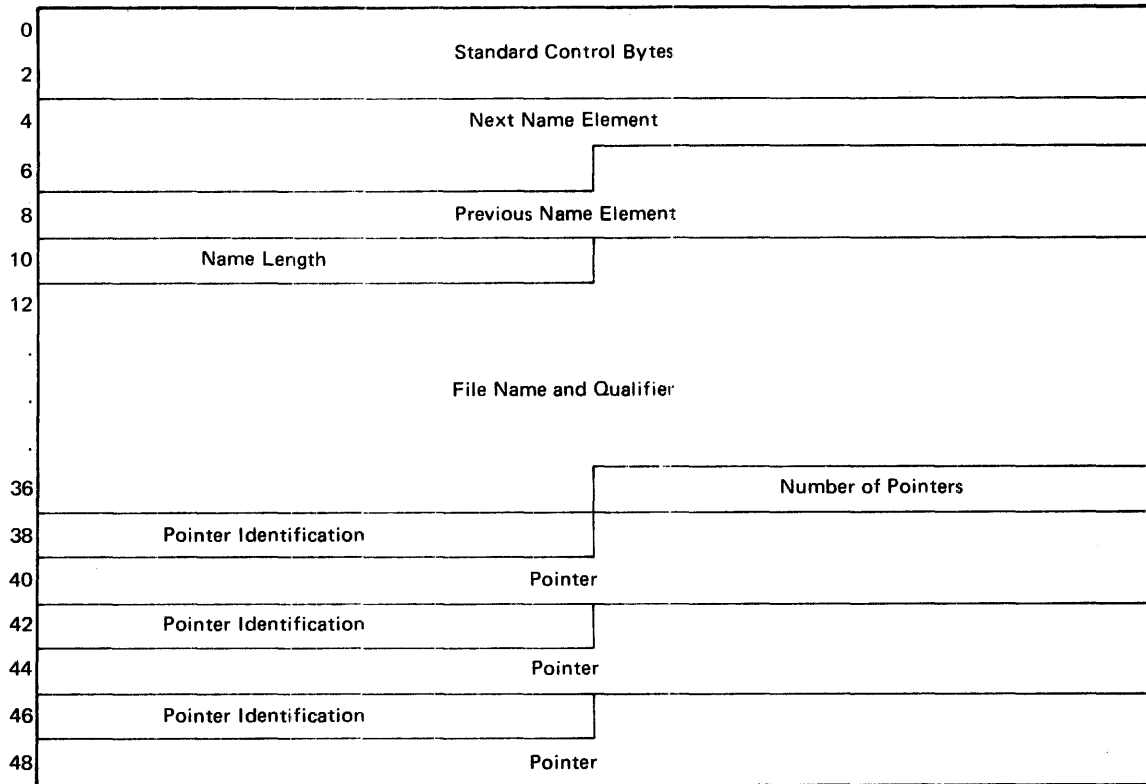
The pack catalog entry is generally divided into the following three elements, each having the form of a common stored data format.

- Name element, to identify the file
- Attribute element, to detail various characteristics of the file
- Space element, to define the space occupied by the file

The central catalog, existing once for a system, contains an entry for each file cataloged in the system. The entry identifies the file and describes the volumes occupied by the file. The central catalog also consists of three elements:

- Name element, to identify the file
- Attribute element, to detail characteristics of the file
- Volume element, to define the volumes occupied by the file

Table 2-1. Name Element Format

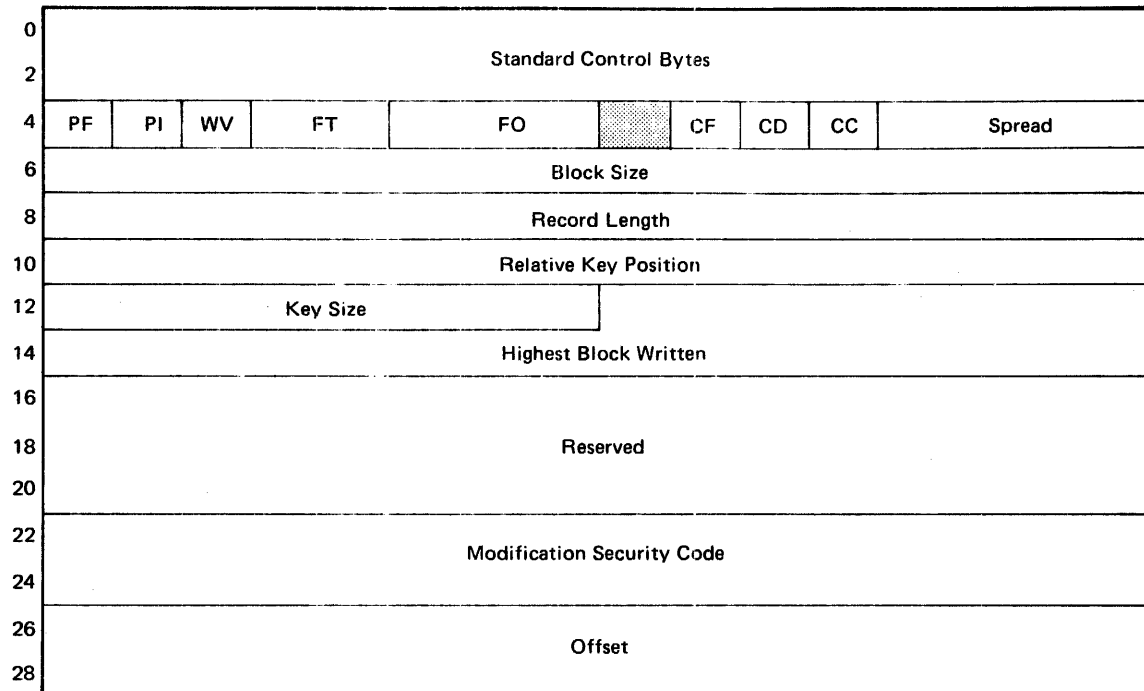


Bytes	Bit	Description
0	0,1	Data type (binary) 01 User data 10 System data 11 Both user and system data
	2-5	Reserved, always 0000 ₂
	6,7	Count of length bytes, always 10 ₂
1		Relative record number within block, with value 0 to m-1 for m records
2,3		Length bytes, giving length of the record (in bytes) with the measurement being exclusive of the control bytes
4-6		Thread to the next name element (in collating sequence). This thread is expressed in a BBR format, where BB is the relative block number (value 1 to n for n blocks) and R is the relative record number (value 0 to m-1 for the next name element's position within block BB). BBR=0 for the thread in the last name element of the chain.
7-9		Thread to the previous name element within the catalog file. BBR=0 for the first name element of the chain.
10		Name length, giving the count of bytes in the File Name and Qualifier field. The total count must be even and not exceed 26.

Table 2-1. Name Element Format (Continued)

Bytes	Bit	Description
11-36		File Name and Qualifier, containing a file name consisting of a maximum of 17 EBCDIC characters (alphabetic, numeric, and dash), a period separator, and a qualifier consisting of a maximum of eight alphanumeric characters. For name elements related to a system catalog file, the EBCDIC requirement is lifted.
37		Number of element pointers which follow within the name element.
38,42, 46		<p>Pointer identification, expressed in hexadecimal within one byte and identifying the pointer, where * denotes pertinent name elements within an entry in the pack catalog, and ** denotes pertinent name elements within an entry in the central catalog. Other pointers are pertinent in both the pack and central catalogs.</p> <p>00 Null pointer</p> <p>10 Pointer to attribute element</p> <p>20* Pointer to space element</p> <p>30** Pointer to volume element</p> <p>40** Pointer (from the indexed file's name element) to the name element of the information file</p> <p>41** Pointer (from the information file's name element) to the name element of the indexed file</p> <p>50 Pointer (from the lower paired file's name element) to the name element of the upper file</p> <p>51 Pointer (from the upper paired file's name element) to the name element of the lower file</p> <p>60* System pointer (from the pack catalog name element describing the pack catalog) to the chain of available entry blocks within the pack catalog</p> <p>61* System pointer (from the pack catalog name element describing the pack catalog) to the chain of available continuation blocks within the pack catalog</p> <p>70** System pointer (from the central catalog name element describing the central catalog) to the chain of available entry blocks within the central catalog</p> <p>71** System pointer (from the central catalog name element describing the central catalog) to the chain of available continuation blocks within the central catalog</p> <p>80** System pointer (from the name element of a SYSIN or subordinate scratch or temporary file) to the name element of the next chronologically sequenced scratch or temporary file. Scratch or temporary files are cataloged like any other file, in that they are entered by collating sequence into the main chain of existing files. In addition, the scratch or temporary file is linked in a secondary chain to the pertinent SYSIN entry through use of the forward system pointer (identification of 80) and backward pointer (identification of 81).</p> <p>81** System pointer (from the name element of a scratch or temporary file) to the name element of previous scratch or temporary file or to the SYSIN entry at the front of the secondary chain</p>
39-41, 43-45, 47-49		Pointers (in BBR format) to the pertinent elements previously described.

Table 2-2. Attribute Element Format

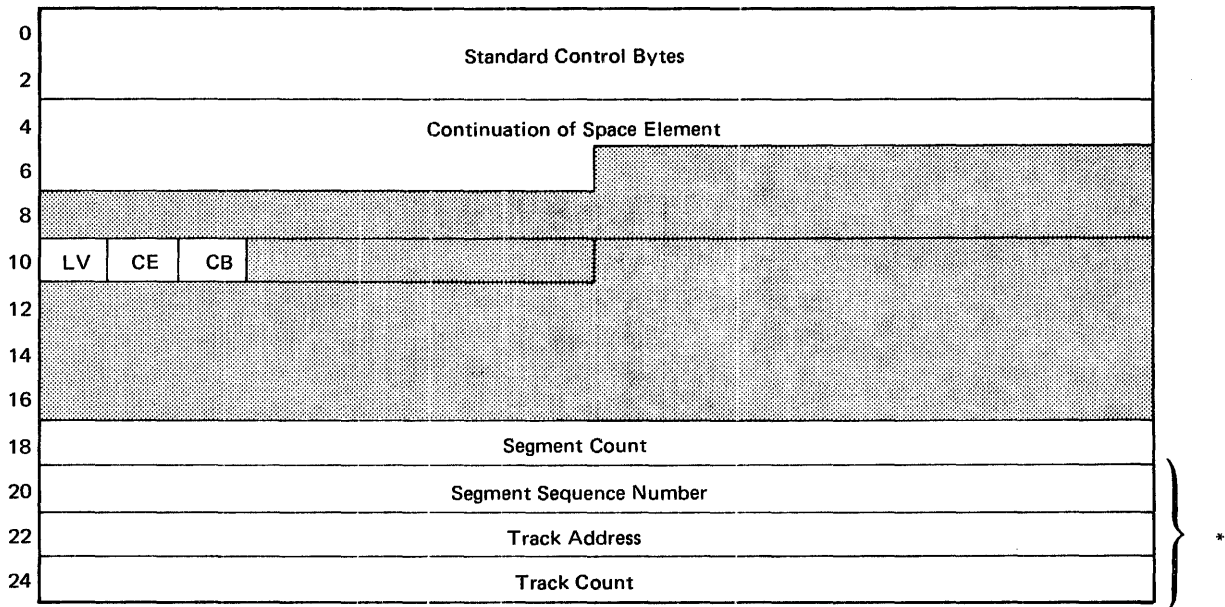


Byte	Bit	Description
0-3		Control bytes (defined in Table 2-1)
4	0	Paired file flag. If PF=1, file is paired
	1	Paired index (PI) 0 Upper file 1 Lower file
	2	Write verify (WV) 0 No write verify 1 Write verify
	3,4	File type (FT) 00 Permanent 01 Scratch 10 Temporary 11 Work
	5-7	File organization (FO) 000 General 001 Indexed

Table 2-2. Attribute Element Format (continued)

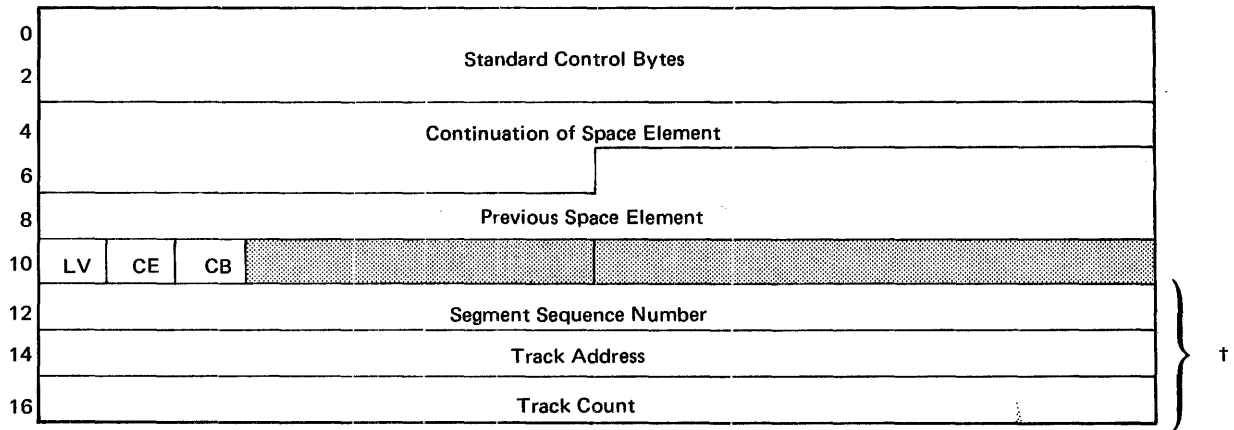
Byte	Bit	Description
		010 Relative 100 Sequential 111 Information (for indexed file)
5	1	Central catalog (CF) 0 File is uncataloged 1 File is cataloged
	2	Common stored data format (CD) 0 Not common stored data format 1 Common stored data format
	3	Control character (CC) 0 ANSI control characters 1 Native device control characters
	4-7	Spread factor for indexed files (SPREAD)
6,7		Block size (in bytes) of data block
8,9		Record length (in bytes) of a record within a data block
10,11		Relative key position; pointer to the primary key in the data portion of a record. Position 0 refers to the first byte following the control bytes.
12		Key size (in bytes) of the primary key
13-15		Highest block written, relative block number for the last block written in the file
16-21		Reserved
22-25		Modification security code
26-29		Offset; lower limit of relative record number at the time a relative file is created or, for a sequential file, the relative block number for the first block of the highest volume written. In the information file's attribute element for an indexed file, the first two bytes have the count of directory blocks that have been allocated. The other two bytes have the count of index blocks.

Table 2-3. Space Element and Space Element Continuation Format



*One per segment

Space Element



†One per segment

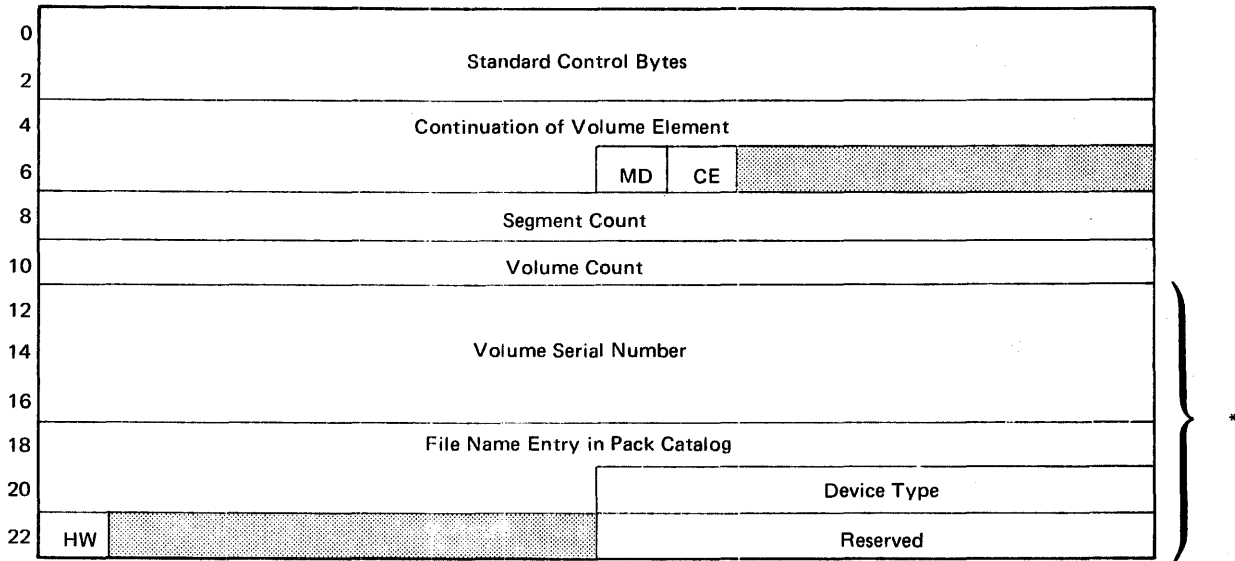
Space Element Continuation

Table 2-3. Space Element and Space Element Continuation Format (Continued)

Byte	Bit	Description
0-3		Control bytes (defined in Table 2-1)
4-6		Continuation of space element, a thread (in BBR format) which points to the next space element continuation for a file
7-9*		Previous space element is a thread (in BBR format) which points to the previous space element.
10	0	If LV=1, this is the last volume for a file
	1	If CE=1, a space element continuation exists
	2	If CB=1, each segment lower boundary is on a cylinder boundary
11-17; 11*		Not used
18,19		Segment count, a count of space elements used by the file on this volume
20,21; 12,13*		Segment sequence number, giving relative segment number for each segment (shows relation to segments existing on same or separate volumes)
22,23; 14,15*		Track address relative to beginning of segment. (All physical disc addresses are expressed in terms of track number rather than cylinder and track within a cylinder.)
24,25; 16,17*		Track count of tracks in segment

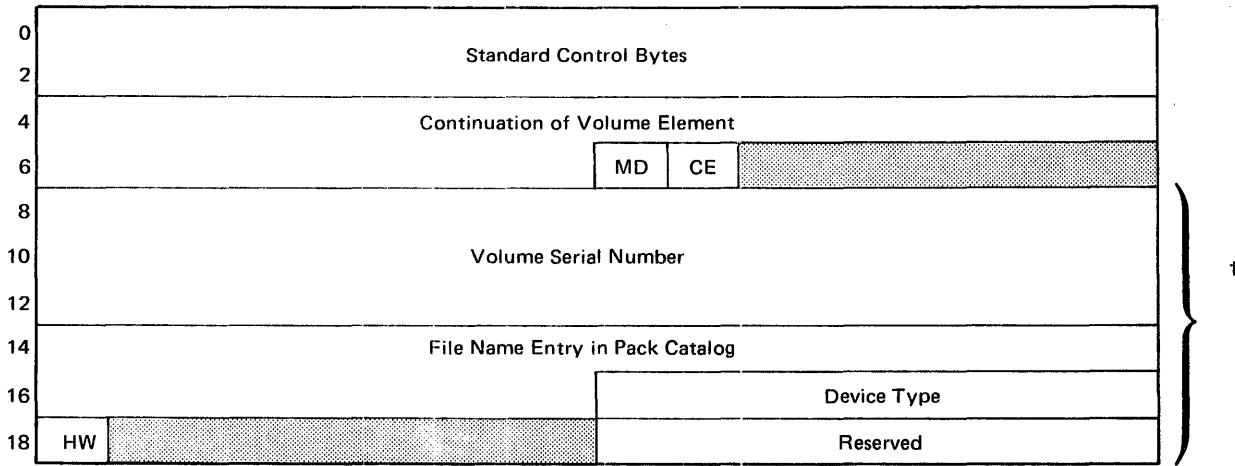
*Byte position(s) unique to space element continuation table.

Table 2-4. Volume Element and Volume Element Continuation Format



*One per volume

Volume Element



†One per volume

Volume Element Continuation

Table 2-4. Volume Element and Volume Element Continuation Format (Continued)

Byte	Bit	Description
0-3		Control bytes (defined in Table 2-1)
4-6		Continuation of volume element, a thread (in BBR format) which points to the next volume element continuation for a file
7	0	Modified volume descriptions. If MD=1, central catalog is created and modified volume descriptions are included.
	1	If CE=1, volume element continuation exists
8,9		Segment count
10,11		Volume count
12-17; 8-13*		Volume serial number
18-20; 14-16*		File entry in pack catalog, a pointer to the file's entry in the specified volume's pack catalog (BBR format)
21;17*		Device type (00 ₁₆ for MEMOREX 3664 Drive)
22;18*	0	If HW=1, highest volume written
23;19*		Reserved

*Byte position(s) unique to volume element continuation table.

TABLES

Tables are primarily for storage information that is easily referenced. The Data Management system has two primary tables for information reference:

- File Description Table (FDT) created by OPEN
- Buffer Description Table (BDT) created by DEFSF, DEFRR, or DEFIF

FILE DESCRIPTION TABLE

A File Description Table (FDT) is built dynamically during a job's execution time in the user area of memory. An OPEN request causes an FDT to be built, and a CLOSE request releases the FDT space. The FDT is 60 bytes in length. The portion of the FDT common to all files is 40 bytes in length, and the remainder of the FDT which is device dependent is 20 bytes in length.

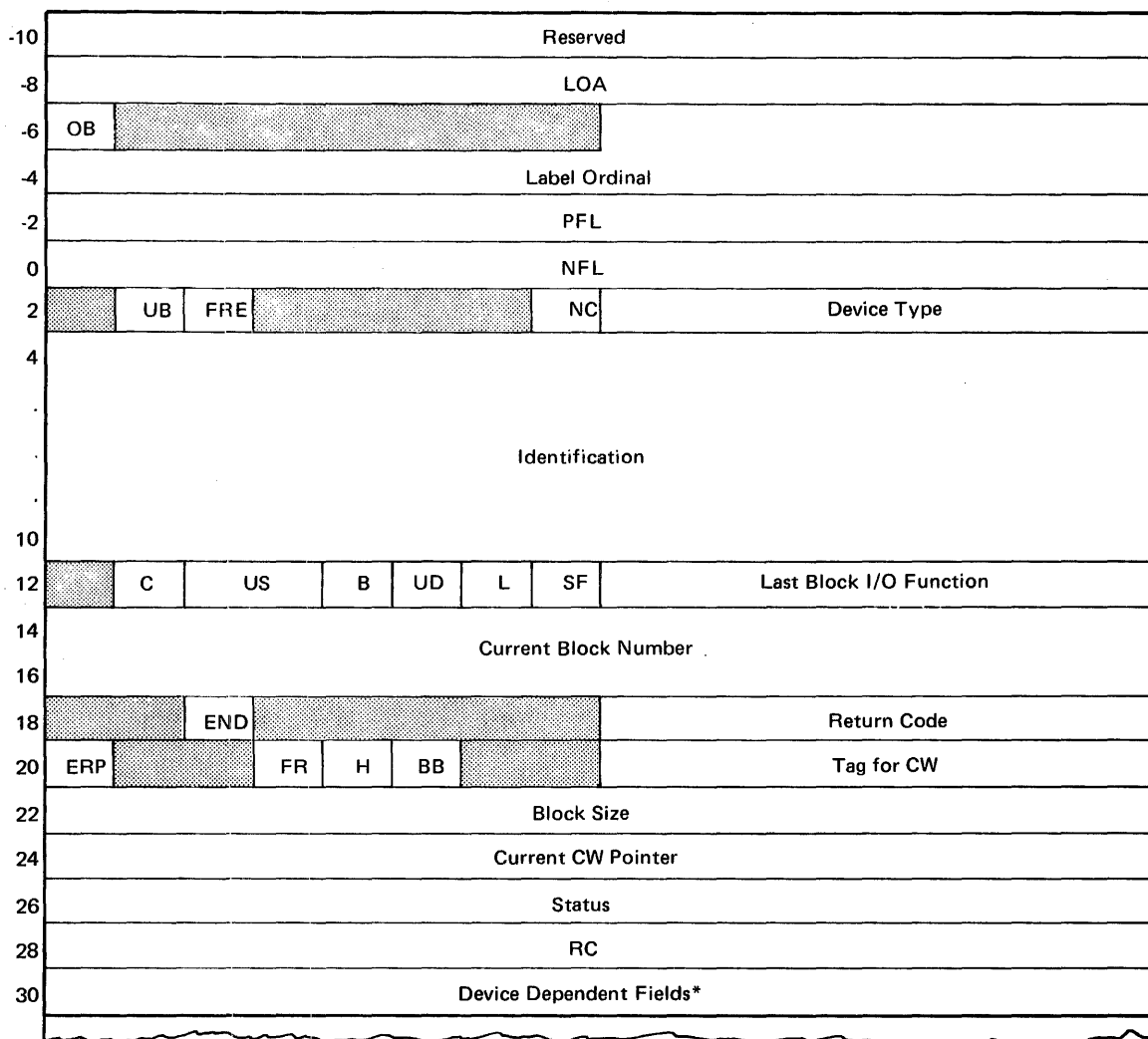
The FDT is linked with block I/O requests through an 8-byte identification field; the identification field in the FDT and the block I/O request must match. The FDT contains a unit table ordinal through which I/O control routines find the unit table and execute the I/O request. Parameters in the FDT protect users from overlapping each other's area on a shared device. Table 2-5 gives the locations of the different information fields of the FDT. Tables 2-6 through 2-8 give the device-dependent fields of the FDT. Manipulation of FDT's in the user partition is discussed in **Control Language Services, Extended Reference**.

BUFFER DESCRIPTION TABLE

A Buffer Description Table (BDT) is created by the file definition macros (DEFSF, DEFRR, and DEFIF). This main-memory table is used by the GET/PUT logic to manage buffers and control logical records. The BDT is responsible for recording location of I/O buffer(s) and record areas.

A BDT is created for each file organization: sequential, relative, indexed. The three BDT format tables follow with the appropriate field descriptions (Tables 2-9 through 2-11).

Table 2-5. File Description Table Format



*Refer to appropriate device table (Tables 2-6 through 2-8).

Byte	Bit	Description
-10,-9		Reserved
-8,-7		Length (in bytes) of FDT (LOA)
-6	0	Ordinal bit (OB) 0 Label ordinal points to central catalog 1 Label ordinal points to Control Language job file
-5,-4,-3		Label ordinal indicates the position of the label in the central catalog or the Control Language job file.
-2,-1		Previous FDT address (PFL)

Table 2-5. File Description Table Format (Continued)

Byte	Bit	Description
0,1		Next FDT address (NFL)
2	1	User bit (UB) reserved for emulator set to zero when FDT is built
	2	Freeze flag (FRE) 0 File is not frozen 1 File is reserved for a recycle of outstanding queue entry block (QEB)
	7	Native character set flag (NC). If NC=1, first data byte is the command code.
3	0-7	Device type
4-11		File identification
12	1	Common stored data format bit (C) 0 Not common stored data format 1 Common stored data format
	2,3	Usage flag (US) 00 Input 01 Update 10 Output
	4	Bypass flag (B) 0 No bypass 1 Bypass, READ goes to EOF and WRITE is a NOP
	5	Update flag (UD) 0 No update 1 Update
	6	Lockout flag (L) 0 No lockout 1 Close with lock
	7	Sequential file (SF) 0 Not sequential 1 Sequential
	13	0-7
14-17		Block number after last function processed. If zero, current block number is unknown.
18	2	End condition bit (END) indicates the sensing of an end condition: EOF, EOA, or printer carriage channels 9 or 12.
19		Return code

Table 2-5. File Description Table Format (Continued)

Byte	Bit	Description
20	0	<p>Error recovery processing flag (ERP)</p> <p>0 Call error recovery when errors encountered</p> <p>1 Do not call error recovery for errors</p>
	3	<p>FDT restore (FR)</p> <p>0 FDT not restored</p> <p>1 FDT restored</p>
	4	<p>Hold up flag (H)</p> <p>0 File not in recovery</p> <p>1 File in error recovery and following requests are held up until recovery complete</p>
	5	<p>Block I/O internal flag (BB); set by block I/O to indicate that a RESET macro has passed through the file. Bit is reset by driver.</p>
21		Tag for command word (CW) address pointer
22,23		Byte size (or maximum size) of each record
24,25		A pointer to the current or last CW executed by this file
26,27		Status of last I/O operation
28,29		Residual count (RC): the difference between bytes requested and bytes received

The device dependent portion of the FDT begins at byte 30.

Table 2-6. Disc Device Format

30	Res.	PF	PA	WF	DF	Reserved	Number of Residual Blocks
32	Highest Block Written						
34	Relative Block Number						
36	Relative Block Number						
38	Relative Block Number						
40	Blks/Track			Gap			
42	Next Segment Link						
44	UORD			Reserved			
46	Number of Contiguous Tracks						
48	Beginning Track Number						

} *

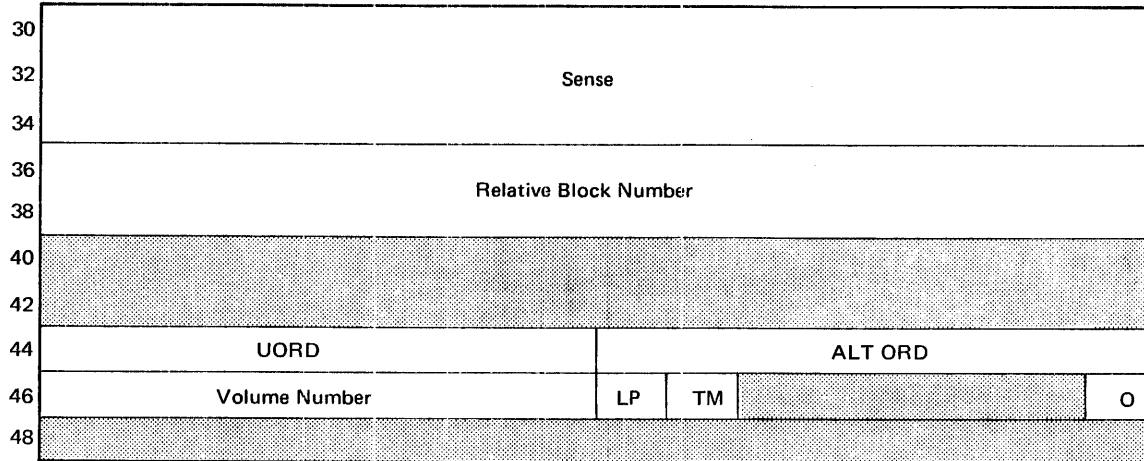
*Extent

Byte	Bit	Description
30	0	Reserved
	1	Paired file flag (PF) 0 Not paired 1 Paired file
	2	Paired file indicator (PA) 0 Upper portion of track used for this file 1 Lower portion of track used for this file
	3	Write check flag (WF) 0 No write check 1 Write check of all writes
	4	Disc driver flag (DF)
	5-7	Reserved
31		Residual block number, the number of blocks remaining to be up for a multiblock read request which crosses tracks
32-35		Highest block number written for volume now mounted
36-39		Relative block number (calculated from beginning of file) of first block on presently mounted volume
40	0-5	Number of records on a disc track (BLK/TRACK)
40,41	6-7, 0-7	Gap in bytes between records for pack rotational speed variation
42-43		Next segment link (extent address)

Table 2-6. Disc Device Format (Continued)

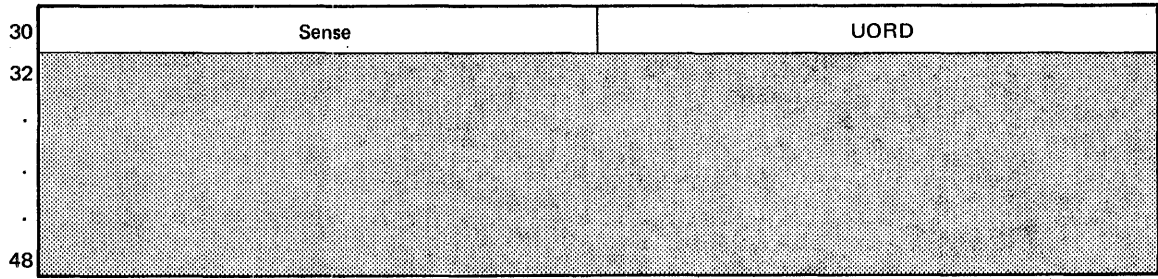
Byte	Bit	Description
44		Unit table ordinal (UORD) which indicates the unit table to which this segment is related
45		Reserved
46,47		Number of contiguous tracks
48,49		Beginning track number for this extent (0-3999)

Table 2-7. Magnetic Tape Device Format



Byte	Bit	Description
30-35		Sense bytes of the unit at the time of the last error
36-39		Relative block number (calculated from beginning of file) of first block on presently mounted volume
44		Current unit table ordinal (UORD)
45		Unit ordinal of alternate tape (ALT ORD)
46		Volume number
47	0	Label processing flag (LP) 0 No label processing complete 1 Label processing complete
	1	Tape mark flag (TM); a tape mark precedes the first data record on the tape
	7	Offset (O), an alternate unit

Table 2-8. Unit Record Device Format

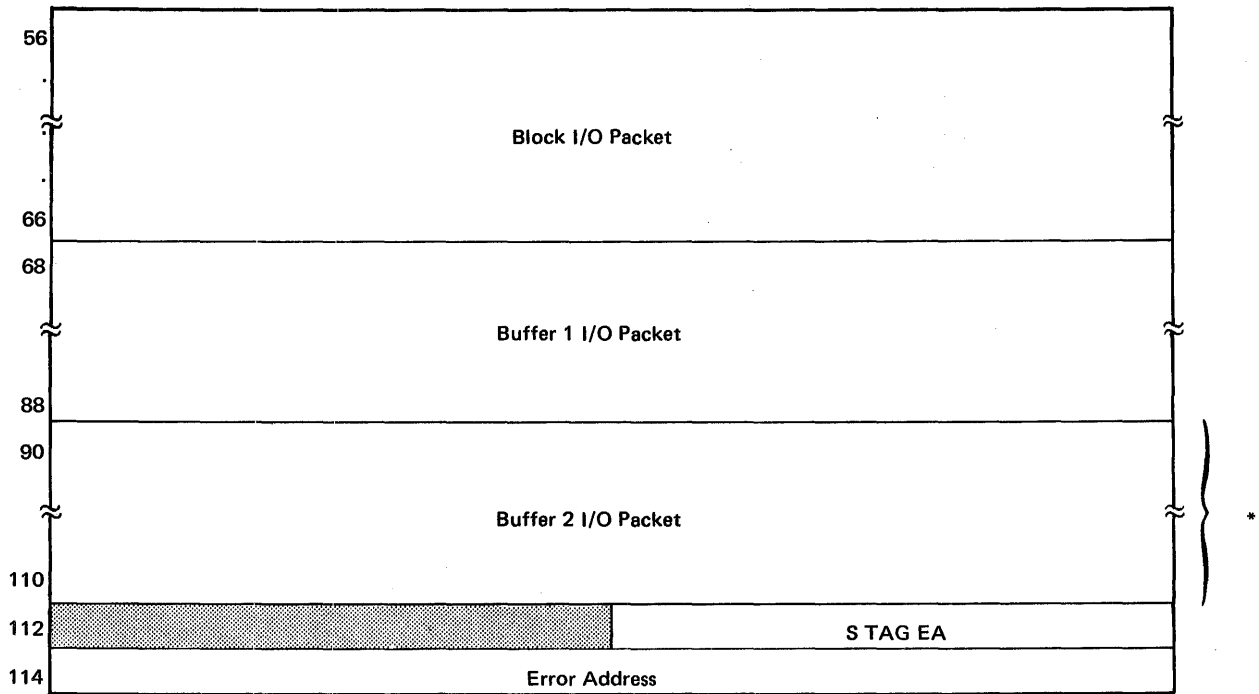


Byte	Bit	Description
30		Sense byte of the unit at the time of last error
31		Unit table ordinal (UORD)

Table 2-9. Buffer Description Table Format for Sequential Files

0	GET/PUT Address																													
2	NB	FTYP	S	LABL	AM	RT	Blocking Factor																							
4	Block Size																													
6	Record Size																													
8	S TAG GP MOD							S TAG RA																						
10	Record Area Address																													
12	File Identification																													
18																														
20																							S TAG SA							
22																Save Area Address														
24																Error Offset														
26																							S TAG RAP							
28	Record Address Pointer																													
30	Relative Record Number																													
32																														
34	Current Block Number																													
36																														
38	Logical Record Number																													
40	PU	AB	P	UR		L	I	V	SK	IG	B				Use															
42	Work Area																													
44																														
46	Move Routine																													
54																														

Table 2-9. Buffer Description Table Format for Sequential Files (Continued)



*Optional

If buffer 2 is absent, bytes 90 through 93 end the BDT as follows:



Byte	Bit	Description
0,1		External address of the GET/PUT module processing this file
2	0	Number of buffers (NB) 0 One buffer 1 Two buffers
	1,2	File type (FTYP) 00 Sequential 01 Relative 10 Indexed
	3	Shared buffer flag (S). If S=1, I/O buffers are shared with other files.
	4,5	Label processing (LABL) 00 No labels

Table 2-9. Buffer Description Table Format for Sequential Files (Continued)

Byte	Bit	Description
		01 Nonstandard labels 10 Standard labels
	6	Access method (AM) 0 Sequential 1 Random
	7	Record type (RT) 0 Fixed length 1 Variable length
3		Blocking factor, number of records per block (0-255)
4,5		Block size (in bytes) of the I/O buffer(s), size must include record headers
6,7		Record size (in bytes) of maximum record excluding header
8		Segment tag for GET/PUT module (S TAG GP MOD)
9		Segment tag for record area address (S TAG RA)
10,11		Record area address, first byte of record area
12-19		File identifier for data file (8 bytes EBCDIC)
21		Segment tag for save area address (S TAG SA)
22,23		Save area address, calling program's linkage and register save area
24,25		Error offset, offset (in bytes) to error address from beginning of BDT
27		Segment tag for record address pointer (S TAG RAP)
28,29		Record address pointer, first byte address of active record in active I/O data buffer
30-33		Relative record number, record number relative to first record in the file
34-37		Current block number, physical block number present in active buffer
38-39		Logical record number, number of active record in active buffer
40	0	If PU=1, the record address pointer has been updated following a PUT or reset to the first record in the active buffer.
	1	Active buffer (AB) 0 First buffer 1 Second buffer
	2	PUT flag (P). If P=1, record has been written in the active data buffer.
	3	Unit record flag (UR). If UR=1, device is not disc or tape.

Table 2-9. Buffer Description Table Format for Sequential Files (Continued)

Byte	Bit	Description
	5	Limits flag (L). If L=1, limits are defined for relative files
	6	I/O register flag (I) 0 Record area 1 I/O register
	7	Verify flag (V). If V=1, write verify is performed.
41	0	Skip flag (SK). If SK=1, error option is to skip error.
	1	Ignore flag (IG). If IG=1, ignore option was selection.
	2	Binary bit (B). If B=1, binary cards are to be read.
	6,7	I/O usage (USE) 00 Input 01 Update 10 Output
42-45		Work area
46-55		Move routine. Data movement routine consisting of a MOVL instruction followed by a BR instruction.
56-67		Block I/O packet used to issue service requests such as CLOVE, POSITION, and RESET.
68-69		Buffer I/O packet for I/O between the data file and the first I/O buffer.
90-111*		Buffer 2 I/O packet for I/O between the data file and the second I/O buffer. (Optional)
113*,91†		Segment tag for error address (S TAG EA)
114-115*; 92-93†		Error address, address of error return

*Byte positions unique to BDT for sequential files when buffer 2 is present.

†Byte positions unique to BDT for sequential file when buffer 2 is absent.

Table 2-10. Buffer Description Table Format for Relative Files

0	GET/PUT Address														
2	NB	FTYP	S	LABL	AM	RT	Blocking Factor								
4	Block Size														
6	Record Size														
8	S TAG GP MOD						S TAG RA								
10	Record Area Address														
12	File Identification														
18															
20							S TAG SA								
22	Save Area Address														
24	Error Offset														
26							S TAG RAP								
28	Record Address Pointer														
30	Relative Record Number														
32															
34	Current Block Number														
36															
38	Logical Record Number														
40	PU	AB	P	UR		L	I	V	SK	IG	B				Use
42	Work Area														
44															
46	Move Routine														
54															
56	Block I/O Packet														
66															
68	Buffer 1 I/O Packet														
88															

Table 2-10. Buffer Description Table Format for Relative Files (Continued)

90	Buffer 2 I/O Packet		}	*
110				
112		STAG AK	}	*
114	Key Address Pointer			
116	Limit X			
118	Limit Y			
120	Offset			
122				
124		STAG EA		
126	Error Address		}	*
128				
130				

*Optional

If buffer 2 is absent, bytes 90 through 109 end the BDT as follows:

90		STAG AK	}	*
92	Key Address Pointer			
94	Limit X			
96	Limit Y			
98	Offset			
100				
102		STAG EA		
104	Error Address		}	*
106				
108				

*Optional

Table 2-10. Buffer Description Table Format for Relative Files (Continued)

Byte	Bit	Description
0,1		External address of the GET/PUT module processing this file
2	0	Number of buffers (NB) 0 One buffer 1 Two buffers
	1,2	File Type (FTYP) 00 Sequential 01 Relative 10 Indexed
	3	Shared buffer flag (S). If S=1, I/O buffers are shared with other files.
	4,5	Label processing (LABL) 00 No labels 01 Nonstandard labels 10 Standard labels
	6	Access Method (AM) 0 Sequential 1 Random
	7	Record type (RT) 0 Fixed length 1 Variable length
3		Blocking factor, number of records per block (0-255)
4,5		Block size (in bytes) of the I/O buffers, size must include record headers
6,7		Record size (in bytes) of maximum record excluding header
8		Segment tag for GET/PUT module (S TAG GP MOD)
9		Segment tag for record area address (S TAG RA)
10,11		Record area address, first byte of record area
12-19		File identifier for data file (8 bytes EBCDIC)
21		Segment tag for save area address (S TAG SA)
22,23		Save area address, calling programs linkage and register save area
24,25		Error offset, offset (in bytes) to error address from beginning of BDT
27		Segment tag for record address pointer (S TAG RAP)
28,29		Record address pointer, first byte address of active record in active I/O data buffer
30-33		Relative record number, record number relative to the first record in the file

Table 2-10. Buffer Description Table Format for Relative Files (Continued)

Byte	Bit	Description
34-37		Current block number, physical block number present in active buffer
38-39		Logical record number, number of active record in active buffer
40	0	If PU=1, the record address pointer has been updated following a PUT or reset to the first record in the active buffer.
	1	Active buffer (AB) 0 First buffer 1 Second buffer
	2	PUT flag (P). If P=1, record has been written in the active data buffer.
	3	Unit record flag (UR). If UR=1, device is not disc or tape.
	5	Limits flag (L). If L=1, limits are defined for relative files.
	6	I/O register flag (I) 0 Record area 1 I/O register
	7	Verify flag (V). If V=1, write verify is performed.
41	0	Skip flag (SK). If SK=1, error option is to skip error.
	1	Ignore flag (IG). If IG=1, ignore option was selected.
	2	Binary bit (B). If B=1, binary cards are to be read.
	6,7	I/O Usage (USE) 00 Input 01 Update 10 Output
42-45		Work Area
46-55		Move routine. Data movement routine consisting of a MOVL instruction followed by a BR instruction.
56-67		Block I/O packet used to issue service requests such as CMOVE, POSITION, and ADD KEY/DELETE KEYS.
68-89		Buffer 1 I/O packet for I/O between the data file and the first I/O buffer.
90-111*		Buffer 2 I/O packet for I/O between the data file and the second I/O buffer.
113*,91†		Segment tag for key address pointer (S TAG AK)
114,115*; 92,93†		Key address pointer. First byte address of key address field for relative file.

Table 2-10. Buffer Description Table Format for Relative Files (Continued)

Byte	Bit	Description
116-119*; 94-97†		Limit X, lower limit for processing. Optional.
120-123*; 98-101†		Limit Y, upper limit for processing. Optional.
124-127*; 102-105†		Offset, lower limits defined when file was first created. Optional.
129*; 107†		Segment tag for Error Address (S TAG EA)
130,131*; 108,109†		Error address. Address of error return.

*Byte positions unique to BDT for relative files when buffer 2 is present.

†Byte positions unique to BDT for relative files when buffer 2 is absent.

Table 2-11. Buffer Description Table Format for Indexed Files

0	GET/PUT Address														
2	NB	FTYP	S	LABL	AM	RT	Blocking Factor								
4	Block Size														
6	Record Size														
8	S TAG GP MOD							S TAG RA							
10	Record Area Address														
12	File Identification														
18															
20	K ₁							S TAG SA							
22	Save Area Address														
24	Error Offset														
26	K ₂							S TAG RAP							
28	Relative Address Pointer														
30	Relative Record Number														
32															
34	Current Block Number 1														
36															
38	Logical Record Number														
40	PU	AB	P	EOF	IS	IF	I	V		IG		PG	IW		Use
42	Work Area														
44															
46															
54	Move Routine														
56															
66	Block I/O Packet														
68															
88	Buffer 1 I/O Packet														

Table 2-11. Buffer Description Table Format for Indexed Files (Continued)

90	Index Buffer I/O Packet	
110		
112	K ₃	S TAG AK
114	KEYADR1 Address Pointer	
116	C ₁	C ₂
118	Index Buffer Block	
120	Current Index Block	
122		
124	Key Size	S TAG IKP
126	Index Block Key Address Pointer	
128	Directory to the Directory Block Address	
130	Number of Directory Blocks	
132	Physical Block Number	
134		
136	Compare Routine	
144		
146	F D [Shaded]	S TAG FK
148	KEYADR2 Address Pointer	
150	Buffer 2 I/O Packet	
170		
172	Current Block Number 2	
174		
176	[Shaded]	S TAG HLDB
178	Directory to the Directory Block Pointer	
180	[Shaded]	S TAG EA
182	Error Address	

*Optional

If buffer 2 is absent, bytes 150 through 157 end the BDT as shown:

Table 2-11. Buffer Description Table Format for Indexed Files (Continued)

150		S TAG HLDB
152	Directory to the Directory Block Pointer	
154		S TAG EA
156	Error Address	

*Optional

Byte	Bit	Description
0,1		External address of the GET/PUT module processing this file
2	0	Number of buffers (NB) 0 One buffer 1 Two buffers
	1,2	File type (FTYP) 00 Sequential 01 Relative 10 Indexed
	3	Shared buffer flag (S). If S=1, I/O buffers are shared with other files.
	4,5	Label processing (LABL) 00 No labels 01 Nonstandard labels 10 Standard labels
	6	Access method (AM) 0 Sequential 1 Random
	7	Record type (RT) 0 Fixed length 1 Variable length
3		Blocking factor, number of records per block (0-255)
4,5		Block size (in bytes) of the I/O buffer(s), size must include record headers
6,7		Record size (in bytes) of maximum record excluding header
8		Segment tag for GET/PUT module (S TAG GP MOD)
9		Segment tag for record area address (S TAG RA)
10,11		Record area address, first byte of record area
12-19		File identifier for data file and information file (8 bytes, EBCDIC)

Table 2-11. Buffer Description Table Format for Indexed File (Continued)

Byte	Bit	Description
20		Spread factor (K_1), indicating how many passes made on each track
21		Segment tag for save area address (S TAG SA)
22,23		Save area address, calling program's linkage and register save area
24,25		Error offset, offset (in bytes) to error address from beginning of BDT
26		Number of blocks per track minus one (K_2)
27		Segment tag for record address pointer (S TAG RAP)
28,29		Record address pointer, first byte address of active record in active I/O data buffer
30-33		Relative record number, record number relative to the first record in the file
34-37		Current block number 1, physical block number present in data buffer 1
40	0	If PU=1, the index block key address pointer was updated by a DELR, PUT, or PUTU instruction in sequential access mode.
	1	Active buffer (AB) 0 First buffer 1 Second buffer
	2	PUT flag (P). If P=1, record has been written in the active data buffer
	3	End-of-file flag (EOF). If EOF=1, the end-of-file has been reached.
	4	Index sharing flag (IS). If IS=1, index buffer is shared with data buffer.
	5	Index file flag (IF). If IF=1, index block was just read or written.
	6	I/O register flag (I) 0 Record area 1 I/O register
	7	Verify flag (V). If V=1, write verify is performed.
41	1	Ignore flag (IG). If IG=1, ignore option was selected.
	3	Previous GET flag (PG). If PG=1, last operation was a GET.
	4	Index write flag (IW). If IW=1, KEY is added or deleted in current index block.
	6,7	I/O usage (USE) 00 Input 01 Update 10 Output
42-45		Work Area
46-55		Move routine. Data movement routine consisting of a MOVL followed by a BR instruction.

Table 2-11. Buffer Description Table Format for Indexed Files (Continued)

Byte	Bit	Description
56-67		Block I/O packet, used to issue service requests such as CLOVE, POSITION, and ADD KEY/DELETE KEYS.
68-89		Buffer 1 I/O packet for I/O between the data file and the first I/O buffer.
90-111		Index buffer I/O packet. I/O parameter packet for I/O between the information file and the index buffer.
112		Number of blocks per pass minus one (K ₃).
113		Segment tag for key address 1 pointer (S TAG AK)
114,115		Key address 1 (KEYADR1) pointer, first byte address of key address 1 field
116		Current pass boundary counter (C ₁)
117		Current track boundary counter (C ₂)
118,119		Index buffer size, in bytes
120-123		Current index block number; physical block number present in index buffer.
124		Key size, size (in bytes) of primary key field
125		Segment tag for index block key address pointer (S TAG IKP)
126,127		Index block key address pointer, first byte address of active key in index buffer
128,129		Block address of directory to the directory block
130,131		Number of directory blocks written
132-135		Physical block number of data block to be read or written
136-145		Compare routine, data comparison consisting of a CMPX instruction followed by a BR instruction.
146	0	KEYADR2 flag (F) 0 KEYADR2 not present 1 KEYADR2 present
	1	Directory to the directory block flag (D) 0 Directory to the directory block in mass storage 1 Directory to the directory block in main storage
147*		Segment tag for key address 2 (KEYADR2) pointer (S TAG FK)
148,149*		Key address 2 (KEYADR2) pointer
150-171*		Buffer 2 I/O packet, parameter packet for I/O between data file and second I/O buffer
172-175*		Current block number 2, physical block number present in data buffer 2
177*; 151†		Segment tag for directory to the directory block pointer (S TAG HLDB)
178,179*; 152,153†		Directory to the directory block pointer, first byte address of main storage directory to the directory block

Table 2-11. Buffer Description Table Format for Indexed Files (Continued)

Byte	Bit	Description
181*;155†		Segment tag for Error Address (S TAG EA)
182,183*; 156,157†		Error address, address of error return

*Byte positions unique to BDT for indexed files when buffer 2 is present.

†Byte positions unique to BDT for indexed files when buffer 2 is absent.

3. BLOCK INPUT/OUTPUT

INTRODUCTION

The block level of input/output processing may be applied either to files which were previously processed at the logical level* or to files intended for use at the block level only. But whatever the organization of the file being processed, the data is referenced by key block number or as next data block for the unit record devices.

The basic unit of data transfer at the block (and physical) level is the block. Thus the block-input/output user may only read or write a whole block of data at a time (not individual records within a block).

If the user is operating on files created specifically for block input/output use, he is free to establish within the blocks any arrangement or grouping of logical records he wishes, and the blocking and deblocking of these records must of course be performed by the user himself. However, it should be noted that if he wishes to operate on previously-created files which were used for logical input/output, he must be aware of the pre-established logical structure of the data within the blocks.

*Sequential and relative files may easily be processed by block input/output but it is not practical to apply block input/output to indexed files.

GENERAL RULES

Tape and disc files for block input/output processing are created in the same way as those created for logical input/output (that is, through Control Language //DEF statements external to the program or ALLOC macros internally). These are standard system file structures and the following rules of file logic must be observed when processing them:

- A file must be opened before it may be processed
- All blocks within a given disc file must be of the same length
- Blocks must be at least two bytes long for disc requests (printer and card punch requests are two bytes long).
- Block numbers may go from 1 to 2³²-1 (disc block is 2²⁴-1)
- All buffers must begin on word boundaries for disc
- Read buffers will terminate on word boundaries for disc

In addition, the following restrictions should be noted:

- Processing across volume boundaries on sequential files is not allowed without calling CLOVE
- Multi-block reading will only be implemented for files with blocks of an even byte length
- Key fields are not supported for disc storage and will be ignored if present

BLOCK INPUT/OUTPUT CODING

A set of system macro instructions is provided for block input/output coding. The basic macros are READ, WRITE, POSITN and CNTRL. With these instructions blocks of data may be read and written, files may be positioned to particular blocks in preparation for processing, and certain hardware commands not involving data transfer may be performed. Other facilities that might be required are provided by the STATUS macro, which allows file status checking; the TYPE macro, which returns device type; and the RESET macro, which enables error conditions to be cleared (to allow for continuation after an error).

A particular block of data within a file is referred to by number. All files are ordered sequentially, with the first block of the file being block number 1. As an aid to processing files in a sequential manner, explicit block numbers are not necessary. Any data request with an implied block number causes the block number to be updated. The block referred to by a request is determined relative to an internal block number maintained by the I/O control routines.

BLOCK READING

The READ macro reads a block of data from a specified file and stores it at a specified buffer address. The number of the block to be read is obtained either directly from the READ call (if specified) or by adding 1 to the current block count saved by the system. After the READ is executed the updated block number will replace the old block number in the file's block counter so that a subsequent READ will automatically read the next block.

If a POSITN macro (to the same file) is executed immediately preceding a READ, the current block number will be used without change. When a file is opened for input (reading) its block counter is initially set at 1.

BLOCK WRITING

The WRITE macro transfers a block of data from a specified buffer address to a specified file. The number of the block into which the data is to be written is obtained either directly from the WRITE call (if specified) or by adding 1 to the current block

Table 3-1. Assumed Block Numbers

Operational Sequence	Effect
READ } READ } READ }	<u>Disc:</u> Each Read causes a one-count increase in the block number before the operation. <u>All:</u> Records are read sequentially.
WRITE } WRITE } WRITE }	<u>Disc:</u> Each Write causes a one-count increase in the block number after the operation. <u>All:</u> Records are written sequentially.
POSITN } POSITN } } or } READ } WRITE }	The block referred to by Positn is used for the Read or Write request.
READ } WRITE }	<u>Disc:</u> The block written replaces the block just read. <u>Reader/Punch:</u> The block is written into the card just read (file open for update); and the block is written into the next card (file open for output). <u>Mag Tape:</u> The block is written after the block just read.
WRITE } READ }	<u>Disc:</u> The block number is increased after the WRITE operation and not prior to the Read operation. This avoids skipping the next sequential block. <u>Reader/Punch:</u> Card n is written. Card n+1 is then read. <u>Mag Tape:</u> Invalid sequence.

count saved by the system. After the WRITE is executed the updated block number will replace the old block number in the file's block counter so that a subsequent WRITE will automatically write into the next block.

If a POSITN macro (to the same file) is executed immediately preceding the WRITE the current block number will be used without change.

If a READ macro immediately precedes a WRITE, the current block number will be increased by 1 in the normal way for a magnetic tape file but will not be altered in the case of a disc file.

When a file is opened for output (writing), what happens to the block counter depends on the type of file. For scratch, temporary and work files, opening the file sets the block counter to 1. For permanent files, however, the block counter will not be altered and will retain the value left in it on the previous use. This enables data already contained on the file to be saved between different job runs by adding new data only to the end of the file (unless otherwise specified).

Table 3-1 gives the assumed block numbers for the READ, WRITE, and POSITN macros.

BLOCK POSITIONING

The current block number of a file may be preset by a POSITN macro for subsequent reading or writing. This macro sets the current block counter of a specified file to a specified value.

DEVICE CONTROL COMMANDS

Input/output device control commands, not involving a data transfer, may be implemented by the CNTRL macro. This macro transmits a specified command to the device containing a specified file. Typical commands transmitted by this macro are *Skip to Top of Form* for line printer, *Rewind Tape* for tape drive.

SPACE MANAGEMENT AND FILE CONTROL

Space management and file control for block input/output are handled by Data Management functions described in Section 2. For space management ALLOC, EXPAND and PURGE are used. For file control OPEN, CLOSE and CLOVE are used.

PROCESSING CONSIDERATIONS

REQUEST OVERLAP

Multiple requests to the same file or the same device can be issued. However, to avoid ambiguous results, separate request blocks and data buffers must be used. Requests will be honored in the order of receipt within a file. Issuing multiple requests, with the aid of the RETURN=YES operand in the requesting macros, improves throughput by enabling the system to overlap input/output set-up time with data transmission.

PRIORITY

Input/output requests are processed by the system according to the priority of the program which issues them. This priority is specified in the Control Language //JOB card and is set at the time the job is initiated.

END CONDITIONS

End conditions are special boundary conditions resulting in return indications to the user but not considered as errors.

- End of File (EOF)

EOF is a logical boundary defined for input files. Each device* capable of reading has a defined end of file condition. EOF provides a condition in the data stream which is uniquely detectable by the system. After EOF has been detected on disc, magnetic tape, or the card reader, a RESET macro must be issued before processing may continue.

On the card reader†, a data record beginning with the characters /* is defined to be an end-of-file condition. When EOF is detected, the entire record will be transferred to the user's buffer.

On magnetic tape the EOF condition is set whenever a tapemark is detected.

* An exception to this is the card reader where the /* card is used to indicate the EOR condition in the EBCDIC mode. In the EBCDIC=NO mode, there is no EOF condition detectable by an I/O driver because all data images are considered to be legal data.

† The operator-selected EOF option on IBM card readers is not supported.

With disc, the EOF condition is set when a block with a data-length specification of zero has been read. Note that an embedded EOF will not be detected, however, during a multi-block read.

- End of Allocation (EOA)

EOA is a physical boundary applying only to disc output files. An EOA indication is returned any time a block is written into the last allocated space for the file.

- End of Tape (EOT)

EOT is a physical boundary applying only to magnetic tape output files. It signifies that the end-of-tape reflective marker has been sensed.

PROCESSING MULTI-VOLUME FILES

When processing sequential files occupying more than one physical volume (disc and tape only), the block input/output user must perform volume switching to move from one volume to another.

At any given time, the File Description Table for a sequential file* may only describe a single physical volume, which limits the current range of processing to the described volume. When, during processing, the user encounters an EOVS indication or wishes to prematurely close the volume and switch to the next volume he must issue a CLOVE (close volume) macro to continue processing on the next consecutive volume. CLOVE ensures that the next volume is mounted, by issuing an operator message if necessary, and then modifies the FDT to describe the next volume.

If there are no more volumes to be processed, the EOF/EOA bit (bit 2 of the return information) in the CLOVE packet is set. The user is responsible for testing this condition.

SENSE INFORMATION

The number of bytes of sense information varies with the device. All the bytes for the device are maintained in the File Description Table, and are updated whenever an abnormal completion occurs for reasons

other than logical errors. A STATUS macro request transfers this information from the File Description Table to the user buffer (to the extent that user buffer allows).

REQUEST TERMINATION

The user requesting a block I/O operation has two options while a request is processed:

- to wait for the request to be completed.
- to return to do parallel processing after the request is recognized and before the request is processed to completion.

If the user elects to wait for the request to be completed before continuing with processing, his program will be suspended by the operating system until the request has been terminated.

If the user has returned to do parallel processing, the COMPLETE indicator in the parameter string is set when the request is returned from the I/O control routines.

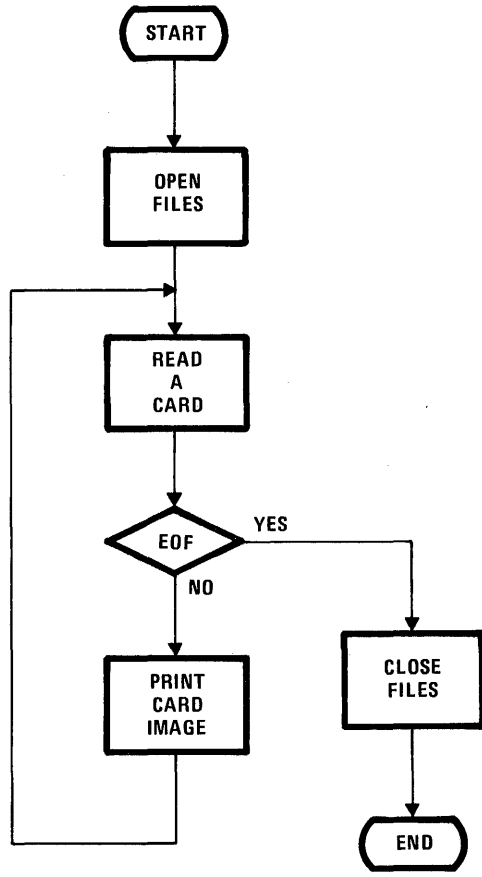
ERROR PROCESSING

At the block input/output level, the system automatically provides attempted recovery from hardware device errors. If peripheral device error recovery is successful, control is returned to the user with no error indication.

In the case of an error return (where peripheral device error recovery has been unable to correct the error), the program (and the rest of the job) normally is aborted. However, if ERRCOMP=YES is coded in the request macro, control is returned to the user together with return information (Appendix C) thus enabling him to process the error condition himself. It should be noted that before issuing another request to the same file following an error return, the user may first have to reset the error condition with a RESET macro (Appendix C).

Examples of block input/output are shown in Figures 3-1 and 3-2.

* An FDT for a relative file describes the whole file, therefore eliminating the need for CLOVE in this case.



LABEL	OPER'N	OPERANDS	COMMENTS
START	OPEN OPEN READ	IDENT=CARDIN,IOTYP=B,USAGE=1 IDENT=PRINT,IOTYP=B,USAGE=0 IDENT=CARDIN,DATABUF=CRDIMG	OPEN CARD INPUT FILE OPEN PRINTER OUTPUT FILE READ A CARD
END	TBIT BOV WRITE B CLOSE CLOSE HALT	2,START+4 END IDENT=PRINT,DATABUF=CRDBUF START IDENT=CARDIN,IOTYP=B IDENT=PRINT,IOTYP=B	CHECK FOR END-OF-FILE IN REQ BLOCK IF EOF, GO CLOSE FILES AND END PROGRAM PRINT CONTENT OF A CARD GO READ ANOTHER CARD CLOSE CARD FILE CLOSE PRINTER FILE TERMINATE PROGRAM
* * * * * DATA AREA * * * * *			
CRDBUF	WRS	80	CARD/PRINT BUFFER
CARDIN	WDD	C'CARDIN	CARD FILE IDENT
PRINT	WDD	C'PRINT	PRINTER FILE IDENT

Figure 3-1. Block I/O Program to Read Cards and Print

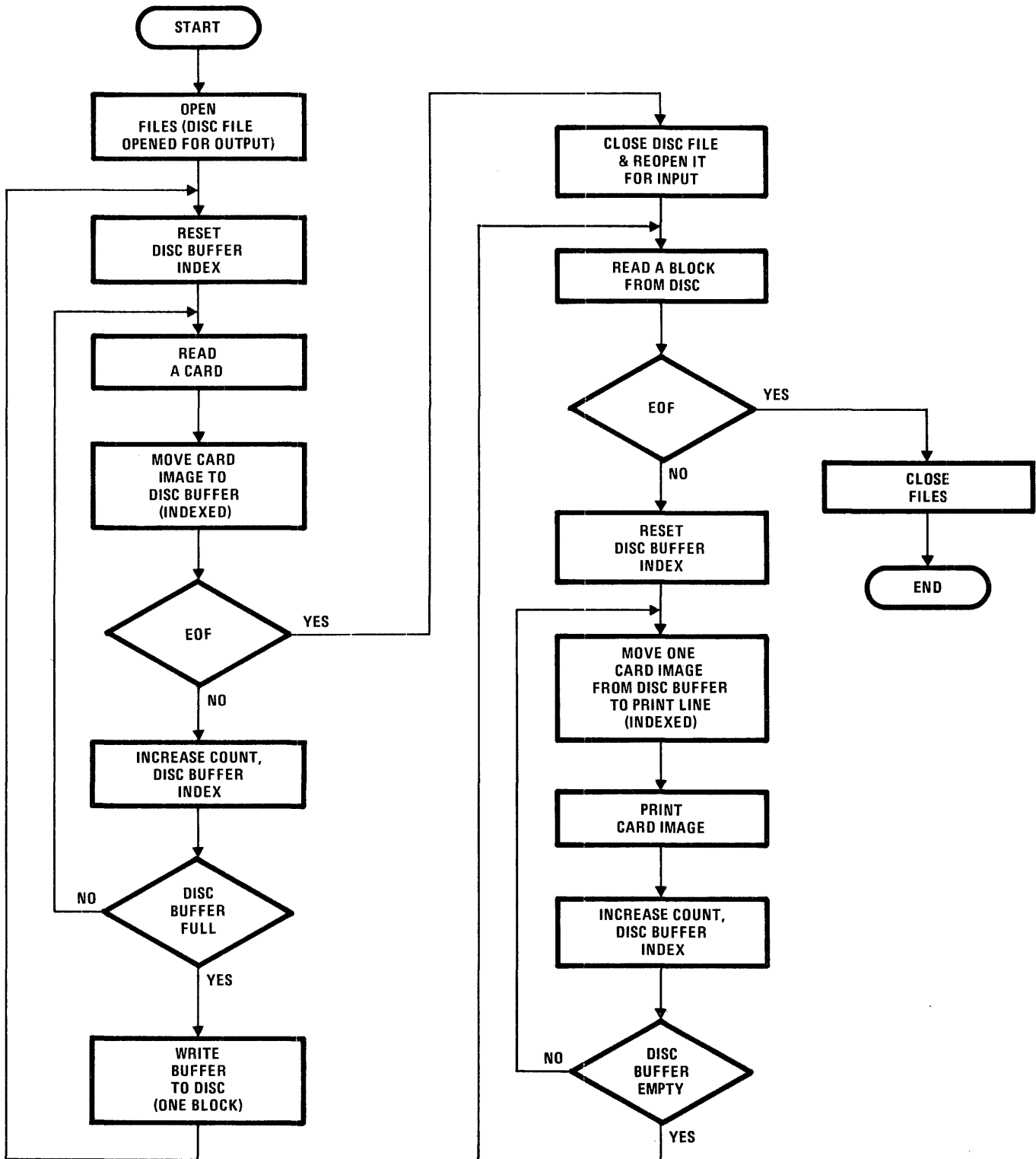


Figure 3-2. Block I/O Program to Read Cards into Disc File

LABEL	OPER'N	OPERANDS	COMMENTS
START CARDRD	OPEN	IDENT=CFIL,IOTYP=B,USAGE=I	OPEN CARD FILE
	OPEN	IDENT=DFIL,IOTYP=B,USAGE=0	OPEN DISC FILE FOR OUTPUT
	OPEN	IDENT=PFIL,IOTYP=B,USAGE=0	OPEN PRINTER FILE
	LODI	3,0	ZERO DISC BUFFER INDEX
	READ	IDENT=CFIL,BUFADR=CBUF	READ A CARD
	MOVX	DBUF(80,3),CBUF(80)	MOVE CARD IMAGE TO DISC BUFFER
	TBIT	2,CARDRD+4	CHECK FOR END-OF-FILE
	BOV	RESET	IF EOF, GO READ DISC FILE
	ADD	3,80	INCREMENT BUFFER INDEX
	CMPD	3,800	CHECK IF BUFFER FULL
	BNE	CARDRD	IF NOT, GO READ ANOTHER CARD
DISCWR	WRITE	IDENT=DFIL,BUFADR=DBUF,RETURN=YES	WRITE 10-CARD BUFFER ON DISC FILE
	B	START	GO READ ANOTHER 10 CARDS
RESET	CLOSE	IDENT=DFIL,IOTYP=B	CLOSE DISC FILE
DISCRD	OPEN	IDENT=DFIL,IOTYP=B,USAGE=I	REOPEN DISC FILE FOR INPUT
	READ	IDENT=DFIL,BUFADR=DBUF	READ A BLOCK FROM DISC
	TBIT	2,DISCRD+4	CHECK FOR END-OF-FILE
	BOV	END	IF EOF, GO END PROGRAM
	LODI	3,0	ZERO DISC BUFFER INDEX
	MOVX	PBUF(80),DBUF(80,3)	MOVE CARD IMAGE FROM DISC BUFFER TO PRINT LINE
	PRINT	WRITE	IDENT=PFIL,BUFADR=PBUF
	ADD	3,80	INCREMENT DISC BUFFER INDEX
	CMPD	3,800	CHECK IF BUFFER HAS BEEN EMPTIED
	BNE	PRINT	IF NOT, GO PRINT ANOTHER CARD IMAGE
	B	DISCRD	GO READ ANOTHER BLOCK FROM DISC FILE
END	CLOSE	IDENT=CFIL,IOTYP=B	CLOSE CARD FILE
	CLOSE	IDENT=DFIL,IOTYP=B	CLOSE DISC FILE
	CLOSE	IDENT=PFIL,IOTYP=B	CLOSE PRINTER FILE
	HALT		TERMINATE PROGRAM
* * * *	* * * * *	* * * * * DATA AREA * * * * *	* *
CBUF	WRS	80	CARD BUFFER
DBUF	WRS	800	DISC BUFFER
PBUF	WRS	80	PRINT LINE
CFIL	WDD	C'CFIL	CARD FILE IDENT
PFIL	WDD	C'PFIL	PRINTER FILE IDENT
DFIL	WDD	C'DFIL	DISC FILE IDENT

Figure 3-2. Block I/O Program to Read Cards into Disc File (Continued)

4. PHYSICAL INPUT/OUTPUT

INTRODUCTION

The physical I/O interface gives the user ability to utilize the device drivers to perform hardware-dependent I/O operations.

Physical input/output is independent of the system's file processing scheme. Whereas block input/output deals with files, physical input/output deals directly with hardware devices. Use of physical I/O assumes the following:

- Devices to be used must be defined by Control Language //DEF statements
- Devices must be opened for data transmission (by the OPEN macro)
- Error recovery will automatically be provided by the system (but may be bypassed if desired)

PHYSICAL INPUT/OUTPUT CODING

DEFINING AND OPENING DEVICES

Each device to be used for a physical input/output operation must first be assigned by the system. This is done through a System Control Language //DEF statement. An OPEN macro must also be coded in the user program to return the assigned unit ordinal used in building the PCB (next paragraph).

PERFORMING THE PHYSICAL I/O OPERATION

To perform a physical I/O operation, three entities must be created in the user's program:

- a command program
- a physical command block (PCB)
- a "do I/O" instruction (EXCP)

The command program does not consist of directly executable codes but is a chain of "command words" which will be operated on by the system's I/O processing routines. The program should be located in a data area of the program and may be built by means of COMMAND macros.

Examples of command word functions are these:

- print a record on a line printer
- select a stacker on a card reader
- seek on a disc file
- read a card record from a card reader

For every physical input/output operation a Physical Control Block (PCB) is required. The PCB should be located in a data area. The Physical Control Block may be built by a PCB macro.

To initiate the execution of physical I/O command programs a single action macro, EXCP, is required; it must be coded in line with the program's executable code. While the above general requirements are fixed, there are several variations in the detailed implementation, some of which allow more efficient use of memory space than others. Efficient use of device may be another consideration for not doing this. First the basic method will be explained, and then the more efficient methods will be described.

Basic Method

In the simplest situation, there is one PCB for each command program and one EXCP for each PCB (Figure 4-1). In this case, the address of the command program may be assembled into the PCB by the CPADR operand in the PCB macro, and the address of the PCB may be assembled (by means of the PCB operand) into the corresponding EXCP macro.

Sharing a PCB

When there are several command programs applying to the same device, coding efficiency may be improved by having these command programs all share the same PCB (Figure 4-2). In this case the address of the appropriate command program must be dynamically supplied to the PCB before each operation. This may be done by supplying the appropriate command program address through the related EXCP (by means of the CP operand) rather than by presetting it into the PCB. Thus a group of EXCP's may all specify the same PCB but different command programs. Input/output operations to different devices may be overlapped by coding RETURN=YES in the appropriate EXCP instruction.

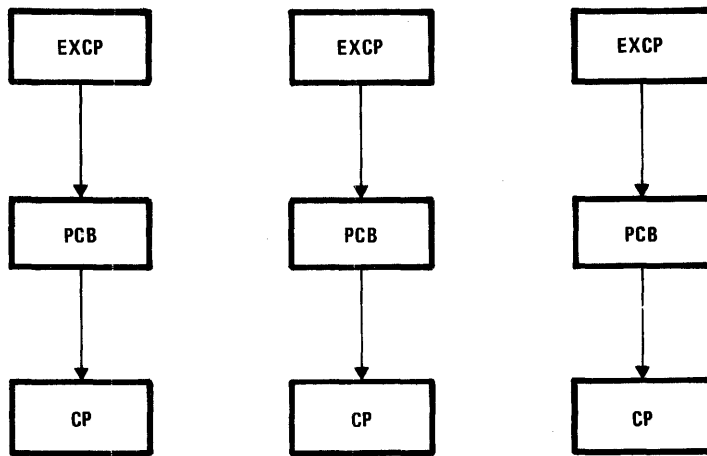


Figure 4-1. Basic Method for Physical I/O

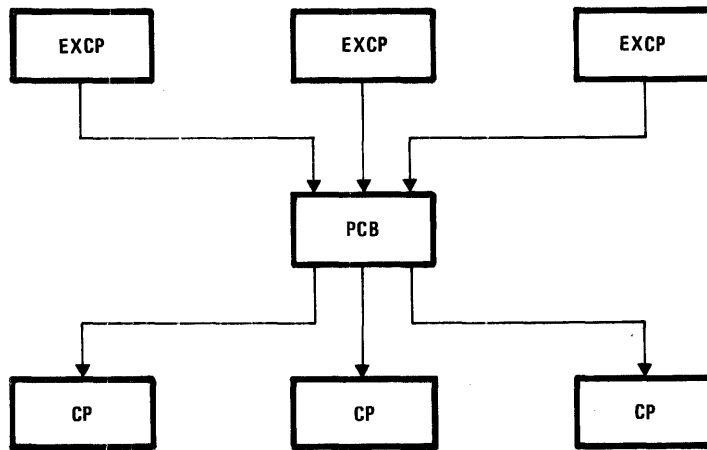


Figure 4-2. Sharing a PCB

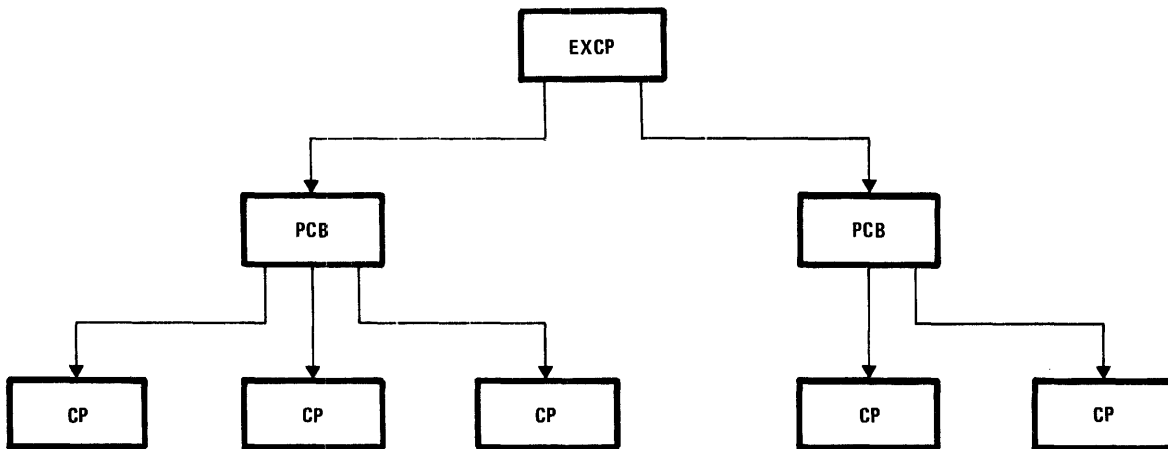


Figure 4-3. Sharing an EXCP

Sharing an EXCP

Sometimes it is possible to employ a single EXCP to operate through a PCB on several command programs or even to operate on several PCB's (Figure 4-3). When this is to be done, the user may set up a table of command program addresses (and PCB addresses, if there are multiple PCB's). The EXCP will then refer to the address tables at execution time by indirect addressing with the CPADR operand and PCBADR operand.

OVERLAPPED OPERATION*

There is a PCB for each input/output device; input/output operations to different devices may be overlapped by coding RETURN=YES in the relevant EXCP instruction(s). This causes control to be retained by the requesting program (while the I/O operation itself is concurrently being processed by the system), thus enabling initiation of another I/O operation before the first has been completed.

Beyond this, it is also possible to overlap requests to the same device, but it should be remembered that a separate PCB and data buffer will be required for each such overlapped request. When multiple requests are issued in this manner, throughput will be improved since the system is able to overlap I/O set-up time with data transmission. (The system will always service multiple requests to the same device in order of receipt.)

PHYSICAL REQUEST TERMINATION

When a physical I/O request is completed, several values are returned to the calling program in the

command block (PCB):

- address of the last command word executed or attempted to be executed
- hardware status indication
- residual byte count of last data operation

PHYSICAL I/O RESTRICTIONS

- Byte count must be less than 216-1 (65K)
- Multi-record read operations must not attempt access to records across track boundaries (disc).
- The user is responsible for validating a "read count field" request (disc).

ERROR PROCESSING

System error recovery (Appendix C) is provided by default just as it is at the block level. However, at the physical level, system error recovery may be bypassed by coding ERROPT=NO in the PCB macro. In any case, an uncorrected error will always cause the program to be aborted unless the operand ERRCOMP=YES is coded in the EXCP macro making the request. When ERRCOMP=YES is used, control is returned to the user in the event of an error, together with return information in the PCB so the user may either ignore it or process the error himself.

EXAMPLE OF PHYSICAL I/O PROGRAM

Figure 4-4 shows a block diagram of a program to read cards and print. A coding form illustrates the code necessary for this physical I/O program.

* The effectiveness of overlapped coding is dependent on the number of Queue Entry Blocks available in the system, a SYSGEN variable.

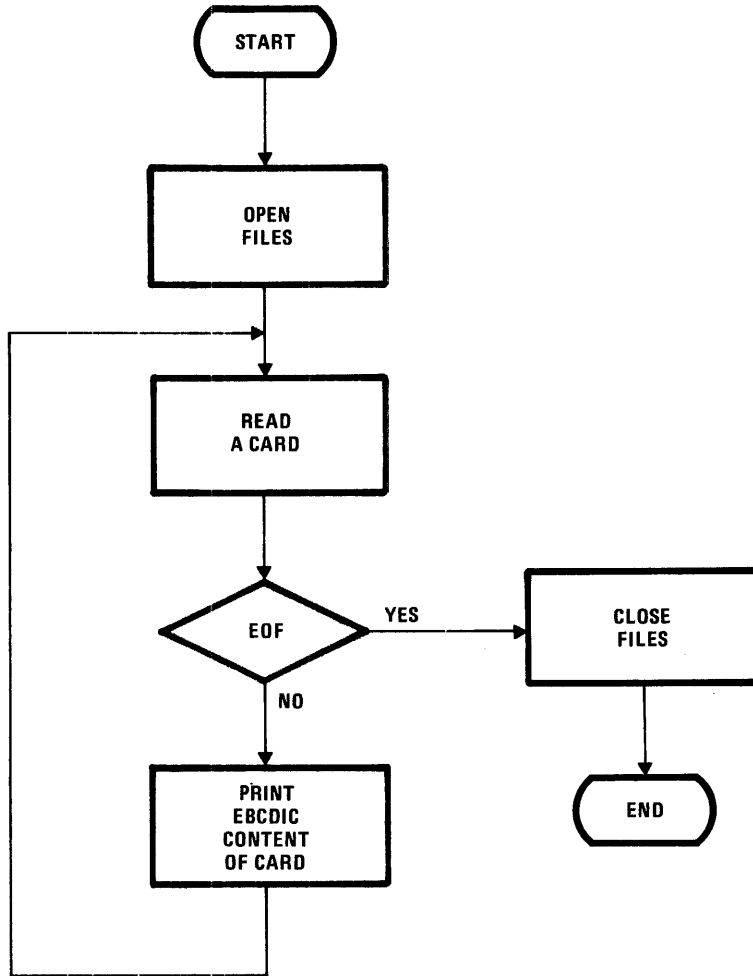


Figure 4-4. Physical I/O Program to Read Cards and Print

MEMOREX

Assembler Coding Form

Punching Instructions

Graphic					
Punch					

Date _____ Page _____ of _____
 Programmer _____
 Program _____

Figure 4-4. Physical I/O Program to Read Cards and Print (Continued)

NAME	OPERATION	OPERAND	IDENTIFICATION
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 65 66 67 68 69 70 71 72	73 74 75 76 77 78 79 80
OPEN	OPEN	IDENT=CARD,IOTYP=P,BUFADR=UNIT1	OPEN CARD FILE
	OPEN	IDENT=PRNT,IOTYP=P,BUFADR=UNIT2	OPEN PRINT FILE
START	EXCP	PCB=LOC1,UNORD=UNIT1	READ A CARD
	CMPX	CBUF(3),=C'/*'(3)	LAST CARD?
	BEQ	CLOSE	YES, GO CLOSE FILES
	MOVX	CBUF(80),PBUF(132)	MOVE CARD DATA TO PRINT LINE
	EXCP	PCB=LOC2,UNORD=UNIT2,RETURN=YES	PRINT THE LINE
	B	START	GO READ ANOTHER CARD
CLOSE	CLOSE	IDENT=CARD,IOTYP=P	CLOSE CARD FILE
	CLOSE	IDENT=PRNT,IOTYP=P	CLOSE PRINT FILE
	HALT		TERMINATE
*****DATA AREA			
READ	COMMAND	OPCODE=X'02',DATBUF=CBUF,DATSIZE=80	READ COMMAND WORD
PRINT	COMMAND	OPCODE=X'09',DATBUF=PBUF,DATSIZE=132	PRINT COMMAND WORD
PCB1	PCB	DEVTYP=UR,CPADR=READ	CARD-READ PCB
PCB2	PCB	DEVTYP=UR,CPADR=PRINT	LINE-PRINT PCB
LOC1	WDD	PCB1	ADDRESS OF PCB1
LOC2	WDD	PCB2	ADDRESS OF PCB2
CARD	WDD	C'CARD'	CARD FILE IDENT
PRNT	WDD	C'PRNT'	PRINT FILE IDENT
UNIT1	WDD	0	OPEN BUFFER FOR CARD READER
UNIT2	WDD	0	OPEN BUFFER FOR LINE PRINTER
CBUF	WRS	80	CARD BUFFER
PBUF	WRS	132	PRINT BUFFER
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 65 66 67 68 69 70 71 72	73 74 75 76 77 78 79 80

5. CONTROL PROGRAM SERVICES

INTRODUCTION

The executive services are those implemented directly by the Control Program itself to assist the user in various areas, such as input/output request control and communication between job steps.

SERVICE REQUEST CONTROL

Control over the issuance of service requests by the user program may be aided by the WAIT, INFORM and DELAY* macros. In one way or another, all of these macros can detect completion of the user's service requests.

WAIT simply suspends program execution until completion of the specified request or requests. WAIT is used to wait for any one or all of the outstanding service requests.

DELAY enables the user to suspend execution for a specific period of time with the option of breaking the delay (resuming execution) on the completion of any service request.

INFORM offers the capability of detecting the completion of a service request in the interval between the issuances of the INFORM and its actual processing by the system. This capability, which is not available with the WAIT macros, is implemented with the aid of a user-supplied count of the number of outstanding requests he has at the time of issuing the INFORM. By comparing this count with its own count of outstanding requests, the system can determine whether any requests had been completed in the time since INFORM was issued by the user. In addition to this, INFORM always returns control immediately to the user (it has an implied RETURN=YES operand) so that he may continue processing and subsequently check completion of the INFORM by testing the *complete* bit in the request block.

INTER STEP AND CONTROL LANGUAGE COMMUNICATION

User programs running as separate job steps in a

multi-step job have two ways of transferring data to one another: The POST/RPOST pair of macros† and the SETCOM/GETCOM pair of macros.

In addition, any program may influence the subsequent course of the job by means of the SETIF macro which posts information to be tested by an //IF statement between later job steps.

The POST/RPOST pair of macros post and read a single byte of data in the Job Control Table, whereas the SETCOM/GETCOM pair post and read a full eight bytes.

FINDING PARTITION SIZE

When a program is being written which is expected to run in partitions of various sizes, it would often be advantageous to code the program in such a way as to occupy as much of the partition as is currently available.

To achieve this flexibility, a dynamic determination of current partition size is necessary; this is provided by the MEMLIM macro.

MEMLIM informs the program of the size of the user portion of the partition in which it is currently running by returning the starting address of the last addressable 256-byte page, expressed as an absolute address. In effect, MEMLIM returns the starting address of the last usable page below the partition space pool. For this reason, programs which make use of MEMLIM should always specify a fixed-length space pool through the appropriate Link Editor directive, since otherwise the space pool will automatically commence on the page boundary following the space allocated to the program.

*DELAY is not available on the minimal system.

†POST/RPOST conform to the IBM UPSI bit scheme.

READING DATA FROM //PAR CARDS

Data may be read from //PAR cards supplied in the job control deck by means of the ACCEPT macro. This macro transfers the contents of a single //PAR card into a specified buffer. On the first execution of ACCEPT, the first //PAR card is read; successive executions automatically read the rest of the //PAR cards consecutively. After the last //PAR card has been read the next execution of an ACCEPT transfers program control to a specified "end" address.

The user may also specify a particular //PAR card by using the PARNUM keyword in the ACCEPT macro.

WRITING TO THE SYSOUT FILE

SYSOUT files are system output files, one of which is created uniquely for each job*. Any user program may write a one-line EBCDIC message on the SYSOUT file for the job by means of the DISPLAY macro. The location of the message buffer should be specified on the related DISPLAY macro call. DISPLAY expands to in-line processing code, including an embedded block I/O WRITE to the SYSOUT file.

*The Control Language Services Reference manuals contain further explanations.

6. INTERACTION OF DATA MANAGEMENT AND THE CONTROL LANGUAGE

Through the //DEFINE statement, a run-time interaction between certain Data Management services and Control Language services is provided. This interaction gives the program independence from reassembling to make changes that can then be made at run-time through the control language statements.

Both the logical I/O level (described in **Control Program and Data Management Services, Basic Reference**) and the block I/O level require space management. Users of the physical I/O level require no space management. The Control Language and/or the space management macros (ALLOC, EXPND, and PURGE) can allocate, expand, or purge files.

If the Control Language is externally allocating a file, the parameters are passed to the ALLOC packet by the Step Initiator. A file is allocated by either the ALLOC macro or by the //DEFINE statement, not both. The ALLOC macro uses the FILENAME, MSC, and VOLUME parameters specified in the //DEFINE statement to override the parameter packet at execution time.

A space management routine is called in response to a service request from either the Control Language processor or the user program. The Control Language request for Allocate or Expand may be generated at step initiation time using parameters given in the //DEFINE statement. The Control Language request for Purge may be generated at step termination time, using parameters given in the pertinent files disc catalog entry. The user program request for Allocate, Expand, or Purge is generated at assembly time, using parameters given as keyword in a macro call. The user request may optionally make reference to FILENAME, MSC, and VOLUME information given in a //DEFINE statement. If so, at execution time, the FILENAME and MSC parameters in the //DEFINE statement override the equivalent parameters contained in the user request.

Table 6-1 summarizes the use of different Control Language parameters as used by the Step Initiator, OPEN, CLOVE, ALLOC, EXPND, and PURGE. Further explanations of the Control Language can be found in **Control Language Services, Extended Reference**.

Table 6-1. Summary of Data Management and Control Language Interaction

Control Language Keyword Parameter	Control Language Step Initiator	OPEN Macro	CLOVE Macro	ALLOC, EXPND, PURGE Macros
IDENTIFIER	Not used	Must match file identifier in OPEN macro.	Must match file identifier in CLOVE macro.	Must match file identifier in macro.
FILENAME	Used to search central catalog to determine volumes of a cataloged file.	Override DEFLB information.	Override DEFLB information.	Override DEFLB information.
STATUS	File usage checked for partition conflicts. File type passed to ALLOCATE packet* if NUMBER is specified.	Not used by OPEN	Not used by CLOVE	Not used by ALLOC, EXPND, or PURGE
MSC	Not used by Step Initiator	Override DEFLB information.	Override DEFLB information.	Override DEFLB information.
DEVICE	Used to allocate peripheral resources to files.	Not used by OPEN	Not used by CLOVE	Not used by ALLOC, EXPND, or PURGE
VOLUME (and IVOLUME for OPEN)	Used to mount packs or tape reels when possible and verify the mount by comparison of volume with volume identifier in the volume label.	When volumes have been mounted by Step Initiator, they are verified by comparison with unit table volume identifier entries. When not mounted, OPEN mounts the volume(s) and verifies against volume label.	Used to mount and verify successive volumes of sequential files.	Volume identifiers specify which packs are to be used for allocation or expansion. If omitted, shared resources are used. PURGE uses volume identifiers for purging uncataloged files.
SPREAD	Passed to the ALLOCATE packet*	Not used by OPEN	Not used by CLOVE	Not used by ALLOC, EXPND, or PURGE
CONTIGUOUS	Passed to the ALLOCATE packet*	Not used by OPEN	Not used by CLOVE	Not used by ALLOC, EXPND, or PURGE
LABEL	Not used by Step Initiator	Indicates type of label processing for the job. Control Language overrides program specification.	Indicates type of label processing for the job. Control Language overrides program specification.	Not used by ALLOC, EXPND, or PURGE
CSD	Used to compute block size from SIZE and BLOCK for ALLOCATE when NUMBER is specified.	Not used by OPEN	Not used by CLOVE	Not used by ALLOC, EXPND, or PURGE
NUMBER	Used to compute number of blocks from BLOCK and NUMBER for ALLOCATE. Also indicates need of ALLOC.	Not used by OPEN	Not used by CLOVE	Not used by ALLOC, EXPND, or PURGE

*Parameters are passed to the ALLOCATE parameter packet by the Step Initiator; the Control Language is allocating the file.

Table 6-1. Summary of Data Management and Control Language Interaction (Continued)

Control Language Keyword Parameter	Control Language Step Initiator	OPEN Macro	CLOVE Macro	ALLOC, EXPND, PURGE Macros
SIZE	Used to compute block size. Key size portion is passed to the ALLOCATE packet* for indexed files.	Not used by OPEN	Not used by CLOVE	Not used by ALLOC, EXPND, or PURGE
BLOCK	Used to compute block size for data and index files.	For tape files, Control Language supplies block size. Default value is 251 bytes.	Not used by CLOVE	Not used by ALLOC, EXPND, or PURGE
LOCATION	Passed to the ALLOCATE packet* when NUMBER is specified.	Not used by OPEN	Not used by CLOVE	Not used by ALLOC, EXPND, or PURGE
ORGANIZATION	Passed to the ALLOCATE packet*	Not used by OPEN	Not used by CLOVE	Not used by ALLOC, EXPND, or PURGE
RETENTION	Not used by Step Initiator	Used to generate an expiration date for tape files.	Used to generate an expiration date for tape files.	Not used by ALLOC, EXPND, or PURGE
CATALOG	Passed to the ALLOCATE packet*	Not used by OPEN	Not used by CLOVE	Not used by ALLOC, EXPND, or PURGE
VERIFY	Passed to the ALLOCATE packet*	If the file was cataloged with VERIFY=YES, OPEN does not interpret the parameter. If not, then VERIFY=YES at OPEN time will temporarily override the catalog attribute.	Not used by CLOVE	Not used by ALLOC, EXPND, or PURGE
EXPAND	Passed to the EXPAND packet	Not used by OPEN	Not used by CLOVE	Not used by ALLOC, EXPND, or PURGE

*Parameters are passed to the ALLOCATE parameter packet by the Step Initiator; the Control Language is allocating the file.

**Parameter is passed to the EXPAND parameter packet by the Step Initiator; the Control Language is expanding the file.

7. MACROS

This section gives the specifications for the block and physical I/O level functions of Data Management, Block I/O action, Physical I/O action, and Control Program Services. In general, all the macros have the following format:

Name	Operation	Operand
------	-----------	---------

The name field is an optional field which contains a 1- to 8-character alphanumeric file address. The first six characters must be unique to accommodate the standard suffixes used by the system. These are discussed in Appendix B of this manual. The names ident, labadr, and tag are used as identifiers for the software function specified in the operation field of the macro prototype.

The operand field contains keyword parameters which may be in any order separated by commas. Optional parameters are denoted by brackets, []. Parameters with a choice of specifications are denoted by braces, { }, and the default value is underlined.

Fields are free-form and are separated by blanks; thus, no imbedded blanks are allowed within the parameter string. If more than one card is necessary, a semicolon must appear after the last parameter on each card except the last.

Symbolic address (as used in macro prototypes) is the 1- to 8-character symbol used to identify a coding statement. The IDENT parameter is the symbolic address of the 8-byte field containing the file identifier (left-justified, blank filled). If the pertinent file is defined in the Control Language, the file identifier referenced by IDENT must be identical to the IDENT specified by the //DEFINE statement. Unless otherwise stated numbers are assumed to be in decimal with no leading zeros.

DATA MANAGEMENT

In this section only the block and physical I/O level of Data Management macros are given. This level of macros include these:

- Block I/O Level Declarative Macro
DEFLB
- Space Management Macros
ALLOC
EXPND
PURGE
- File Control Macros
OPEN
CLOSE
CLOVE
- I/O Service Macro
LABRTN
- Block I/O Macros
READ
WRITE
POSITN
CNTRL
STATUS
TYPE
RESET
- Physical I/O Macros
EXCP
PCB
- Control Program Macros
WAIT
DELAY
INFORM
POST-
RPOST
SETCOM
GETCOM
ACCEPT
DISPLAY
MEMLIM
SETIF
HALT
EHALT
ABEND
TIME
SDATE
JDATE
- Console Communication Macros
CONSOLE
MESSAGE

Appendix B details the service request mechanism and details the expansion of these macros.

BLOCK I/O LEVEL DECLARATIVE MACRO

A block I/O level declarative macro, DEFLB, defines the file label for Data Management.

DEFLB – Define File Label

The DEFLB macro generates file label data into a main-memory buffer for creating and checking disc file names. The Control Language may also be used to create and check file names. The block I/O level space management and file control macros use DEFLB. The format is as follows:

<u>Name</u>	<u>Operation</u>	<u>Operand</u>
labadr	DEFLB	FILENAM=name [MSC=code]

FILENAM=name

Specifies a 1- to 17-character alphanumeric file name. The first character may be A-Z, 0-9, or \$; A-Z, 0-9, and dash are allowed as succeeding characters. Index file names are created internally by adding an asterisk at the end of the associated data file name.

MSC=code (Optional)

Designates a 4-byte EBCDIC modification security code, which is used for work and permanent files. If omitted, blanks are assumed.

BLOCK I/O LEVEL AND PHYSICAL I/O LEVEL EXECUTIVE REQUESTS

The executive request macros generate requests for space management, file control, file processing, and file positioning at the block I/O level. These requests specify a file identifier and a specific block of data within the file. The file control and file processing macros are also available at the physical I/O level, specifying a unit identifier rather than a file identifier.

Files which are open for block I/O level file processing requests are described by a File Description Table (FDT) created by OPEN. FDT contains a physical description of the file, a 1- to 8-character file identifier, and current processing information about the file (for example, current block number and status of last request). File processing and positioning requests are made by file identifiers and controlled by the FDT.

Units which are open for physical I/O level processing are assigned by the Control Language. The identifier field in OPEN is used to find the correct Control

Language entry which gives the unit assignment. The //DEFINE statement indicates physical I/O by stating P/O in the file name field. Physical I/O requests are made by a unit ordinal which has been returned to the user after an OPEN.

Space Management Macros

The ALLOC, EXPND, and PURGE macros manage space for all disc files. Usually these three functions are generated through the Control Language and by a space management utility program, which allows changes without recompilation of the program; however, a user program can directly manage file space.

The allocation of paired files must be done through a utility if not done directly from a user program. The utility programs may also purge permanent and work files if not done directly from the user program.

Space management is required for usage of block I/O level and logical I/O level interfaces. Users of physical I/O level require no space management.

Space is allocated in increments of tracks. Paired file allocation of two files provides minimum arm movement for file processing. A contiguous or non-contiguous segment of space is allocated to both files starting at the chosen segment. Alternate tracks from this space are then assigned to each of the two files. File space allocated will be contiguous unless contiguous space is unavailable, in which case up to 12 segments will be allocated. Other allocation options are as follows:

- Multipack files (up to seven packs)
- Suppression of automatic segmentation
- Cylinder number specification

Each disc pack contains a volume label which is created at disc initialization time. A MRX/OS utility program may change the volume label and its parameters.

In its device label each disc pack contains pack status indication of one of the following conditions.

<u>Pack Status</u>	<u>Meaning</u>
Nonrestricted	Any allocation request may obtain space from this pack.
Restricted	Only allocation requests with packs specified by the Control Language may obtain space from this pack.
Locked	No further allocation is allowed on this pack.

Some disc drives are classified as shared resource drives at initial program load time. The operator mounts a set of packs on these drives, which will be available for any allocation requests. The remaining disc drives (non-shared-resource drives) are each assigned by job control to a specific user partition for the job step requesting the drives, and may be used only by allocation requests associated with the job step in the partition to which they were assigned.

ALLOC – Allocate Space

The ALLOC macro allocates space for user data files and system data files. When volume identifiers have not been specified through job control, space will be allocated on nonrestricted packs mounted on shared resource drives. When volume identifiers have been specified through job control, the disc drives on which they are mounted will have been assigned by job control to the partition (or system function) of the program calling ALLOC. Duplicate file names are not allowed on the disc catalogs. The format is as follows:

Name	Operation	Operand
[tag]	ALLOC	BLKSIZ=n [CATALOG= { YES / NO }] [CON= { YES / NO }] [CSD= { YES / NO }] [DATACYL= { YES / NO / n }] [ERRCOMP= { YES / NO }] [FILEORG=code] FILESIZE=n [FILETYP=code] [IDENT=symbolic address] [INDCYL= { YES / NO / n }] INDSIZ=n* KEYSIZ=n* LABDEF1=symbolic address [LABDEF2=symbolic address] [LIST= { YES / NO }] RECSIZ=n [RETURN= { YES / NO }] [SPREAD=n] * [VERIFY= { YES / NO }]

*For indexed files only.

BLKSIZ=n

Designates the number of 8-bit bytes per block for the file. The value n ranges from 18 bytes through 32K bytes for magnetic tape and 7294 bytes for disc.

CATALOG= { YES / NO } (Optional)

CATALOG=NO specifies that the file should not be centrally cataloged. If omitted or CATALOG=YES, the file is cataloged. The parameter is ignored for scratch and temporary files.

CON= { YES / NO } (Optional)

CON=YES specifies that contiguous space is to be obtained. Absence of this parameter or CON=NO allows segmentation.

CSD= { YES / NO } (Optional)

CSD=YES (default case) specifies that the file will be created with the common stored data format. CSD=NO specifies that a data format other than CSD will be used.

DATACYL= { YES / NO / n } (Optional)

DATACYL=n designates a cylinder number boundary for the beginning of the data file. The value n ranges from 1 to 199. DATACYL=YES ensures the file beginning on some cylinder boundary; whereas DATACYL=NO does not. If the parameter is omitted, the data file will not necessarily begin on a cylinder boundary.

ERRCOMP= { YES / NO } (Optional)

ERRCOMP=YES specifies that the caller of the request will examine return information on nonfatal errors; therefore, nonfatal errors should not be aborted by the system. Fatal errors such as illegal instructions, irrecoverable memory parity errors, privilege violations, and bound errors cause an unconditional abort. If ERRCOMP=NO or is omitted, no return information is examined.

FILEORG=code (Optional)

Determines the file organization. The possible codes are as follows:

Code	Organization
S	Sequential
R	Relative
I	Indexed

If the code is omitted, sequential file organization is used.

FILESIZ=n

Indicates an estimate of the number of blocks expected in the data file.

FILETYP=code (Optional)

Designates the file type. The possible codes are as follows:

Code	Type
S	Scratch
T	Temporary
W	Work
P	Permanent

If the code is omitted, the temporary file type is used.

IDENT=symbolic address (Optional)

Designates the 8-character file identifier. It should be identical to the operand of the IDENTIFIER keyword in a Control Language DEFINE statement specifying the volume on which to allocate the file. If IDENT does not match the IDENTIFIER specification (or if omitted) and the file is to be centrally cataloged, the file will be allocated on shared resources. IDENT is required if the LABDEF1 parameter is not specified.

INDCYL= $\begin{cases} \text{YES} \\ \text{NO} \\ n \end{cases}$ (Indexed files only)

INDCYL=n specifies a cylinder number boundary for the beginning of the index file. The value n ranges from 1 to 199. INDCYL=YES ensures the file will begin on some cylinder boundary; whereas INDCYL=NO does not. If the parameter is omitted, the index file will not necessarily begin on a cylinder boundary.

INDSIZ=n (Indexed files only)

Specifies the estimated byte size of the index blocks. The value of n ranges from 18 bytes to 7294 bytes.

KEYSIZ=n (Indexed files only)

Gives an estimate of the byte length of the primary key for indexed files. The value of n ranges from 2 bytes to 100 bytes.

LABDEF1=symbolic address

Specifies the symbolic address of a file label area which must correspond to the symbolic address specified as the name field of the DEFLB macro. LABDEF1 is required if the IDENT parameter is not specified.

LABDEF2=symbolic address (Optional)

Specifies a second file label address for paired file allocation requests. This option is available for sequential and relative files only.

LIST= $\begin{cases} \text{YES} \\ \text{NO} \end{cases}$ (Optional)

If LIST=YES, only the parameter packet is generated, and the RETURN and ERRCOMP options are not used. If LIST=NO, only the 2-byte standard *executive request* instruction is generated. Loading of general-purpose register 6 (R6) with the address of the parameter packet and general-purpose register 7 (R7) with the save area address prior to issuing the macro call is a user responsibility. If the LIST parameter is omitted, both the execution request instruction and the parameter packet are generated following the macro call. Appendix B contains a detailed discussion of the LIST parameter.

RECSIZ=n

Indicates the length of a logical record in bytes. The value n ranges from 18 bytes through 32K bytes for magnetic tape and 7294 bytes for disc. If omitted, RECSIZ equals BLKSIZ.

RETURN= $\begin{cases} \text{YES} \\ \text{NO} \end{cases}$ (Optional)

RETURN=YES specifies that the processor issuing the executive request should be given control (reactivated) as soon as the request is queued to its destination routine. If RETURN=NO or if the parameter is omitted, the calling processor is reactivated when the request processing is complete or terminates abnormally if ERRCOMP=YES.

SPREAD=n (Indexed files only)

Specifies the number of physical blocks separating two logically consecutive blocks in the indexed data file. The number of blocks between logically consecutive blocks is one less than the n value whose range is 1 through 10. If this parameter is omitted, the logically consecutive blocks are adjacent.

VERIFY= $\begin{cases} \text{YES} \\ \text{NO} \end{cases}$ (Optional)

VERIFY=YES specifies read after write verification of all writes to this file. If omitted or if VERIFY=NO, write verification will not be a permanent file characteristic. The VERIFY=YES option may be specified as a temporary override through logical level file definition macros or through Control Language DEFINE statements at file open time.

EXPND – Add Mass Storage Space

The EXPND macro obtains additional mass storage space for sequential files. The file may be open but cannot have input or update usage at the time of the EXPND request. The format is as follows:

<u>Name</u>	<u>Operation</u>	<u>Operand</u>
[tag]	EXPND	[CATALOG= $\left\{ \begin{array}{l} \text{YES} \\ \underline{\text{NO}} \end{array} \right\}$] [CON= $\left\{ \begin{array}{l} \text{YES} \\ \underline{\text{NO}} \end{array} \right\}$] [ERRCOMP= $\left\{ \begin{array}{l} \text{YES} \\ \underline{\text{NO}} \end{array} \right\}$] [FILESIZ=n] [IDENT=symbolic address] LABDEF=symbolic address [LIST= $\left\{ \begin{array}{l} \text{YES} \\ \underline{\text{NO}} \end{array} \right\}$] [PAIRED= $\left\{ \begin{array}{l} \text{YES} \\ \underline{\text{NO}} \end{array} \right\}$] [RETURN= $\left\{ \begin{array}{l} \text{YES} \\ \underline{\text{NO}} \end{array} \right\}$]
		CATALOG= $\left\{ \begin{array}{l} \text{YES} \\ \underline{\text{NO}} \end{array} \right\}$ (Optional)
		CATALOG=NO specifies the file is not centrally cataloged. If the parameter is omitted or CATALOG=YES, the file is centrally cataloged. The parameter is ignored for scratch or temporary files.
		CON= $\left\{ \begin{array}{l} \text{YES} \\ \underline{\text{NO}} \end{array} \right\}$ (Optional)
		CON=YES specifies addition of contiguous space. If absent or CON=NO, segmentation is allowed within the space added.
		ERRCOMP= $\left\{ \begin{array}{l} \text{YES} \\ \underline{\text{NO}} \end{array} \right\}$ (Optional)
		ERRCOMP=YES specifies that the caller of the request will examine return information on nonfatal errors; therefore, nonfatal errors should not be aborted by the system. Fatal errors such as illegal instructions, irrecoverable memory parity errors, privilege violations, and bounds errors cause an unconditional abort. If ERRCOMP=NO or if omitted, no return information is examined.
		FILESIZ=n (Optional)
		Indicates the number of blocks to add to the file.
		IDENT=symbolic address (Optional)
		Specifies the 8-character file identifier. If the file is

open at the time of expansion, it must match the name specified by the IDENT parameter of the OPEN macro. If the file is closed, it should match the operand of the IDENTIFIER keyword in a Control Language DEFINE statement specifying a volume identifier for expansion. If IDENT does not match the IDENTIFIER specification (or if omitted) and the file is centrally cataloged, the expansion will be to the shared device. IDENT is required if LABDEF is not specified. IDENT may be omitted only for cataloged files.

LABDEF=symbolic address

Specifies the symbolic address of a file label which must be identical to the label address of the DEFLB macro. This parameter is required if the IDENT parameter is not specified.

LIST= $\left\{ \begin{array}{l} \text{YES} \\ \underline{\text{NO}} \end{array} \right\}$ (Optional)

If LIST=YES, only the parameter packet is generated, and the RETURN and ERRCOMP options are not used. If LIST=NO, only the 2-byte standard executive request instruction is generated. Loading of R6 with the address of the parameter packet and R7 with the save area address prior to issuing the macro call is a user responsibility. If the LIST parameter is omitted, both the executive request instruction and the parameter packet are generated following the macro call. Appendix B contains a detailed discussion of the LIST parameter.

PAIRED= $\left\{ \begin{array}{l} \text{YES} \\ \underline{\text{NO}} \end{array} \right\}$ (Optional)

PAIRED=YES indicates the file is paired. If PAIRED=NO or if the parameter is omitted, there is no pairing of files.

RETURN= $\left\{ \begin{array}{l} \text{YES} \\ \underline{\text{NO}} \end{array} \right\}$ (Optional)

RETURN=YES specifies that the processor issuing the executive request should be given control (reactivated) as soon as the request is queued to its destination routine. If RETURN=NO or if the parameter is omitted, the calling processor is reactivated when the request processing is complete or terminates abnormally if ERRCOMP=YES.

PURGE – Release Disc File Space

The PURGE macro releases the space allocated to a disc file. For paired file allocation, both files are purged. The format is as follows:

Name	Operation	Operand
[tag]	PURGE	[CATALOG= $\begin{cases} \text{YES} \\ \text{NO} \end{cases}$] [ERRCOMP= $\begin{cases} \text{YES} \\ \text{NO} \end{cases}$] [IDENT=symbolic address] LABDEF=symbolic address [LIST= $\begin{cases} \text{YES} \\ \text{NO} \end{cases}$] [PAIRED= $\begin{cases} \text{YES} \\ \text{NO} \end{cases}$] [RETURN= $\begin{cases} \text{YES} \\ \text{NO} \end{cases}$]

CATALOG= $\begin{cases} \text{YES} \\ \text{NO} \end{cases}$ (Optional)

CATALOG=NO specifies that the file is not centrally cataloged. If the parameter is omitted or CATALOG=YES, the file is cataloged. The parameter is ignored for scratch and temporary files.

ERRCOMP= $\begin{cases} \text{YES} \\ \text{NO} \end{cases}$ (Optional)

ERRCOMP=YES specifies that the caller of the request will examine return information on nonfatal errors; therefore, nonfatal errors should not be aborted by the system. Fatal errors such as illegal instructions, irrecoverable memory parity errors, privilege violations, and bounds errors cause an unconditional abort. If ERRCOMP=NO or if omitted, no return information is examined.

IDENT=symbolic address (Optional)

Specifies the 8-character file identifier. It must match the operand of the IDENTIFIER keyword in a Control Language DEFINE statement specifying the volumes of the file purge. IDENT may be omitted only for cataloged files. If omitted, Data Management finds the file in the central catalog and then locates the volumes.

LABDEF=symbolic address

Specifies the address of a file label, which must correspond to the label address of the DEFLB macro. This parameter is required if the IDENT parameter is not specified.

LIST= $\begin{cases} \text{YES} \\ \text{NO} \end{cases}$ (Optional)

If LIST=YES, only the parameter packet is generated, and the RETURN and ERRCOMP options are not used. If LIST=NO, only the 2-byte standard executive request instruction is generated. Loading of R6 with the address of the parameter packet and R7 with the

save area address prior to issuing the macro call is a user responsibility. If the LIST parameter is omitted, both the executive request instruction and the parameter packet are generated following the macro call. Appendix B contains a detailed discussion of the LIST parameter.

PAIRED= $\begin{cases} \text{YES} \\ \text{NO} \end{cases}$ (Optional)

PAIRED=YES indicates the file is paired. If the parameter is omitted or PAIRED=NO, the file is not paired.

RETURN= $\begin{cases} \text{YES} \\ \text{NO} \end{cases}$ (Optional)

RETURN=YES specifies that the processor issuing the executive request should be given control (reactivated) as soon as the request is queued to its destination routine. If RETURN=NO or is omitted, the calling processor is reactivated when the request processing is complete or terminates abnormally if ERRCOMP=YES.

FILE CONTROL

The file control macros — OPEN, CLOSE, and CLOVE — direct data transmission. OPEN and CLOSE control a file or unit at the block or physical I/O level. CLOVE performs volume switching at the block I/O level.

OPEN — Open File for Data Transmission

The OPEN macro makes the file or unit assessible for data transmission. The format is as follows:

Name	Operation	Operand
[tag]	OPEN	[BUFADR=symbolic address] [CONTROL= $\begin{cases} \text{ANS} \\ \text{NATIVE} \end{cases}$] [ERRCOMP= $\begin{cases} \text{YES} \\ \text{NO} \end{cases}$] IDENT=symbolic address [IOTYP= $\begin{cases} \text{P} \\ \text{B} \end{cases}$] [LABDEF=symbolic address] [LIST= $\begin{cases} \text{YES} \\ \text{NO} \end{cases}$] [RETURN= $\begin{cases} \text{YES} \\ \text{NO} \end{cases}$] [REWIND= $\begin{cases} \text{YES} \\ \text{NO} \end{cases}$] [USAGE= $\begin{cases} \text{I} \\ \text{U} \end{cases}$]

BUFADR=symbolic address (Optional)

Specifies the address of a buffer for use in initiating a file. BUFADR is required when USAGE=O for block I/O level openings of a relative file. The user generates the desired data as dummy data in this buffer prior to the OPEN request. The initialization is performed only once. BUFADR is required if IOTYP=P.

0	Unit ordinal	Length
2	Unit Table Word 1	
4	Unit Table Word 2	
6	Volume Id	
8		
10		

CONTROL = { ANS / NATIVE } (Optional)

CONTROL=NATIVE indicates that the control characters are native to that particular device. ANSI control characters are used if it is omitted or CONTROL=ANS.

ERRCOMP = { YES / NO } (Optional)

ERRCOMP=YES specifies that the caller of the request will examine return information on nonfatal errors; therefore, nonfatal errors should not be aborted by the system. Fatal errors such as illegal instructions, irrecoverable memory parity errors, privilege violations, and bounds errors cause an unconditional abort. If ERRORCOMP=NO or is omitted, no return information is examined.

IDENT=symbolic address

Specifies the 8-character file identifier. IDENT must be identical to the operand of the IDENTIFIER keyword in a Control Language DEFINE statement.

IOTYP = { P / B } (Optional)

Determines the type of I/O. IOTYP=P for physical open. IOTYP=B (block open) creates the FDT for block I/O. The default value is B. If IOTYP=P, then BUFADR must be specified.

LABDEF=symbolic address

Specifies the address of a main-storage buffer that contains a file label. The symbolic address should be

identical with the label address name specified in the DEFLB macro. This parameter is ignored for files on unit record or magnetic tape (tape labels are assigned by the control language). For disc files, label information may have been specified by Control Language DEFINE statements which override LABDEF information. For temporary or scratch files, the file name is concatenated with the job name.

LIST = { YES / NO } (Optional)

If LIST=YES, only the parameter packet is generated, and the RETURN and ERRCOMP options are not used. If LIST=NO, only the 2-byte standard executive request instruction is generated. Loading of R6 with the address of the parameter packet and R7 with the save area address prior to issuing the macro call is a user responsibility. If the LIST parameter is omitted, both the executive request instruction and the parameter packet are generated following the macro call. Appendix B contains a detailed discussion of the LIST parameter.

RETURN = { YES / NO } (Optional)

RETURN=YES specifies that the processor issuing the executive request should be given control (reactivated) as soon as the request is queued to its destination routine. If RETURN=NO or is omitted, the calling processor is reactivated when the request processing is complete or terminates abnormally if ERRCOMP=YES.

REWIND = { YES / NO } (Optional)

REWIND=NO indicates that no initial rewind of magnetic tape files is to be performed. If REWIND=YES or is omitted, initial rewind is performed.

USAGE = { I / U / O } (Optional)

Specifies input (I), update (U), or output (O) processing. Update usage is allowed for sequential files only if the record type is fixed length and the file is assigned to mass storage. The default value is I.

CLOSE – Close File for Data Transmission

The CLOSE macro removes the availability of the file for data transmission. The format is as follows:

Name	Operation	Operand
[tag]	CLOSE	[ERRCOMP={ <u>YES</u> NO }] IDENT=symbolic address [IOTYP={ <u>P</u> B }] [LIST={ <u>YES</u> NO }] [LOCK={ <u>YES</u> NO }] [RETURN={ <u>YES</u> NO }] [REWIND={ <u>YES</u> NO }]

ERRCOMP={YES
NO } (Optional)

ERRCOMP=YES specifies that the caller of the request will examine return information on nonfatal errors; therefore, nonfatal errors should not be aborted by the system. Fatal errors such as illegal instructions, irrecoverable memory parity errors, privilege violations, and bounds errors cause an unconditional abort. If ERRCOMP=NO or is omitted, no return information is examined.

IDENT=symbolic address

Specifies the 8-character file identifier. It must be identical to the one specified in the OPEN macro.

IOTYP={P
B } (Optional)

Indicates the type of I/O. IOTYP=P for physical close. IOTYP=B (block close) releases the FDT. The default value is B.

LIST={YES
NO } (Optional)

If LIST=YES, only the parameter packet is generated, and the RETURN and ERRCOMP options are not used. If LIST=NO, only the 2-byte standard executive request instruction is generated. Loading of R6 with the address of the parameter packet and R7 with the save area address prior to issuing the macro call is a user responsibility. If the LIST parameter is omitted, both the executive request instruction and the parameter packet are generated following the macro call. Appendix B contains a detailed discussion of the LIST parameter.

LOCK={YES
NO } (Optional)

Specifies the disposition of the file. When

LOCK=YES is used, the FDT is flagged and OPEN may not be executed again during the same job; files assigned to magnetic tape are unloaded. When this parameter is not used or LOCK=NO, the file may be reopened from within the same job; files assigned to magnetic tape may be rewound.

RETURN={YES
NO } (Optional)

RETURN=YES specifies that the processor issuing the executive request should be given control (reactivated) as soon as the request is queued to its destination routine. If RETURN=NO or is omitted, the calling processor is reactivated when the request processing is complete or terminates abnormally if ERRCOMP=YES.

REWIND={YES
NO } (Optional)

REWIND=NO specifies that magnetic tape files are not rewound after closing. If the parameter is omitted or REWIND=YES, the tapes are rewound/unloaded.

CLOVE – Close Volume

The CLOVE macro performs volume switching at the block I/O level. It is used for sequential multivolume files assigned to tape or disc. A user program may close a volume at any time and switch to the next sequential volume. CLOVE must be used when a switch to the next volume is indicated by either an EOF record or an end of space indicator from block I/O.

CLOVE performs header and trailer label processing on tapes, alternate unit processing on tapes, and disc pack mounting and dismounting (via operator control). The format is as follows:

Name	Operation	Operand
[tag]	CLOVE	[ERRCOMP={ <u>YES</u> NO }] IDENT=symbolic address [LIST={ <u>YES</u> NO }] [RETURN={ <u>YES</u> NO }]

ERRCOMP={YES
NO } (Optional)

ERRCOMP=YES specifies that the caller of the request will examine return information on nonfatal errors; therefore, nonfatal errors should not be aborted by the system. Fatal errors such as illegal instructions, irrecoverable memory parity errors, privilege violations, and bounds errors cause an unconditional abort. If ERRCOMP=NO or is omitted, no return information is examined.

IDENT=symbolic address

Specifies the 8-character file identifier.

LIST= $\begin{cases} \text{YES} \\ \underline{\text{NO}} \end{cases}$ (Optional)

If LIST=YES, only the parameter packet is generated, and the RETURN and ERRCOMP options are not used. If LIST=NO, only the 2-byte standard executive request instruction is generated. Loading of R6 with the address of the parameter packet and R7 with the save area address prior to issuing the macro call is a user responsibility. If the LIST parameter is omitted, both the executive request instruction and the parameter packet are generated following the macro call. Appendix B contains a detailed discussion of the LIST parameter.

RETURN= $\begin{cases} \text{YES} \\ \underline{\text{NO}} \end{cases}$ (Optional)

RETURN=YES specifies that the processor issuing the executive request should be given control (reactivated) as soon as the request is queued to its destination routine. If RETURN=NO or if the parameter is omitted, the calling processor is reactivated when the request processing is complete or terminates abnormally if ERRCOMP=YES.

I/O SERVICE MACRO

Data Management also has available I/O service macros which can report the status information of various aspects of the file.

LABRTN – Return File Label Information

The LABRTN macro returns the disc catalog elements for uncataloged and cataloged files. The file may be open or closed. LABRTN searches the FDT string in partition first; and if there is no FDT, LABRTN then searches the table created by the Control Language DEFINE statement. If the IDENT is found, then the proper disc file information is returned in the buffer specified by the user in INFOADR. If a LABRTN macro is issued against a nondisc device and the IDENT specification matches a Control Language //DEFINE statement for a nondisc device, LABRTN returns an error code and returns the device type in the first byte of the user buffer. The format is as follows:

<u>Name</u>	<u>Operation</u>	<u>Operand</u>
[tag]	LABRTN	[ELEMENT= $\begin{cases} \text{NAME} \\ \text{ATTRIBUTE} \end{cases}$] [ERRCOMP= $\begin{cases} \text{YES} \\ \underline{\text{NO}} \end{cases}$] IDENT=symbolic address INFOADR=symbolic address [LIST= $\begin{cases} \text{YES} \\ \underline{\text{NO}} \end{cases}$] [RETURN= $\begin{cases} \text{YES} \\ \underline{\text{NO}} \end{cases}$]

ELEMENT= $\begin{cases} \text{NAME} \\ \text{ATTRIBUTE} \\ \text{ALL} \end{cases}$ (Optional)

Specifies that the name element, attribute element, or both elements will be returned in the buffer specified by INFOADR. If the parameter is omitted, the NAME element is returned. If ELEMENT=ALL is specified, the name and attribute elements are returned respectively. For an indexed file, the data portion of the file is returned first and then the index portion of the file.

ERRCOMP= $\begin{cases} \text{YES} \\ \underline{\text{NO}} \end{cases}$ (Optional)

ERRCOMP=YES specifies that the caller of the request will examine return information on nonfatal errors; therefore, nonfatal errors should not be aborted by the system. Fatal errors such as illegal instructions, irrecoverable memory parity errors, privilege violations, and bounds errors cause an unconditional abort. If ERRCOMP=NO or if omitted, no return information is examined.

IDENT=symbolic address

Specifies the 8-character file identifier which was used to open the file.

INFOADR=symbolic address

Specifies the address of a main-storage buffer where the information is returned.

LIST= $\begin{cases} \text{YES} \\ \underline{\text{NO}} \end{cases}$ (Optional)

If LIST=YES, only the parameter packet is generated, and the RETURN and ERRCOMP options are not used. If LIST=NO, only the 2-byte standard executive request instruction is generated. Loading of R6 with the address of the parameter packet and R7 with the save area address prior to issuing the macro call is a user responsibility. If the LIST parameter is omitted, both the executive request instruction and the parameter packet are generated following the macro call. Appendix B contains a detailed discussion of the LIST parameter.

RETURN= $\left\{ \begin{array}{l} \text{YES} \\ \underline{\text{NO}} \end{array} \right\}$ Optional

RETURN=YES specifies that the processor issuing the executive request should be given control (reactivated) as soon as the request is queued to its destination routine. If RETURN=NO or if the parameter is omitted, the calling processor is reactivated when the request processing is complete or terminates abnormally if ERRCOMP=YES.

BLOCK INPUT/OUTPUT MACROS

READ

This macro transfers a block of data into the specified buffer. If the location of the block is not explicitly stated in the request, it will be generated by adding 1 to the block number last obtained (this applies to disc and tape files only). If the preceding reference to this file involved a WRITE or POSITN request, the current block number is used without updating. When the BLKNUM keyword is not used, the file is addressed in a sequential manner.

Standard error recovery is provided, with irrecoverable errors terminating the operation and posting the return code in the request block. Attempts to read outside of the portion of the file which is currently accessible (current volume) will result in an error.

Name	Operation	Operand
[tag]	READ	IDENT=symbolic address DATBUF=symbolic address [DATSIZ=symbolic address] [LIST= $\left\{ \begin{array}{l} \text{YES} \\ \underline{\text{NO}} \end{array} \right\}$] [RETURN= $\left\{ \begin{array}{l} \text{YES} \\ \underline{\text{NO}} \end{array} \right\}$] [ERRCOMP= $\left\{ \begin{array}{l} \text{YES} \\ \underline{\text{NO}} \end{array} \right\}$] [EBCDIC= $\left\{ \begin{array}{l} \text{YES} \\ \underline{\text{NO}} \end{array} \right\}$] [OPER=SSn] [BLKNUM=symbolic address] [MULTBLK= $\left\{ \begin{array}{l} \text{YES} \\ \underline{\text{NO}} \end{array} \right\}$]

IDENT=symbolic address

Specifies the address of a storage location containing the file identifier — the name by which the file is known to the program (as opposed to the external catalog name specified through Control Language). The file identifier must be eight bytes in length.

DATBUF=symbolic address

The address of the data buffer. Requests to disc

storage must specify a buffer which begins and ends on word boundaries even though the data count may be odd.

DATSIZ=symbolic address Optional

An address pointer to the number of bytes in the buffer. If this is omitted, the block size defined for the file is used as the byte count. The buffer size may be up to 65,535 bytes. Any DATSIZ value given which is greater than the device record size will result in a nonzero residual count returned at the end of the operation.

LIST= $\left\{ \begin{array}{l} \text{YES} \\ \underline{\text{NO}} \end{array} \right\}$ Optional

To be used when either the system linkage (service request instruction) or the parameter table is to be generated separately. LIST=NO generates the service request only; LIST=YES generates the parameter table (list) only. When no LIST operand is specified, both the service request and the parameter table are generated.

RETURN= $\left\{ \begin{array}{l} \text{YES} \\ \underline{\text{NO}} \end{array} \right\}$ Optional

To be used when the user program wishes to be given control immediately on recognition of the request, but before completion (RETURN=YES). When RETURN=NO is coded, or the default is taken, control will only be returned on completion of the request.

ERRCOMP= $\left\{ \begin{array}{l} \text{YES} \\ \underline{\text{NO}} \end{array} \right\}$ Optional

To be used when the user decides to retain control in the event of an error (ERRCOMP=YES). When ERRCOMP=NO is coded, or the default is taken, the program will be aborted if an error occurs.

The following parameters apply to the card reader only.

EBCDIC= $\left\{ \begin{array}{l} \text{YES} \\ \underline{\text{NO}} \end{array} \right\}$ Optional

This determines the type of conversion to be performed on the data between card format and memory format. If EBCDIC=YES, the translation is made in accordance with the defined EBCDIC formats. If EBCDIC=NO, the data on the card is accepted in a binary format and is transferred to storage without modification (card image storage format, shown in Figure 7-1). If the parameter is omitted the EBCDIC=YES option is assumed.

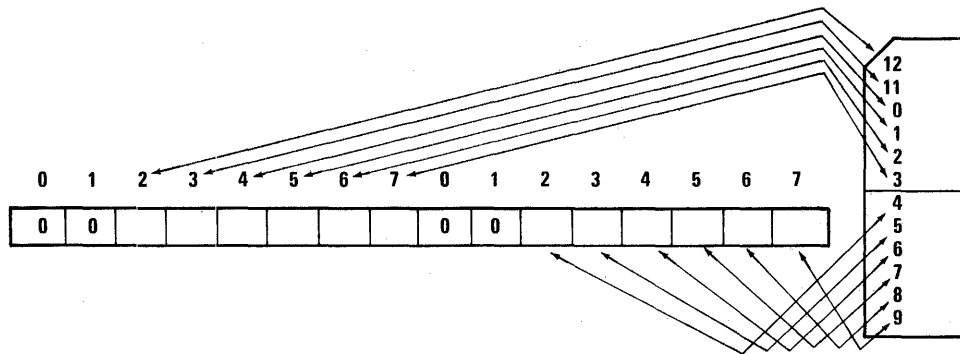


Figure 7-1. Card Image/Storage Relationship

OPER=SSn Optional

Selects a stacker for the card read, designated by n. The value n is determined by the hardware device being used. This parameter is ignored except for card readers with multiple stackers. If it is omitted, the normal stacker is used.

The following parameter applies to magnetic tape and disc files only.

BLKNUM=address Optional

The address of a four-byte hexadecimal field containing the block number to be read. If it is omitted, the current block number of the file is updated by 1 before the READ, unless the last operation was a WRITE* or POSITN, in which case the current block number is used. Block numbers up to $2^{32}-1$ are allowed, but attempts to read beyond the limits of a file (as defined by allocation) are returned with return information noted.

Block numbers up to $2^{32}-1$ are allowed, but attempts to read beyond the limits of a file (as defined by allocation) are returned with return information noted.

MULTBLK= $\left\{ \begin{array}{l} \text{YES} \\ \underline{\text{NO}} \end{array} \right\}$ (Optional)

If this operand is specified, as many disc records as the program buffer will hold will be loaded. Any remaining buffer space will be noted in the residual count field.

*READ after WRITE on magnetic tape is illegal.

WRITE

This macro transfers a block of data from the specified buffer to an I/O device. The location on the I/O device where the data is to be placed may be controlled by the BLKNUM keyword (disc), or the current setting of the file block number will be used. In either case, the file block number is updated after the WRITE operation by adding 1 to the current block number.

When writing to the line printer or card punch, the first character of the user's buffer is used for carriage or stacker control, and is deleted from the data line. This feature may be overridden by the OPER option. The first character of the buffer is still deleted from the line.

Errors detected by the I/O routines initiate error recovery procedures. If these fail to correct the error, the return code is posted in the request block and the operation is terminated.

Name	Operation	Operand
[tag]	WRITE	IDENT=symbolic address DATBUF=symbolic address [DATSIZ=symbolic address] [LIST= $\left\{ \begin{array}{l} \text{YES} \\ \text{NO} \end{array} \right\}$] [RETURN= $\left\{ \begin{array}{l} \text{YES} \\ \text{NO} \end{array} \right\}$] [ERRCOMP= $\left\{ \begin{array}{l} \text{YES} \\ \text{NO} \end{array} \right\}$] [EBCDIC= $\left\{ \begin{array}{l} \text{YES} \\ \text{NO} \end{array} \right\}$] [BLKNUM=symbolic address] [OPER= $\left\{ \begin{array}{l} \text{SKnn} \\ \text{SPnn} \\ \text{SSn} \end{array} \right\}$]

IDENT=symbolic address

Specifies the address of a storage location containing the file identifier — the name by which the file is known to the program (as opposed to the external catalog name specified through Control Language). The file identifier must be eight bytes in length.

DATBUF=symbolic address

Specifies the address of the data buffer. Requests to disc must specify a buffer which begins a word boundary.

DATSIZ=symbolic address Optional

The address of the memory word which contains the byte count of the buffer. If it is omitted, the block size defined for the file is used as the byte count. The buffer size may be up to 65,535 bytes.

LIST= $\left\{ \begin{array}{l} \text{YES} \\ \text{NO} \end{array} \right\}$ Optional

To be used when either the system linkage (service request instruction) or the parameter table is to be generated separately. LIST=NO generates the service request only; LIST=YES generates the parameter table (list) only. When no LIST operand is specified, both the service request and the parameter table are generated.

RETURN= $\left\{ \begin{array}{l} \text{YES} \\ \text{NO} \end{array} \right\}$ Optional

To be used when the user program wishes to be given control immediately on recognition of the request, but before completion (RETURN=YES). When RETURN=NO is added, or the default is taken, control will only be returned on completion of the request.

ERRCOMP= $\left\{ \begin{array}{l} \text{YES} \\ \text{NO} \end{array} \right\}$ Optional

To be used when the user decides to retain control in the event of an error (ERRCOMP=YES). When ERRCOMP=NO is coded, or the default is taken, the program will be aborted if an error occurs.

Magnetic Tape and Disc

BLKNUM=symbolic address Optional

The address of a four-byte hexadecimal field containing the block number to be read. This parameter is valid only for direct-access storage devices. If it is omitted, the current block number of the file is updated by 1 after the WRITE operation.

Block numbers through 2³²-1 are allowed, but attempts to write beyond the limits of a file (as defined by allocation) are returned with an error indication.

Line Printer

OPER=SKnn Optional

Defines a carriage control tape channel to be used when the data is written by a line printer. If the file is not a printer file, the value of *nn* is converted to a comparable ASA standard control character which overlays the first character of the written block. If the file is a printer file, this parameter overrides the carriage operation specified by the first character of the data record. Regardless of the method used, the first character in the data buffer is deleted from the print line.

OPER=SPnn Optional

Defines the number of print lines to be spaced after printing. It is treated in a manner like that of the OPER=SKnn parameter. From 1 to 15 lines may be specified depending on hardware capabilities.

Card Punch

OPER=SSn Optional

Defines a stacker select operation to a card punch. If this parameter is used, the first character in the data buffer is written in the first position of the card. If the parameter is not used, the first character in the data buffer is treated as a stacker select code and is deleted from the punch data. The absence of multiple stackers on the card punch has no effect on this procedure.

EBCDIC= $\left\{ \begin{array}{l} \text{YES} \\ \text{NO} \end{array} \right\}$ Optional

This determines the type of conversion to be performed on the data between card format and memory format. With EBCDIC=YES, the translation is made in accordance with the defined EBCDIC formats. With EBCDIC=NO, the data on the card is treated as being in a binary format (card image) and is transferred from storage without modification (Figure 7-1). If the parameter is omitted the EBCDIC=YES option is assumed.

POSITN — Change Current Block Number

The POSITN request is used to change the current block number for the file. The subsequent READ or WRITE macro begins processing at the new block number. POSITN is only allowed for files assigned to disc storage or magnetic tape.

A POSITN on a magnetic tape file will always cause the tape to be physically repositioned, whereas with disc files a physical seek operation will not be performed unless explicitly requested by SEEK=YES.

Name	Operation	Operand
[tag]	POSITN	IDENT=symbolic address [BLKNUM= $\left\{ \begin{array}{l} \text{BOV} \\ \text{EOV} \\ \text{symbolic address} \end{array} \right\}$] [LIST= $\left\{ \begin{array}{l} \text{YES} \\ \text{NO} \end{array} \right\}$] [RETURN= $\left\{ \begin{array}{l} \text{YES} \\ \text{NO} \end{array} \right\}$] [ERRCOMP= $\left\{ \begin{array}{l} \text{YES} \\ \text{NO} \end{array} \right\}$] [SEEK= $\left\{ \begin{array}{l} \text{YES} \\ \text{NO} \end{array} \right\}$]

IDENT=symbolic address

Specifies the address of a storage location containing the file identifier — the name by which the file is known to the program (as opposed to the external catalog name specified through Control Language). The file identifier must be eight bytes in length.

BLKNUM= $\left\{ \begin{array}{l} \text{BOV} \\ \text{EOV} \\ \text{symbolic address} \end{array} \right\}$ Optional

Specifies the block number at which the file is to be positioned. BOV indicates beginning of volume, EOV indicates end of volume*. If a symbolic address is coded this is taken as the address of a four-byte field containing the block number as an unsigned hexadecimal number.

If this parameter is omitted, a position to BOV is performed. Standard error recovery is provided, with irrecoverable errors terminating the operation and posting the exception indicator and return code in the request block. A POSITN to block number 1 is a request to position at the beginning of the file. A POSITN to block number 0 will return an error. A position to a block number above the highest block currently mounted will also be returned with an error; however, the file will be left positioned on the last block of the volume or file.

LIST= $\left\{ \begin{array}{l} \text{YES} \\ \text{NO} \end{array} \right\}$ Optional

To be used when either the system linkage (service request instruction) or the parameter table is to be generated separately. LIST=NO generates the service

* For magnetic tape EOV indicates a position forward to the first tapemark. For disc, BOV and EOV result in positioning, across physical pack boundaries if necessary, to the lowest (BOV) or highest (EOV) block number which is currently described in the File Description Table.

request only; LIST=YES generates the parameter table (list) only. When no LIST operand is specified, both the service request and the parameter table are generated.

RETURN= $\left\{ \begin{array}{l} \text{YES} \\ \text{NO} \end{array} \right\}$ Optional

To be used when the user program wishes to be given control immediately on recognition of the request, but before completion (RETURN=YES). When RETURN=NO is added, or the default is taken, control will only be returned on completion of the request.

ERRCOMP= $\left\{ \begin{array}{l} \text{YES} \\ \text{NO} \end{array} \right\}$ Optional

To be used when the user devices to retain control in the event of an error (ERRCOMP=YES). When ERRCOMP=NO is coded, or the default is taken, the program will be aborted if an error occurs.

The following description applies only to disc storage.

SEEK= $\left\{ \begin{array}{l} \text{YES} \\ \text{NO} \end{array} \right\}$ Optional

This parameter causes the disc drive arm to be physically positioned on the cylinder in which the specified block number is to be found. If the parameter is omitted, the seek operation will be performed when the following READ or WRITE macro is executed and the POSITN will merely update the block number.

SEEK=YES should only be included when operating on an unshared file; this is because the position of the disc arm is not locked after a POSITN macro has been executed, and with a shared file the arm could therefore be repositioned by another user before data transfer had begun. If the SEEK on POSITN capability was not included in the operating system (SYSGEN), this operand is ignored.

CNTRL — Hardware Control Operation

The CNTRL request is used to perform a specific hardware control operation (this does not apply to spooled files going to disc storage). The operations allowed are dependent upon the device type. Standard error recovery is provided, with irrecoverable and logical errors terminating the operation and posting the return code in the request block. Control of the internal block number is maintained where possible, but if an operation is performed which does not maintain the block number, any subsequent request which needs a block number will be terminated with an error.

Name	Operation	Operand
[tag]	CNTRL	IDENT=symbolic address [LIST={YES NO}] [RETURN={YES NO}] [ERRCOMP={YES NO}] OPER=operation code

IDENT=symbolic address

Specifies the address of a storage location containing the file identifier — the name by which the file is known to the program (as opposed to the external catalog name specified through Control Language). The file identifier must be eight bytes in length.

LIST={YES
NO} Optional

To be used when either the system linkage (service request instruction) or the parameter table is to be generated separately. LIST=NO generates the service request only; LIST=YES generates the parameter table (list) only. When no LIST operand is specified, both the service request and the parameter table are generated.

RETURN={YES
NO} Optional

To be used when the user program wishes to be given control immediately on recognition of the request, but before completion (RETURN=YES). When RETURN=NO is added, or the default is taken, control will only be returned on completion of the request.

ERRCOMP={YES
NO} Optional

To be used when the user decides to retain control in the event of an error (ERRCOMP=YES). When ERRCOMP=NO is coded, or the default is taken, the program will be aborted if an error occurs.

OPER=operation code

- Reader/Punch

OPER=SSn

Select stacker on a card reader/punch. The numeric value of *n* is the stacker number. This command may only follow a READ to a file opened for modify usage.

- Line Printer

OPER=SPnn

Space the printer form *nn* lines immediately. From 1 to 15 lines may be specified, depending on the hardware capability.

OPER=SKnn

Skip to carriage control tape channel *nn* immediately.

The permissible values of *nn* are 1 through 12, corresponding to channels 1 through 12 of the printer carriage control.

- Magnetic Tape

OPER=REW

Rewind tape. The internal block number will not be maintained.

OPER=RUN

Rewind and unload tape. The block number will not be maintained.

OPER=ERG

Erase gap (write blank tape). The block number will be maintained.

OPER=BSR

Backspace to interrecord gap. The block number will be maintained.

OPER=FSR

Forward space to interrecord gap. The block number will be maintained.

OPER=FSF

Forward space to tapemark (EOF). The block number will not be maintained.

OPER=BSF

Backspace to tapemark (EOF) or to Load Point if no tapemark is present. The block number will not be maintained.

OPER=EOF

Write End-of-File mark (tapemark). The block number will be maintained.

- Disc Files

OPER=EOF*

Write End-of-File mark (a record containing a count field specifying a data length of zero). When read, this record

*For disc, CNTRL will always write an EOF mark regardless of whether the OPER=EOF operand is included or not.

will cause the system to set the EOF flag in the request block and in the File Description Table. The block number will be maintained.

STATUS — Report of Status

This request causes the system to pass information to a specified data buffer regarding the status of the file at the completion of the last operation to the file.

<u>Name</u>	<u>Operation</u>	<u>Operand</u>
[tag]	STATUS	IDENT=symbolic address BUFADR=symbolic address BUFSIZ=symbolic address [LIST={YES NO }] [RETURN={YES NO }] [ERRCOMP={YES NO }]

IDENT=symbolic address

Specifies the address of a storage location containing the file identifier — the name by which the file is known to the program (as opposed to the external catalog name specified through Control Language). The file identifier must be eight bytes in length.

BUFADR=symbolic address

Specifies the address of the data buffer.

BUFSIZ=symbolic address

Specifies the address of the memory word containing the byte count of the buffer.

LIST={YES
 NO } Optional

To be used when either the system linkage (service request instruction) or the parameter table is to be generated separately. LIST=NO generates the service request only; LIST=YES generates the parameter table (list) only. When no LIST operand is specified, both the service request and the parameter table are generated.

RETURN={YES
 NO } Optional

To be used when the user program wishes to be given control immediately on recognition of the request but before completion (RETURN=YES). When RETURN=NO is coded, or the default is taken, control will only be returned on completion of the request.

ERRCOMP={YES
 NO } Optional

To be used when the user decides to retain control in the event of an error (ERRCOMP=YES). When ERRCOMP=NO is coded, or the default is taken, the program will be aborted if an error occurs.

Between 20 and 24 bytes of file status information are available from the system, depending on the type of device (Table 7-1). The BUFSIZ specification determines how much of this information is to be returned, starting from the top of the table (this could be less than the total amount available, if specified).

Table 7-2 gives the information for the status word of the Basic Data Channel operation; Table 7-3 gives the status word for Disc Channel operation.

TYPE — Device and File Type

This request causes the system to pass two bytes of information to the data buffer regarding the type of file and the type of device for which this file is prepared.

<u>Name</u>	<u>Operation</u>	<u>Operand</u>
[tag]	TYPE	IDENT=symbolic address BUFADR=symbolic address [LIST={YES NO }] [RETURN={YES NO }] [ERRCOMP={YES NO }]

IDENT=symbolic address

Specifies the address of a storage location containing the file identifier — the name by which the file is known to the program (as opposed to the external catalog name specified through Control Language). The file identifier must be eight bytes in length.

BUFADR=symbolic address

Specifies the address of the data buffer.

Table 7-1. Returned Information Format

	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7
0		C	US	B	UD	L	SF	Last BIO Function								
2	Current Block Number															
4																
6	Error Field															
8	EP		FR	H												
10	Block Size															
12	Pointer to Current Command Word															
14	Status of Last I/O Operation															
16	Residual Count															

Disc:

18		PF	PA	WC	DF		No. of Residual Blocks									
20																
22	Highest Block Written (per volume)															

Magnetic Tape:

18																
20	Sense															
22																

Unit Record:

18	Sense								Unit Ordinal							
----	-------	--	--	--	--	--	--	--	--------------	--	--	--	--	--	--	--

Table 7-1. Returned Information Format (Continued)

Byte	Bit(s)	Field Name	Definition
0	1	Common stored data format flag (C)	0 = not common stored data format 1 = common stored data format
	2,3	Usage Flag (US)	00 = input 01 = update 10 = output
	4	Bypass Flag (B)	0 = no bypass 1 = bypass: READ goes to EOF WRITE is a NOP
	5	Update Flag (US)	0 = not update mode 1 = update mode
	6	Lockout Flag (L)	0 = no lockout 1 = file has been closed with lock
	7	Sequential File Flag (SF)	0 = not sequential 1 = sequential
1	0-7	Last BIO Function	Last function processed in BIO
2-5	0-7 0-7 0-7 0-7	Current Block Number (four bytes)	Block number after last function processed. If zero, current block number is unknown
6,7	0-7	Error Field	When this field is non-zero, a RESET macro must be issued before making another request to the file
8	0	System Error Processing (EP)	0 = Call Error Recovery when an error is encountered 1 = Bypass Error Recovery
	1,2	Not Used	
	3	FDT Restore (FR)	0 = FDT not restored 1 = FDT restored
	4	Hold Up Flag (H)	Set when file is in error recovery to prevent further requests from being serviced
	5-7	Not Used	
9	0-7	Not Used	
10,11	0-7	Block Size	Number of bytes in a physical record

Table 7-1. Returned Information Format (Continued)

Byte	Bit(S)	Field Name	Definition
12,13	0-7	Current CW Pointer	Address of the current or last command word executed on this file
14,15	0-7	Status	Status of last I/O operation (see Tables 7-2 and 7-3)
16,17	0-7	Residual Count (RC)	The difference between the number of bytes requested and the number of bytes transferred

Disc

18	0	Not Used	
	1	Paired File Flag (PF)	0 = not paired 1 = paired
	2	Paired File Indicator (PA)	0 = first track of paired tracks 1 = second track of paired tracks
	3	Write Check (WC)	0 = no write check 1 = write check on all writes
	4	Disc Driver Flag	Used internally by the driver only
	5-7	Not Used	
19	0-7	Residual Blocks	Number of blocks remaining to be set up for a multiblock read request which crosses tracks
20-23	0-7 0-7 0-7 0-7	Highest Block Written (HBW)	Highest block number written for volume now mounted

Magnetic Tape

18-23		Sense	Sense bytes of the device at time of last error
-------	--	-------	---

Unit Record

18	0-7	Sense	Sense byte of the device at time of last error
19	0-7	Unit Ordinal (UORD)	Unit table ordinal

Table 7-2. Status Word for Basic Data Channel Operations

Byte	Bit Set to 1	Meaning	
0	0	Attention	
	1	Status Modifier	
	2	Control Unit End	
	3	Busy	
	4	Channel End	
	5	Device End	
	6	Unit Check	
	7	Unit Exception	
1	0	Initial selection sequence error, caused by one of the following: Unit not there Parity error on bus out Bad address Unit off-line	
		1	Main storage buffer not exhausted
		2	Wrong Address-In returned on initial selection or bad parity on 'ADDRESS IN'
		3	No Request In on SIO Poll Sequence Request
	4	Control check	
	5	Examine check (for data transfer only)	
	6	Invalid command or zero byte count	
	7	Unused	

Table 7-3. Status Word for Disc Channel Operations

Byte	Bit Set to 1	Meaning
0	0	IFA status not valid or command early
	1	IFA missed window or command early
	2	IFA window
	3	IFA track boundary
	4	IFA read/write termination
	5	IFA burst check error
	6	IFA lost data
	7	IFA no sync compare
1	0	IFA 3rd rev sync find
	1	Disc (not on line) or (seek incomplete and not file unsafe)
	2	Disc (file unsafe) or (seek incomplete and not file unsafe)
	3	Disc read only
	4	Disc pack change
	5	Disc end of cylinder
	6	Disc write current sense or search fail (may be EOF)
	7	Disc busy
	All bits set	Invalid function code in command program

LIST= $\left\{ \begin{array}{l} \text{YES} \\ \text{NO} \end{array} \right\}$ Optional

To be used when either the system linkage (service request instruction) or the parameter table is to be generated separately. LIST=NO generates the service request only; LIST=YES generates the parameter table (list) only. When no LIST operand is specified, both the service request and the parameter table are generated.

RETURN= $\left\{ \begin{array}{l} \text{YES} \\ \text{NO} \end{array} \right\}$ Optional

To be used when the user program wishes to be given control immediately on recognition of the request but before completion (RETURN=YES). When RETURN=NO is coded, or the default is taken, control will only be returned on completion of the request.

ERRCOMP= $\left\{ \begin{array}{l} \text{YES} \\ \text{NO} \end{array} \right\}$ Optional

To be used when the user decides to retain control in the event of an error (ERRCOMP=YES). When ERRCOMP=NO is coded, or the default is taken, the program will be aborted if an error occurs.

The system responds to a TYPE request by placing two bytes of file information in the buffer in the following format:

0	1	2	3	4	5	6	7	8	15
C	US	B	UD	L	SF	Device Type			

The first byte of this word is the same as the first byte returned by the STATUS macro. The second byte is a value representing the device type as defined following:

- 0F-2F input/output devices
- 20-3F input only devices
- 40-5F output only devices
- 60-7F input/output devices
- 80-FF communications devices

A more specific description of the bit significance in the device type code is illustrated in Table 7-4.

RESET – Reset Exception Conditions

The RESET request allows a user to reset exception conditions in the file description table. The exception conditions lock out new I/O operations which would overlay the exception status indications. Examples of exception conditions are EOF and irrecoverable hardware conditions.

Name	Operation	Operand
[tag]	RESET	IDENT=symbolic address [LIST= $\left\{ \begin{array}{l} \text{YES} \\ \text{NO} \end{array} \right\}$] [RETURN= $\left\{ \begin{array}{l} \text{YES} \\ \text{NO} \end{array} \right\}$] [ERRCOMP= $\left\{ \begin{array}{l} \text{YES} \\ \text{NO} \end{array} \right\}$]

IDENT=symbolic address

Specifies the address of a storage location containing the file identifier — the name by which the file is known to the program (as opposed to the external catalog name specified through Control Language). The file identifier must be eight bytes in length.

LIST= $\left\{ \begin{array}{l} \text{YES} \\ \text{NO} \end{array} \right\}$ Optional

To be used when either the system linkage (service request instruction) or the parameter table is to be generated separately. LIST=NO generates the service request only; LIST=YES generates the parameter table (list) only. When no LIST operand is specified, both the service request and the parameter table are generated.

RETURN= $\left\{ \begin{array}{l} \text{YES} \\ \text{NO} \end{array} \right\}$ Optional

To be used when the user program wishes to be given control immediately on recognition of the request, but before completion (RETURN=YES). When RETURN=NO is coded, or the default is taken, control will only be returned on completion of the request.

ERRCOMP= $\left\{ \begin{array}{l} \text{YES} \\ \text{NO} \end{array} \right\}$ Optional

To be used when the user decides to retain control in the event of an error (ERRCOMP=YES). When ERRCOMP=NO is coded, or the default is taken, the program will be aborted if an error occurs.

Table 7-4. Bit Significance

8	9	10	11	12	13	14	15	Bit assignment
0	0	0	1	0	x	b	a	Card Reader/Punches
0	0	1	0	0	x	b	a	Card Readers
0	1	0	0	0	x	b	a	Card Punches
0	1	0	1	x	x	b	a	Line Printers
0	1	1	0	x	x	b	a	Magnetic Tape Units
0	1	1	1	0	x	x	x	Disc Drives
1	0	0	0	0	x	x	x	Communications Devices

x = modifier bit

a = 1 means hardware-buffered device

b = 1 means early termination (on channel end rather than device end)

PHYSICAL INPUT/OUTPUT MACROS

EXCP – Input/Output Action

Name	Operation	Operand
[tag]	EXCP	$[PCB = \left\{ \begin{array}{l} @register\ number \\ symbolic\ address \end{array} \right\}]$ $[CP = \left\{ \begin{array}{l} @register\ number \\ symbolic\ address \end{array} \right\}]$ $[UNORD = symbolic\ address]$ $[RETURN = \left\{ \begin{array}{l} YES \\ NO \end{array} \right\}]$ $[ERRCOMP = \left\{ \begin{array}{l} YES \\ NO \end{array} \right\}]$

This is the physical I/O action macro (the "do I/O" macro). Basically, it generates the service request necessary to execute a command program in conjunction with a Physical Control Block (PCB) and for this purpose does not necessarily require any operands. However, it should be noted that two preconditions must be satisfied before the EXCP can function properly:

- general register 6 must be loaded with a pointer to the associated PCB
- the appropriate command program address must be loaded into the PCB

If these two conditions are not already satisfied at the time the macro is issued, the following optional operands may be included in the macro for this purpose.

$PCB = \left\{ \begin{array}{l} @register\ number \\ symbolic\ address \end{array} \right\}$ Optional

This operand specifies the address of the PCB. If the parameter is omitted, the PCB location is assumed to be R6. An alternate register or a symbolic address may be designated to give the location of the PCB. The use of this keyword causes the contents of R6 to be destroyed.

$CP = \left\{ \begin{array}{l} @register\ number \\ symbolic\ address \end{array} \right\}$ Optional

This operand specifies the address of the appropriate command program. If this operand is omitted, the command program address is assumed to be in the PCB. If the keyword is used, a register (other than R6) or a symbolic address may specify the location of the command program to be executed. If the keyword is specified, the address of the command program is moved to the CPADR area in the PCB.

$UNORD = symbolic\ address$ Optional

The operand must point to a user location containing the unit ordinal of the device on which the operation is to be performed. This will normally be the same address as the buffer specified in the physical OPEN macro associated with the device, since OPEN returns this value.

RETURN = $\left\{ \begin{array}{l} \text{YES} \\ \underline{\text{NO}} \end{array} \right\}$ Optional

To be used when the user program wishes to be given control immediately on recognition of the request, but before completion (RETURN=YES). When RETURN=NO is coded, or the default is taken, control will only be returned on completion of the request.

ERRCOMP = $\left\{ \begin{array}{l} \text{YES} \\ \underline{\text{NO}} \end{array} \right\}$ Optional

To be used when the user decides to retain control in the event of an error (ERRCOMP=YES). When ERRCOMP=NO is coded, or the default is taken, the program will be aborted if an error occurs.

PHYSICAL CONTROL BLOCK

This is a data macro which generates the Physical Command Block necessary for performing a physical input/output operation.

Name	Operation	Operand
[tag]	PCB	[CPADR=symbolic address] [ERROPT = $\left\{ \begin{array}{l} \text{YES} \\ \underline{\text{NO}} \end{array} \right\}]$ [DEVTYP = $\left\{ \begin{array}{l} \text{TAPE} \\ \underline{\text{UREC}} \\ \text{DISC} \end{array} \right\}]$ [FUNCTN = $\left\{ \begin{array}{l} \text{ASKATT} \\ \underline{\text{REMOVE}} \end{array} \right\}]$

ERROPT = $\left\{ \begin{array}{l} \text{YES} \\ \underline{\text{NO}} \end{array} \right\}$ Optional

Determines whether system error recovery is to be employed in the event that an error occurs when the command program is executed (by an EXCP macro).

DEVTYP = $\left\{ \begin{array}{l} \underline{\text{MT}} \\ \text{UR} \\ \text{DISC} \\ \text{COMM} \end{array} \right\}$ Optional

Determines how much space is to be reserved in the generated PCB to receive sense information.

- MT reserves three words
- UR reserves one word
- DISC and COMM reserve no space
- Omitting the DEVTYP operand causes reservation of three words

In the event that an uncorrectable error occurs, sense information will always be returned to the PCB and

the user may then use this information for his own error processing provided that ERRCOMP=YES was coded in the corresponding EXCP macro.

CPADR=symbolic address Optional

This parameter specifies the location of the command program to be used with this PCB. The symbolic address should be the label used on the command program. If this parameter is omitted, the PCB will be assembled without a command program address; then this address must therefore be supplied dynamically at execution time by the corresponding EXCP macro.

FUNCTN = $\left\{ \begin{array}{l} \text{ASKATT} \\ \underline{\text{REMOVE}} \end{array} \right\}$ Optional

This optional parameter is used for executing functions which do not require a command program. When this parameter is coded the CPADR parameter is not required and will be ignored if present. FUNCTN=ASKATT causes the program to wait for an asynchronous attention from the device (such as disc pack change, line printer ready, magnetic tape loaded). If no attention is received following a FUNCTN=ASKATT request, the outstanding request must be cleared before the end of the job by issuing a FUNCTN=REMOVE request. FUNCTN=REMOVE clears all ASKATT requests from the particular device and returns them with an explanatory error code (see Appendix C).

NOTE

Since the completion of a FUNCTN=ASKATT request is dependent on outside action (operator, hardware) care should be exercised in its use (it is possible to give up control and never regain it).

REMOVE and ASKATT functions cannot be chained.

COMMAND

This single macro will generate a command word (or words)* for a physical input/output operation on unit record, magnetic tape, disc or communications** devices. Since the macro takes a wide variety of possible forms according to the type of device and operation for which it is being used, separate descriptions are given for each of the different classes as reflected in the value of the OPCODE operand.

* The generated word is not directly executable code and must therefore be coded in a data area of the user's program.

** Details in **Telecommunications Reference** manual.

COMMAND Macro for Basic Data Channel (Unit Record Devices and Magnetic Tape)

This macro generates a command word for physical input/output operations on card reader, card punch, line printer, magnetic tape, etc. (devices connected to the basic data channel). The generated command word specifies the hardware operation to be executed and, for commands that require data transfer, it designates the storage area associated with the operation. In addition, various operation modifiers may be supplied.

<u>Name</u>	<u>Operation</u>	<u>Operand</u>
[tag]	COMMAND	OPCODE=X'nn' [BUFADR=symbolic address] [BUFSIZ=decimal number] [CHAIN= { YES } { NO }] [SIZERR= { YES } { NO }] [SKIP= { YES } { NO }]

OPCODE=X'nn'

This parameter is a two-digit hexadecimal code specifying the hardware operation to be performed (see Table 7-5), or a command program jump command (OPCODE=X'08').

BUFADR=symbolic address Optional

Specifies the address of the data buffer.

BUFSIZ=decimal number Optional

Specifies the length of the data buffer in bytes. The maximum allowable value is 65,535 bytes. BUFSIZ=0 is not valid at execution time.

CHAIN= { YES }
 { NO } Optional

Indicates that another command word follows contiguously. Omission of this operand indicates that this is the last (or only) command word in the command program. CHAIN=NO represents the command word being the last one in the command program.

SIZERR= { YES }
 { NO } Optional

Any difference between the length of the data record processed at the hardware-software interface and the size of the data buffer is detected by the I/O routines. This "incorrect length indication" is normally treated as an error condition and causes termination of a command program at that point. The SIZERR=YES operand (suppress length check) allows chaining to

proceed regardless. SIZERR=NO causes termination of a command program at that point.

SKIP= { YES }
 { NO } Optional

SKIP=YES causes suppression of the transfer of data into storage during this command word execution. SKIP=NO causes the transfer of the data storage.

COMMAND Macro for DCABLE Operation

Creates a command word for a physical disc operation to return the disc drive cable address. When an EXCP executes this command word it will read the status of the physical device identified in the related PCB and will place status indication in a two-byte buffer specified by BUFADR.

<u>Name</u>	<u>Operation</u>	<u>Operand</u>
[tag]	COMMAND	OPCODE=DCABLE BUFADR=symbolic address [CHAIN= { YES } { NO }]

OPCODE=DCABLE

Operation code for returning disc drive cable address.

BUFADR=symbolic address

Specifies the two-byte buffer area where the disc drive cable address will be returned.

CHAIN= { YES }
 { NO } Optional

Indicates that another command word follows contiguously in storage and is to be executed immediately after this one. CHAIN=NO represents the command word being the last one in the command program.

COMMAND Macro for a DCSEEK Operation

Creates a command word for a physical disc seek operation. When executed by an EXCP, this command word will cause a disc seek to the cylinder and track specified in a buffer pointed to by BUFADR.

<u>Name</u>	<u>Operation</u>	<u>Operand</u>
[tag]	COMMAND	OPCODE=DCSEEK BUFADR=symbolic address [CHAIN= { YES } { NO }]

OPCODE=DCSEEK

Operation code for a disc seek.

Table 7-5. Peripheral Device Basic Hardware Operation Codes

Device	Operation	OPCODE
Card Reader (8010)	Read (and Feed): EBCDIC mode Card Image mode	02 22
	Test I/O No Operation Sense	00 03 04
	Write (no line spacing)	01
Line Printer (5120, 5060)	Write and Space Space 1 line Space 2 lines Space 3 lines Space 4 lines . . Space 14 lines Space 15 lines	09* 11 19 21 . . 71 79
	Space Immediate: Space 1 line Space 2 lines Space 3 lines Space 4 lines . . Space 14 lines Space 15 lines	0B* 13 1B 23 . . 73 7B
	Write and Skip: Skip to Channel 1 (top of form) Skip to Channel 2 Skip to Channel 3 Skip to Channel 4 . . Skip to Channel 11 Skip to Channel 12	89* 91 99 A1 . . D9 E1
	Skip Immediate: Skip to Channel 1 (top of form) Skip to Channel 2 Skip to Channel 3 Skip to Channel 4 . . Skip to Channel 11 Skip to Channel 12	8B* 93 9B A3 . . DB E3
	Test I/O No Operation Sense	00 03 04

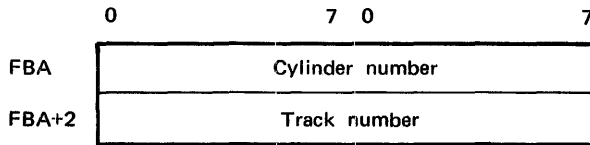
*These numbers in the series are being incremented by eight.

Table 7-5. Peripheral Device Basic Hardware Operation Codes (Continued)

Device	Operation	OPCODE
Card Reader/Punch (8025)	Read Only: EBCDIC mode Card Image mode	0A 2A
	Feed and Select Stacker: without offset with offset	23 A3
	Read, Feed and Select Stacker: EBCDIC mode, without offset EBCDIC mode, with offset Card Image mode, without offset Card Image mode, with offset	02 82 22 A2
	Punch, Feed and Select Stacker: EBCDIC mode, without offset EBCDIC mode, with offset Card Image mode, without offset Card Image mode, with offset	01 81 21 A1
	Test I/O No Operation Sense	00 03 04
Magnetic Tape (3237)	Write Forward Read Forward Rewind Rewind and Unload	01 02 07 0F
	Backspace: Block File	27 2F
	Forward Space: Block File	37 3F
	Write Tapemark (End-of-File) Erase Gap	1F 17
	Force Error Mode: Set Clear	E3 D3
	Test I/O No Operation Sense	00 X5 X6 XD XE 04

BUFADR=symbolic address

This specifies the address of a four-byte buffer which is assumed to have been preset by the user with the cylinder and track numbers to which the seek is to be made. The buffer format should be:



CHAIN= { YES }
 { NO } Optional

Indicates that another command word follows contiguously in storage and is to be executed immediately after this one. This parameter must be included to reserve the unit until the subsequent data transfer is complete. CHAIN=NO represents the command word being the last one in the command program.

COMMAND Macro for a DCSRCH Operation

Creates a command word for a physical disc search operation. When executed by an EXCP, this command word causes a disc search to be performed. The search command allows the user to locate, on the current track, the data he wishes to process. This is done by analyzing the physical track records for type and content, searching for the record specified by BUFADR. When the specified record is found, the next contiguous command word in the command program is executed*. If the search fails to find the specified record on the track, the command program is discontinued at this point and the *abnormal completion* bit is set in the request block.

<u>Name</u>	<u>Operation</u>	<u>Operand</u>
[tag]	COMMAND	OPCODE=DCSRCH BUFADR=symbolic address BUFSIZ=decimal number [FIELD= { HA } { RZCNT } { RNCNT }] [CHAIN= { YES } { NO }]

OPCODE=DCSRCH

Operation code for a disc search.

*This will normally be a DCREAD or DCWRIT to read or write the actual data record associated with the search. The search should always be made for a field (usually a count field) preceding the record to be processed. Appendix E contains further information on the organization and structure of physical disc records.

BUFADR=symbolic address

Specifies the address of a buffer containing the data to be used in comparing to the field being searched. This buffer must begin on a word boundary.

BUFSIZ=decimal number

Specifies the length of the field being searched for.

FIELD= { HA }
 { RZCNT }
 { RNCNT } Optional

Specifies the type of field for which to search. HA=home address, RZCNT=count field of record zero and RNCNT=count field of record *n*. If this parameter is omitted, HA is assumed. Following is a table specifying the field to search in order to read or write a desired field.

To Perform this DCREAD or DCWRIT Operation:	This DCSRCH Operation is First Required:
HA	No search required
Data	RNCNT
Key + Data	RNCNT
Count	R(N-1)CNT or HA
Count + Data	R(N-1)CNT or HA
Count + Key + Data	R(N-1) CNT or HA

Note that, when searching in preparation for a write operation, any fields that may be situated between the search field and the field to be written must be skipped using a DCREAD without transfer.

CHAIN= { YES }
 { NO } Optional

Indicates that another command word follows contiguously in storage and is to be executed immediately after this one (almost always the case with the DCSRCH). CHAIN=NO represents the command word being the last one in the command program.

COMMAND Macro for a DCREAD Operation

Creates a command word or chained group of command words for a physical disc read operation. When executed by an EXCP instruction, this command word (or group of command words) will cause a field (or number of fields) to be copied from disc into the specified buffer (or buffers). A multi-record data field read may be specified by including a BUFSIZ parameter which is a multiple of the DATSIZ parameter. However, when this is done, no other fields may be read by the same macro.

Name	Operation	Operand
[tag]	COMMAND	OPCODE=DCREAD { HABUF=symbolic address, HASIZ=decimal number DATBUF=symbolic address, DATSIZ=decimal number KEYBUF=symbolic address, KEYSIZ=decimal number CNTBUF=symbolic address, CNTSIZ=decimal number } [BUFSIZ=decimal number] [SKIP= { YES } { NO }] [CHAIN= { YES } { NO }] [FIELD= { RZCNT { RNCNT }]

OPCODE=DCREAD

Operation code for a disc read.

HABUF=symbolic address Optional
 HASIZ=decimal number

These two parameters are coded as a pair to indicate that a home address field is to be read. HABUF specifies the address of the buffer into which the home address is to be placed and HASIZ specifies the buffer length in bytes.

DATBUF=symbolic address Optional
 DATSIZ=decimal number

These two parameters are coded as a pair to indicate that a data field is to be read. DATBUF specifies the address of the buffer and DATSIZ specifies the length of the buffer, except in the case of multi-record reads (see BUFSIZ).

KEYBUF=symbolic address Optional
 KEYSIZ=decimal number

These two parameters are coded as a pair to indicate that a data field is to be read. KEYBUF specifies the address of the buffer, and KEYSIZ its length.

CNTBUF=symbolic address Optional
 CNTSIZ=decimal number

These two parameters are coded as a pair to indicate that a count field is to be read. CNTBUF specifies the address of the buffer, and CNTSIZ its length in bytes.

BUFSIZ=decimal number Optional

This parameter is to be coded for multi-record reads only. It indicates the overall length of the data buffer (DATBUF); it should be a multiple of the value of DATSIZ.

SKIP= { YES }
 { NO } Optional

When this parameter is included no data will be transferred to memory so that fields may be skipped and/or the cyclic burst bytes* of a record may be checked. The SKIP=NO option (default) reads data into the buffer area.

FIELD= { RZCNT }
 { RNCNT } Optional

This option gives the record in the count field. FIELD=RZCNT is record 0 in the count field; FIELD=RNCNT is record n in the count field.

CHAIN= { YES }
 { NO } Optional

Indicates that another command word follows contiguously in storage and is to be executed immediately after this one. CHAIN=NO (default) represents the command word being the last one in the command program.

The following table illustrates the valid combinations of parameters which may be coded in a DCREAD macro. (In all cases SKIP=YES may be included to suppress data transfer.)

Operation	Buffer	Multi-Record (BUFSIZ)
Read Home Address	HABUF	No
Read Data	DATBUF	Yes
Read Key and Data	KEYBUF+DATBUF	No
Read Count	CNTBUF	No
Read Count and Data	CNTBUF+DATBUF	No
Read Count, Key and Data	CNTBUF+KEYBUF+DATBUF	No

*Appendix E has further details.

CHAIN= $\left\{ \begin{array}{l} \text{YES} \\ \underline{\text{NO}} \end{array} \right\}$ Optional

Indicates that another command word follows contiguously in storage and is to be executed immediately after this one. CHAIN=NO represents the command word being the last in the command program.

FIELD= $\left\{ \begin{array}{l} \text{RZCNT} \\ \underline{\text{RNCNT}} \end{array} \right\}$ Optional

This option gives the record in the count field. FIELD=RZCNT is record 0 in the count field; FIELD=RNCNT is record n in the count field.

COMMAND Macro for a RESTORE Operation

Creates a physical command word which, when executed by an EXCP, causes a disc restore operation.

Name	Operation	Operand
[tag]	COMMAND	OPCODE=RESTORE

OPCODE=RESTORE

Operation code for a disc restore.

COMMAND Macro for a DCJUMP Operation

Creates a physical command word which, when executed by an EXCP, causes the EXCP to next execute the command word located at the address specified by CWADR. This may be used to link command programs.

Name	Operation	Operand
[tag]	COMMAND	OPCODE=DCJUMP CWADR=symbolic address

CWADR=symbolic address

Specifies the address of the command word which is to be executed immediately following this one.

CONTROL PROGRAM MACROS

WAIT – Wait for Service Request Completion

This macro is used to wait for the completion of one or more outstanding service requests. Execution of the requesting program will be suspended until completion of the specified request(s)*. Except for

the optional LIST and ERRCOMP, only one other operand may be coded with this macro; depending upon the particular parameter used, the macro will wait for completion of:

- any one of the outstanding requests (MODE = ANY)
- all of the outstanding requests (MODE = ALL)
- a specific outstanding request (REQADR = symbolic address)

Note that when the REQCNT option is used the WAIT macro is satisfied on any of the following conditions:

- When the WAIT macro is issued and the system detects that no requests are outstanding.
- when the WAIT macro is issued and the count of outstanding requests is not equal to that specified by REQCNT
- otherwise, when the first of any outstanding request operations is completed

Name	Operation	Operand
[tag]	WAIT	$\left[\text{MODE} = \left\{ \begin{array}{l} \text{ANY} \\ \underline{\text{ALL}} \end{array} \right\} \right]$ [REQADR=symbolic address] [REQCNT=symbolic address] $\left[\text{LIST} = \left\{ \begin{array}{l} \text{YES} \\ \underline{\text{NO}} \end{array} \right\} \right]$ $\left[\text{ERRCOMP} = \left\{ \begin{array}{l} \text{YES} \\ \underline{\text{NO}} \end{array} \right\} \right]$

MODE= $\left\{ \begin{array}{l} \text{ANY} \\ \underline{\text{ALL}} \end{array} \right\}$ Optional

This operand provides the option of waiting for the completion of any one or all of the currently outstanding requests.

REQADR=symbolic address Optional

This operand should be used when it is desired to wait for the completion of a single specific request. The symbolic address must be the address of the first word in the request block (parameter list) of the specific request to wait for (not service request instruction itself).

REQCNT=symbolic address Optional

This operand specifies the location of a one-byte field containing the hexadecimal count of the number of requests known to be outstanding. Use of this operand gives a wait for the completion of any service request.

*If none of the specified requests is outstanding at the time the WAIT is executed, control will be returned immediately to the requesting program.

LIST= $\left\{ \begin{array}{l} \text{YES} \\ \text{NO} \end{array} \right\}$ Optional

To be used when either the system linkage (service request instruction) or the parameter table is to be generated separately. LIST=NO generates the service request only; LIST=YES generates the parameter table (list) only. When no LIST operand is specified, both the service request and the parameter table are generated in line.

ERRCOMP= $\left\{ \begin{array}{l} \text{YES} \\ \text{NO} \end{array} \right\}$ Optional

To be used when the user decides to retain control in the event of an error (ERRCOMP=YES). When ERRCOMP=NO is coded, or the default is taken, the program will be aborted if an error occurs.

DELAY – Suspend Program Execution†

This macro causes program execution to be suspended for a specified period of time, with the option of resuming on completion of any outstanding service request. The period of suspension may be specified either by the SECONDS operand or by the CYCLES operand (one of which is always required).

Name	Operation	Operand
[tag]	DELAY	SECONDS=symbolic address CYCLES=symbolic address [BREAK= $\left\{ \begin{array}{l} \text{YES} \\ \text{NO} \end{array} \right\}$] [LIST= $\left\{ \begin{array}{l} \text{YES} \\ \text{NO} \end{array} \right\}$] [ERRCOMP= $\left\{ \begin{array}{l} \text{YES} \\ \text{NO} \end{array} \right\}$]

SECONDS=symbolic address Optional

Specifies the location of a one-word field containing the desired delay in seconds (hexadecimal).

CYCLES=symbolic address Optional

Specifies the location (hexadecimal) of a one-word field containing the desired delay in cycles (one cycle = 50.2 milliseconds).

BREAK= $\left\{ \begin{array}{l} \text{YES} \\ \text{NO} \end{array} \right\}$ Optional

BREAK=YES indicates that program execution is to be resumed immediately after any outstanding service request is completed. When this operand is omitted or BREAK=NO, the program will be resumed only when the specified delay time has elapsed.

†Not available on minimal system.

LIST= $\left\{ \begin{array}{l} \text{YES} \\ \text{NO} \end{array} \right\}$ Optional

To be used when either the system linkage (service request instruction) or the parameter table is to be generated separately. LIST=NO generates the service request only; LIST=YES generates the parameter table (list) only. When no LIST operand is specified, both the service request and the parameter table are generated.

ERRCOMP= $\left\{ \begin{array}{l} \text{YES} \\ \text{NO} \end{array} \right\}$ Optional

To be used when the user decides to retain control in the event of an error (ERRCOMP=YES). When ERRCOMP=NO is coded, or the default is taken, the program will be aborted if an error occurs.

INFORM – Service Request Completed

The INFORM macro informs the requesting program that one of the program's outstanding service requests has been completed. INFORM returns control immediately to the user and subsequently indicates that a service request has been completed by setting the *complete* bit in the INFORM request block (bit 0 of the third byte of the parameter list). The REQCNT operand points to a location containing a one-word count of the number of service requests which are not known to have been completed. The purpose of the count specification is to indicate whether any of the requests whose completion status is unknown have already been completed by the time the INFORM macro is executed (this will include the possibility of a request being completed in the interval between the issuance and actual processing of the INFORM). Thus the INFORM macro is satisfied by any one of the following conditions:

- when no service requests are outstanding
- when the number of outstanding requests is less than that specified by REQCNT
- when the first outstanding request is completed

If when INFORM is issued, the number of outstanding requests exceeds the REQCNT specification, an error is assumed and the INFORM request is completed as abnormal.

Name	Operation	Operand
[tag]	INFORM	REQCNT=symbolic address [LIST= $\left\{ \begin{array}{l} \text{YES} \\ \text{NO} \end{array} \right\}$] [ERRCOMP= $\left\{ \begin{array}{l} \text{YES} \\ \text{NO} \end{array} \right\}$]

REQCNT=symbolic address

Specifies the address of a one-byte location containing the number of service requests that are not known to have completed (outstanding).

LIST= $\left\{ \begin{matrix} \text{YES} \\ \text{NO} \end{matrix} \right\}$ Optional

To be used when either the system linkage (service request instruction) or the parameter table is to be generated separately. LIST=NO generates the service request only; LIST=YES generates the parameter table (list) only. When no LIST operand is specified, both the service request and the parameter table are generated.

ERRCOMP= $\left\{ \begin{matrix} \text{YES} \\ \text{NO} \end{matrix} \right\}$ Optional

To be used when the user decides to retain control in the event of an error (ERRCOMP=YES). When ERRCOMP=NO is coded, or the default is taken, the program will be aborted if an error occurs.

POST – Create Compressed Communication Byte

This macro, in conjunction with RPOST, is used for communication between user job steps. POST takes the first eight bytes located at INFOADR and interprets each byte to modify the corresponding bit in a compressed communication byte located in the Job Control Table*. At the start of a job the POST communication byte is initialized to zero.

<u>Name</u>	<u>Operation</u>	<u>Operand</u>
[tag]	POST	INFOADR=symbolic address $\left[\text{LIST} = \left\{ \begin{matrix} \text{YES} \\ \text{NO} \end{matrix} \right\} \right]$ $\left[\text{RETURN} = \left\{ \begin{matrix} \text{YES} \\ \text{NO} \end{matrix} \right\} \right]$ $\left[\text{ERRCOMP} = \left\{ \begin{matrix} \text{YES} \\ \text{NO} \end{matrix} \right\} \right]$

INFOADR=symbolic address

Specifies the location of an eight-byte buffer containing the information to be posted. The buffer should contain only EBCDIC 0's, 1's or X's. These bytes will be interpreted as follows to modify the corresponding eight bits in the communication byte:

- EBCDIC 0 = reset bit to 0
- EBCDIC 1 = set bit to 1
- EBCDIC X = leave bit unchanged

*This scheme is compatible with IBM's UPSI bit mechanism.

LIST= $\left\{ \begin{matrix} \text{YES} \\ \text{NO} \end{matrix} \right\}$ Optional

To be used when either the system linkage (service request instruction) or the parameter table is to be generated separately. LIST=NO generates the service request only; LIST=YES generates the parameter table (list) only. When no LIST operand is specified, both the service request and the parameter table are generated.

RETURN= $\left\{ \begin{matrix} \text{YES} \\ \text{NO} \end{matrix} \right\}$ Optional

To be used when the user program wishes to be given control immediately on recognition of the request, but before completion (RETURN=YES). When RETURN=NO is coded, or the default is taken, control will only be returned on completion of the request.

ERRCOMP= $\left\{ \begin{matrix} \text{YES} \\ \text{NO} \end{matrix} \right\}$ Optional

To be used when the user decides to retain control in the event of an error (ERRCOMP=YES). When ERRCOMP=NO is coded, or the default is taken, the program will be aborted if an error occurs.

RPOST – Expand from Communication Byte

This macro is used to read information, located in an interjob step communication area, which was posted there either by a POST macro in a previous job step or by the SWITCH=parameter in a //SET Control Language statement*. Each bit in the one-byte communication area is expanded to an EBCDIC equivalent and placed in the buffer specified by INFOADR.

<u>Name</u>	<u>Operation</u>	<u>Operand</u>
[tag]	RPOST	INFOADR=symbolic address $\left[\text{LIST} = \left\{ \begin{matrix} \text{YES} \\ \text{NO} \end{matrix} \right\} \right]$ $\left[\text{RETURN} = \left\{ \begin{matrix} \text{YES} \\ \text{NO} \end{matrix} \right\} \right]$ $\left[\text{ERRCOMP} = \left\{ \begin{matrix} \text{YES} \\ \text{NO} \end{matrix} \right\} \right]$

INFOADR=symbolic address

Specifies the location of an eight-byte buffer where the interpreted information will be placed. Each bit in the communication byte will be converted to an EBCDIC byte as follows:

- Binary 0 = EBCDIC 0
- Binary 1 = EBCDIC 1

LIST= $\left\{ \begin{array}{l} \text{YES} \\ \text{NO} \end{array} \right\}$ Optional

To be used when either the system linkage (service request instruction) or the parameter table is to be generated separately. LIST=NO generates the service request only; LIST=YES generates the parameter table (list) only. When no LIST operand is specified, both the service request and the parameter table are generated.

RETURN= $\left\{ \begin{array}{l} \text{YES} \\ \underline{\text{NO}} \end{array} \right\}$ Optional

To be used when the user program wishes to be given control immediately on recognition of the request, before completion (RETURN=YES). When RETURN=NO is coded, or the default is taken, control will only be returned on completion of the request.

ERRCOMP= $\left\{ \begin{array}{l} \text{YES} \\ \underline{\text{NO}} \end{array} \right\}$ Optional

To be used when the user decides to retain control in the event of an error (ERRCOMP=YES). When ERRCOMP=NO is coded, or the default is taken, the program will be aborted if an error occurs.

SETCOM – Transfer to Job Control Table

This macro, in conjunction with GETCOM, is used for communication between user job steps. It transfers, without modification, eight bytes of information located in a user-specified buffer to an eight-byte communication area in the Job Control Table where it may be picked up by a GETCOM in a subsequent job step. At the start of a job the communication area is initialized to all blanks.

<u>Name</u>	<u>Operation</u>	<u>Operand</u>
[tag]	SETCOM	INFOADR=symbolic address $\left[\text{LIST} = \left\{ \begin{array}{l} \text{YES} \\ \text{NO} \end{array} \right\} \right]$ $\left[\text{RETURN} = \left\{ \begin{array}{l} \text{YES} \\ \underline{\text{NO}} \end{array} \right\} \right]$ $\left[\text{ERRCOMP} = \left\{ \begin{array}{l} \text{YES} \\ \underline{\text{NO}} \end{array} \right\} \right]$

INFOADR=symbolic address

Specifies the location of an eight-byte buffer containing the data to be transferred to the communication area.

LIST= $\left\{ \begin{array}{l} \text{YES} \\ \text{NO} \end{array} \right\}$ Optional

To be used when either the system linkage (service request instruction) or the parameter table is to be generated separately. LIST=NO generates the service request only; LIST=YES generates the parameter table (list) only. When no LIST operand is specified, both the service request and the parameter table are generated.

RETURN= $\left\{ \begin{array}{l} \text{YES} \\ \underline{\text{NO}} \end{array} \right\}$ Optional

To be used when the user program wishes to be given control immediately on recognition of the request but before completion (RETURN=YES). When RETURN=NO is coded, or the default is taken, control will only be returned on completion of the request.

ERRCOMP= $\left\{ \begin{array}{l} \text{YES} \\ \underline{\text{NO}} \end{array} \right\}$ Optional

To be used when the user decides to retain control in the event of an error (ERRCOMP=YES). When ERRCOMP=NO is coded, or the default is taken, the program will be aborted if an error occurs.

GETCOM – Transfer from Job Control Table Communication Area

This macro is used to obtain the information which had been placed in an eight-byte communication area by a SETCOM executed in a previous job step. The content of the communication area, located in the Job Control Table, is moved, without modification, to an eight-byte buffer specified by INFOADR.

<u>Name</u>	<u>Operation</u>	<u>Operand</u>
[tag]	GETCOM	INFOADR=symbolic address $\left[\text{LIST} = \left\{ \begin{array}{l} \text{YES} \\ \text{NO} \end{array} \right\} \right]$ $\left[\text{RETURN} = \left\{ \begin{array}{l} \text{YES} \\ \underline{\text{NO}} \end{array} \right\} \right]$ $\left[\text{ERRCOMP} = \left\{ \begin{array}{l} \text{YES} \\ \underline{\text{NO}} \end{array} \right\} \right]$

INFOADR=symbolic address

Specifies the location of an eight-byte user buffer into which the contents of the communication area will be moved.

LIST= $\left\{ \begin{array}{l} \text{YES} \\ \text{NO} \end{array} \right\}$ Optional

To be used when either the system linkage (service request instruction) or the parameter table is to be generated separately. LIST=NO generates the service request only; LIST=YES generates the parameter table (list) only. When no LIST operand is specified, both the service request and the parameter table are generated.

RETURN= $\left\{ \begin{array}{l} \text{YES} \\ \text{NO} \end{array} \right\}$ Optional

To be used when the user program wishes to be given control immediately on recognition of the request, but before completion (RETURN=YES). When RETURN=NO is coded, or the default is taken, control will only be returned on completion of the request.

ERRCOMP= $\left\{ \begin{array}{l} \text{YES} \\ \text{NO} \end{array} \right\}$ Optional

To be used when the user decides to retain control in the event of an error (ERRCOMP=YES). When ERRCOMP=NO is coded, or the default is taken, the program will be aborted if an error occurs.

ACCEPT – Read //PAR Card

This macro is used to obtain data contained on //PAR cards supplied in the Control Language deck. ACCEPT transfers the EBCDIC content of a single //PAR card to the buffer specified by DATBUF1.

The particular //PAR card read by the ACCEPT is determined by a pointer located in the Job Control Table. At the start of the job step, this pointer is initialized to point to the first //PAR card so that the first ACCEPT executed in the step will read the first //PAR card supplied with this step. Each time an ACCEPT is executed the //PAR card pointer is updated to the next consecutive card so that on the next execution the next //PAR card will be read.

If, when an ACCEPT is executed, there are no more //PAR cards left to be read, control will be transferred to the location specified by the ENDADR operand.

The optional operand PARNUM allows the //PAR pointer to be set to any desired //PAR card by specifying a user location containing a count (since the //PAR cards are considered to be numbered 1 through *n* by implication). Each time an ACCEPT macro with a PARNUM specification is executed, the

content of the user counter specified by PARNUM will be incremented by 1 so that this same ACCEPT may be re-executed to read the next //PAR card.

The format of the data transferred to the buffer consists of a four-byte Common Stored Data Format (CSDF) control header preceding the data from the //PAR card, unless the optional operand CSD=NO is included. The content of the //PAR card is always stripped of characters through the first blank (so as not to include “//PAR” itself) and is also stripped of the sequence number field (columns 73 through 80) unless the optional operand STRIP=NO is included.

Name	Operation	Operand
[tag]	ACCEPT	DATBUF1=symbolic address [CSD=NO] [STRIP=NO] [PARNUM=symbolic address] ENDADR=symbolic address [LIST= $\left\{ \begin{array}{l} \text{YES} \\ \text{NO} \end{array} \right\}$] [RETURN= $\left\{ \begin{array}{l} \text{YES} \\ \text{NO} \end{array} \right\}$] [ERRCOMP= $\left\{ \begin{array}{l} \text{YES} \\ \text{NO} \end{array} \right\}$]

DATBUF1=symbolic address

Specifies the location of the buffer into which the data from the //PAR card will be transferred.

CSD=NO (Optional)

If this operand is used the parameter line will not be preceded by the CSDF control header.

STRIP=NO (Optional)

If this operand is used the Sequence Number Field (card columns 73 through 80) will not be stripped.

PARNUM=symbolic address (Optional)

Specifies the location of a one-word binary counter whose value is to be used as a //PAR card pointer. A value of 1 will cause the first //PAR card to be read, and 2 the second, and so on. Each time the ACCEPT is executed, the value of the counter will be increased by 1.

ENDADR=symbolic address

This required operand specifies the program address where control is to be transferred if all the //PAR cards have been read (value of system's //PAR counter greater than number of //PAR cards supplied) or if the PARNUM specified is greater than the number of //PAR cards supplied.

LIST= $\begin{cases} \text{YES} \\ \text{NO} \end{cases}$ Optional

To be used when either the system linkage (service request instruction) or the parameter table is to be generated separately. LIST=NO generates the service request only; LIST=YES generates the parameter table (list) only. When no LIST operand is specified, both the service request and the parameter table are generated.

RETURN= $\begin{cases} \text{YES} \\ \text{NO} \end{cases}$ Optional

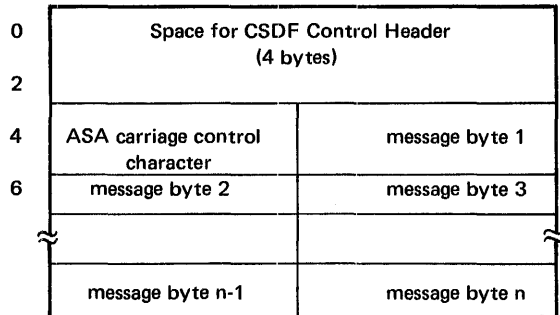
To be used when the user program wishes to be given control immediately on recognition of the request, but before completion (RETURN=YES). When RETURN=NO is coded, or the default is taken, control will only be returned on completion of the request.

ERRCOMP= $\begin{cases} \text{YES} \\ \text{NO} \end{cases}$ Optional

To be used when the user decides to retain control in the event of an error (ERRCOMP=YES). When ERRCOMP=NO is coded, or the default is taken, the program will be aborted if an error occurs.

DISPLAY – Write Message on SYSOUT

The DISPLAY macro is used for writing a one-line message on the SYSOUT file. The address of the message buffer, which must be aligned on a word boundary, is specified by DATBUF. The format of the buffer must be as follows:



The first four bytes of the buffer are reserved for a Common Stored Data Format (CSDF) control header to be generated by the system. The fifth byte, preceding the message, is an ASA carriage control character* (not to be printed) and the message itself is an EBCDIC character string up to 132 bytes in length, starting in the sixth byte. If the message is shorter than 132 bytes, a storage location containing the length must be set up and the DATSIZ operand must be included in the call.

*Details in Control Program and Data Management Services – Basic Reference.

Name	Operation	Operand
[tag]	DISPLAY	DATBUF=symbolic address [DATSIZ=symbolic address] [ERRCOMP= $\begin{cases} \text{YES} \\ \text{NO} \end{cases}$]

DATBUF=symbolic address

Specifies the address of the first byte in the message buffer. This will be the address of the space reserved for the CSDF control header and the first byte of the message itself (the ASA control character) should be displaced 4 bytes from this address.

DATSIZ=symbolic address Optional

Specifies the address of a one-word location containing the length of the message. If this operand is omitted, the message length is assumed to be 132 bytes.

ERRCOMP= $\begin{cases} \text{YES} \\ \text{NO} \end{cases}$ Optional

If ERRCOMP=YES is specified, the program is not aborted. When ERRCOMP=NO is coded, or the default is taken, the program will be aborted if an error occurs.

MEMLIM – Identify Partition Limit

This macro returns the first byte address of the highest addressable 256-byte page of storage available to the problem program in the partition in which the program is running. The returned address is expressed as the absolute address of the first word of the last addressable page. MEMLIM is intended for enabling programs which are expected to run in partitions of various different sizes to “spread themselves out” to occupy as much space as is available (in the interest of efficiency). In this situation, the partition space pool should always be fixed by means of the appropriate Linkage Editor directive.*

Name	Operation	Operand
[tag]	MEMLIM	INFOADR=symbolic address [LIST= $\begin{cases} \text{YES} \\ \text{NO} \end{cases}$] [RETURN= $\begin{cases} \text{YES} \\ \text{NO} \end{cases}$] [ERRCOMP= $\begin{cases} \text{YES} \\ \text{NO} \end{cases}$]

*Linkage Editor portion of Program Library Services Reference.

INFOADR=symbolic address

Specifies the location of a two-byte area where the address of the highest addressable 256-byte page is to be placed.

LIST= $\left\{ \begin{array}{l} \text{YES} \\ \text{NO} \end{array} \right\}$ Optional

To be used when either the system linkage (service request instruction) or the parameter table is to be generated separately. LIST=NO generates the service request only; LIST=YES generates the parameter table (list) only. When no LIST operand is specified, both the service request and the parameter table are generated.

RETURN= $\left\{ \begin{array}{l} \text{YES} \\ \text{NO} \end{array} \right\}$ Optional

To be used when the user program wishes to be given control immediately on recognition of the request but before completion (RETURN=YES). When RETURN=NO is coded, or the default is taken, control will only be returned on completion of the request.

ERRCOMP= $\left\{ \begin{array}{l} \text{YES} \\ \text{NO} \end{array} \right\}$ Optional

To be used when the user decides to retain control in the event of an error (ERRCOMP=YES). When ERRCOMP=NO is coded, or the default is taken, the program will be aborted if an irrecoverable error occurs.

SETIF – Post Code for Control Language Test

This macro enables a program, running as one step of a multi-step job, to post in the Job Control Table (usually at step completion) a code which may be tested by a Control Language //IF statement* in order to govern the subsequent course of the job. The code to be posted is one byte in length (EBCDIC character); its address is specified by INFOADR.

Name	Operation	Operand
[tag]	SETIF	INFOADR=symbolic address $\left[\text{LIST} = \left\{ \begin{array}{l} \text{YES} \\ \text{NO} \end{array} \right\} \right]$ $\left[\text{RETURN} = \left\{ \begin{array}{l} \text{YES} \\ \text{NO} \end{array} \right\} \right]$ $\left[\text{ERRCOMP} = \left\{ \begin{array}{l} \text{YES} \\ \text{NO} \end{array} \right\} \right]$

INFOADR=symbolic address

Specifies the location of the one-byte (EBCDIC character) of data to be posted in the IF code field of the Job Control Table.

LIST= $\left\{ \begin{array}{l} \text{YES} \\ \text{NO} \end{array} \right\}$ Optional

To be used when either the system linkage (service request instruction) or the parameter table is to be generated separately. LIST=NO generates the service request only; LIST=YES generates parameter table (list) only. When no LIST operand is specified, both the service request and the parameter table are generated.

RETURN= $\left\{ \begin{array}{l} \text{YES} \\ \text{NO} \end{array} \right\}$ Optional

To be used when the user program wishes to be given control immediately on recognition of the request but before completion (RETURN=YES). When RETURN=NO is coded, or the default is taken, control will only be returned on completion of the request.

ERRCOMP= $\left\{ \begin{array}{l} \text{YES} \\ \text{NO} \end{array} \right\}$ Optional

To be used when the user decides to retain control in the event of an error (ERRCOMP=YES). When ERRCOMP=NO is coded, or the default is taken, the program will be aborted if an error occurs.

HALT – Terminate Program

The HALT macro is used to perform normal termination of a user's program step. This macro will not result in a memory dump unless DUMP=YES has been coded in the //EXECUTE statement associated with the program. The format for HALT is as follows:

Name	Operation	Operand
[tag]	HALT	

*See Control Language Services Reference.

EHALT – Terminate Program

The EHALT (error halt) macro is used to request termination of a user's job. This macro will not automatically give a memory dump unless DUMP=YES has been coded in the //EXECUTE statement associated with the program. The format for EHALT is as follows:

<u>Name</u>	<u>Operation</u>	<u>Operand</u>
[tag]	EHALT	

ABEND – Terminate Program Abnormally

The ABEND macro is used to request abnormal termination of a job, and to pass a completion code to Job Monitor for display. A dump will be given unless DUMP=NO is specified on the //EXECUTE statement. The completion code is a 16-bit binary value. The format for ABEND is as follows:

<u>Name</u>	<u>Operation</u>	<u>Operand</u>
[tag]	ABEND	INFOADR=symbolic address

INFOADR=symbolic address

Specifies the first byte address of the area containing the completion code.

TIME – Retrieve Time of Day

The TIME macro returns the current time of day in the operand specified. The time of day is returned in an unpacked decimal format: hhmmss, where hh is the hour, mm is the minute, and ss is the second. The format for TIME is as follows:

<u>Name</u>	<u>Operation</u>	<u>Operand</u>
[tag]	TIME	INFOADR=symbolic address

INFOADR=symbolic address

Specifies the first byte address of the area which receives the time.

SDATE – Retrieve System Data

The SDATE (system date) macro returns the system date in the operand specified. The date is returned in one of two unpacked decimal formats: mmddyy or yyjjj, where mm is the month, dd is the day, yy is the year, and jjj is the Julian day. The format for SDATE is as follows:

<u>Name</u>	<u>Operation</u>	<u>Operand</u>
tag	SDATE	INFOADR=symbolic address MODE={ $\frac{C}{J}$ }

INFOADR=symbolic address

Specifies the first byte address of the area that receives the data.

MODE={ $\frac{C}{J}$ }

Specifies the current date in the calendar (C) 6-byte format mmddyy or the Julian (U) 5-byte format yyjjj. The default value is C.

JDATE – Retrieve Job Date

The JDATE (job date) macro returns the date provided for by a //SET statement. The date returned will be the system date unless the //SET statement has specified a job date. The date is returned in one of two unpacked decimal formats: mmddyy or yyjjj, where mm is the month, dd is the day, yy is the year, and jjj is the Julian day. The format for JDATE is as follows:

<u>Name</u>	<u>Operation</u>	<u>Operand</u>
tag	JDATE	INFOADR=symbolic address MODE={ $\frac{C}{J}$ }

INFOADR=symbolic address

Specifies the first byte address of the area which receives the date.

MODE={ $\frac{C}{J}$ }

Specifies the current date in the calendar (C) 6-byte format mmddyy or Julian (J) 5-bit format yyjjj. The default value is C.

CONSOLE COMMUNICATION MACROS

Two macros, CONSOLE (an active macro) and MESSAGE (a data macro), are available for communicating with the operator's console.

CONSOLE – Transmit Message to Console and Optionally Receive Reply

The CONSOLE macro enables programs to transmit messages to the operator's console and optionally receive replies. The main-storage format of the message to be sent to the console must include two fields, a control block which is not typed and the text field which contains the actual message. The buffer set up to receive a reply from the console (if any) must contain a control block followed by the actual buffer for the reply text. The format of CONSOLE is as follows:

Name	Operation	Operand
[tag]	CONSOLE	DATBUF1=symbolic address [DATBUF2=symbolic address]

DATBUF1=symbolic address

Specifies the address of the message control block which is followed by the message test.

DATBUF2=symbolic address (Optional)

Specifies the address of the reply control block which is followed by the reply buffer area.

MESSAGE – Set Up Message Format

The MESSAGE macro, to be used in conjunction with the CONSOLE macro, simplifies the generation of messages by creating the correct format required by CONSOLE. A tag is required for all MESSAGE macros so that the corresponding CONSOLE macro may locate it. Two formats exist for the MESSAGE macro, one for generating an output message and one for generating a reply buffer.

Generation of an Output Message

The format for generating an output message is as follows:

Name	Operation	Operand
tag	MESSAGE	[DATBUF1=symbolic address] { DATSIZ1=decimal number } { DATATXT=character string } [MODE={ I / D }]

DATBUF1=symbolic address (Optional)

Enables a name to be attached to the beginning of the message text field.

DATSIZ1=decimal number

Specifies the decimal length (in bytes) of the message text. If no length is specified, the text length will be used. If there is no message, a length of zero is assumed. The maximum value of DATSIZ1 is 100.

DATATXT=character string

Specifies the actual message to be placed in the message text field. It is created in EBCDIC and may be up to 100 characters (bytes) long. The default is a string of blanks, with length being determined by the parameter DATSIZ1. The character string must be coded in the form, C'message text'.

MODE={ I / D } (Optional)

Specifies whether the message is informative (I) or directive (D). MODE=D indicates that the message calls for some operator action. The default value of MODE is I.

Generation of a Reply Buffer

The format for generating a reply buffer is as follows:

Name	Operation	Operand
tag	MESSAGE	[DATBUF2=symbolic address] DATSIZ2=decimal number

DATBUF2=symbolic address (Optional)

Enables a name to be attached to the beginning of the reply text field.

DATSIZ2=decimal number

Specifies the decimal length in characters (bytes) of the reply buffer to be generated. The reply field will be assembled with blanks. The maximum value of DATSIZ2 is 100.

A. PACK CATALOG AND CENTRAL CATALOG FORMATS

The formats used by the pack catalog and central catalog of the disc are discussed in the following paragraphs.

PACK CATALOG

Figures A-1 through A-5 illustrate the use of the name, attribute and space elements in generating the pack catalog. The normal block of the pack catalog is 128 bytes, and the continuation element is 64 bytes for normal files and 128 bytes for a continuation to the description of pack space, which is catalog blocks.

At disc initialization time, the pack catalog is created with block formatting as follows (assuming the pack contains the central catalog):

- Block 1 Entry that describes the pack catalog itself
- Block 2 Entry that describes the space available on the pack
- Block 3 Entry that describes the space occupied by the central catalog
- Block 4 First block of space available in the pack catalog. Name element fields are filled as follows:
 - Control bytes X'8200007C'
 - Next name X'000500'
 - Previous name X'000000'

- Block 5-n Remaining blocks of space are constructed in the same format as block 4 with proper linking through the *next name* and *previous name* fields.

During the process of allocating a data file, one of the available blocks is removed from the string of available pack catalog blocks and is linked into the chain describing existing files. When space is needed for a continuation element, another block is removed from the available chain. Half of the block is used for a continuation element and the other half is linked to start the chain of available continuation space.

CENTRAL CATALOG

Figures A-6 through A-8 illustrate the use of the name, attribute, and volume elements in constructing entries for the central catalog. The normal block size of the central catalog is 128 bytes, and the continuation element is 64 bytes.

At disc initialization time, the first block is formatted as shown in Figure A-7. Remaining blocks are considered available and are linked as those in the pack catalog.

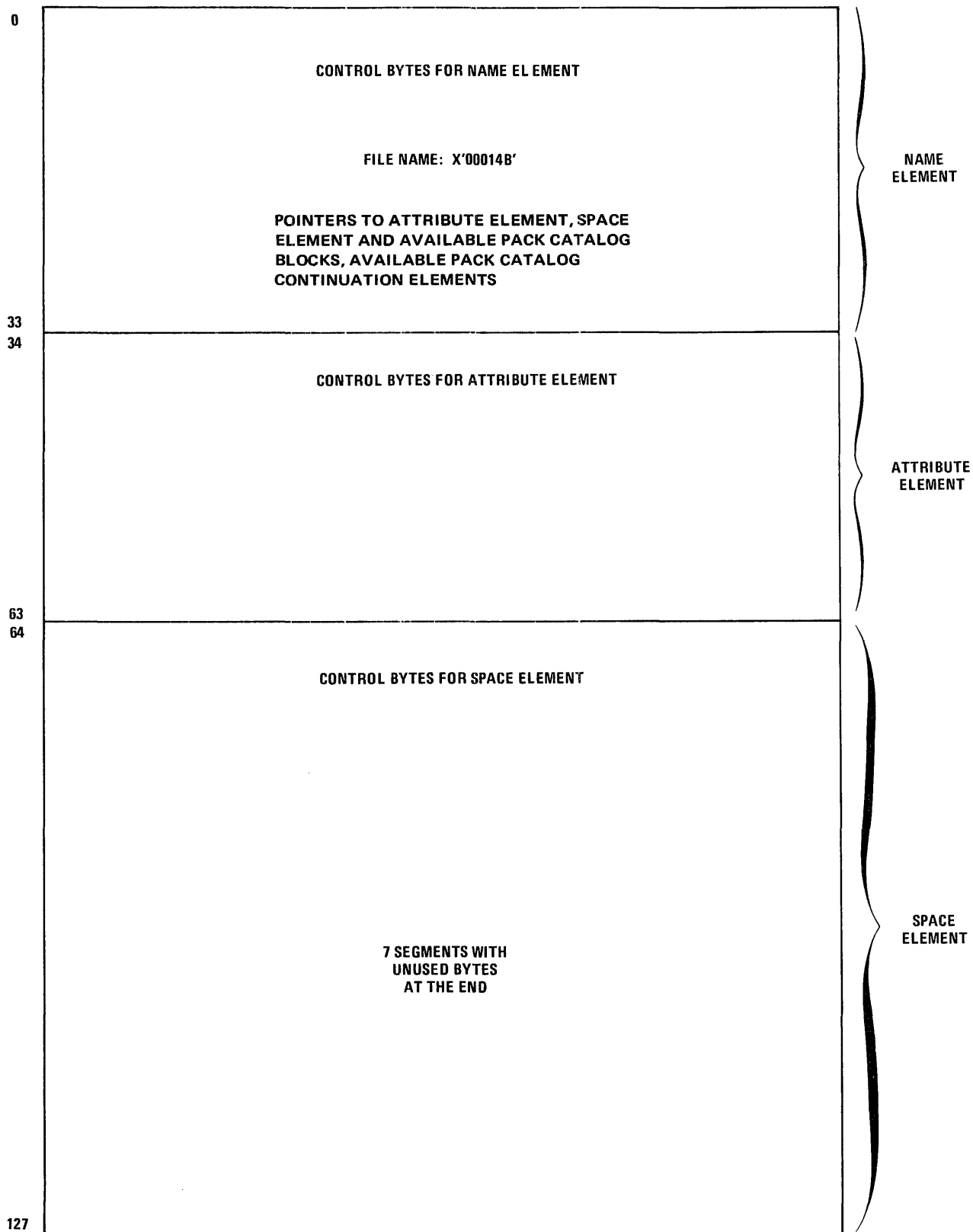


Figure A-1. Block 1 – Pack Catalog Entry in Pack Catalog

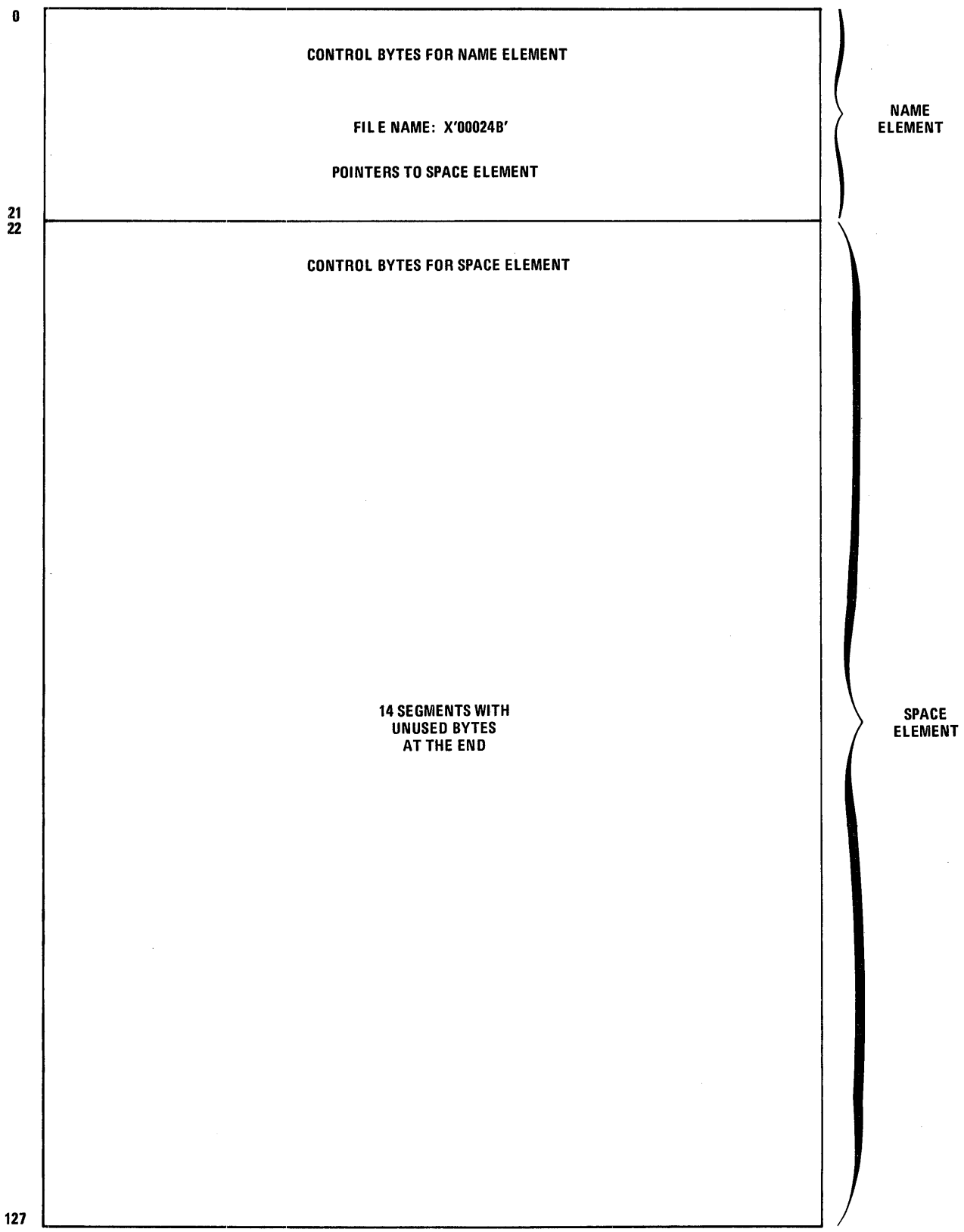


Figure A-2. Block 2 – Space Entry in Pack Catalog

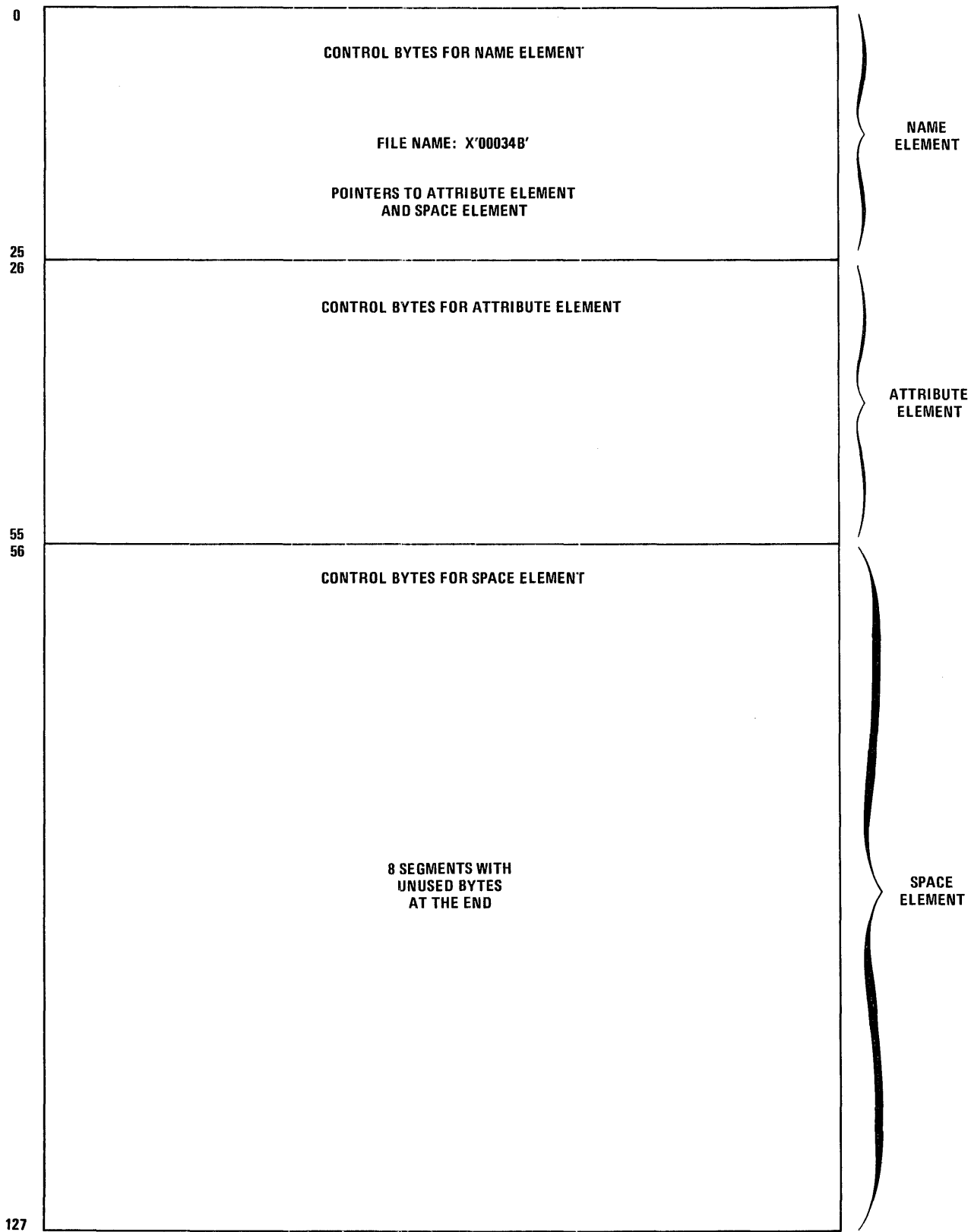


Figure A-3. Block 3 – Central Catalog Entry in Pack Catalog

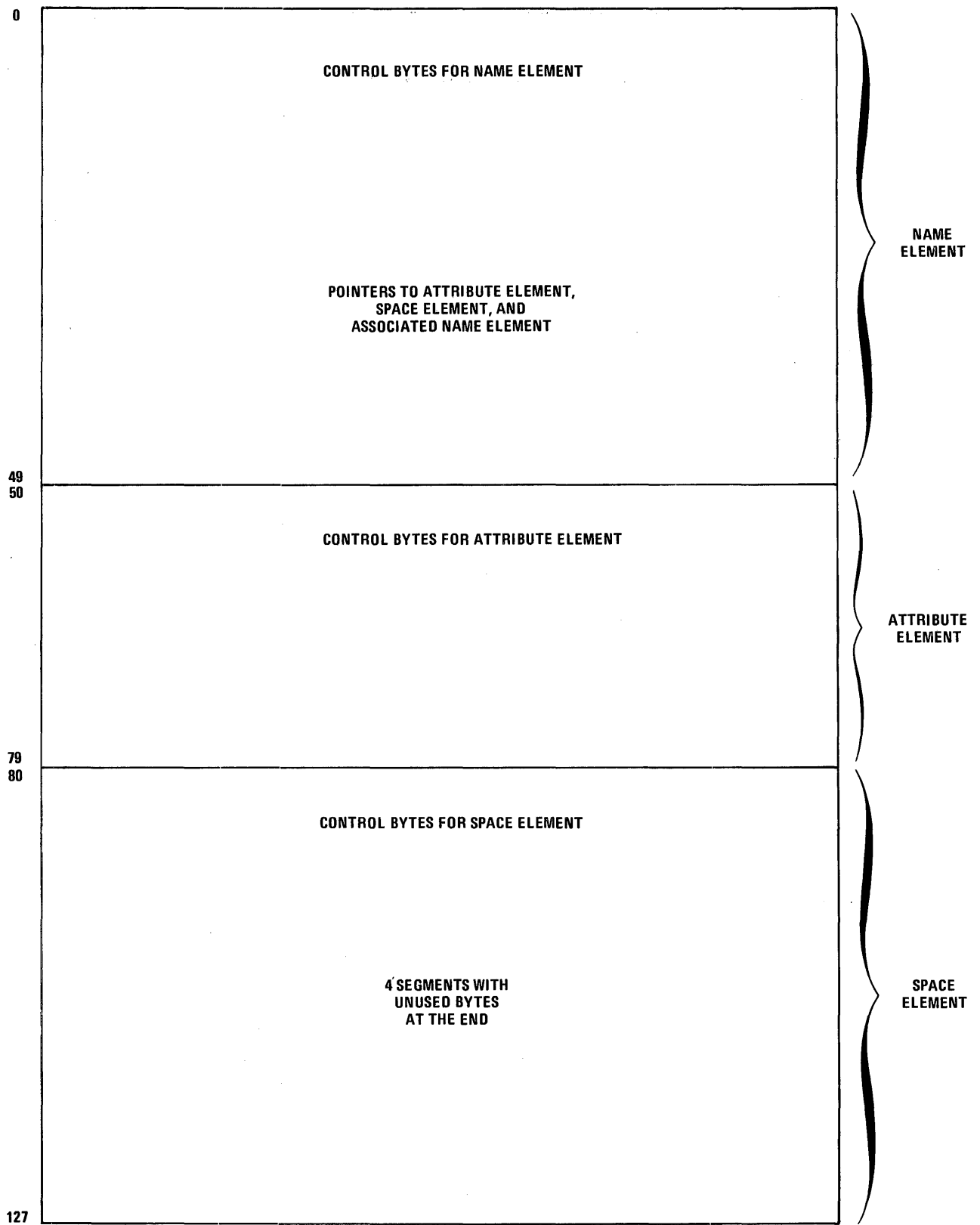


Figure A-4. Block 4-n – Normal File Entry in Pack Catalog

0

CONTROL BYTES FOR SPACE CONTINUATION

63

Figure A-5. Pack Catalog Space Element Continuation

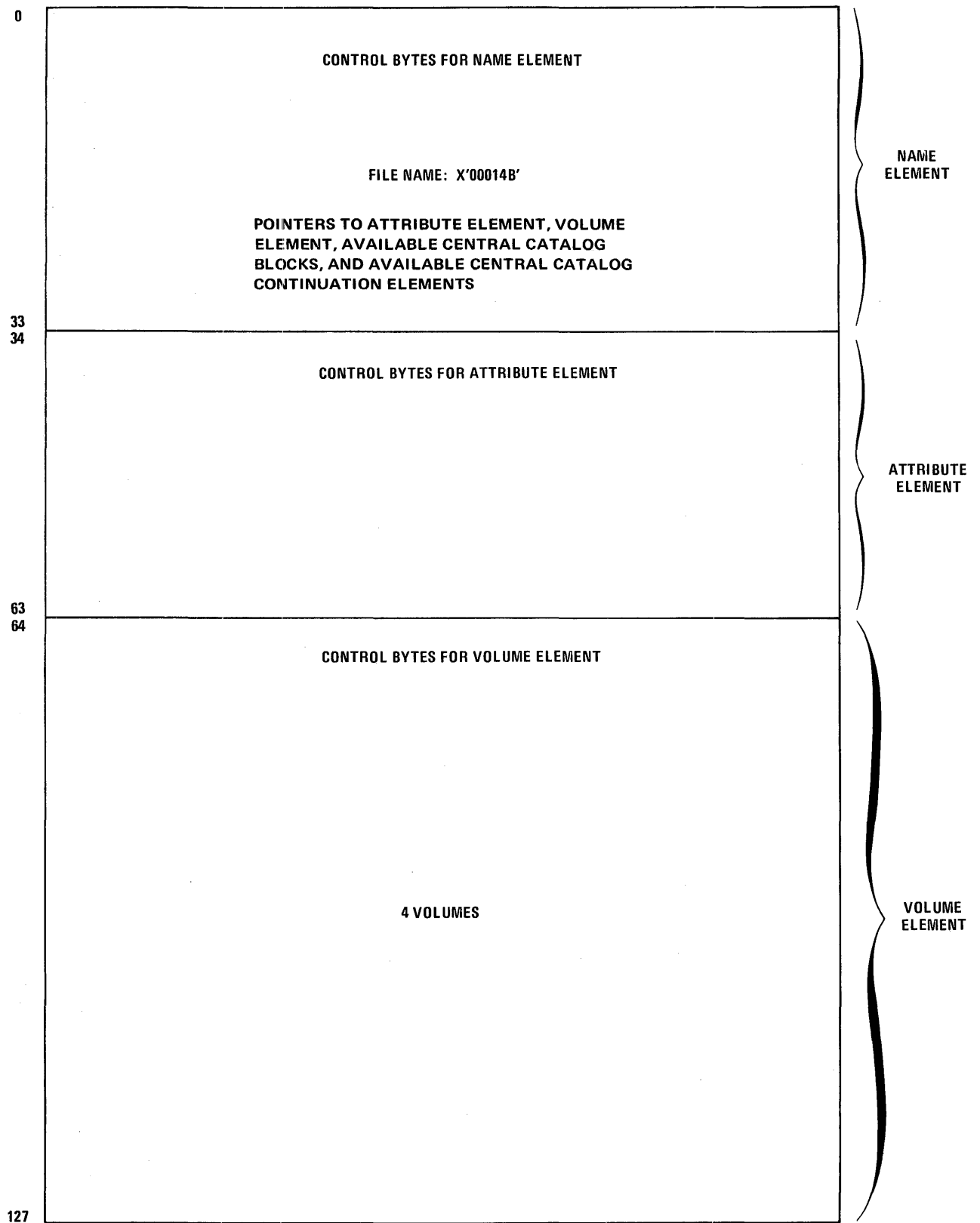


Figure A-6. Block 1 – Central Catalog Entry in Central Catalog

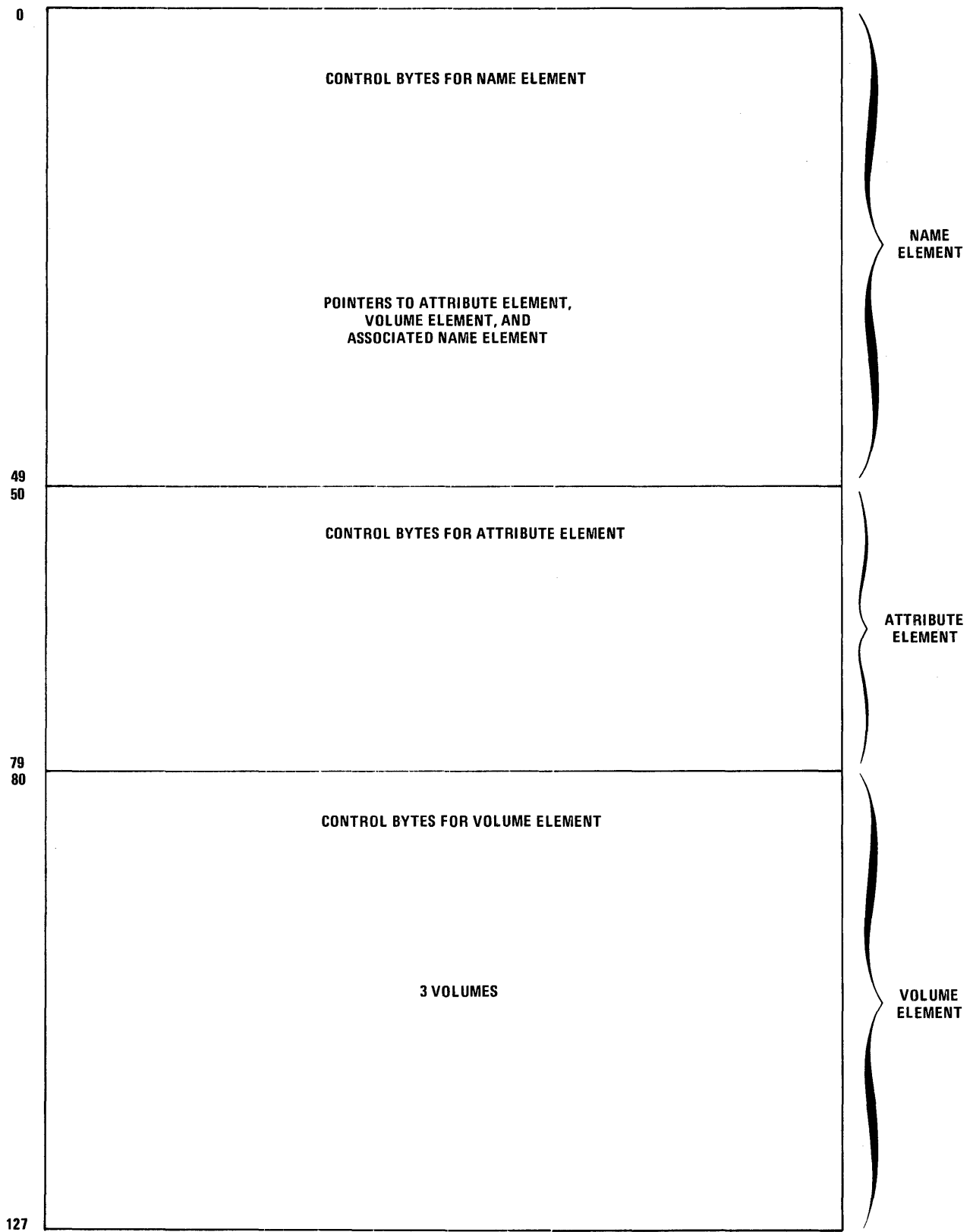


Figure A-7. Block 2-n – Normal File Entry in Central Catalog

0

CONTROL BYTES FOR VOLUME CONTINUATION

63

Figure A-8. Central Catalog Volume Element Continuation

B. SERVICE REQUEST MECHANISM AND MACRO EXPANSIONS

INTRODUCTION

There are essentially two types of system macro instructions: action macros and data macros. Action macros, which expand into executable code, may be further subdivided into user code macros and service request macros. User code macros expand into code which is primarily executed within the user's program (except for possible nested service request macros). Service request macros, on the other hand, link to system routines for their implementation.

Examples:

GET, DISPLAY user code macros
 READ, GETCOM service request macros
 MESSAGE, COMMAND data macros

Most of the control program and block input/output macros are service request macros whereas most of the physical input/output macros are data macros.

The expansions of all service request macros contain a service request machine instruction.

SERVICE REQUEST INSTRUCTION

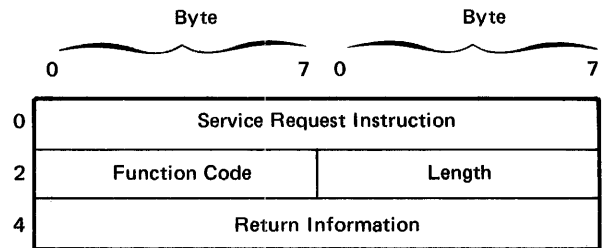
The *Service Request* instruction (mnemonic SR) is the only means by which a user program* may communicate with the operating system. In practice, however, there is no need for the user to code this instruction directly, since it will automatically be

*This same method is also often employed by system programs to communicate with each other.

generated in the expansion of a service request macro instruction. Figure B-1 illustrates the control path between a user program and operating system.

SERVICE REQUEST MACRO EXPANSION CONVENTIONS

The expansions of all* service request macros will normally begin with the following standard three words (with no LIST operand used):



Any additional words in the expansion are uniquely defined for each macro.

SERVICE REQUEST INSTRUCTION (BYTES 0 AND 1)

This instruction establishes the link with the operating system; it is defined as follows.

* Except for the physical input/output macro, EXCP.

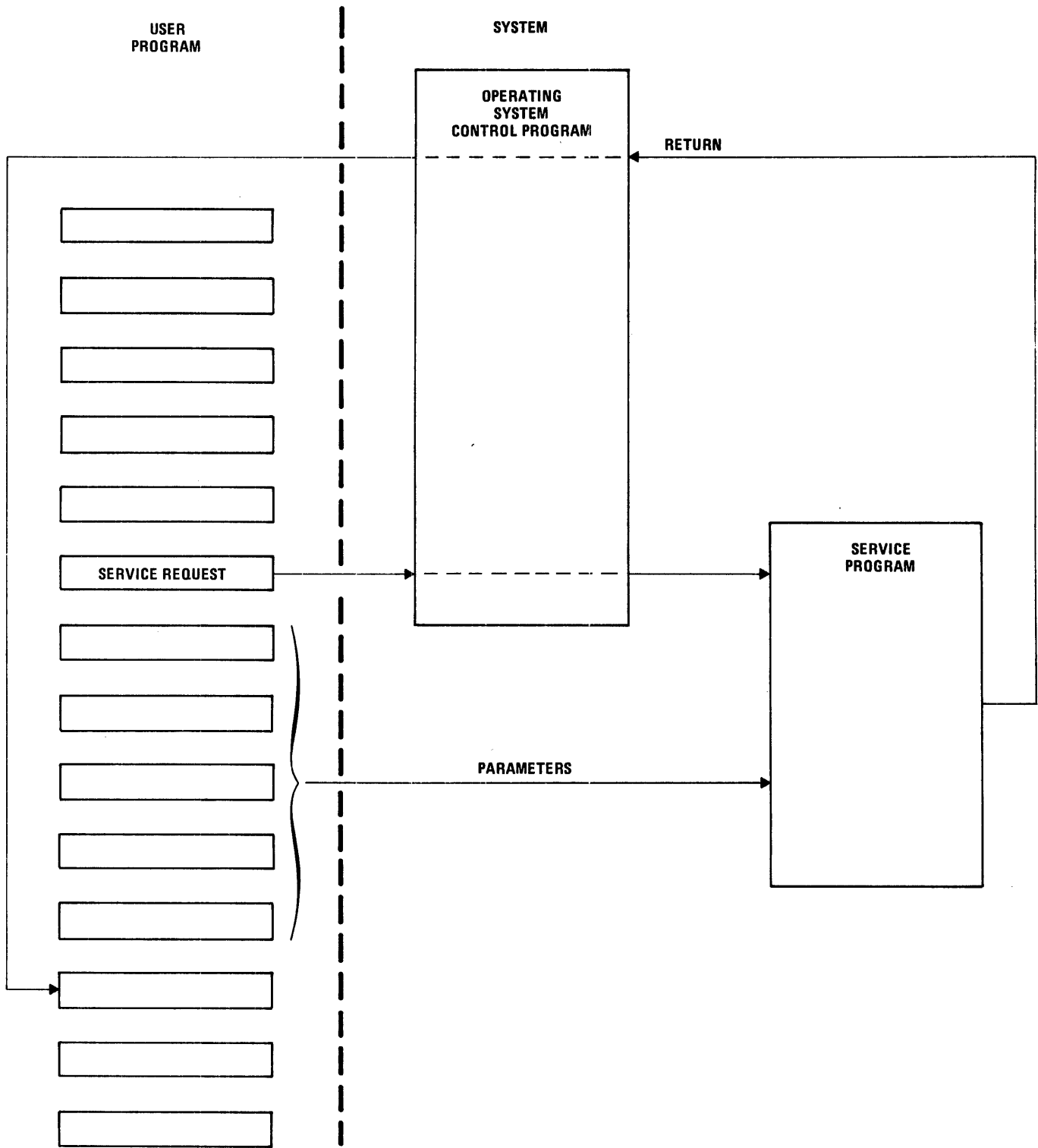
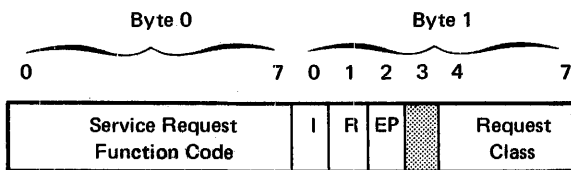


Figure B-1. Service Request Linkage to System



Service Request Function Code: SR = X'13'

- I:** parameter list location indicator
- 0 immediately following this instruction
 - 1 at location whose address is contained in register 6
- R:** Return indicator
- 0 return control only after request is completed
 - 1 return control immediately after request is accepted
- EP:** User error processing indicator
- 0 user program to be aborted on occurrence of an error
 - 1 user to retain control after an error
- Request Class:** Major class of service request
- 0 = Debug Request (system only)
 - 1 = Restricted Request (system only)
 - 2 = Control Program Request
 - 3 = Block Input/Output Request
 - 4 = Illegal
 - 5 = Supervisor Request
 - 6 = Telecommunications Request
 - 7 = Physical Input/Output Request

FUNCTION CODE (BYTE 2)

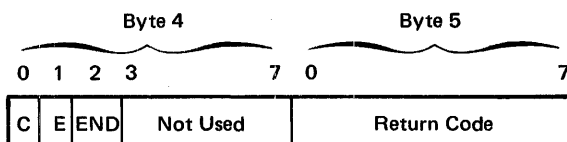
Function or operation code for this request (see individual expansions).

LENGTH (BYTE 3)

Number of words in this request block, excluding the service request instruction (see individual expansions).

RETURN INFORMATION (BYTES 4 AND 5)

This word is an area where the system will return information developed during the execution of the request. On initial receipt of the request the whole word will be set to zero. The return information field is defined as follows:



Complete Bit (C)

This bit will be set by the system to indicate that the request has been completed (normally or otherwise).

Error Flag (E)

When this bit is set an abnormal completion is indicated in which case the remainder of the word will contain information connected with the error. If the error flag is not set, no further information will be contained in this word (except in the case of END). This bit may be set on completion even if the EP bit is not set in the SR.

End Flag (END)

When set, this bit indicates that one of a number of possible end conditions has been detected. The particular end condition is specified by the return code in byte 5, as follows:

- 00 = Disc EOA (end of allocation)
- 01 = Printer Channel 12
- 02 = Disc EOF (end of file)
- 03 = Card Reader EOF
- 04 = Printer Channel 9
- 05 = Printer Channels 9 and 12

Return Code

This eight-bit code contains specific information about the error (return code, Table C-1, Appendix C).

OPTIONAL PARAMETERS

Three keyword parameters are usable by most block I/O level service request macros: RETURN, ERRCOMP, and LIST. The LIST parameter gives the option of separating the parameter packet and service request instruction, or not doing so.

If LIST=YES, the RETURN and ERRCOMP parameters cannot be used; if LIST=NO, the RETURN and ERRCOMP parameters are the only other parameters which can be used. However, if LIST is omitted, these two parameters can be used independently.

LIST

If LIST=YES, only the appropriate parameter packet is generated. The LIST=YES option allows the generation of the macro parameter packet only once in the program.

If LIST=NO, only a 2-byte standard service request instruction is generated. The user must load into general-purpose register 6 (R6) the address of the parameter packet to be referenced and into register 7 (R7) the address of the save area address prior to issuing the LIST=NO option.

The 2-byte service request instruction has the following format, where 05 defines the Data Management expansions and 13 the function code.



Example:

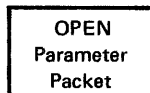
NAME	OPERATION	OPERAND
1 2 3 4 5 6 7 8 9	10 11 12 13 14 15 16 17 18	19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44
	LODD	SAVEIT, R7
	LODD	TAG5, R6
TAG6	OPEN	LIST=NO, RETURN=YES
	:	:
TAG5	OPEN	BUFFER=H,SKIP;
	:	IDENT=FILEB, IOTYP=B;
	:	LIST=YES, USAGE=I
SAVEIT	WDD	30

With the LODD instruction, the user loads into R6 the address of a parameter packet which has been specified elsewhere in the program (TAG 5), and the save area address (SAVEIT) into R7.

With the OPEN macro and a LIST=NO parameter specified, only the 2-byte executive request code is generated.

At TAG 5, the OPEN parameter list is generated.

If the LIST option is omitted, the service request instruction is generated followed by the appropriate parameter list. The user must load the address of the save area into R7 prior to issuing the call.



NAME	OPERATION	OPERAND
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
	LODD	SAVEIT, R7
TAG8	OPEN	IDENT=FILEB;
	:	BUFFER=H,SKIP;
	:	IOTYP=B, USAGE=I
SAVEIT	WDD	30

The LIST default for the OPEN request results in the generation of the service request instruction followed by the OPEN parameter packet.

RETURN

The RETURN parameter enables the user to request that control be returned to his program immediately after the service request has been received and recognized by the system (but before the function it requests has been completed). By coding RETURN=YES, the user may save time by proceeding with his own processing while the system is processing the service request. When RETURN=YES is used, detection of request completion becomes a user responsibility (he must check the *complete* bit in the return information word of the request parameter list). The RETURN parameter is mostly applicable to input/output service requests, which are relatively slow to complete. The default specification, RETURN=NO may be coded explicitly if desired. (Note that there are certain service request macros on which the RETURN parameter is not available.)

ERRCOMP

The ERRCOMP parameter allows the user to retain control when an error occurs which the system was unable to correct. By coding ERRCOMP=YES the user has the option of attempting to correct any errors or of simply ignoring* them and continuing. The default condition ERRCOMP=NO, which may be coded explicitly if desired, results in a program abort in the event of an error (which the system was unable to correct).

STANDARD SYSTEM SUFFIXES

If the programmer wishes to reference a particular word of a parameter packet, he may do so by adding the standard system suffix to his tag name (name symbol). If the tag name is six characters or less, the standard suffix is appended to the tag name. If the tag name is seven or eight characters, this (these) last character(s) is (are) truncated and the standard suffix is appended. For this reason the first six characters must be unique. The standard suffixes are listed alongside the macro expansion with the suffix across from the appropriate parameter packet word.

*Occasionally, a condition may arise which is classified by the system as an error but which does not affect the outcome of the user program.

For an example: if the programmer wishes to reference the word containing the file type of the ALLOC macro, it is located in byte 5 which is referenced by the F1 suffix. If his tag name is JOB65, the word may be referenced with JOB65F1. However, if his tag name is MYJOB658, the reference is MYJOB6F1.

EXPANSION TABLES

The expansion of each macro is detailed in the following pages; generally there is a map of the bytes as stored, followed by detailed assignments of bytes and bits.

The macros are listed in these groupings:

- Data Management

- ALLOCATE
- EXPAND
- PURGE
- OPEN
- CLOSE
- CLOVE
- LABRTN

- Block I/O

- STATUS/TYPE
- CONTRL/POSITN
- READ/WRITE
- RESET

- Physical I/O

- EXCP
- PCB
- COMMAND (various applications)

- Control Program

- WAIT
- POST/RPOST
- SETCOM/GETCOM
- ACCEPT
- DISPLAY
- MEMLIM
- INFORM
- SETIF
- DELAY
- HALT
- EHALT
- ABEND
- TIME
- SDATE
- JDATE

- Console Communications

- CONSOLE

These expansion explanations show the parameter tables or lists without the service request instruction (for those macros including a service request). Byte displacement shown at the side of the tables is without the service request; considering the service request, it is necessary to increase the displacement count by 2.

Where applicable, bytes containing Segment Tags are shown. This tag is mandatory in systems having more than 64K bytes of storage, and may be implemented in smaller systems. A programmer should be aware that the showing of Segment Tags in these format drawings does not necessarily imply that the tags will be in the system he is using.

Table B-1. ALLOCATE Parameter Packet Format

Suffix††

0	Request Code							Length							FL	
2	C	Return Information						Return Code							CR	
4	PF	UC	CS	CB	WV	CD						FT	FO		F1	
6	1	1	1	1	L1	L1	L2	L2	IA	IA	IB	IK	DC	IC	SF	F3
8	Data Block Count														BC	
10	Data Block Size														BS	
12	Data Record Size														RS	
14	Data Record Size														RS	
16								S TAG LAB1								
18	LAB1 ADR														L1	
20								S TAG LAB2								
22	LAB2 ADR														L2	
24								S TAG IDENT								
26	IDENT ADR														IA	
28	Index Block Size														IB	
30	Index Key Size														IK	
32	Data Cylinder Number														DC	
34	Index Cylinder Number														IC	
36	Spread Factor														SF	

Byte	Bit	Description
0		Request code, 1 designates ALLOCATE.
1		Length, number of words in the parameter packet (10 to 19 words)
2	0	Request complete indicator (C) 0 Service request in process 1 Request complete
	1-7	Return information
3		Return code
4	0	Paired file (PF). PF=1 for paired file.

† Bytes 6 and 7 reflect which optional parameters are present. If a parameter is not present, as evidenced by the presence of a zero in that parameter bit, the corresponding 2-byte word in bytes 16 through 37 is omitted, and those parameters that follow are telescoped upward by one word. Four-byte parameters such as IDENT ADR with its segment tag and reserved byte, is represented by two bits in the pattern. Both bits (IA) must be set if IDENT ADR is present.

†† The 2-character suffix for unique file identifier.

Table B-1. ALLOCATE Parameter Packet Format (Continued)

Byte	Bit	Description
	1	Uncataloged (UC). UC=1 for not recording file in the central catalog.
	2	Contiguous space (CS). CS=1 for contiguous space on each volume.
	3	Cylinder boundary (CB). CB=1 for space segments starting on cylinder boundary.
	4	Write verify (WV). WV=1 for write verification of files.
	5	Common stored data format (CD). CD=1 for common stored data format.
5	3,4	File type (FT) 00 Permanent 01 Scratch 10 Temporary 11 Work
	5-7	File organization (FO) 000 General 001 Indexed 010 Relative 100 Sequential
6	4,5	LAB1 ADR specified (L1=1)†
	6,7	LAB2 ADR specified (L2=1)†
7	0,1	IDENT ADR specified (IA=1)†
	2	Index block size specified (IB=1)
	3	Index key size specified (IK=1)
	4	Data cylinder number specified (DC=1)
	5	Index cylinder number specified (IC=1)
	6	Spread factor specified (SF=1)
8-11		Data block count, number of blocks for data file
12,13		Data block size, size (in bytes) of data block
14,15		Data record size, size (in bytes) of record
17*		Segment tag for LAB1 ADR (S TAG LAB1)
18,19*		LAB1 ADR, address of label information for the primary data file LABDEF1, required if IDENT ADR is not specified.

*Optional parameter.

†Two fields are shown on format expansion to accommodate paired files.

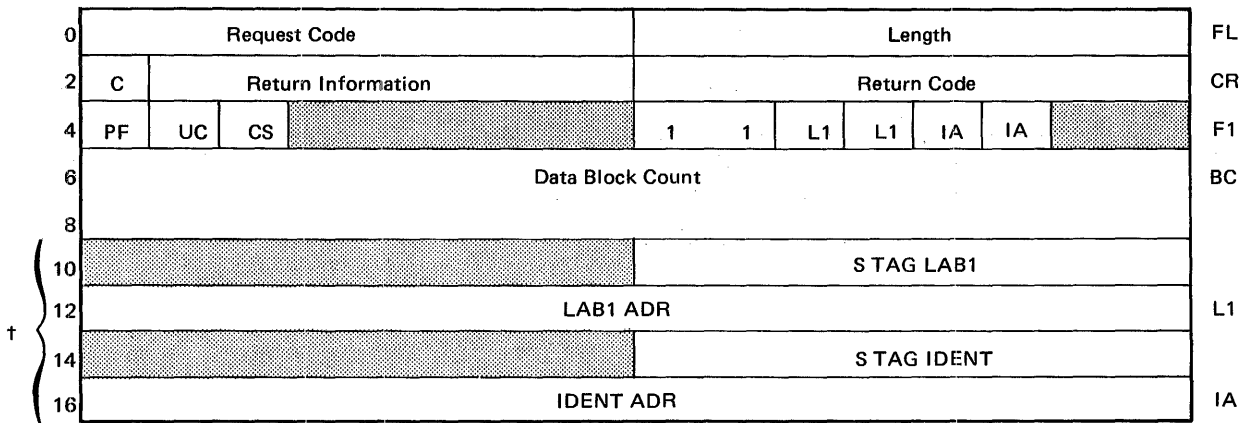
Table B-1. ALLOCATE Parameter Packet Format (Continued)

Byte	Bit	Description
21*		Segment tag for LAB2 ADR (S TAG LAB2)
22,23*		LAB2 ADR, address of label information for a paired file LABDEF2.
25*		Segment tag for IDENT ADR (S TAG IDENT)
26,27*		IDENT ADR, address of file identifier as used by Control Language to identify volumes for use in allocation
28,29*		Index block size, size (in bytes) of index file
30,31*		Index key size, size (in bytes) of key
32,33*		Data cylinder number, cylinder number at which allocation of the data file starts
34,35*		Index cylinder number, cylinder number at which allocation of index file starts
36,37*		Spread factor, physical record interlace to be used in an indexed data file

*Optional parameter

Table B-2. EXPAND Parameter Packet Format

Suffix††



Byte	Bit	Description
0		Request code, 2 designates EXPND
1		Length, number of words in parameter packet (7 to 9 words)
2	0	Request complete indicator (C) 0 Service request in process 1 Request complete
	1-7	Return information
3	0-7	Return code
4	0	Paired file (PF), PF=1 for paired file
	1	Uncataloged (UC), UC=1 for uncataloged file
	2	Contiguous Space (CS), CS=1 if space on each volume is to remain contiguous.
5	2,3	LAB1 ADR specified (L1=1)†
	4,5	IDENT ADR specified (IA=1)†
6-8		Data block count, number of data blocks to add to the file

† Byte 5 reflects which optional parameters are present. If a parameter is not present, the corresponding 4-byte field in bytes 10 through 17 is deleted from the parameter packet and the subsequent parameter words are moved upward. The 4-byte field is represented by two bits in byte 5; both bits must be set (or cleared) to reflect the presence of the field.

†† The 2-character suffix for unique file identifier.

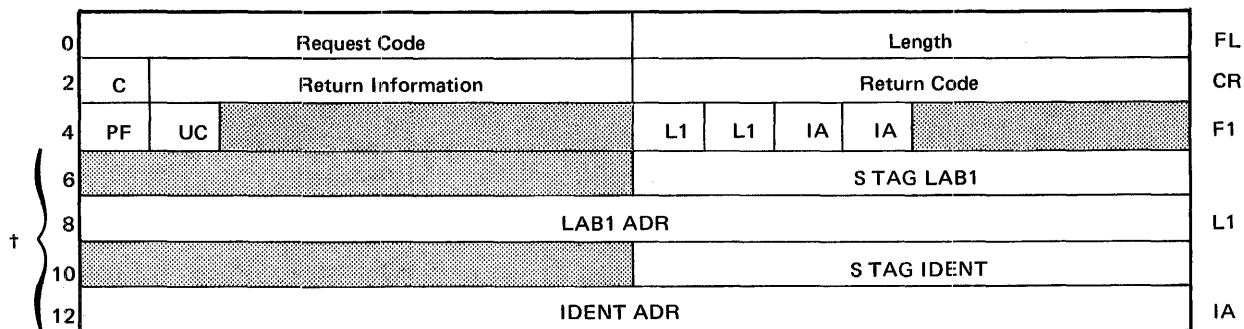
Table B-2. EXPAND Parameter Packet Format (Continued)

Byte	Bit	Description
11*		Segment tag for LAB1 ADR (S TAG LAB1)
12,13*		LAB1 ADR, address of label information for the file. This parameter is required if IDENT ADR is not specified.
15*		Segment tag for IDENT ADR (S TAG IDENT)
16,17*		IDENT ADR, address of file identifier as used by Control Language. This parameter is required if the file is open at the time of expansion.

*Optional parameter

Table B-3. PURGE Parameter Packet Format

Suffix††



Byte	Bits	Description
0		Request code, 3 designates PURGE
1		Length, number of words in parameter packet (5 to 7 words)
2	0	Request complete indicator (C) 0 Service request in process 1 Request complete
	1-7	Return information
3		Return code
4	0	Paired file (PF), PF=1 for paired files
	1	Uncataloged (UC), UC=1 for uncataloged file
5	0,1	LAB1 ADR specified (L1=1)
	2,3	IDENT ADR specified (IA=1)
7*		Segment tag for LAB1 ADR (S TAG LAB1)
8,9*		LAB1 ADR, address of label information for data file. This parameter is required if IDENT ADR is not specified.
11*		Segment tag for IDENT ADR
12,13*		IDENT ADR, address of file identifier as used by Control Language

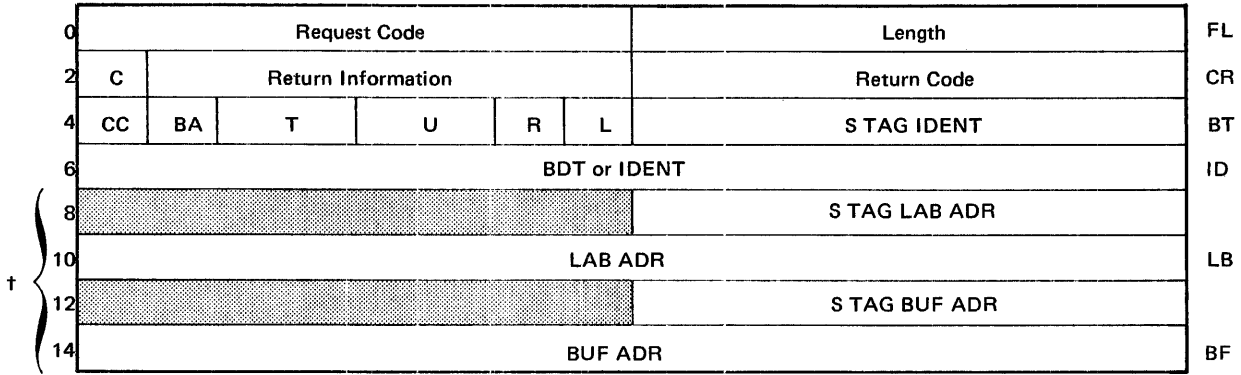
† Byte 5 reflects which optional parameters are present. If a parameter is not present, the corresponding 4-byte field in bytes 6 through 13 is deleted from the parameter packet and the subsequent parameter words are moved upward. The 4-byte field is represented by two bits in byte 5. Both bits must be set to reflect the presence of the field.

* Optional parameter.

†† The 2-character suffix for unique file identifier.

Table B-4. OPEN Parameter Packet Format

Suffix ††



Byte	Bit	Description
0		Request code, 4 designates OPEN
1		Length, number of words in parameter packet (4 to 8 words)
2	0	Request completion indicator (C) 0 Service request in process 1 Request complete
	1-7	Return information
3	0-7	Return code
4	0	Control character (CC) 0 ANSI control characters 1 Device control characters
	1	Buffer address (BA) 0 No buffer 1 Buffer present
	2-3	Type of I/O (T) 00 Logical 01 Block 10 Physical

† Byte 4 reflects which optional parameters are present. If a parameter is not present, the corresponding 2-byte word in bytes 8 through 15 is deleted from the parameter packet and the subsequent parameter words are moved.

†† The 2-character suffix for unique file identifier.

Table B-4. OPEN Parameter Packet Format (Continued)

Byte	Bit	Description
	4-5	Usage (U) 00 Input 01 Update 10 Output
	6	Rewind (R) 0 No rewind 1 Rewind
	7	Label (L) 0 No label 1 Label address
5		Segment tag for BDT or IDENT (S TAG IDENT)
6,7		BDT or IDENT, address of BDT or address of file identifier
9*		Segment tag for LAB ADR (S TAG LAB ADR)
10,11*		LAB ADR, label address
13*		Segment tag for BUF ADR (S TAG BUF ADR)
14,15*		BUF ADR, buffer address

*Optional parameter.

Table B-5. CLOSE Parameter Packet Format

Suffix

0	Request Code					Length		FL
2	C	Return Information				Return Code		CR
4		L	T		R		S TAG IDENT	BT
6	BDT or IDENT							ID

Byte	Bit	Description
0		Request code, 5 designates CLOSE
1		Length, number of words in parameter packet (4 words)
2	0	Request complete indicator (C) 0 Service request in process 1 Request complete
1-7		Return information
3		Return code
4	2	Lock (L) 0 No lock 1 Lock
	3,4	Type of I/O (T) 00 Logical I/O 01 Block I/O 10 Physical I/O
	6	Rewind (R) 0 No rewind 1 Rewind
5		Segment tag for IDENT or BDT (S TAG IDENT)
6,7		BDT or IDENT, address of BDT or address of file identifier

†The 2-character suffix for unique file identifier.

Table B-6. CLOVE Parameter Packet Format

Suffix

0	Request Code		Length		FL	
2	C	Return Information		Return Code	CR	
4	T		S TAG IDENT			
6	BDT or IDENT					ID

Byte	Bit	Description
0		Request code, 6 designates CLOVE
1		Length, number of words in parameter packet (4 words)
2	0	Request complete indicator (C) 0 Service request in process 1 Request complete
	1-7	Return information
3		Return code
4	3,4	Type of I/O (T) 00 Logical I/O 01 Block I/O
5		Segment tag for BDT or IDENT (S TAG IDENT)
6,7		BDT or IDENT, address of BDT or address of file identifier

†The 2-character suffix for unique file identifier.

Table B-7. LABRTN Parameter Packet Format

Suffix†

0	Request Code		Length	FL
2	C	Return Information	Return Code	CR
4	ELEM		S TAG IDENT	BT
6	IDENT ADR			ID
8			S TAG BUFF	
10	BUFF ADDRESS			IN

Byte	Bit	Description
0		Request code, 7 designates LABRTN
1		Length, number of words in parameter packet (6 words)
2	0	Request completion indicator (C) 0 Service request in process 1 Request complete
	1-7	Return information
3		Return code
4	5-7	Element field (ELEM) 000 Name element returned 001 Attribute element returned 010 Both elements returned
5		Segment tag for IDENT (S TAG IDENT)
6,7		IDENT ADR, address of file identifier
9		Segment tag for buffer address (S TAG BUFF)
10,11		Address of buffer containing status information

†The 2-character suffix for unique file identifier.

Table B-8. STATUS and TYPE Macros Parameter Packet Format

Suffix*

0	Request Code	Length	FL
2	Return Information		CR
4		S TAG	
6	Pointer to File Identifier		ID
8		S TAG	
10	Buffer Address		BA
12		S TAG	
14	Pointer to Byte Counter		BS

Byte	Bit	Description
0	0-7	Request Code: STATUS=40, TYPE=41)
1	0-7	Length, number of words in parameter: 08
2, 3	0-15	Return Information
4	0-7	Not used
5	0-7	Segment Tag
6, 7	0-7	Pointer to file ID: Address of 8-byte file identifier
8	0-7	Not used
9	0-7	Segment Tag
10, 11	0-7	Buffer Address: first byte address of user-specified buffer area
12	0-7	Not used
13	0-7	Segment Tag
14, 15	0-7	Pointer to Byte Count: address of 2-byte area containing byte count (length of buffer)

*The 2-character suffix for the unique file identifier.

Table B-9. CNTRL/POSITN Macros Parameter Packet Format

Suffix*

0	Request Code	Length	FL
2	Return Information		CR
4	Sub-Function Code	S TAG	SF
6	Pointer to File ID		ID
8		S TAG	
10	Pointer to Block Number		BN
12	Residual Byte Count		RC

Byte	Bit	Description
0	0-7	Request Code: CNTRL=03,POSITN=04
1	0-7	Length, number of words in the parameter packet, 07
3	0-7	Return Information
4	0-7	Sub-Function Code (the indicated bits are set for the function):
	7	Select Stacker
	6	Space Line Printer
	5	Skip Line Printer Form
	0-2	Rewind (000)
		Rewind and Unload (001)
		Erase gap (010)
		Backspace Record (100)
		Forward space a record (110)
		Forward space a file (111)
		Backspace a file (101)
		Write End-of-File mark (011)
	0-3	The value of nn when SS, SK, or SP is used
	7	Beginning of volume
	6	End of Volume
	0	SEEK on position
5	0-7	Segment Tag
6,7	0-7	Pointer to File ID: address of 8-byte file identifier in FDT
	0-7	
8	0-7	Not used
9	0-7	Segment Tag
10,11	0-7	Pointer to Block No.: address of 4-byte block number (applies to POSITN)
	0-7	
12,13	0-7	Residual Byte Count: difference between number of bytes requested and number of bytes received.
	0-7	

*The 2-character suffix for unique file identifier.

Table B-10. READ/WRITE Macros Parameter Packet Format

Suffix*

0	Request Code	Length	FL
2	Return Information		CR
4	Sub-Function Code	S TAG	SF
6	Pointer to File ID		ID
8		S TAG	
10	Pointer to Block Number		BN
12	Residual Byte Count		RC
14		S TAG	
16	Buffer Address		BA
18		S TAG	
20	Pointer to Byte Count		BS

Byte	Bit	Description
0	0-7	Function Code: READ=02, WRITE=01
1	07	Length, number of words in the parameter packet: 11
2,3	0-7 0-7	Return Information
4	0-7 7 6 5 4 0-3 0-2	Sub-Function Code (the indicated bits are set for the function): Select Stacker Space Line Printer Skip Line Printer Form EBCDIC=NO The value of nn when SK or SP is used The value of nn when SS is used
5	0-7	Segment Tag
6,7	0-7 0-7	Pointer to File ID: address of 8-byte file identifier
8	0-7	Not used
9	0-7	Segment Tag
10,11	0-7 0-7	Pointer to Block No.: address of 4-byte area containing block number
12,13	0-7 0-7	Residual Byte Count: difference between number of requested bytes and number of bytes transferred
14	0-7	Not used
15	0-7	Segment Tag

*The 2-character suffix for unique file identifier.

Table B-10. READ/WRITE Macros Parameter Packet Format (Continued)

Byte	Bit	Description
16,17	0-7 0-7	Buffer Address: first byte address of user specified buffer area
18	0-7	Not used
19	0-7	Segment Tag
20,21	0-7 0-7	Pointer to Byte Count: address of 2-byte area containing the byte count (length of buffer)

Table B-11. RESET Macro Parameter Packet Format

Suffix*

0	Request Code	Length	FL
2	Return Information		CR
4		S TAG	
6	Pointer to File ID		ID

Byte	Bit	Description
0	0-7	Function Code: RESET=31
1	0-7	Length, number of words in the parameter packet: 04
2, 3	0-7	Return Information
4	0-7	Not used
5	0-7	Segment Tag
6, 7	0-7	Pointer to File ID: address of 8-byte file identifier

*The 2-character suffix for unique file identifier.

Table B-12. EXCP Macro Expansion

<p>The expansion of the EXCP macro does not include a parameter table (LIST=NO is implied) but it may, depending on the operands coded in the call, contain executable code for request setup purposes:</p>	
No operands:	SR
PCB=@register number:	MOVR Rn,6 SR
PCB=symbolic address:	LOD symbolic address,6 SR
CP=@register number:	STO 6(6),Rn SR
CP=symbolic address:	MOVM symbolic address, 6(6) SR
UNORD=symbolic address:	MOVB symbolic address, 0(6) SR
UNORD=@register number:	STOB 0(6),Rn SR

Table B-13. PCB Macro Parameter Packet Format

0*	Unit Ordinal	Length
2	Return Information	
4	ERP	CP
6	Pointer to CP or FC	
8	Pointer to Current CW	
10	Status	
12	Residual Byte Count	
14	Sense	
16	Sense	
18	Sense	

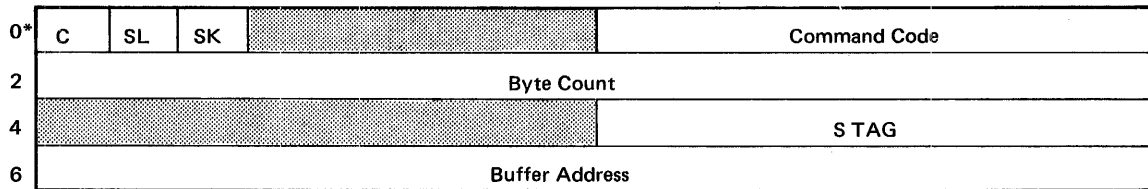
*This macro does not use the standard service request prefix.

Byte	Bit	Description
0	0-7	Unit Ordinal: Device identifier
1	0-7	Length, number of words in this block excluding Device Ident field
2,3	0-7 0-7	Return Information (same as for service request macros)
4	0	Error Recovery Flag (ERP) 0 = Call system error recovery when an error occurs 1 = Bypass error recovery
	1	Not used
	2	Command Program Flag 0 = Next word is address of command program to be executed 1 = Next word is function code to be executed
	3-7	Not used
5	0-7	Not used
6,7	0-7 0-7	Pointer to CP or FC If CP=0, pointer to command program If CP=1, pointer to function code
8,9	0-7 0-7	Pointer to Current CW: Address of command word being executed
10,11	0-7 0-7	Status indication

Table B-13. PCB Macro Parameter Packet Format (Continued)

Byte	Bit	Description
12,13	0-7 0-7	Residual Byte Count: difference between number of bytes requested and number of bytes transferred
14,15, 16,17, 18,19		Sense bytes: variable number of sense bytes depending on device Disc = none Unit Record = 2 Mag Tape = 6

Table B-14. COMMAND Macro Parameter Packet Format (Basic Data Channel)



*This macro does not use the standard service request prefix.

Byte	Bit	Description
0	0	Chain Flag (C) 0 = No CW follows this one 1 = Another CW follows
	1	Suppress Length Check (SL): if it is set, the CP will not be terminated when the transferred byte count differs from the specified byte count. Not used when C = 0.
	2	Skip (SK): If it is set, data will not be transferred to memory on a data input operation.
	3-7	Not used
	8-15	Command Code: The eight-bit hardware command code (OPCODE).
2, 3	0-7	Byte Count: The length of the data buffer (BUFSIZ) in hexadecimal; length must be non-zero
4	0-7	Not used
5	0-7	Segment Tag
6, 7	0-7	Buffer Address: the FBA address of the data buffer (BUFADR).

Table B-15. COMMAND Macro Parameter Packet Format (DCABLE)

0*	C		2	1
2	0	0	0	2
4			S TAG	
6	Buffer Address			

*This macro does not use the standard service request prefix.

Byte	Bit	Description
0	0	Chain Flag 0 = No CW follows this one 1 = Another CW follows
	1-7	Not used
1	0-7	Command Code: DCABLE=21
2, 3	0-7	Byte Count: DCABLE=0002
4	0-7	Not used
5	0-7	Segment Tag
6, 7	0-7	Buffer Address: address of buffer into which the disc drive cable address is to be transferred.

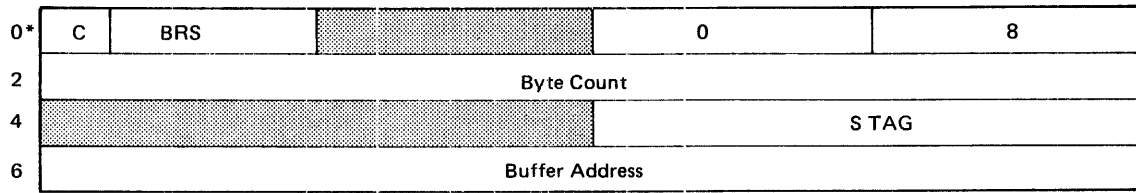
Table B-16. COMMAND Macro Parameter Packet Format (DCSEEK)

0*	C		2	0
2	0	0	0	4
4			STAG	
6	Buffer Address			

*This macro does not use the standard service request prefix.

Byte	Bit	Description
0	0	Chain Flag 0 = No CW follows this one 1 = Another CW follows
	1-7	Not used
1	0-7	Command Code: DCSEEK=20
2, 3	0-7	Byte Count: DCSEEK=0004
4	0-7	Not used
5	0-7	Segment Tag
6, 7	0-7	Buffer Address: address of buffer containing cylinder and track numbers.

Table B-17. COMMAND Macro Parameter Packet Format (DCSRCH)

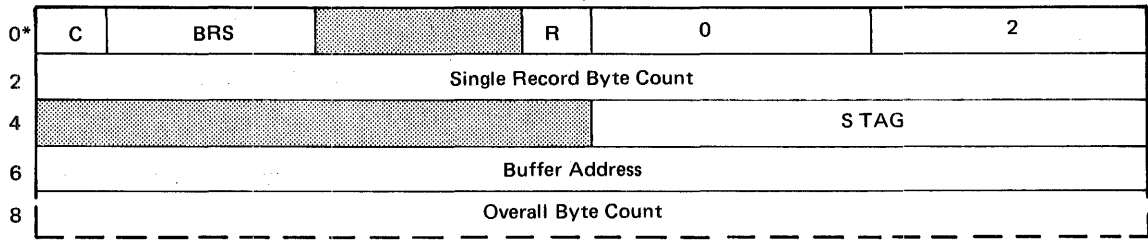


*This macro does not use the standard service request prefix.

Byte	Bit	Description
0	0	Chain Flag 0 = No CW follows this one 1 = Another CW follows
	1-3	Bit Ring Sync: least-significant three bits of Bit Ring Sync Code 101 = home address field 011 = RO count field 110 = RN count field
1	0-7	Command Code: DCSRCH=08
2, 3	0-7	Byte Count: Length of field for which search is being made BC = 0005 for home address BC = 0009 for count field
4	0-7	Not used
5	0-7	Segment Tag
6, 7		Buffer Address: address of buffer containing field data for which search is being made

Table B-18. COMMAND Macro Parameter Packet Format (DCREAD)

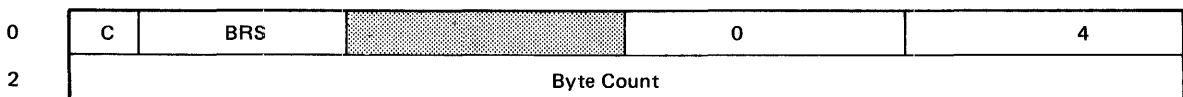
(1) Read with Transfer



*This macro does not use the standard service request prefix.

Byte	Bit	Description
0	0	Chain Flag 0 = No CW follows this one 1 = Another CW follows
	1-3	Bit Ring Sync: least-significant three bits of the Bit Ring Sync code: 001 = data field 010 = key field 011 = R _O count field 110 = R _n count field 101 = home address field 100 = control storage data
	4-6	Not used
	7	Repeat Flag 0 = Execute read command once 1 = repeat read command until overall byte count is satisfied
1	0-7	Command Code: READ WITH TRANSFER = 02
2, 3	0-7	Single Record Byte Count: number of bytes to be transferred in each execution of the read command. If R=0 this will be the total number of bytes transferred.
4	0-7	Not used
5	0-7	Segment Tag
6, 7	0-7	Buffer Address: address of the buffer into which the data is to be placed.
8, 9	0-7	Overall Byte Count: total number of bytes to be transferred. Only present when R=1 for a multi-record read.

(2) Read Without Transfer



Command Code = 04 (Bits 0-7 of byte 1)

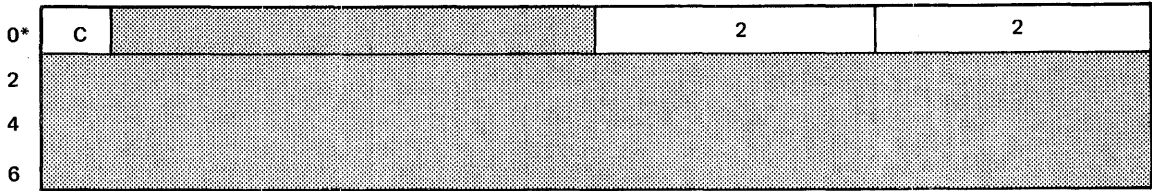
Table B-19. COMMAND Macro Parameter Packet Format (DCWRIT and DCFWRIT)

0*	C	BRS		Command Code
2	Byte Count			
4	Gap Length		S TAG	
6	Buffer Address			

*This macro does not use the standard service request prefix.

Byte	Bit	Description
0	0	Chain Flag 0 = No CW follows this one 1 = Another CW follows
	1-3	Bit Ring Sync: least-significant three bits of the Bit Ring Sync Code 001 = Data field 010 = Key field 011 = R ₀ count field 110 = R _n count field 101 = Home address field 100 = Control Storage data
	4-7	Not used
1	0-7	Command Code 01 = Write 10 = Format Write
2, 3	0-7	Byte Count: number of bytes of data to be written (length of buffers).
4	0-7	Gap Length: one's complement of the length of the gap to be written preceding the field itself.
5	0-7	Segment Tag
6, 7	0-7	Buffer Address: address of the buffer containing the data to be written

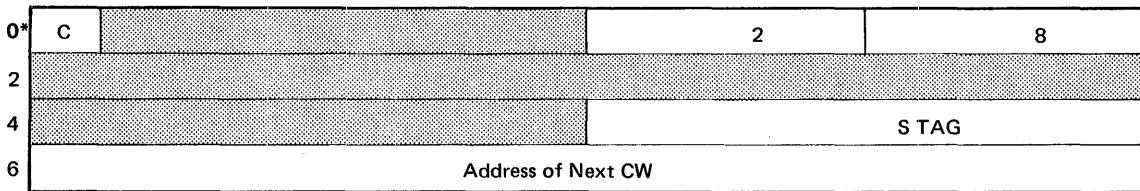
Table B-20. COMMAND Macro Parameter Packet Format (RESTORE)



*This macro does not use the standard service request prefix.

Byte	Bit	Description
0	0	Chain Flag 0 = No CW follows this one 1 = Another CW follows
	1-7	Not used
1	0-7	Command Code: RESTORE = 22
2-6	-	Not used

Table B-21. COMMAND Macro Parameter Packet Format (DCJUMP)



*This macro does not use the standard service request prefix.

Byte	Bit	Description
0	0	Chain Flag 0 = No CW follows this one 1 = Another CW follows
	1-7	Not used
1	0-7	Command Code: DCJUMP = 28
2-4	0-7	Not used
5	0-7	Segment Tag
6,7	0-7	Next CW Address: locations of the CW to be executed following this one

Table B-22. WAIT Macro Parameter Packet Format

Suffix*

0	Request Code	Length	FL
2	Return Information		CR
4	C	L A	BT
6	Request Block Address (Optional)		BF

Byte	Bit	Description
0	0-7	Request Code: WAIT=03
1	0-7	Length, number of words in the parameter packet: 3 or 4
2, 3	0-7	Return Information
4	0-3	Not used
	4	Count Flag: C=1 means wait for any request with count option
	5	Not used
	6	Wait - all flag (L) L=1 means wait for all outstanding requests.
	7	Wait - any flag (A) A=1 means wait is for any outstanding request.
5	0-7	Segment Tag
6, 7	0-7	Request Block Address: address of request block for wait-specific. Only present when L=0 and A=0.

When L=0 and A=0, wait is for specific request.

*The 2-character suffix for unique file identifier.

Table B-23. POST/RPOST Macros Parameter Packet Format

Suffix*

0	Request Code	Length	FL
2	Return Information		CR
4		S TAG	
6	Buffer Address		IN

Byte	Bit	Description
0	0-7	Request Code: POST=43,RPOST=44
1	0-7	Length, number of words in the parameter packet: 04
2, 3	0-7	Return Information
4	0-7	Not used
5	0-7	Segment Tag
6, 7	0-7	Buffer Address: address of an 8-byte buffer. This will either contain the information to be posted (POST) or will receive the interpreted information from the system (RPOST).

*The 2-character suffix for unique file identifier.

Table B-24. SETCOM/GETCOM Macros Parameter Packet Format

Suffix*

0	Request Code	Length	FL
2	Return Information		CR
4		S TAG	
6	Buffer Address		IN

Byte	Bit	Description
0	0-7	Request Code: SETCOM=47,GETCOM=48
1	0-7	Length, number of words in the parameter packet: 04
2, 3	0-7	Return Information
4	0-7	Not used
5	0-7	Segment Tag
6, 7	0-7	Buffer Address: address of an 8-byte buffer. This will either contain the information to be transferred (SETCOM) or will receive information from the system (GETCOM).

*The 2-character suffix for unique file identifier.

Table B-25. ACCEPT Macro Parameter Packet Format

Suffix**

0	Request Code	Length	FL		
2	Return Information		CR		
4	N	S	H	STAG	BT
6	Buffer Address		BF		
8	S TAG		EN		
10	End Address		EN		
12	S TAG		IN		
14	Number Address (Optional)		IN		

Byte	Bit	Description
0	0-7	Request Code: 13
1	0-7	Length, number of words in the parameter packet: 6 or 8
2, 3	0-7	Return Information
4	0-4	Not used
	5	Number Flag 0 = number not specified (current value of //PAR card pointer will be used) 1 = //PAR card number is specified in word at displacement 16
	6	Strip Flag 0 = do not strip sequence number field (card cols 73-80) 1 = strip sequence number field
	7	Control Header Flag 0 = do not include CSDF control header 1 = include CSDF header
5	0-7	Segment Tag
6, 7	0-7	Buffer Address: address of buffer into which data from //PAR card is to be transferred.
8	0-7	Not used
9	0-7	Segment Tag
10,11	0-7	End Address: address to which program control will be transferred when all //PAR cards have been read.
14*	0-7	Not used (optional)
15*	0-7	Segment Tag (optional)
16*,17*	0-7	Number Address (optional): address of one-word location containing the //PAR card number

*Last four bytes will only be present when number flag (N) = 1.

**The 2-character suffix for unique file identifier.

Table B-26. MEMLIM Macro Parameter Packet Format

Suffix*

0	Request Code	Length	FL
2	Return Information		CR
4		S TAG	
6	Buffer Address		IN

Byte	Bit	Description
0	0-7	Request Code: MEMLIM=46
1	0-7	Length, number of words in the parameter packet: 04
2,3	0-7	Return Information
4	0-7	Not used
5	0-7	Segment Tag
6,7	0-7	Buffer Address: Address of buffer into which last page address is to be placed.

*The 2-character suffix for unique file identifier.

Table B-27. INFORM Macro Parameter Packet Format

Suffix*

0	Request Code	Length	FL
2	Return Information		CR
4		S TAG	
6	Count Address		IN

Byte	Bit	Description
0	0-7	Request Code: INFORM=05
1	0-7	Length, number of words in the parameter packet: 04
2, 3	0-7	Return Information
4	0-7	Not used
5	0-7	Segment Tag
6,7	0-7	Count address: address of a one-word location containing a count of the number of requests not known to have completed at this time.

*The 2-character suffix for unique file identifier.

Table B-28. SETIF Macro Parameter Packet Format

Suffix*

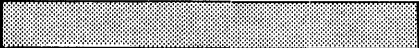
0	Request Code	Length	FL
2	Return Information		CR
4		S TAG	
6	Buffer Address		IN

Byte	Bit	Description
0	0-7	Request Code: SETIF=45
1	0-7	Length, number of words in the parameter packet: 04
2,3	0-7	Return Information
4	0-7	Not used
5	0-7	Segment Tag
6,7	0-7	Buffer Address: address of a one-byte buffer containing the data to be transferred.

*The 2-character suffix for unique file identifier.

Table B-29. DELAY Macro Parameter Packet Format

Suffix**

0	Request Code	Length	FL
2	Return Information		CR
4		B T S TAG	BT
6	Delay Address		IN

Byte	Bit	Description
0	0-7	Request Code: DELAY=04
1	0-7	Length, number of words in the parameter packet: 04
2,3	0-7	Return Information
4	0-5	Not used
	6	Break Flag (B) 0 = no delay break 1 = delay to be broken on any service request completion
	7	Type Flag (T) 0 = delay in seconds 1 = delay in cycles*
5	0-7	Segment Tag
6,7	0-7	Delay Address: address of a one-word buffer containing the duration of the delay in seconds or cycles* (see Type Flag).

*Cycle equals 50.2 milliseconds.

**The 2-character suffix for unique file identifier.

Table B-30. HALT Macro Parameter Packet Format

Suffix*

0	Request Code	Length	FL
2	Return Information		CR

Byte	Bit	Description
0	0-7	Request Code: HALT=64
1	0-7	Length, number of words in the parameter packet: 02
2,3	0-7	Return Information

*The 2-character suffix for unique file identifier.

Table B-31. EHALT Macro Parameter Packet Format

Suffix*

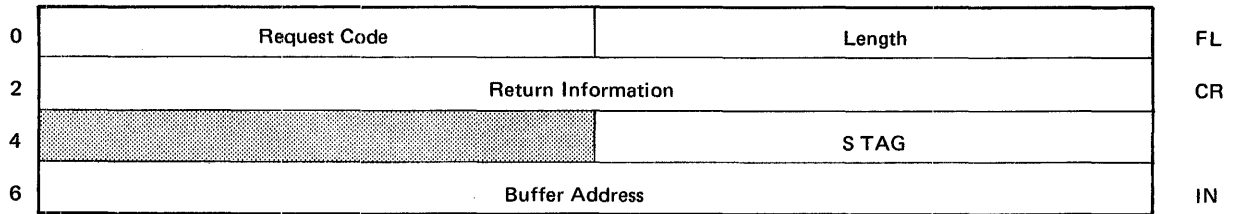
0	Request Code	Length	FL
2	Return Information		CR

Byte	Bit	Description
0	0-7	Request Code: EHALT=65
1	0-7	Length, number of words in the parameter packet: 02
2,3	0-7	Return Information

*The 2-character suffix for unique file identifier.

Table B-32. ABEND Macro Parameter Packet Format

Suffix*

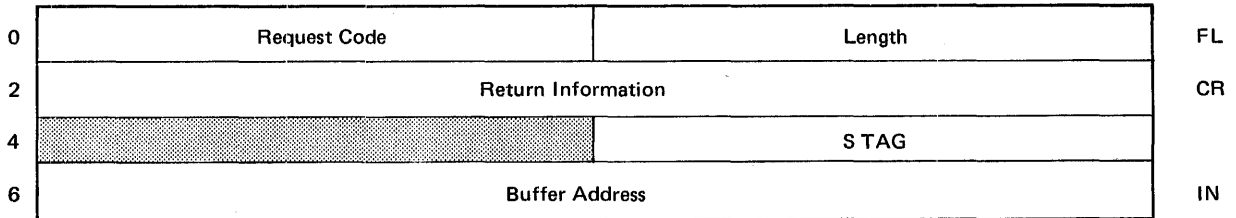


Byte	Bit	Description
0	0-7	Request Code: ABEND=75
1	0-7	Length, number of words in the parameter packet: 04
2,3	0-7	Return Information
5	0-7	Segment Tag
6,7	0-7	Buffer Address: address of location containing the completion code.

*The 2-character suffix for unique file identifier.

Table B-33. TIME Macro Parameter Packet Format

Suffix*



Byte	Bit	Description
0	0-7	Request Code: TIME=73
1	0-7	Length, number of words in the parameter packet: 04
2,3	0-7	Return Information
5	0-7	Segment Tag
6,7	0-7	Buffer Address: address of location containing the time

*The 2-character suffix for unique file identifier.

Table B-34. SDATE Macro Parameter Packet Format

Suffix*

0	Request Code	Length	FL
2	Return Information		CR
4	D	S TAG	BT
6	Buffer Address		IN

Byte	Bit	Description
0	0-7	Request Code: SDATE=74
1	0-7	Length, number of words in the parameter packet: 04
2,3	0-7	Return Information
4	7	Date Flag 0 = Calendar date 1 = Julian date
5	0-7	Segment Tag
6,7	0-7	Buffer Address: address of location containing the date

*The 2-character suffix for unique file identifier.

Table B-35. JDATE Macro Parameter Packet Format

Suffix*

0	Request Code	Length	FL
2	Return Information		CR
4	D	S TAG	BT
6	Buffer Address		IN

Byte	Bit	Description
0	0-7	Request Code: JDATE=66
1	0-7	Length, number of words in the parameter packet: 04
2,3	0-7	Return Information
4	7	Date Flag 0 = Calendar date 1 = Julian date
5	0-7	Segment Tag
6,7	0-7	Buffer address: address of location containing the date

*The 2-character suffix for unique file identifier.

Table B-36. CONSOLE Macro Parameter Packet Format

Suffix*

0	Request Code	Length	FL
2	Return Information		CR
4		S TAG	
6	Buffer Address		BF
8		S TAG	
10	Reply Address		IN

Byte	Bit	Description
0	0-7	Request Code: CONSOLE=01
1	0-7	Length, number of words in the parameter packet: 04 (without reply) or 06 (with reply)
2,3	0-7	Return Information
5	0-7	Segment Tag
6,7	0-7	Buffer Address: address of location containing message
9	0-7	Segment Tag
10,11	0-7	Reply Address: address of location containing reply message.

*The 2-character suffix for unique file identifier.

C. ERROR RECOVERY

INTRODUCTION

Of the various classes of possible errors associated with a service request, the operating system's error recovery program is concerned with only one particular class: the input/output hardware malfunctions. Attempted recovery from these input/output hardware malfunctions will be performed for all levels of input/output coding by default; however, at the physical level it may be bypassed if specified.

If an error proves to be irrecoverable, information describing the error is returned to the user's request or command block (Table C-1) and then, normally, the job is aborted. However, if the operand, ERRCOMP=YES had been coded in the original request, control is returned so that the user may either ignore the error or attempt to process it himself.

In addition to its basic task of attempting to recover from errors, the error recovery program is also responsible for handling certain exceptional non-error situations, such as EOF detection and indication.

TYPES OF ERROR

The following major categories of error are handled by the error recovery program.

INTERVENTION REQUIRED

When operator intervention is required, a console message is issued. Once the operator has serviced the device, he responds to the message indicating either to continue processing or to return control to the user. The MRX/OS Messages manual contains the operator console messages.

ERRORS REQUIRING RETRIES

Input/output requests completed but including specific types of errors are retried a given number of times. If the error persists, a message is typed on the operator's console. The operator response indicates either to retry the request another specific number of times or to return the error condition indication to the user program. If one of the retries succeeds, a *normal completion* indication is returned to the user program as though no error had occurred.

CONDITIONS OF UNCERTAINTY

There are certain error conditions where the error recovery program is unable to perform accurate error recovery, or any error recovery at all; for example, not being sure where magnetic tape is positioned.

Although the error recovery program checks all defined hardware conditions, it is not inconceivable that an undefined condition from a non-supported hardware device might arise; in this case the error recovery program is unable to process it.

With regard to the first example, it should be noted that although the error recovery program checks all defined hardware conditions, it is not inconceivable that an undefined condition might arise; in this case the error recovery program is unable to process it.

When error conditions of this kind occur, a message is typed on the operator's console. If the message requires a response and the operator chooses to continue, the user may either gain or lose information. If the message does not require a response, the request is routed back to the user with a return code.

IMMEDIATE IRRECOVERABLE CONDITIONS

An example of an error so classified is that of magnetic tape running off the end of the reel. In situations such as this, a console message is typed and the request is routed back to the user with a return code.

CONDITIONS THAT DO NOT REQUIRE RECOVERY

Certain conditions routed to the error recovery program are considered to be normal completions and do not require the issuance of any return codes or error messages. Two examples are these:

- a parity error while spacing magnetic tape (no recovery is needed because tape is properly positioned)
- detection of an end-of-file condition

ERROR LOGGING

Error logging is an integral part of the operating system. The error log file, which is built at system generation time, consists of forty-byte records (Table C-2).

Records of error conditions and retries are maintained in the two error flag bytes and two error counters, respectively (located in the extension area of each unit table entry). The bits of the error flag bytes represent the error conditions that can occur. If multiple errors occur, bits are set as the error conditions are detected.

The error counters are incremented by 1 whenever a retry error condition occurs. Each error counter can contain more than one error condition.

Table C-1. I/O Error Recovery Information

Condition	Hexadecimal Status Completion Code	FDT Return Information	PCB/Request Block Return Information Field	Message Code Displayed On Console	Console Message Reply	No. of Retries*	Must File be Reset	Error Log Record Written
EOA on this disc request	0	0	A000	—	—	0	N	
Channel 12 on the printer carriage control tape	0	0	A001	—	—	0	N	N
This request was not processed because an exception condition occurred on a previous request and the file has not been reset	—	Old	C001	—	—	0	Y	N
Disc EOF	0	0002	A002	—	—	0	Y	N
Unit down	5002	0002	C002	002	—	0	Y	N
Card reader EOF	0	0003	A003	—	—	0	Y	N
Invalid function code in command program	5003	—	C003	—	—	0	—	N
Channel 9 on the printer carriage control tape	0	0	A004	—	—	0	N	N
A Remove request removed this request	5004	—	C004	—	—	0	—	N
Length error. A record was read which was longer than the buffer space provided for it	0	0	C005	—	—	0	N	N
Tape mark sensed on any Read operation not including Search command	0	0006	A006	—	—	0	Y	N
An ASKATT request is being rejected because an ASKATT is already pending against the device	5006	—	C006	—	—	0	—	N
Error status indication returned from an I/O operation	5010	0010	C010	—	—	0	—	N
Operation timed out	5011	0011	C011	011	N/Y	1	Y	Y
Unsolicited attention set	5012	0012	C012	012	—	1	Y	N
Bad I/O status indication from Seek or Restore	5013	0013	C013	—	—	0	Y	N
No error recovery for device	503F	003F	C03F	—	—	0	Y	N
Invalid function code in BIO macro	5020	0	C020	—	—	0	N	N
Invalid block number	5021	0	C021	—	—	0	N	N
Invalid byte count	5022	0	C022	—	—	0	N	N
Invalid CNTRL request	5023	0	C023	—	—	0	N	N
No FDT	5024	—	C024	—	—	0	N	N
Usage error	5025	0	C025	—	—	0	N	N
Operation to locked file	5026	0	C026	—	—	0	N	N
Invalid sequence of operations to a device	5027	0	C027	—	—	0	N	N
Subfunction field error	5028	0	C028	—	—	0	N	N
Invalid position (to magnetic tape file that is not maintaining block numbers)	5029	0	C029	—	—	0	N	N
EOF on a read to a bypassed file	0	0	A02A	—	—	0	N	N
Invalid unit ordinal on privileged PIO request	5030	—	C030	—	—	0	—	N

*Number of retries automatically performed by Error Recovery before completion is declared.

Table C-1. I/O Error Recovery Information (Continued)

Condition	Hexadecimal Status Completion Code	FDT Return Information	PCB/Request Block Return Information Field	Message Code Displayed On Console	Console Message Reply	No. of Retries	Must File be Reset	Error Log Record Written
DISC								
Timeout	5041	0041	C041	041	—	10	Y	Y
Command reject	5042	0042	C042	042	—	0	Y	N
Disc write current	5043	0043	C043	043	—	0	Y	Y
Seek incomplete	5044	0044	C044	044	—	10	Y	Y
Not on line	5045	0045	C045	045	Y/N	1	Y	N
File unsafe	5046	0046	C046	046	—	0	Y	Y
Pack change	5047	0047	C047	047	—	0	Y	N
Status not valid	5048	0048	C048	048	—	0	Y	Y
Command early	5049	0049	C049	049	—	0	Y	Y
Unsolicited Attention	504A	004A	C04A	04A	—	1	Y	N
Catastrophic error	504B	004B	C04B	04B	Y/N	0	Y	Y
Missed window	504C	004C	C04C	04C	—	10	Y	Y
IFA window	504D	004D	C04D	04D	—	10	Y	Y
Track boundary	504E	004E	C04E	04E	—	10	Y	N
Read Write terminate	504F	004F	C04F	04F	—	10	Y	Y
Burst check	5050	0050	C050	050	—	10	Y	Y
Lost data	5051	0051	C051	051	—	10	Y	Y
No sync compare	5052	0052	C052	052	—	10	Y	Y
Write operation issued to a drive in read only mode	5053	0053	C053	053	Y/N	1	Y	N
End of cylinder	5054	0054	C054	054	—	10	Y	Y
Busy	5055	0055	C055	055	—	10	Y	Y
Invalid seek address	5056	0056	C056	056	—	0	Y	N
Search failed (arm mispositioned)	5058	0058	C058	058	—	5	Y	Y
Search failed (no record found)	5059	0059	C059	059	—	5	Y	N

PRINTER								
Command reject	5064	0065	C064	—	—	0	Y	N
I/O Channel error	5065	0065	C065	065	Y/N	1	Y	Y
Data check	5066	0066	C066	—	—	0	Y	Y
Not ready	5067	0067	C067	067	Y/N	1	Y	N
Bus out check	5069	0069	C069	069	Y/N	1	Y	Y
Catastrophic error	506A	006A	C06A	06A	Y/N	1	Y	Y
Both channel 9 and 12 on the printer carriage control tape	0	0	A06B	—	—	0	N	N

Table C-1. I/O Error Recovery Information (Continued)

Condition	Hexadecimal Status Completion Code	FDT Return Information	PCB/Request Block Return Information Field	Message Code Displayed On Console	Console Message Reply	No. of Retries	Must File be Reset	Error Log Record Written
CARD READER								
Catastrophic error	5070	0070	C070	070	Y/N	0	Y	N
Command reject	5071	0071	C071	—	—	0	Y	N
Not ready	5072	0072	C072	072	Y/N	1	Y	N
Busy	5073	0073	C073	—	—	10	Y	Y
Feed check or jam	5074	0074	C074	074	Y/N	1	Y	Y
Read check	5075	0075	C075	075	Y/N	1	Y	Y
Data check (illegal EBCDIC char.)	5076	0076	C076	076	Y/N	1	Y	N
Time out	5077	0077	C077	—	—	10	Y	Y
Unsolicited attention	5078	0078	C078	078	—	1	Y	N
Initial selection error	5079	0079	C079	—	—	10	Y	Y

CARD READER PUNCH								
Catastrophic error	5080	0080	C080	080	Y/N	0	Y	N
Command Reject	5081	0081	C081	—	—	0	Y	N
Not ready	5082	0082	C082	082	Y/N	1	Y	N
Busy	5083	0083	C083	—	—	10	Y	Y
Feed check or jam	5084	0084	C084	084	Y/N	1	Y	Y
Data check (illegal EBCDIC char.)	5085	0085	C085	085	Y/N	1	Y	N
Read check	5086	0086	C086	086	Y/N	1	Y	Y
Time out	5087	0087	C087	—	—	10	Y	Y
Unsolicited attention	5088	0088	C088	088	—	1	Y	N
Initial selection error	5089	0089	C089	—	—	10	Y	Y
Punch check	508A	008A	C08A	08A	Y/N	1	Y	Y

MAGNETIC TAPE								
Command reject (read reverse)	50A1	00A1	C0A1	—	—	0	Y	N
Command reject (protected tape)	50A1	00A1	C0A1	0A1	—	0	Y	N
Intervention required (without equipment check)	50A2	00A2	C0A2	0A2	Y/N	1	Y	Y
Intervention required (with equipment check)	50A3	00A3	C0A3	0A3	—	0	Y	Y
Bus out check (on command)	50A4	00A4	C0A4	0A4	Y/N	10	Y	Y
Bus out check (on data)	50A5	00A5	C0A5	0A5	Y/N	10	Y	Y
Equipment check (Read Write reg.)	50A6	00A6	C0A6	0A6	Y/N	10	Y	Y
Equipment check (Write register)	50A7	00A7	C0A7	0A7	Y/N	1	Y	Y
Equipment check (Read register) on data	50A8	00A8	C0A8	0A8	Y/N	1	Y	Y
Equipment check (Read register) on command	50A8	00A8	C0A8	0A8	—	0	Y	Y
Equipment check (Delay register) on data	50A9	00A9	C0A9	0A9	Y/N	1	Y	Y

Table C-1. I/O Error Recovery Information (Continued)

Condition	Hexadecimal Status Completion Code	FDT Return Information	PCB/Request Block Return Information Field	Message Code Displayed On Console	Console Message Reply	No. of Retries	Must File be Reset	Error Log Record Written
Equipment check (Delay register) on command	50A9	00A9	C0A9	0A9	—	0	Y	Y
Data check (Write)	50AA	00AA	C0AA	0AA	Y/N	10-5	Y	Y
Data check (Write File Mark)	50AB	00AB	C0AB	0AB	Y/N	10	Y	Y
Data check (multiple track error on read)	50AC	00AC	C0AC	0AC	Y/N	10	Y	Y
Data check (single track error on read) Includes preamble and post-amble errors	50AD	00AD	C0AD	0AD	Y/N	10	Y	Y
Data check (phase track in error and VRC without Read Write register error)	0	0	0	—	—	0	N	Y
Data check (Erase)	50AE	00AE	C0AE	0AE	Y/N	1	Y	Y
Over run	50AF	00AF	C0AF	0AF	Y/N	10	Y	Y
Word count zero (Write Command Count ≠ 0)	50B0	00B0	C0B0	0B0	—	10	Y	Y
Not capable	50B1	00B1	C0B1	0B1	Y/N	1	Y	N
Backspace into BOT	0	00B2	A0B2	—	—	0	Y	N
Reverse Read command at BOT	50B4	00B4	C0B4	—	—	0	Y	N
EOT during any write operation	0	00B5	A0B5	—	—	0	Y	N
Unsolicited attention	50B6	00B6	C0B6	0B6	Y/N	1	Y	N
Time out	50B7	00B7	C0B7	0B7	—	0	Y	Y
Undefined IOC error	50B8	00B8	C0B8	0B8	N	0	Y	N
Internal recovery error No. 1*	50B9	00B9	C0B9	0B9	N	10	Y	Y
Internal recovery error No. 2*	50BA	00BA	C0BA	0BA	N	0	Y	Y
Internal recovery error No. 3*	50BB	00BB	C0BB	0BB	N	0	Y	N
ISS channel error	50E1	00E1	C0E1	0E1	Y/N	10	Y	Y
Wrong address-in channel error	50E2	00E2	C0E2	0E2	Y/N	10	Y	Y
Control check channel error	50E3	00E3	C0E3	0E3	Y/N	10	Y	Y
Transmission check channel error	50E4	00E4	C0E4	0E4	Y/N	10	Y	Y
Zero byte count	50E5	00E5	C0E5	—	—	0	Y	N

*Catastrophic error.

Table C-2. Error Log Record

0	ID	DATE/TIME
2	DATE/TIME	DATE/TIME
4	DATE/TIME	DATE/TIME
6	DATE/TIME	DATE/TIME
8	DATE/TIME	DATE/TIME
10	DATE/TIME	DATE/TIME
12	DATE/TIME	PROCESSOR
14	DEVICE ADDR	CMD IN ERROR
16	STATUS	STATUS
18	ER FLAG BYTE 0	ER FLAG BYTE 1
20	ERROR COUNTER 1	ERROR COUNTER 2
22	ERROR COUNTER 3 OR CYLINDER	ERROR COUNTER 4 OR HEAD
24	VOL ID	VOL ID
26	VOL ID	VOL ID
28	VOL ID	VOL ID
30	SENSE	SENSE
32	SENSE	SENSE
34	SENSE	SENSE
36		
38	RECORD SEQ. NUMBER	RECORD SEQ. NUMBER

<u>Bytes</u>	<u>Mnemonics</u>	<u>Description</u>																		
0	ID	<p>Identification of the record. The value of this field is as follows:</p> <table border="0"> <thead> <tr> <th>Hex Value</th> <th>Explanation</th> </tr> </thead> <tbody> <tr> <td>01</td> <td>from disc error recovery</td> </tr> <tr> <td>02</td> <td>from tape error recovery</td> </tr> <tr> <td>03</td> <td>from card reader error recovery</td> </tr> <tr> <td>04</td> <td>from card reader punch error recovery</td> </tr> <tr> <td>05</td> <td>from printer error recovery</td> </tr> <tr> <td>06</td> <td>from logical communications error recovery</td> </tr> <tr> <td>07</td> <td>from error correction code feature (optional)</td> </tr> </tbody> </table>	Hex Value	Explanation	01	from disc error recovery	02	from tape error recovery	03	from card reader error recovery	04	from card reader punch error recovery	05	from printer error recovery	06	from logical communications error recovery	07	from error correction code feature (optional)		
Hex Value	Explanation																			
01	from disc error recovery																			
02	from tape error recovery																			
03	from card reader error recovery																			
04	from card reader punch error recovery																			
05	from printer error recovery																			
06	from logical communications error recovery																			
07	from error correction code feature (optional)																			
1-12	DATE/TIME	<p>The date and time which are supplied by the Exec are in a binary format as follows:</p> <table border="0"> <thead> <tr> <th>Byte</th> <th>Explanation</th> </tr> </thead> <tbody> <tr> <td>1</td> <td>unused</td> </tr> <tr> <td>2</td> <td>year</td> </tr> <tr> <td>3</td> <td>month</td> </tr> <tr> <td>4</td> <td>day</td> </tr> <tr> <td>5</td> <td>hour</td> </tr> <tr> <td>6</td> <td>minutes</td> </tr> <tr> <td>7-10</td> <td>seconds</td> </tr> <tr> <td>11-12</td> <td>hardware clock</td> </tr> </tbody> </table>	Byte	Explanation	1	unused	2	year	3	month	4	day	5	hour	6	minutes	7-10	seconds	11-12	hardware clock
Byte	Explanation																			
1	unused																			
2	year																			
3	month																			
4	day																			
5	hour																			
6	minutes																			
7-10	seconds																			
11-12	hardware clock																			

Table C-2. Error Log Record (Continued)

<u>Bytes</u>	<u>Mnemonics</u>	<u>Description</u>																																																												
13	PROCESSOR	Processor state which the device uses. The processor codes are as follows: <table border="1"> <thead> <tr> <th>Hex Code</th> <th>Processor</th> </tr> </thead> <tbody> <tr> <td>08</td> <td>zero</td> </tr> <tr> <td>04</td> <td>one</td> </tr> <tr> <td>02</td> <td>two</td> </tr> <tr> <td>01</td> <td>three</td> </tr> </tbody> </table>	Hex Code	Processor	08	zero	04	one	02	two	01	three																																																		
Hex Code	Processor																																																													
08	zero																																																													
04	one																																																													
02	two																																																													
01	three																																																													
14	DEVICE ADDR	The physical device address. The physical device address for disc is the cable address (not the plug address).																																																												
15	CMD IN ERROR	The command code that the driver is attempting to execute when the error condition occurs.																																																												
16-17	STATUS	The two bytes of hardware status indication.																																																												
18-19	ER FLAG BYTE 0 & 1	Flag bits set by error recovery modules when the different error conditions occur. The two flag bytes are described as follows: <table border="1"> <thead> <tr> <th colspan="2"><u>Magnetic Tape</u></th> </tr> <tr> <th colspan="2">Flag Byte 0</th> </tr> <tr> <th><u>Bit</u></th> <th><u>Explanation</u></th> </tr> </thead> <tbody> <tr> <td>0</td> <td>intervention required</td> </tr> <tr> <td>1</td> <td>bus out check</td> </tr> <tr> <td>2</td> <td>equipment check (register parity)</td> </tr> <tr> <td>3</td> <td>equipment check (internal counter)</td> </tr> <tr> <td>4</td> <td>data check</td> </tr> <tr> <td>5</td> <td>over run</td> </tr> <tr> <td>6</td> <td>word count zero</td> </tr> <tr> <td>7</td> <td>preamble/postamble error</td> </tr> </tbody> </table> <table border="1"> <thead> <tr> <th colspan="2">Flag Byte 1</th> </tr> <tr> <th><u>Bit</u></th> <th><u>Explanation</u></th> </tr> </thead> <tbody> <tr> <td>0</td> <td>timeout error</td> </tr> <tr> <td>1</td> <td>ISS channel error</td> </tr> <tr> <td>2</td> <td>wrong address-in channel error</td> </tr> <tr> <td>3</td> <td>control check channel error</td> </tr> <tr> <td>4</td> <td>transmission check channel error</td> </tr> <tr> <td>5-7</td> <td>unused</td> </tr> </tbody> </table> <table border="1"> <thead> <tr> <th colspan="2"><u>Disc</u></th> </tr> <tr> <th colspan="2">Flag Byte 0</th> </tr> <tr> <th><u>Bit</u></th> <th><u>Explanation</u></th> </tr> </thead> <tbody> <tr> <td>0</td> <td>command early</td> </tr> <tr> <td>1</td> <td>timeout</td> </tr> <tr> <td>2</td> <td>disc arm mispositioning</td> </tr> <tr> <td>3</td> <td>unused</td> </tr> <tr> <td>4</td> <td>read/write terminate</td> </tr> <tr> <td>5</td> <td>catastrophic error</td> </tr> <tr> <td>6</td> <td>lost data</td> </tr> <tr> <td>7</td> <td>no sync compare</td> </tr> </tbody> </table>	<u>Magnetic Tape</u>		Flag Byte 0		<u>Bit</u>	<u>Explanation</u>	0	intervention required	1	bus out check	2	equipment check (register parity)	3	equipment check (internal counter)	4	data check	5	over run	6	word count zero	7	preamble/postamble error	Flag Byte 1		<u>Bit</u>	<u>Explanation</u>	0	timeout error	1	ISS channel error	2	wrong address-in channel error	3	control check channel error	4	transmission check channel error	5-7	unused	<u>Disc</u>		Flag Byte 0		<u>Bit</u>	<u>Explanation</u>	0	command early	1	timeout	2	disc arm mispositioning	3	unused	4	read/write terminate	5	catastrophic error	6	lost data	7	no sync compare
<u>Magnetic Tape</u>																																																														
Flag Byte 0																																																														
<u>Bit</u>	<u>Explanation</u>																																																													
0	intervention required																																																													
1	bus out check																																																													
2	equipment check (register parity)																																																													
3	equipment check (internal counter)																																																													
4	data check																																																													
5	over run																																																													
6	word count zero																																																													
7	preamble/postamble error																																																													
Flag Byte 1																																																														
<u>Bit</u>	<u>Explanation</u>																																																													
0	timeout error																																																													
1	ISS channel error																																																													
2	wrong address-in channel error																																																													
3	control check channel error																																																													
4	transmission check channel error																																																													
5-7	unused																																																													
<u>Disc</u>																																																														
Flag Byte 0																																																														
<u>Bit</u>	<u>Explanation</u>																																																													
0	command early																																																													
1	timeout																																																													
2	disc arm mispositioning																																																													
3	unused																																																													
4	read/write terminate																																																													
5	catastrophic error																																																													
6	lost data																																																													
7	no sync compare																																																													

Table C-2. Error Log Record (Continued)

<u>Bytes</u>	<u>Mnemonics</u>	<u>Description</u>
Flag Byte 1		
<u>Bit</u>		<u>Explanation</u>
0		third revolution sync find
1		seek incomplete
2		file unsafe
3		missed window
4		burst check
5		end of cylinder
6		disc write current
7		busy
Card Reader and Reader Punch		
Flag Byte 0		
<u>Bit</u>		<u>Explanation</u>
0		timeout
1		busy
2		ISS channel error
3		catastrophic error
4		punch check
5		command reject
6		read check
7		card feed check or jam
Flag Byte 1		
<u>Bit</u>		<u>Explanation</u>
0-7		unused
Printer		
Flag Byte 0		
<u>Bit</u>		<u>Explanation</u>
0		timeout
1-4		unused
5		bus out check
6		data check
7		invalid sense information
Flag Byte 1		
<u>Bit</u>		<u>Explanation</u>
0		ISS error
1		unused
2		wrong address in initial selection or address/status parity
3		no request in SIO poll sequence request
4		control check
5		data transmission check
6-7		unused

Note: The printer hardware status bytes that are written in the error log file represent all of the conditions that occurred while printer error recovery was in progress for a given error. Therefore, if a channel error occurs during recovery procedures, it will appear in the two hardware status bytes.

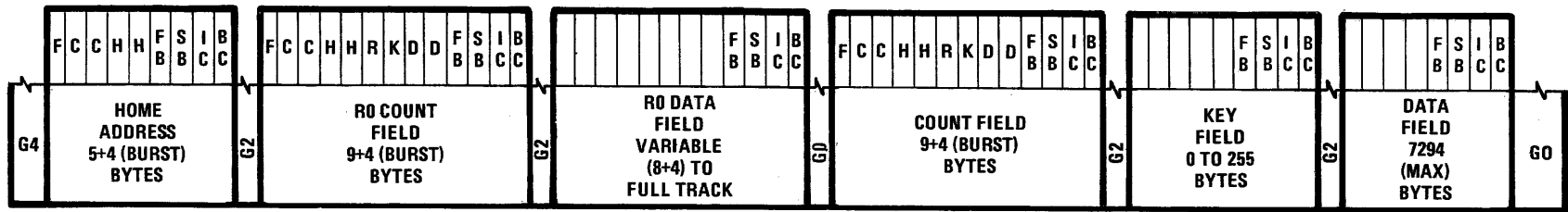
Table C-2. Error Log Record (Continued)

<u>Bytes</u>	<u>Mnemonics</u>	<u>Description</u>
20	ERROR COUNTER 1	The following errors use Counter 1: <ol style="list-style-type: none">1. Card Reader and Card Reader Punch Error Recovery error conditions.2. Disc Error Recovery error conditions except search failures.3. Magnetic Tape Error Recovery for user request error condition.
21	ERROR COUNTER 2	The following errors use Counter 2: <ol style="list-style-type: none">1. Disc Error Recovery for a search failure.2. Magnetic Tape Error Recovery for recording the number of erase operations performed during the recovery of a write operation.
22	ERROR COUNTER 3 or CYLINDER	Error Counter 3 is incremented by the Magnetic Tape Error Recovery for positioning errors. Or, the disc's cylinder number is extracted from the seek argument of the first command word in a command program. If the first command word is not a seek, the cylinder number is not supplied.
23	ERROR COUNTER 4 or HEAD	Magnetic Tape Error Recovery uses Counter 4 as a record counter. Bits 0-3 are used when positioning forward and bits 4-7 are used when positioning backward. The disc's head number is extracted from the seek argument of the first command word in a command program. If the first command word is not a seek, the head number is not supplied.
24-29	VOLID	Volume ID. This field applies only to disc.
30-35	SENSE	From one to six sense bytes depending on the type of device. See the associated hardware specification for the bit positions and their meanings.
36-37		Unused.
38-39		Record sequence number.

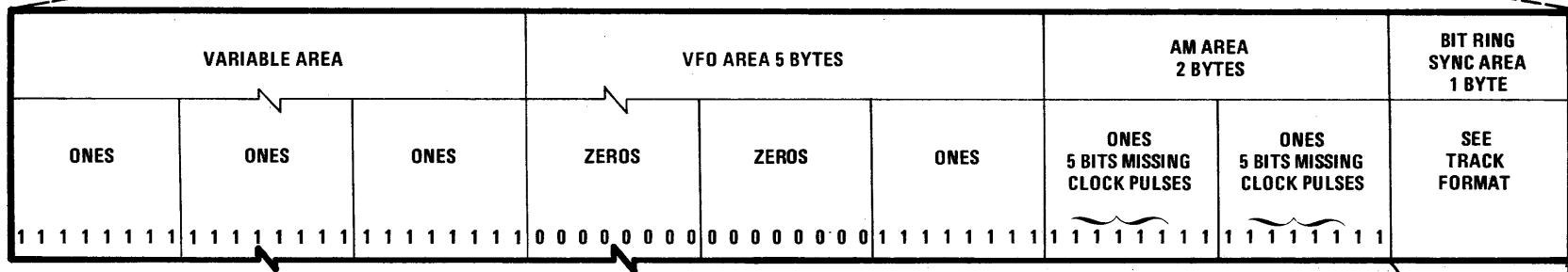
D. GAP SPECIFICATIONS

Table of Gap specifications to be used with the COMMAND/OPCODE=DCWRIT macro when working with fixed record lengths (and no key fields).

Record Size in Bytes	Number of Records in Track	GAP (Inverted Hex)
3521 – 7294	1	None
2299 – 3520	2	67
1694 – 2298	3	9C
1333 – 1693	4	B6
1093 – 1332	5	C5
922 – 1092	6	D0
794 – 921	7	D7
695 – 793	8	DC
616 – 694	9	E1
551 – 615	10	E4
497 – 550	11	E7
451 – 496	12	E9
412 – 450	13	EB
378 – 411	14	ED
348 – 377	15	EE
322 – 347	16	F0
299 – 321	17	F1
277 – 298	18	F2
259 – 276	19	F3
242 – 258	20	F3
227 – 241	21	F4
212 – 226	22	F5
200 – 211	23	F5
188 – 199	24	F6
177 – 187	25	F6
167 – 176	26	F7
158 – 166	27	F7
149 – 157	28	F8
140 – 148	29	F8
133 – 139	30	F9
126 – 132	31	F9
119 – 125	32	F9
113 – 118	33	F9
107 – 112	34	FA
101 – 106	35	FA
95 – 100	36	FA
81 – 94	37-39	FB
62 – 80	40-44	FC
46 – 61	45-49	FD
33 – 45	50-54	FE
23 – 32	55-59	FE
13 – 22	60-64	FE
0 – 12	65-73	FF



GAP BIT CONFIGURATION



SYNC BYTE CONFIGURATIONS

HA SYNC BYTE	RO COUNT SYNC BYTE	RO DATA SYNC BYTE	R _N COUNT SYNC BYTE	KEY SYNC BYTE	R _N DATA SYNC BYTE
13 ₁₀	11 ₁₀	9 ₁₀	14 ₁₀	10 ₁₀	9 ₁₀
0 0 0 0 1 0 1	0 0 0 0 1 0 1 1	0 0 0 0 1 0 0 1	0 0 0 0 1 1 1 0	0 0 0 0 1 0 1 0	0 0 0 0 1 0 0 1

- F = FLAG BYTE
- CC = CYLINDER NUMBER IN BINARY
- HH = HEAD NUMBER IN BINARY
- R = TRACK RECORD NUMBER IN BINARY
- K = KEY FIELD LENGTH IN BINARY
- DD = DATA FIELD LENGTH IN BINARY
- FB = FIRST (CYCLIC BURST) BYTE
- SB = SECOND (CYCLIC BURST) BYTE
- IC = INDICATOR BYTE
- BC = BIT COUNT BYTE
- G0 = 41 BYTES + 0.043 X (K_L + D_L)*
- G2 = 41 BYTES
- G4 = 73 BYTES

E1 * BURST CHECK BYTES NOT INCLUDED

F. INDEX—BLOCK SIZE FOR INDEXED FILES

There is a *minimum index block size* for every indexed file depending on key size and file size. The user may utilize any index block size larger than the minimum, if he has memory space for a larger index block. The larger the index block the better retrieval becomes on random processing. If the user goes below the minimum index block size there is the possibility of not being able to create the file size as planned.

When planning the creation of indexed files, the user must decide whether he wants to process the directory-directory, which resides on mass storage, in a main memory buffer. This option speeds up random processing, but requires extra space for the buffer. If the mode of processing is with a main-memory buffer there is a well-defined *optimum index block size* which minimizes memory space for the index buffer and directory-directory buffer.

Once the user has determined his mode of processing, Table F-1 is used to determine minimum-keys/block and Table F-2 is used to determine optimum keys/block. Note that in using Table F-1 and Table F-2, the larger of the two values in the file size is the determining factor. Also note that these tables were computed for consistency for maximum key size and one million records as the upper limit. There will be some index block sizes generated that exceed one track in number of bytes. This exceeds the system limit for block sizes. The user will have to choose a smaller key size or smaller file size.

The keys/block is entered in the Control Language //DEFINE statement along with key size. The corresponding minimum or optimum index block size can be calculated from Table F-3. The resulting index block size is then entered in the COBOL source program via the INDEX-BLOCK Clause.

If the user has determined to calculate the optimum keys/block and optimum index block size, Table F-4 is used to calculate the number of bytes for the main-memory buffer for the directory-directory entries.

The user must be careful not to exceed the file maximum at creation time when using the optimum block size — when he utilizes the main-memory buffer to hold the directory-directory entries for random processing, the buffer would not be able to hold all the entries, thus writing over the user program. Thus, when choosing an index block size other than the optimum and the main-memory buffer is used to process the directory-directory entries, the buffer size should be the size of the index block, as the system checks for overflow at creation time.

If the user wishes to calculate keys/block based on a different file maximum than given in Tables F-1 and F-2, the following algorithms, along with Table F-5, can be used to compute minimum and optimum keys/block. The constants K_o and K_m are taken from Table F-5 based on key size.

$$\text{Optimum (OKB)} = \left\{ \sqrt[3]{\frac{FS}{K_o}} \right\}$$

$$\text{Minimum (MKB)} = \left\{ \sqrt[3]{\frac{FS}{K_m}} \right\}$$

FS = Maximum File Size

NOTE: $\left\{ \right\}$ = Round up if result not whole integer.

Table F-1. Minimum Keys/Block

Blocks in File	Key Size in Bytes														
	2	3	4	5	6	7	8	9	10	11 to 15	16 to 20	21 to 25	26 to 35	36 to 50	51 to 100
0 - 5,000	13	14	14	14	15	15	15	15	15	16	16	16	16	17	17
5,000 - 10,000	16	17	18	18	18	19	19	19	19	20	20	20	21	21	21
10,000 - 15,000	19	20	20	21	21	21	22	22	22	23	23	23	23	24	24
15,000 - 20,000	20	21	22	23	23	23	24	24	24	25	25	26	26	26	26
20,000 - 25,000	22	23	24	24	25	25	25	26	26	27	27	27	28	28	28
25,000 - 30,000	23	24	25	26	26	27	27	27	28	28	29	29	29	30	30
30,000 - 35,000	25	26	27	27	28	28	28	29	29	30	30	31	31	31	32
35,000 - 40,000	26	27	28	28	29	29	30	30	30	31	32	32	32	33	33
40,000 - 45,000	27	28	29	30	30	31	31	31	31	32	33	33	34	34	34
45,000 - 50,000	28	29	30	31	31	32	32	32	33	34	34	34	35	35	36
50,000 - 60,000	29	31	32	32	33	34	34	34	35	36	36	37	37	37	38
60,000 - 70,000	31	32	34	34	35	35	36	36	36	37	38	38	39	39	40
70,000 - 80,000	32	34	35	36	36	37	37	38	38	39	40	40	41	41	42
80,000 - 90,000	34	35	36	37	38	38	39	39	40	41	41	42	42	43	43
90,000 - 100,000	35	36	37	38	39	40	40	41	41	42	43	43	44	44	45
100,000 - 125,000	37	39	40	41	42	43	43	44	44	45	46	47	47	48	48
125,000 - 150,000	40	41	43	44	45	45	46	46	47	48	49	49	50	51	51
150,000 - 175,000	42	44	45	46	47	48	48	49	49	50	51	52	53	53	54
175,000 - 200,000	44	46	47	48	49	50	50	51	51	53	54	54	55	56	56
200,000 - 250,000	47	49	51	52	53	54	54	55	55	57	58	58	59	60	61
250,000 - 300,000	50	52	54	55	56	57	58	58	59	61	61	62	63	64	64
300,000 - 350,000	52	55	57	58	59	60	61	62	62	64	65	65	66	67	68
350,000 - 400,000	55	57	59	61	62	63	63	64	65	67	68	68	69	70	71
400,000 - 450,000	57	60	62	63	64	65	66	67	67	69	70	71	72	73	74
450,000 - 500,000	59	62	64	65	67	68	68	69	70	72	73	74	74	75	76
500,000 - 600,000	63	66	68	69	71	72	73	73	74	76	77	78	79	80	81
600,000 - 700,000	66	69	71	73	74	75	76	77	78	80	81	82	83	84	85
700,000 - 800,000	69	72	74	76	78	79	80	81	81	84	85	86	87	88	89
800,000 - 900,000	72	75	77	79	81	82	83	84	85	87	88	89	91	91	93
900,000 - 1,000,000	74	78	80	82	84	85	86	87	88	90	92	93	94	95	96

Table F-2. Optimum Keys/Block

Blocks in File	Key Size in Bytes														
	2	3	4	5	6	7	8	9	10	11 to 15	16 to 20	21 to 25	26 to 35	36 to 50	51 to 100
0 - 5,000	16	17	18	18	18	19	19	19	19	20	20	20	21	21	21
5,000 - 10,000	20	21	22	23	23	23	24	24	24	25	25	25	26	26	26
10,000 - 15,000	23	24	25	26	26	27	27	27	27	28	29	29	29	30	30
15,000 - 20,000	26	27	28	28	29	29	29	30	30	31	32	32	32	33	33
20,000 - 25,000	28	29	30	31	31	32	32	32	33	34	34	34	35	35	36
25,000 - 30,000	29	31	32	32	33	34	34	34	35	36	36	37	37	37	38
30,000 - 35,000	31	32	33	34	35	35	36	36	37	37	38	38	39	39	40
35,000 - 40,000	32	34	35	36	36	37	37	38	38	39	40	40	41	41	42
40,000 - 45,000	34	35	36	37	38	38	39	39	40	41	41	42	42	43	43
45,000 - 50,000	35	36	37	38	39	40	40	41	41	42	43	43	44	44	45
50,000 - 60,000	37	39	40	41	42	42	43	43	43	45	45	46	46	47	48
60,000 - 70,000	39	41	42	43	44	44	45	45	46	47	48	48	49	49	50
70,000 - 80,000	40	42	44	45	46	46	47	47	48	49	50	50	51	52	52
80,000 - 90,000	42	44	45	47	47	48	49	49	50	51	52	52	53	54	54
90,000 - 100,000	44	46	47	48	49	50	50	51	51	53	54	54	55	56	56
100,000 - 125,000	47	49	51	52	53	54	54	55	55	57	58	58	59	60	61
125,000 - 150,000	50	52	54	55	56	57	58	58	59	61	61	62	63	64	64
150,000 - 175,000	52	55	57	58	59	60	61	61	62	64	65	65	66	67	68
175,000 - 200,000	55	57	59	61	62	63	63	64	65	67	68	68	69	70	71
200,000 - 250,000	59	62	64	65	67	68	68	69	70	72	73	74	74	75	76
250,000 - 300,000	63	66	68	69	71	72	73	73	74	76	77	78	79	80	81
300,000 - 350,000	66	69	71	73	74	75	76	77	78	80	81	82	83	84	85
350,000 - 400,000	69	72	74	76	78	79	80	81	81	84	85	86	87	88	89
400,000 - 450,000	72	75	77	79	81	82	83	84	85	87	88	89	91	91	93
450,000 - 500,000	74	78	80	82	84	85	86	87	88	90	92	93	94	95	96
500,000 - 600,000	79	82	84	87	89	90	91	92	93	96	97	98	100	101	102
600,000 - 700,000	83	87	90	92	94	95	96	97	98	101	102	103	105	106	107
700,000 - 800,000	87	91	94	96	98	99	100	102	102	105	107	108	110	111	112
800,000 - 900,000	90	94	97	100	102	103	105	106	106	110	111	112	114	115	116
900,000 - 1,000,000	93	98	101	103	105	107	108	109	110	113	115	116	118	120	121

Table F-3. Optimum or Minimum Index Block Size

<p>Optimum block size = $10 + \left\{ \frac{(10) (OKB) (KS+4)}{9} \right\}$</p> <p>OKB = Optimum keys/block KS = Key size</p> <p>Minimum block size = $10 + \left\{ \frac{(10) (MKB) (KS+4)}{9} \right\}$</p> <p>MKB = Minimum keys/block KS = Key size</p> <p>NOTE: $\left\{ \right\}$ = Round up if result not whole integer.</p>
--

Table F-4. Bytes Required in Buffer for Directory-Directory Entries

<p>Usage = $\left[\frac{9(OBS-10)}{10} \right] = US$</p> <p>Number keys/primary index block = $\left[\frac{US}{KS+4} \right] = NKP$</p> <p>Number keys/directory block = $\left[\frac{US}{KS+2} \right] = NKD$</p> <p>Total number keys represented/ directory block = (NKP) (NKD) NKRD</p> <p>Number entries in Directory-directory block = $\left\{ \frac{\text{file size}}{NKRD} \right\} = NKDD$</p> <p>Number of bytes required for buffer for directory-directory entries = $10 + (KS+2) (NKDD)$</p> <p>NOTE: $\left\{ \right\}$ = Round up if result not whole integer. $\left[\right]$ = Round down if result not whole integer.</p>

Table F-5. Constants for Alternate Algorithm

KS	K _o	K _m	KS	K _o	K _m
2	1.2500	2.5000	52	.5975	1.1950
3	1.0888	2.1776	53	.5967	1.1934
4	.9877	1.9753	54	.5959	1.1918
5	.9184	1.8367	55	.5952	1.1904
6	.8681	1.7361	56	.5945	1.1890
7	.8299	1.6598	57	.5939	1.1878
8	.8000	1.6000	58	.5932	1.1864
9	.7759	1.5518	59	.5926	1.1852
10	.7562	1.5124	60	.5920	1.1840
11	.7396	1.4792	61	.5914	1.1828
12	.7256	1.4512	62	.5908	1.1816
13	.7136	1.4272	63	.5903	1.1806
14	.7031	1.4062	64	.5897	1.1794
15	.6940	1.3880	65	.5892	1.1784
16	.6859	1.3718	66	.5887	1.1774
17	.6787	1.3574	67	.5882	1.1764
18	.6722	1.3444	68	.5878	1.1756
19	.6664	1.3328	69	.5873	1.1746
20	.6612	1.3224	70	.5868	1.1736
21	.6564	1.3128	71	.5864	1.1728
22	.6520	1.3040	72	.5860	1.1720
23	.6480	1.2960	73	.5856	1.1712
24	.6443	1.2886	74	.5852	1.1704
25	.6409	1.2818	75	.5848	1.1696
26	.6378	1.2756	76	.5844	1.1688
27	.6348	1.2696	77	.5840	1.1680
28	.6321	1.2642	78	.5837	1.1674
29	.6296	1.2592	79	.5833	1.1666
30	.6272	1.2544	80	.5830	1.1660
31	.6249	1.2498	81	.5827	1.1654
32	.6228	1.2456	82	.5823	1.1646
33	.6209	1.2418	83	.5820	1.1640
34	.6190	1.2380	84	.5817	1.1634
35	.6172	1.2344	85	.5814	1.1628
36	.6156	1.2312	86	.5811	1.1622
37	.6140	1.2280	87	.5808	1.1616
38	.6125	1.2250	88	.5805	1.1610
39	.6111	1.2222	89	.5802	1.1604
40	.6097	1.2194	90	.5800	1.1600
41	.6084	1.2168	91	.5797	1.1594
42	.6072	1.2144	92	.5794	1.1588
43	.6060	1.2120	93	.5792	1.1584
44	.6049	1.2098	94	.5789	1.1578
45	.6038	1.2076	95	.5787	1.1574
46	.6028	1.2056	96	.5785	1.1570
47	.6018	1.2036	97	.5782	1.1564
48	.6009	1.2018	98	.5780	1.1560
49	.6000	1.2000	99	.5778	1.1556
50	.5991	1.1982	100	.5776	1.1552
51	.5983	1.1966			

INDEX

ABEND macro	7-1,37;B-37	Buffer description table	
ACCEPT macro	5-2;7-1,34; B-32	general description	2-12
Access		indexed files	2-29-34
random	2-1	relative files	2-24-28
sequential	2-1	sequential files	2-20-23
Action macros	B-1	BUFSIZ parameter	7-15,24,27,28
Add mass storage space	7-5	Catalog elements	
ALLOC macro	3-1,3;6-1; 7-1,2,3;B-6	attribute	2-3,6
Allocate space	7-3	continuation	2-3;A-6,9
Assumed block number	3-2	name	2-3,4; A-4,5
Attribute element	2-3,6	space	2-3,8;A-3
BDT	2-12,20-34	volume	2-3,10
sequential files	2-20-23	CATALOG parameter	7-3,5,6
relative files	2-24-28	Central catalog	2-3;A-1;4,7
indexed files	2-29-34	CHAIN parameter	7-24,27, 28,30
Bit significance	7-22	Change current block number	7-12
BLKNUM parameter	7-11,12,13	Close file for data transmission	7-7
BLKSIZ parameter	7-3	CLOSE macro	3-3;7-1,7; B-14
Block I/O		Close volume	7-8
coding	3-2	CLOVE macro	3-1,3;6-1; 7-1,8;B-15
declarative macro	7-1,2	CNTBUF parameter	7-28,29
device control commands	3-3	CNTRL macro	3-2,3;7-1, 13;B-18
error processing	3-4	CNTSIZ parameter	7-28,29
file control	3-3	COMMAND macro	7-24
general description	1-2	basic data channel	7-24;B-23
macros		DCABLE	7-24;B-24
CNTRL	7-1,13;B-18	DCJUMP	7-30;B-29
POSITN	7-1,12;B-18	DCREAD	7-28;B-27
READ	7-1,10;B-19	DCSEEK	7-24;B-25
RESET	7-1,21	DCSRCH	7-27;B-26
STATUS	7-1,15;B-17	DCWRIT	7-29;B-28
TYPE	7-1,15;B-17	RESTORE	7-30;B-29
WRITE	7-1,11;B-19	Command program	4-1
positioning	3-3	Common stored data format	2-1
program	3-5,6	CON parameter	7-3,5
reading	3-2	Console communication macros	
request termination	3-4	CONSOLE	7-1,37,38; B-39
rules	3-1	MESSAGE	7-1,38
sense information	3-4	CONSOLE macro	7-1,38;B-39
space management	3-3		
writing	3-2		
BREAK parameter	7-31		
BUFADR parameter	7-7,15,24,27		

CONTROL parameter	7-7	Disc catalog	
Control program macros		central	2-2,3;A-1,4,7
ABEND	7-1,37;B-37	pack	2-2,3;A-1,2
ACCEPT	7-1,34;B-32	Disc track format	E-1
DELAY	7-1,31;B-35	DISPLAY macro	5-2;7-1,35
DISPLAY	7-1,35		
EHALT	7-1,37;B-36	EBCDIC parameter	7-10,12
GETCOM	7-1,33;B-31	EHALT macro	7-1,37;B-36
HALT	7-1,36;B-36	ELEMENT parameter	7-9
INFORM	7-1,31;B-33	End conditions	
JDATE	7-1,37;B-38	EOF	3-3
MEMLIM	7-1,35;B-33	EOA	3-4
POST	7-1,32;B-31	EOT	3-4
RPOST	7-1,32;B-31	ENDADR parameter	7-34
SDATE	7-1,37;B-38	ERRCOMP parameter	3-4;7-3,5 thru 10,12 thru 15,21,23,30 thru 36;B-4
SETCOM	7-1,33;B-31		
SETIF	7-1,36;B-34	ERROPT parameter	4-3;7-23
TIME	7-1,37;B-37	Error log record	C-7
WAIT	7-1,30;B-30	Error logging	C-2
Control program services	5-1	Error processing	3-4
finding partition size	5-1	Error recovery	
inter-step and control language communication	5-1	types of errors	C-1
reading data from //PAR cards	5-2	intervention required	C-1
service request control	5-1	retries	C-1
writing to SYSOUT	5-2	information	C-3
CP parameter	7-22	uncertain conditions	C-1
CPADR parameter	7-23	irrecoverable conditions	C-2
Create communication byte	7-32	error logging	C-2,7
CSD parameter	7-3,34	EXCP instruction	4-1
CWADR parameter	7-30	EXCP macro	7-1,22;B-21
CYCLES parameter	7-31	Expand from communication byte	7-32
		EXPND macro	3-3;6-1;7-1,2,5;B-9
Data macros	B-1		
Data structures	2-1	FDT	2-12,13 thru 19;3-4;7-2
DATA CYL parameter	7-3	FIELD parameter	7-27,28,30
DATATXT parameter	7-38	File control	
DATBUF parameter	7-10,12,28,29,35	general description	3-3
DATABUF1 parameter	7-34,38	macros	
DATABUF2 parameter	7-38	CLOSE	7-1,6,7;B-14
DATSIZ Parameter	7-10,12,28,29,35	CLOVE	7-1,6,8;B-15
DATSIZ1 parameter	7-38	OPEN	7-1,6;B-12
DATSIZ2 parameter	7-38	File description table	2-12,13 thru 19;3-4;7-2
Define file label	7-2	File label (tape)	2-1,2
Defining and opening devices	4-1	File organization	2-1
DEFLB macro	7-1,2	FILENAM parameter	7-2
DELAY macro	5-1;7-1,31;B-35		
Device and file type	7-15		
Device control commands	3-3		
DEVTYP parameter	7-23		

FILEORG parameter	7-3	LIST parameter	7-3 thru 10, 12 thru 15,21, 30 thru 36;B-3
Files			
indexed	2-1		
relative	2-1	LOCK parameter	7-8
sequential	2-1	Logical I/O	1-1
FILESIZ parameter	7-3,4,5		
FILETYP parameter	7-3,4	Macro expansions	B-1,5
Fixed length records	2-1	Macros	
FUNCTN parameter	7-23	ABEND	7-1,37;B-37
		ACCEPT	5-2;7-1,34; B-32
GAP parameter	7-29	ALLOC	3-1,3;6-1; 7-1,2,3;B-6
Gap specification	D-1	CLOSE	3-3;7-1,7; B-14
Generation of reply buffer	7-38	CLOVE	3-1,3;6-1; 7-1,8;B-15
GETCOM macro	5-1;7-1,33; B-31	CNTRL	3-2,3;7-1, 13;B-18
		COMMAND	4-1;7-24 thru 30;B-23 thru 29
HABUF parameter	7-28,29	CONSOLE	7-1,38;B-39
HALT macro	7-1,36;B-36	DEFLB	7-1,2
Hardware control operation	7-13	DELAY	5-1;7-1,31; B-35
HASIZ parameter	7-28,29	DISPLAY	5-2;7-1,35
		EHALT	7-1,37;B-36
IDENT parameter	7-3 thru 10, 12 thru 15,21	EXCP	7-1,22;B-21
Identify partition limit	7-35	EXPND	3-3;6-1;7-1, 2,5;B-9
INDCYL parameter	7-3,4	GETCOM	5-1;7-1,33; B-31
Index block size	F-1	HALT	7-1,36;B-36
Indexed files	2-1	INFORM	5-1;7-1,31; B-33
INDSIZ parameter	7-3,4	JDATE	7-1,37;B-38
INFOADR parameter	7-9,32,33, 36,37	LABRTN	7-1,9;B-16
INFORM macro	5-1;7-1,31; B-33	MEMLIM	5-1;7-1,35; B-33
Input/output action	7-22	MESSAGE	7-1,38
Input/output levels		OPEN	3-3;6-1;7-1, 6;B-12
logical	1-1	PCB	7-1,23;B-22
block	1-1,2	POSITN	3-2,3;7-1, 12;B-18
physical	1-1,2	POST	5-1;7-1,32; B-31
Interaction of Data Management and Control Language	6-1	PURGE	3-3;6-1;7-1, 2,5;B-11
I/O service macro (LABRTN)	7-1,9;B-16	READ	3-2,3;7-1, 10;B-19
IOTYP parameter	7-7,8	RESET	3-2,3,4; 7-1,21
JDATE macro	7-1,37;B-38		
KEYBUF parameter	7-28,29		
KEYSIZ parameter	7-3,4,28,39		
LABDEF parameter	7-5,6,7		
LABDEF1 parameter	7-3,4		
LABDEF2 parameter	7-3,4		
Labels			
tape	2-1,2		
disc	2-2,3		
LABRTN macro	7-1,9;B-16		

Macros (continued)

RPOST	5-1;7-1,32; B-31
SDATE	7-1,37;B-38
SETCOM	5-1;7-1,33; B-31
SETIF	5-1;7-1,36; B-34
STATUS	3-2;7-1,15; B-17
TIME	7-1,37;B-37
TYPE	3-2;7-1,15; B-17
WAIT	5-1;7-1,30; B-30
WRITE	3-2,3;7-1, 11;B-19
MEMLIM macro	5-1;7-1,35; B-33
MESSAGE macro	7-1,38
MODE parameter	7-30,37,38
MSC parameter	7-2
MULTBLK parameter	7-11
Multi-volume file processing	3-4
Name element	2-3,4;A-5
OPCODE parameter	7-24,27 thru 30
OPEN macro	3-3;6-1; 7-1,6;B-12
OPER parameter	7-11,12,14
Pack catalog	2-3;A-1,2
PAIRED parameter	7-5,6
Parameters	
BLKNUM	7-11,12,13
BLKSIZ	7-3
BREAK	7-31
BUFADR	7-7,15,24,27
BUFSIZ	7-15,24,27, 28
CATALOG	7-3,5,6
CHAIN	7-24,27,28, 30
CNTBUF	7-28,29
CNTSIZ	7-28,29
CON	7-3,5
CONTROL	7-7
CP	7-22
CPADR	7-23
CSD	7-3,34
CWADR	7-30
CYCLES	7-31

Parameters (continued)

DATATXT	7-38
DATA CYL	7-3
DATBUF	7-10,12,28, 29,35
DATBUF1	7-34,38
DATBUF2	7-38
DATSIZ	7-10,12,28, 29,35
DATSIZ1	7-38
DATSIZ2	7-38
DEVTYP	7-23
EBCDIC	7-10,12
ELEMENT	7-9
ENDADR	7-34
ERRCOMP	3-4;7-3,5 thru 10, 12 thru 15, 21,23,30 thru 36;B-4
ERROPT	4-3;7-23
FIELD	7-27,28,30
FILENAM	7-2
FILEORG	7-3
FILESIZ	7-3,4,5
FILETYP	7-3,4
FUNCTN	7-23
GAP	7-29
HABUF	7-28,29
HASIZ	7-28,29
IDENT	7-3 thru 10, 12 thru 15,21
INDCYL	7-3,4
INDSIZ	7-3,4
INFOADR	7-9,32,33, 36,37
IOTYP	7-7,8
KEYBUF	7-28,29
KEYSIZ	7-3,4,28,39
LABDEF	7-5,6,7
LABDEF1	7-3,4
LABDEF2	7-3,4
LIST	7-3 thru 10, 12 thru 15, 21, 30 thru 36;B-3
LOCK	7-8
MODE	7-30,37,38
MSC	7-2
MULTBLK	7-11
OPCODE	7-24,27 thru 30
OPER	7-11,12,14
PAIRED	7-5,6

Parameters (continued)			
PARNUM	5-2;7-34	PURGE macro	3-3;6-1;7-1, 2,5;B-11
PCB	7-22		
RECSIZ	7-3,4	Random access	2-1
REQADR	7-30	READ macro	3-2,3;7-1, 10;B-19
REQCNT	7-30,32	Reading data from //PAR cards	5-2
RETURN	4-1;7-3 thru 10,12 thru 15, 21,23,32 thru 36;B-4	Read //PAR card	7-34
		Records	
REWIND	7-7,8	fixed length	2-1
SECONDS	7-31	variable length	2-1
SEEK	7-13	RECSIZ parameter	7-3,4
SIZERR	7-24	Relative files	2-1
SKIP	7-24,28	Release disc file space	7-5
SPREAD	7-3,4	Report of status	7-15
STRIP	7-34	REQADR parameter	7-30
UNORD	7-22	REQCNT parameter	7-30,32
USAGE	7-7	Request	
VERIFY	7-3,4	overlap	3-3
PARNUM parameter	5-2;7-34	termination	3-4
PCB		Reset exception conditions	7-21
general description	4-1	RESET macro	3-2,3,4; 7-1,21
macro	7-1,23;B-22		
parameter	7-22	Retrieve	
Peripheral device hardware codes	7-25	system date	7-37
Physical		time of day	7-37
command block	4-1	Return file label information	7-9
control block	7-23	RETURN parameter	4-1;7-3 thru 10,12 thru 15, 21,23,32 thru 36;B-4
request termination	4-3		
Physical I/O		Returned information format	7-16
coding	4-1	REWIND parameter	7-7,8
error processing	4-3	RPOST macro	5-1;7-1,32; B-31
general description	1-2		
macros			
EXCP	7-1,22;B-21	SDATE macro	7-1,37;B-38
PCB	7-1,23;B-22	SECONDS parameter	7-31
operation	4-1	SEEK parameter	7-13
overlap	4-3	Sense information	3-4
program	4-4	Sequential	
request termination	4-3	access	2-1
restrictions	4-3	files	2-1
POSITN macro	3-2,3;7-1, 12;B-18	Service request mechanism	
		complete bit	B-3
Post code for Control Language	7-36	CONTROL	5-1
POST macro	5-1;7-1,32; B-31	end flag	B-3
		error flag	B-3
Priority	3-3	function code	B-3
Processing considerations		instruction	B-1
end conditions	3-3	length	B-3
multi-volume files	3-4	linkage	B-2
priority	3-3	return code	B-3
request overlap	3-3	return information	B-3

Set up message format	7-38	Tables	
SETCOM macro	5-1;7-1, 33;B-31	FDT	2-12,13 thru 19;3-4;7-2
SETIF macro	5-1;7-1,36; B-34	BDT	2-12,20 thru 34
Sharing		Tape labels	
an EXCP	4-2,3	volume	2-1,2
a PCB	4-1,2	file	2-1,2
SIZERR parameter	7-24	Terminate program	7-36,37
SKIP parameter	7-24,28	TIME macro	7-1,37;B-37
Space element	2-3,8; A-3,A-6	TYPE macro	3-2;7-1,15; B-17
Space management		UNORD parameter	7-22
general description	3-3	USAGE parameter	7-7
macros		Variable length records	2-1
ALLOC	7-1,2,3; B-6	VERIFY parameter	7-3,4
EXPND	7-1,2,5; B-9	Volume element	2-3,10;A-9
PURGE	7-1,2,5; B-11	Volume label	
SPREAD parameter	7-3,4	disc	2-2,3
Standard system suffixes	B-4	tape	2-1
STATUS macro	3-2;7-1,15; B-17	Wait for service request completion	7-30
Status word		WAIT macro	5-1;7-1,30; B-30
data channel	7-19	WRITE macro	3-2,3;7-1, 11;B-19
disc channel	7-20	Write message on SYSOUT	7-35
STRIP parameter	7-34	Writing to SYSOUT file	5-2
Suspend program execution	7-31		

MEMOREX

First Class
Permit No. 14831
Minneapolis,
Minnesota 55427

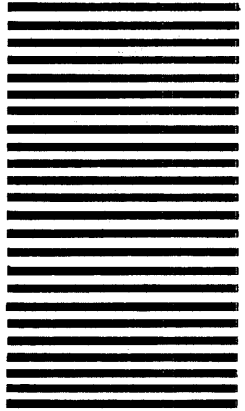
Business Reply Mail

No Postage Necessary if Mailed in the United States

Postage Will Be Paid By

Memorex Corporation

Midwest Operations – Publications
8941 Tenth Avenue North
Minneapolis, Minnesota 55427



Thank you for your information.

Our goal is to provide better, more useful manuals, and your
comments will help us to do so.

..... Memorex Publications