MRX/OS Control Program and Data Management Services

Basic Reference Manual

2200.001-01

MEMOREX

Computer System Products

November 1972 Edition

This edition (2200.001-01) is a major revision of and obsoletes, the previous preliminary edition (2200.001).

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

Comments may be addressed to the Memorex Corporation, Publications Dept., 8941 Tenth Avenue North (Golden Valley) Minneapolis, Minnesota 55427.

MRX/OS CONTROL PROGRAM AND DATA MANAGEMENT SERVICES
BASIC REFERENCE MANUAL

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. This document, 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. 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.

This document provides a detailed discussion of the basic MRX/OS facilities provided. Also included is a list of all macros and associated keyword parameters needed to execute Data Management, Control Program Services, and console communication facilities at the logical I/O level.

Additional and more detailed information may be found in the following documents.

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

TABLE OF CONTENTS

Section		Page
1	INTRODUCTION	1-1
2	RECORDS, FILES, AND LABELS	2-1
	Records	2-1
	Data Format	2-1
	Records and Record Headers	2-2
	Block	2-2
	Record Type	2-3
	Record Size	2-3
	Control Characters	2-3
	Files	2-3
	File Organization	2-4
	Sequential Files	2-4
	Relative Files	2-4
	Indexed Files	2-4
	Data Portion of Indexed File	2-4
	Index Portion of Indexed File	2-7
	File Type	2-7
	Scratch	2-7
	Temporary	2-7
	Work	2-7
	Permanent	2-7
	Summary of File Types by Organization	2-8
	Labels and Catalogs	2-8
	Tape Labels	2-8
	Standard Tape Volume Label	2-9
	Standard Tape File Label	2-9
	Disc Labels	2-10
	Disc Storage Catalogs	2-10
3	I/O PROCESSING	3-1
	Access Methods	3-1
	Sequential Access	3-1
	Random Access	3-3
	Logical Record Processing	3-3
	Buffer Switching	3-3
	Buffer Sharing	3-3
	Sequential Processing with Record Area	3-3
	Sequential Processing Without a Record Area	3-8
	Random Processing	3-8
	Magnetic Tape Processing	3-8
	Open Tape File	3-11
	Bypass and Ignore Tape Labels	3-11
	Standard Tape Labels	3-11
	Nonstandard Tape Labels	3-11
	Unlabeled Tapes	3-11

TABLE OF CONTENTS (Continued)

Section		Page
	Close Tape File	3-12
	Ignore and Bypass Tape Labels	3-12
	Standard Labels	3-12
	Nonstandard Labels and Unlabeled Tapes	3-12
	Volume Switching	3-12
	Input Files without Standard Labels	3-12
	Output Files with No Labels	3-12
	Files with Standard Labels	3-12
	IBM Tape Processing	3-12
	Interaction of Data Management and MRX/OS Control Language	3-13
4	CONTROL PROGRAM SERVICES	4-1
	Program Termination	4-1
	Time and Date Retrieval	4-1
	Console Communication	4-1
	Messages to Console	4-2
	Replies from Console	4-2
5	PROGRAM DEBUGGING	5-1
	Checkout Operation	5-1
	Requesting Debugging Services	5-2
	BREAK Directive	5-2
	PROG Directive	5-4
	Conditional Requests	5-4
	Example of Job Control Deck with Checkout Directives	5-4
6	MACROS	6-1
	Data Management Macros	6-1
	File and Label Definition Macros	6-1
	DEFSF – Define Sequential File	6-2
	DEFRF – Define Relative File	6-3
	DEFIF – Define Indexed File	6-5
	Summary of File Definition Parameters	6-7
	DEFLB – Define Label	6-7
	Logical I/O File Control Macros	6-8
	OPENL – Open Logical File	6-8
	CLOVEL – Volume Switching	6-10
	Data Transmission Macros	6-10
	GET – Get Record	6-10
	PUT – Put Record	6-12
	SETL – Set Limits	6-12
	LOCRC – Locate Record	6-13
	PUTU – Put Update	6-14
	DELR – Delete Record	6-14

TABLE OF CONTENTS (Continued)

Section		Page
	Console Communications Macros	6-15
	CONSOLE — Transmit Message to Console and Optionally Receive Reply	6-15
	MESSAGE — Set up Message Format	6-15
	Generation of an Output Message	6-15
	Generation of a Reply Buffer	6-16
	Control Program Services Macros	6-16
	Program Termination Macros	6-16
	HALT — Terminate Program	6-17
	EHALT — Terminate Program	6-17
	ABEND — Terminate Program Abnormally	6-17
	TIME — Retrieve Time of Day	6-17
	SDATE — Retrieve System Date	6-17
	JDATE — Retrieve Job Date	6-18
APPENDIX A	— LIST PARAMETER	A-1
APPENDIX B	— LINKAGE CONVENTIONS	B-1
APPENDIX C	— PROGRAM FLOW	C-1
APPENDIX D	— INDEX PORTION OF INDEXED FILE	D-1

LIST OF FIGURES

Figure		Page
2-1	Sequential File Organization	2-5
2-2	Relative File Organization	2-5
2-3	Indexed File Organization	2-6
2-4	Tape Organization	2-8
2-5	Standard Tape Volume Label	2-9
2-6	Standard Tape File Label	2-9
2-7	Disc Label	2-10
3-1	Sequential Access Method	3-2
3-2	Random Access Method	3-4
3-3	Buffer Switching with GET Request	3-5
3-4	Buffer Switching with PUT Request	3-5
3-5	GET Request with Record Area	3-6
3-6	PUT Request with Record Area	3-6
3-7	GET Request Without Record Area	3-7
3-8	PUT Request Without Record Area	3-7
3-9	Random Access with GET Macro	3-9
3-10	Random Access with PUT Macro	3-10
5-1	Job Deck with Debug Requests	5-2
5-2	Example of Use of Conditional Request	5-5
5-3	Typical Job Deck with Debugging Directives	5-6

LIST OF TABLES

Table		Page
2-1	Legal Usage by Device	2-1
2-2	File Types by Organization	2-8
3-1	Usage Conflicts	3-3
6-1	RECREG Use	6-7
6-2	File Definition Parameters	6-8
6-3	GET/PUT Macro Usage	6-11

1. INTRODUCTION

Through the use of Data Management and Control Program Services, the MRX/OS provides the easiest and most convenient coding methods for organizing and performing input/output (I/O) operations and for direct implementation by the system's Control program.

The system provides the basic user with the following facilities at the Assembler language level.

1. Data Management

   At the logical I/O level of processing, the Data Management facility provides sequential and random access methods, three file organizations (sequential, relative, and indexed), device independence, and logical record processing with and without a record area.

2. Control Program Services

   Basic Control Program Services available include time and date retrieval, and console communication.

3. Debugging

   Debugging capabilities are provided by the CHECKOUT routine which allows the programmer, through debug request cards in the Control Language statements, to call for a selection of register and main-storage dumps.

2. RECORDS, FILES, AND LABELS

Before considering how processing works within the MRX/OS, an idea of what will be processed is necessary. This section describes the data in terms of records, files, and labels, and Section 3 discusses the data in terms of processing.

The processing of data files is partially determined by the type of file organization. Sequential files can only be processed sequentially, while relative files and indexed files can be processed both sequentially and randomly. Efficiency in processing is provided by the common data formats of the file records and labeling of the files.

Throughout this section, several macros are referenced in the discussions of the records, files, and labels. Section 6 contains a detailed discussion of each macro and its function.

Table 2-1 gives the legal device usage for file organization, record type, and access.

Table 2-1. Legal Usage by Device

Type & Usage \ Device	Unit Record	Tape	Disc
Organization			
Sequential	YES	YES	YES
Relative	NO	NO	YES
Indexed	NO	NO	YES
Record Type			
Fixed	YES	YES	YES
Variable	NO	YES	YES
Access			
Sequential	YES	YES	YES
Random	NO	NO	YES

RECORDS

A record can be fixed or variable length and can vary in size up to 32K bytes of alphanumeric and/or special characters.

The actual size of a file (determined by the length and number of records), file organization, and programming requirements are all important to file processing. Variable length records can be placed on either sequential or relative files; however, the maximum record length must be determined and stated in program coding. Indexed files are limited to fixed length records.

DATA FORMAT

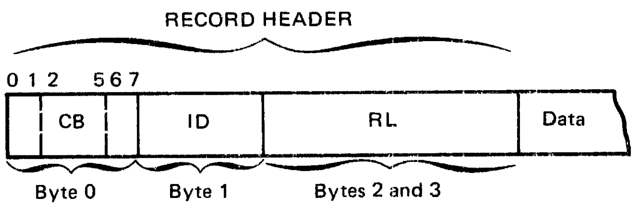
All data stored by Data Management at the logical I/O level on disc or magnetic tape is in a common format, except for absolute program images. This format includes logical record headers and optional space headers (disc files) in each block of data. The common format provides the following advantages:

- Efficiency of processing and decreased memory requirements for the GET/PUT level blocking and deblocking functions
- Increased data recoverability by making it possible to process logical records (at the physical I/O level) without reference to associated cataloging or indexing information
- More complex file organizations or access methods for which the record headers will contain additional information

The control information (record headers and space headers) is managed by the system for users working at the logical I/O level.

Records and Record Headers

A logical record is a contiguous string of user-defined data within a block of data. A record header is a 4-byte field which precedes each logical record in the block. The first byte of the record header is a record identifier which is unique to that block. The format of a record header is as follows:



CB — control byte

Bits 0 and 1 specify data type.

01	user data
10	system data (for example, catalog records)
11	user and system data (for example, index records)

Bit 2 specifies last data record for Library member only.

0	data record in library member
1	last data record in library member

Bits 3 through 5 are reserved for future system use.

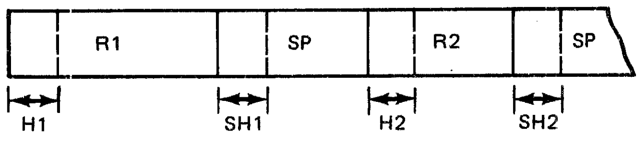
Bits 6 and 7 are always 10_2 to specify two bytes as the length of the record length field (bytes 2 and 3 of the record header).

ID — record identifier

RL — logical record length (in bytes) exclusive of record header

Block

A block is the smallest unit of data which is processed at the block I/O level. Files are collections of blocks, all of equal length on disc and unit record devices and of variable length on magnetic tape. Blocks are made up of logical records with record headers and may have unused space which is preceded by a space header. In a sequential file, records (fixed or variable length) follow in succession without space between records; available space is at the end of the block. Since indexed files are made up of fixed length records only, there is no available space between records. In a relative file, however, records may have available space between variable length records (blocks are formated in increments of the longest variable length record) as shown:



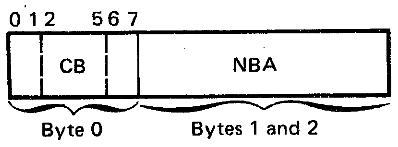
H1, H2 — record headers

R1, R2 — logical records

SH1, SH2 — space headers (optional)

SP — available space (optional)

Space headers are 1- to 3-byte fields which describe and precede unused space in blocks. The format of a space header is as follows:



CB — control byte

Bits 0 and 1 are 00 to specify available space rather than data.

Bits 2 through 5 are reserved for future system use.

Bits 6 and 7 specify the byte length of the NBA.

00	one byte of space available (only CB present)
01	two bytes of space available (CB and one byte of NBA containing zero)
10	three bytes or more of space available (CB and two bytes of NBA containing zero or the NBA)

NBA — number of bytes of space available exclusive of the header. This field may be absent, a single byte containing zero, or two bytes containing zero or the NBA.

At the logical I/O level, record headers are generated by the system when data is stored on disc or tape and are removed from the data by the system when stored data is retrieved from these media.

RECORD TYPE

The data portion of a logical record may be fixed or variable length for files residing on magnetic tape or disc. Records must be fixed length for files processed from unit record devices.

RECORD SIZE

Data Management will support (except when maximum block length for a particular device imposes lower limits) block length between 18 bytes and 32K bytes and blocking of logical records up to 256 records per block.

CONTROL CHARACTERS

The control characters for the printer and punch are included as part of the data record itself, but they are not transferred to the output units. (Therefore, if the output is to the punch, 81 characters are specified for the record size.) The character code defines the operation of the carriage control tape of a printer and the stacker selection of a card punch. The following lists the ANS control character codes and corresponding operations.

Character Code	Operation
(blank)	Space one line before printing
0	Space two lines before printing
-	Space three lines before printing
+	Suppress space before printing
1	Skip to channel 1 before printing
2	Skip to channel 2 before printing
3	Skip to channel 3 before printing
4	Skip to channel 4 before printing
5	Skip to channel 5 before printing
6	Skip to channel 6 before printing
7	Skip to channel 7 before printing
8	Skip to channel 8 before printing
9	Skip to channel 9 before printing
A	Skip to channel 10 before printing
B	Skip to channel 11 before printing
C	Skip to channel 12 before printing
V	Select stacker 1
W	Select stacker 2

The native device control characters are listed in the applicable peripheral device reference manual.

The I/O processing accepts both the ANS character codes and native device codes. The CONTROL parameter of the OPENL macro specifies which set of characters is being used.

In calculating buffer, record area, and record sizes, the appropriate number of bytes must be added for the control characters. For example, if the record size is 96 bytes, another byte is added for the control character; thus the record size and record area size is 97 bytes. However, only the 96 bytes of data will be printed or punched.

FILES

A file is a set of records containing related information. Such a set of records may be punched into cards (card file), printed on forms (printer file), or written on magnetic tape (tape file) or disc (disc file).

Files are organized in specific manners: sequential, relative, and indexed. Files can be classified according to their function within the job structure: scratch, temporary, work, or permanent. The following paragraphs describe the files according to organization and type (classification).

FILE ORGANIZATION

Files assigned to unit record and magnetic tapes are always organized as sequential files.

Random access devices, such as the disc, remove the restrictions of sequential organization and sequential processing. Disc flexibility allows for all three types of organization and, consequently, the associated methods of file processing.

Sequential Files

Sequential files apply to all media: disc storage, magnetic tape, and unit record devices. Records are written in consecutive locations for sequential file organization. The order of the records when the file is created (Figure 2-1) determines the sequence for processing the file later.

Sequential organization limits processing. Records can only be retrieved sequentially beginning with the first record of the file and subsequently retrieving each record in the file. Any changes, additions, or deletions to a magnetic tape or card file involves recopying the entire file. The additions are included in the file and the deletions are excluded. For a sequentially organized disc file, changes can be made to individual records without retrieving unchanged records, providing no additions or deletions exist.

Relative Files

The position of logical records in a relative file is given by a record number relative to the beginning of the file. The GET/PUT logic changes this record number into a block number and a relative position (record identification) within that block.

The records are written into preformated blocks on the basis of the block record number (Figure 2-2). Preformating is accomplished at file creation time by the OPENL macro. Blocks are preformated over the entire allocated space in the common stored data format with binary zeros in the logical record area. Record deletion or further maintenance of blank records is a user responsibility.

Relative file organization provides for sequential or random creation. The records are placed in a predetermined order which could be sequential or relative. The ACCESS parameter of the DEFRF macro determines the processing environment of the file.

Indexed Files

When indexed file organization is specified, two types of data are generated: the data portion of the file and the index portion of the file.

Allocation of space for the index portion is performed automatically when the data file is allocated. The user requests allocation through the Control Language DEFINE cards and may control placement of both the data portion and the index portion to the pack and cylinder level if desired (index portion may, therefore, be on same pack as data portion or on a separate pack).

The number of blocks needed for the index portion is calculated by the allocation routine from the number of data blocks estimated, the key length, and the number of keys per block specified.

Appendix D gives the layout of the index portion of the file.

Data Portion of Indexed File

The data portion of an indexed file is written into blocks that have been preformated by the OPENL macro at disc file creation. As successive blocks are written, they are scattered around each disc track so that a fixed number of blocks may separate two logically consecutive blocks on a track. The number of blocks separating logically consecutive blocks can vary from 0 to 9. Scattering is done to increase the probability that two successive data block readings can be performed without losing a disc revolution. The number of blocks which separate logically consecutive blocks is determined by a spread factor established as a file attribute through the allocation process.

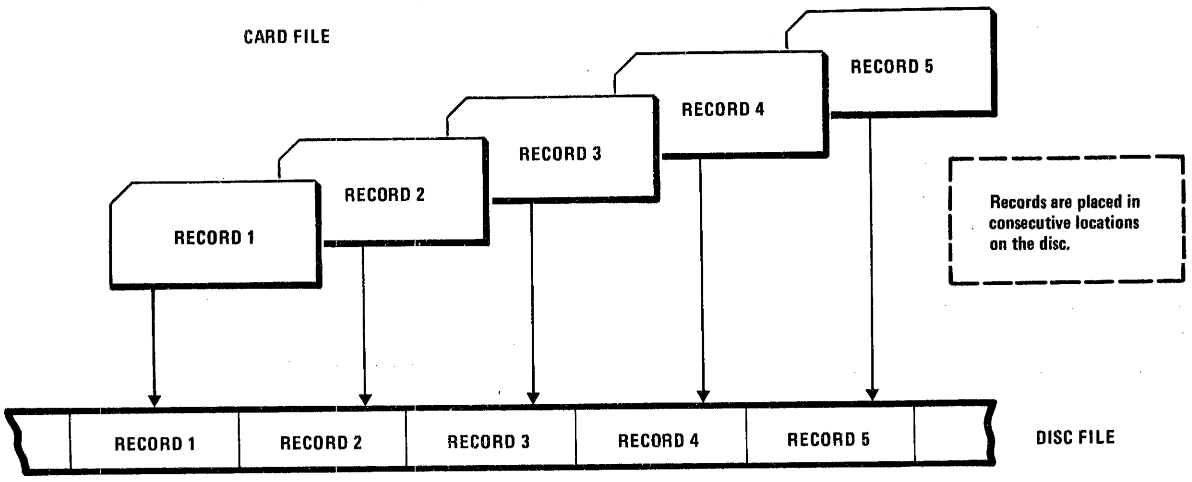


Figure 2-1. Sequential File Organization

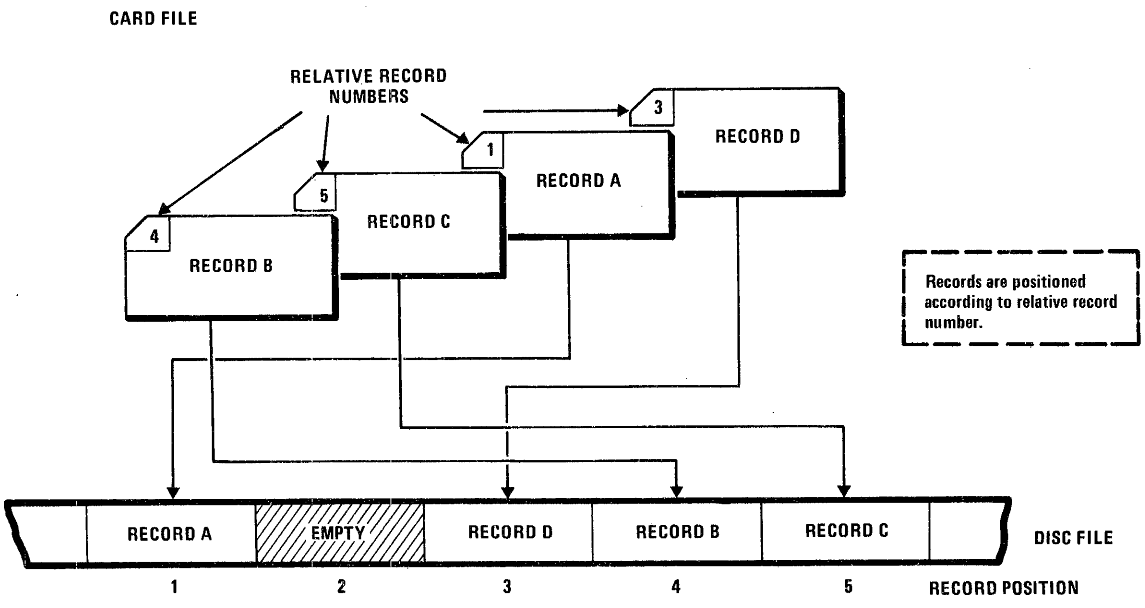


Figure 2-2. Relative File Organization

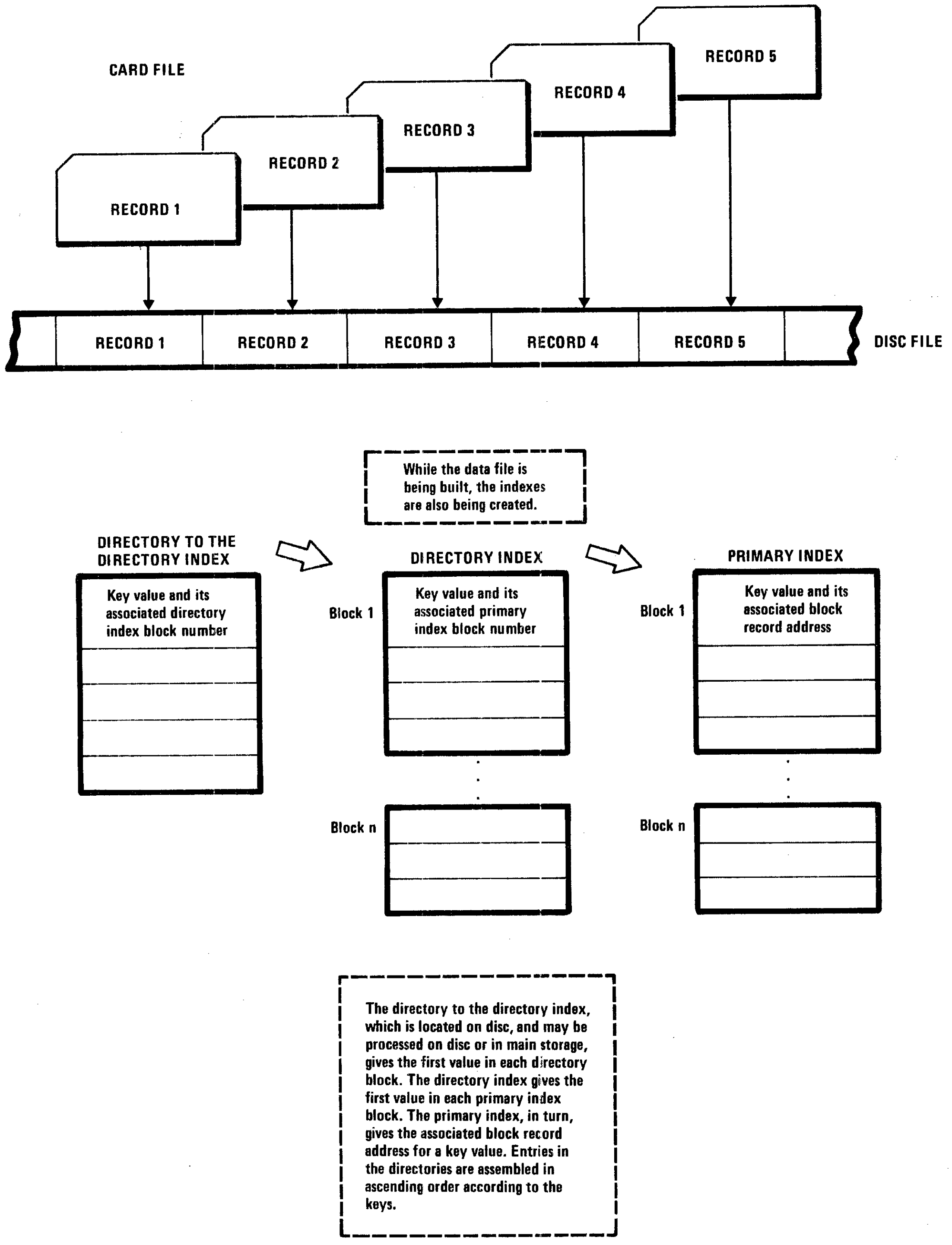

CARD FILE
RECORD 1
RECORD 2
RECORD 3
RECORD 4
RECORD 5
RECORD 1
RECORD 2
RECORD 3
RECORD 4
RECORD 5
DISC FILE
While the data file is being built, the indexes are also being created.
DIRECTORY TO THE DIRECTORY INDEX
Key value and its associated directory index block number
DIRECTORY INDEX
Block 1
Key value and its associated primary index block number
Block n
PRIMARY INDEX
Block 1
Key value and its associated block record address
Block n
The directory to the directory index, which is located on disc, and may be processed on disc or in main storage, gives the first value in each directory block. The directory index gives the first value in each primary index block. The primary index, in turn, gives the associated block record address for a key value. Entries in the directories are assembled in ascending order according to the keys.

Figure 2-3. Indexed File Organization

Index Portion of Indexed File

The locations of logical records in data files are recorded in an index portion of the file by means of a three-level table look-up indexing scheme (Figure 2-3). Each record has a unique primary index key value, which is recorded with the block record number in the primary level index block at file creation time. A directory level index, which locates the primary index blocks for random access, is also generated at file creation time. Another index, a directory to the directory level index is also generated. This additional directory is read into main storage (if specified by the INMAIN parameter of the DEFIF macro) at the opening of the file, thus increasing the efficiency of random retrieval.

Random access is performed by reading the directory to the directory index and the directory blocks, followed by an access to the appropriate primary index block. The data is then accessed by the block record number found in the primary index block.

Sequential access is performed by main-storage processing of primary index blocks. A look-ahead process for a change in block number within each index block provides the overlap of data block transfer with logical record processing. A linkage chain among primary index blocks is used to handle overflow of the primary index blocks.

Creation of indexed files requires that the data be presented in the ascending order of the collating sequence of the primary key values. As data blocks are created, the index information is generated by the GET/PUT modules for indexed files and written in the index blocks. A ten percent margin remains in each index and directory block to accommodate the addition of data records.

Indexed files may be extended by addition of more data records, presorted in the sequence of the primary index key values, if the key value of the first data record of the extension is greater than the highest key value processed during creation. This extension is performed by specifying output usage in the OPENL macro. The positioning to the end of current information and initializing tables is performed by the OPENL macro. No previous updates are allowed if the file is to be extended.

Data may be added, deleted, or updated either randomly or sequentially by opening for update usage. When data records are added they are written after the current last record in the data portion of the file. A record address is then inserted into the appropriate primary index block. If additions of many records in the same region of primary index causes the primary index block to overflow, a new block is linked into the first level linkage chain and some of the record addresses in the block which overflowed are moved into this new block to minimize the chance of further overflow in the same area.

When data is deleted, the index information for the deleted record is removed, but the data record is not modified.

FILE TYPE

Each file has a disposition type (scratch, temporary, work, or permanent) established when space is allocated for the file.

Scratch

Scratch files are used by jobs that have several job steps. These files exist only for the duration of a job step and, at the end of the job step, are purged by system control. The file is positioned at the beginning of the file by the OPENL macro. Scratch files cannot have an indexed file organization.

Temporary

Temporary files provide a temporary storage area for job processing. They exist only for the duration of the job and, at the end of the job, are purged by system control. The file is positioned at the beginning of the file by the OPENL macro. Temporary files cannot have an indexed file organization.

Work

Work files occupy permanently allocated space; however, the file data exists only for the duration of a job. If a work file is opened for output use, the pointer for placement of the data is positioned to the beginning of the file. Work files must have sequential file organization.

Permanent

Permanent files occupy permanently allocated space and the file data is permanent. If a permanent sequential file is opened for output use, the data is added to the end of the file. For relatively organized files, the records are added wherever specified. For indexed files data is added to the end and must be presented in the collating sequence of the primary key beginning above the current high key value.

SUMMARY OF FILE TYPES BY ORGANIZATION

Table 2-2 provides an easy reference to the file types allowed for each file organization. It notes how records are added to permanent files.

Table 2-2. File Types By Organization

File Type \ File Organization	Sequential	Relative	Indexed
Scratch	Legal	Legal	Illegal
Temporary	Legal	Legal	Illegal
Work	Legal	Illegal	Illegal
Permanent	Legal	Legal	Legal
	Records are added to the end of file	Records are added wherever user specifies	Records are added to the end of file

LABELS AND CATALOGS

File labels ensure the opening of a correct file for a particular job. The OPENL macro checks the file for the proper name and attributes. Since Data Management handles labels and catalogs, the basic user need not be concerned with the physical aspects of label and catalog checking (the Control Program and Data Management Services, Extended Reference manual provides this information).

TAPE LABELS

Tape labels are special records at the beginning and end of magnetic tape files (Figure 2-4). They are used to identify the reel of tape and the succeeding file, as well as certain information for file maintenance. All labels may not have the same format. Some tapes may be unlabeled and others may have user-supplied nonstandard label formats.

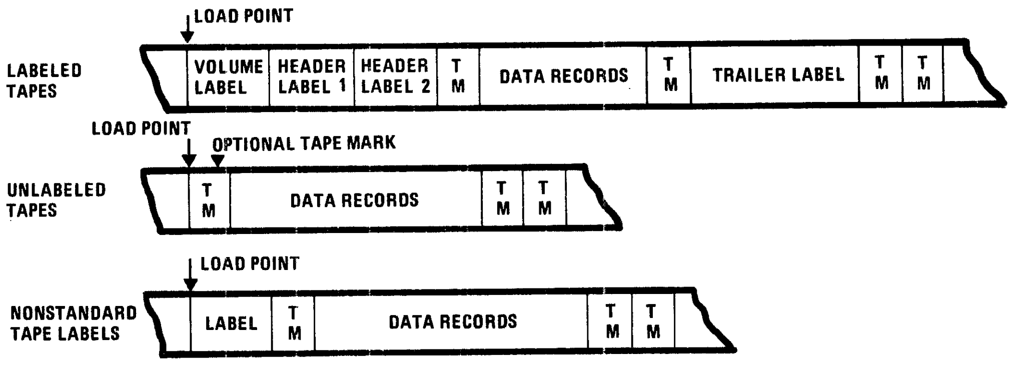


Figure 2-4. Tape Organization

Standard Tape Volume Label

The volume label is located at the beginning of a tape reel and is identified by the characters, VOL, found in the first three positions (Figure 2-5). The volume label number is always 1 for compatibility with IBM. The volume serial number occupies positions 4 through 9 and identifies the volume. A unique owner name and address code identifies the installation.

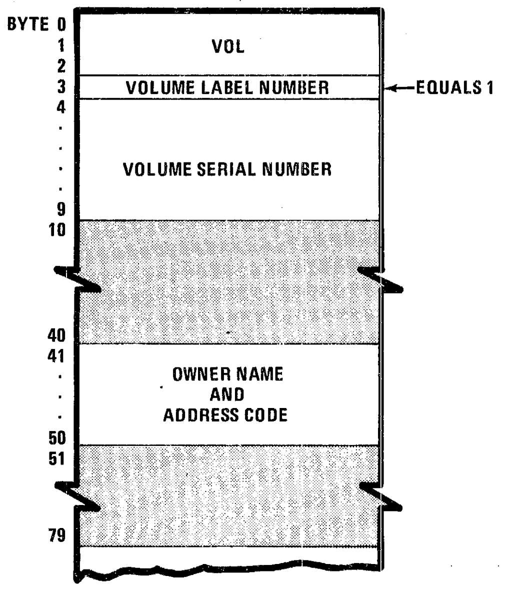


Figure 2-5. Standard Tape Volume Label

Standard Tape File Label

The standard tape file label provides information concerning the user's file such as creation and expiration dates, file name, and sequence number (Figure 2-6). The label identification field identifies the type of standard label with a three letter abbreviation. Three types of labels supported by the system are header labels (HDR), end-of-file labels (EOF), and end-of-volume labels (EOV). The file label number found in byte 3 is 1 or 2. The file serial number, found in positions 21-26, is identical to the volume serial number in the volume label of the first volume. The volume sequence number identifies the order of the volume of data records in a multivolume logical file. The block count provides the number of physical records written in a file at creation.

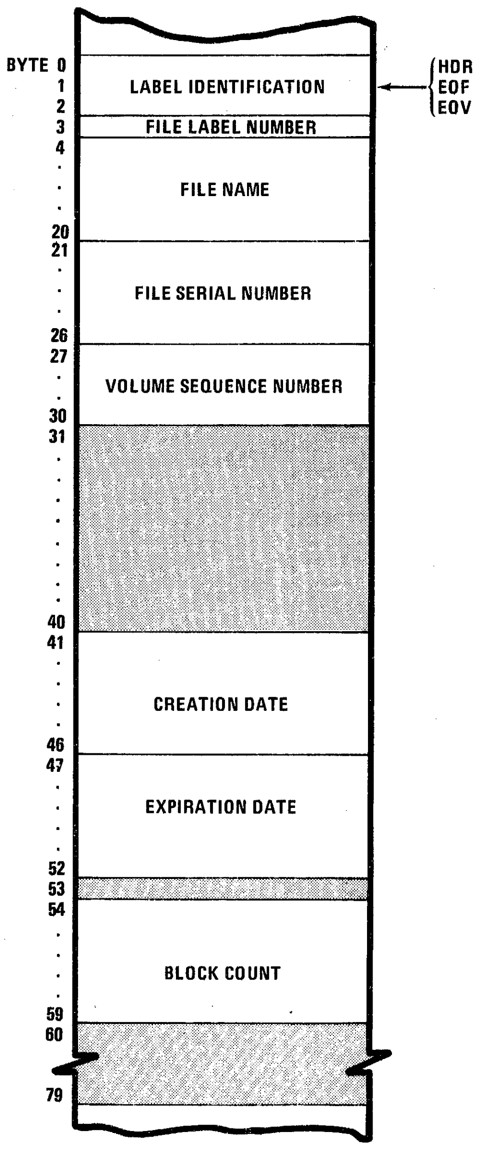


Figure 2-6. Standard Tape File Label

DISC LABELS

Disc packs have a standard volume label containing volume identification and owner information (Figure 2-7). The device type, owner, and state of the volume – unrestricted (0), restricted (1), or locked (2) – are included along with the starting track address of the pack catalog for this volume. The actual file identification is found in the disc catalogs.

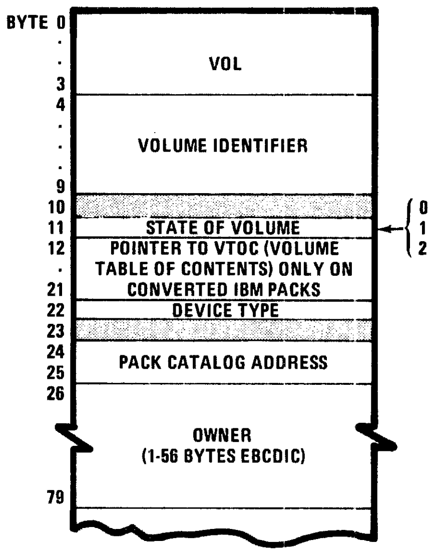


Figure 2-7. Disc Label

Disc files are described by file labels found in the pack catalogs, which are resident on each disc pack in the system. The file labels contain the file name, file attributes, and a physical description of the segments of disc space allocated to the file on this pack. The actual fields of the catalogs can be found in the Control Program and Data Management Services, Extended Reference manual.

DISC STORAGE CATALOGS

The space management routines (explained in the Control Program and Data Management Services, Extended Reference manual), in performing their function, maintain the central catalog and pack catalogs on disc. 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 volume(s) occupied by the file.

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.

3. I/O PROCESSING

The I/O processing capabilities of MRX/OS are divided into three levels: logical, block, and physical. The logical I/O processing level, discussed in this section, blocks and deblocks data for the user; block I/O processing recognizes no logical records (data is read or written as a data block); physical I/O allows the user to do his own I/O processing. The block and physical I/O processing levels are discussed in detail in the Control Program and Data Management Services, Extended Reference manual.

The logical I/O level Data Management functions are provided by relocatable subroutines which are loaded with and linked to the user programs which request them. They require MRX/OS control program, system control, loader, and library functions.

ACCESS METHODS

Two access methods are provided at the logical I/O level, sequential and random.

SEQUENTIAL ACCESS

Sequential access is a method of retrieving records consecutively in the order of logical arrangement in the file. All file organizations are sequentially accessible.

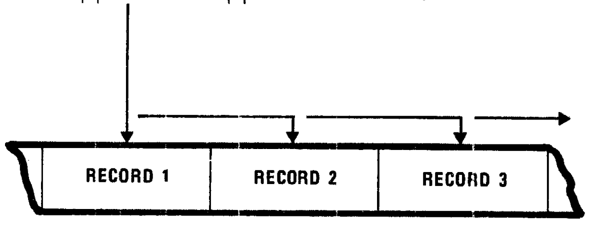
Sequential access is the only method that can be used for retrieving sequentially organized file records. Relatively organized file records can be processed sequentially by beginning at the start of the file and processing each successive record. Indexed file records also can be processed sequentially by retrieving in the sequence of the primary key values. By using the SETL macro, all the records in an indexed file may be retrieved starting at the beginning of file, or a portion of the file may be retrieved by selecting a key value for the beginning of the processing. The SETL macro, however, is not necessary to start processing at the beginning of an indexed file; Data Management will process from the beginning unless otherwise specified.

Figure 3-1 illustrates the sequential access method.

SEQUENTIAL OR RELATIVE FILE

Sequential File
or
Relative File

```
FILE1    DEFSF    BLKSIZ=100,RECSIZ=96,RECTYP =F,BLKFAC=1
                          or
FILE1    DEFRF    ACCESS=S,BLKSIZ=100,RECSIZ=96,;
                  RECTYP=F
                    :
         GET      IDENT=FILE1,RECADR=AREA1,RTNADR=ISIT
```

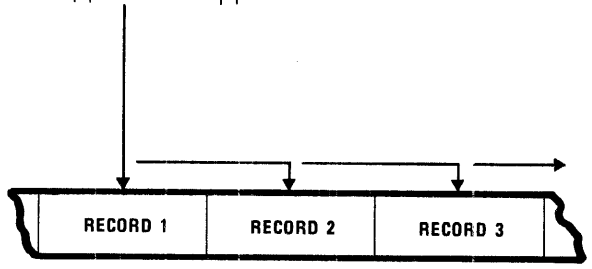


Retrieves first record and each succeeding record.

INDEXED FILE

From beginning of file

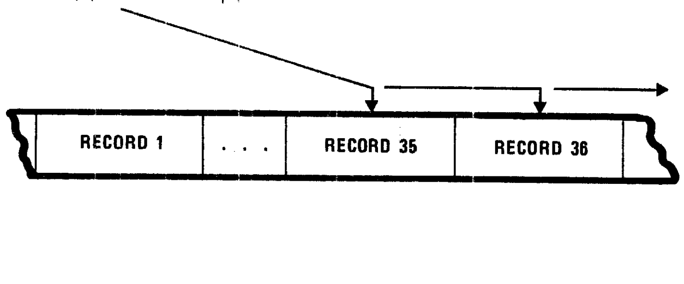
```
FILE2    DEFIF    ACCESS=S,BLKSIZ=100,RECADR=AREA2,;
                  RECSIZ=96,INDSIZ=50,KEYSIZ=3
                    :
         SETL     IDENT=FILE2,START=BOF
         GET      IDXSA=YES,IDENT=FILE2,RTNADR=ISIT
```



Retrieves first record and each succeeding record. Allocated with a spread factor of one.

From selected file record

```
FILE3    DEFIF    ACCESS=S,BLKSIZ=100,KEYADR2=SKEY,;
                  RECADR=AREA2,RECSIZ=96,INDSIZ=50,KEYSIZ=3
                    :
SKEY     BDD      35(3,1)
                    :
         SETL     IDENT=FILE3,START=KEY
         GET      IDXSA=YES,IDENT=FILE3,RTNADR=ISIT
```



Retrieves record specified in key field and each succeeding record. Allocated with a spread factor of one.

Figure 3-1. Sequential Access Method

RANDOM ACCESS

Random access is a method of retrieving individual records from a file. Only relative and indexed file records are randomly accessible.

For relative files, the record number of the desired record is changed into a block number and a relative position within that block. From this computed information the record is directly accessible.

For indexed files, the key values specified as the actual keys indicate the desired records. A search in the directory to the directory index block, the directory index block, and the primary index block gives the appropriate block record number of the desired record. The data record is then retrieved by the block record number.

Figure 3-2 illustrates the random access method.

LOGICAL RECORD PROCESSING

Blocking/deblocking of logical records is provided by GET/PUT macros, which move logical records between a record area and an I/O buffer or which process records in an I/O buffer. Processing may be either sequential or random with a record area and one or two I/O buffers defined.

BUFFER SWITCHING

If two buffers are specified, a buffer switching process is performed. For a GET request, as the records are transferred from the second buffer, the next data block is simultaneously transferred into the first buffer. Figure 3-3 illustrates the buffer switching involved for a GET request.

Buffer switching for PUT is similar to GET but in reverse order. Records which have been processed in the record area are transferred to the appropriate I/O buffer. As soon as the last record* has been transferred to buffer 1, the next record is transferred to buffer 2, and simultaneously buffer 1 has its records put on the output data file in the appropriate block. Figure 3-4 illustrates buffer switching for PUT requests.

* For variable length records, this is the last whole record the buffer can contain.

Buffer switching increases efficiency of processing because reading and processing operations occur simultaneously with no waiting involved.

FILE SHARING

OPENL checks shared files on volumes which are mounted on shared resources. Output usage ensures exclusive use of the file. Update usage ensures that no other program can update the file until it is closed, but other programs may have access to the file. Table 3-1 indicates the legality of usage when program 1 opened the file and an OPENL macro from program 2 is checking for usage conflicts.

Table 3-1. Usage Conflicts

Program 1 Usage / Program 2 Usage	Input	Update	Output
Legal	Input Update	Input	None
Illegal	Output	Update Output	Input Update Output

BUFFER SHARING

When buffer sharing has been specified in the define macro, an additional two bytes must be added to the buffer size. This 2-byte field is used by OPENL and CLOSEL to resolve buffer usage conflicts.

SEQUENTIAL PROCESSING WITH RECORD AREA

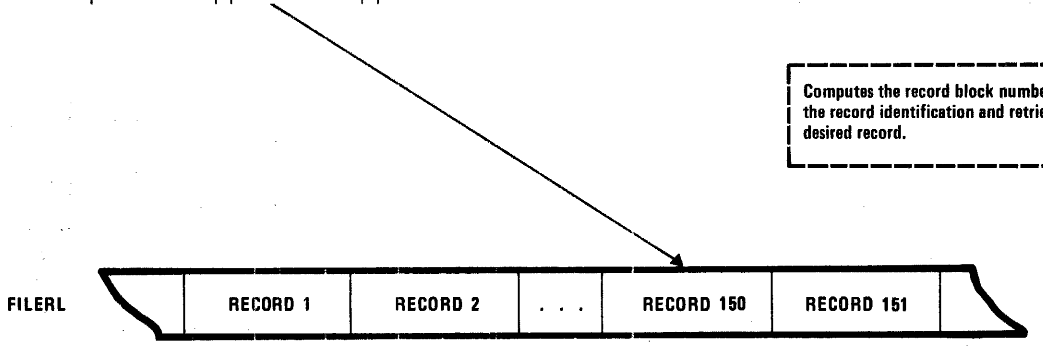
The GET macro transfers a block of input data into an I/O buffer. From the I/O buffer, the logical record goes to a record area. When the last logical record in a data block has moved to a record area, the buffer is refilled. If a second I/O buffer is specified, buffer switching is performed; control returns to the user when the buffer switching is completed. Figure 3-5 illustrates the GET request using both a record area and a buffer.

The PUT macro moves a logical record from a record area into an I/O buffer. Buffers are emptied after the last record has been put into the current I/O buffer or when

RELATIVE FILE

```
NAME      OPERATION  OPERAND
FILERL    DEFRF      ACCESS=R,BLKSIZ=100,KEYADR=RKEY,;
                     RECSIZ=96,RECTYP=F
              :
RKEY      WDD        150
              :
          GET        IDENT=FILERL,RECADR=AREA3,RTNADR=MIST
```

Computes the record block number from the record identification and retrieves the desired record.



INDEXED FILE

```
NAME      OPERATION  OPERAND
FILEIN    DEFIF      ACCESS=R,BLKSIZ=100,KEYADR1=IKEY,;
                     RECADR=AREA4,RECSIZ=96,KEYSIZ=4,INDSIZ=50
              :
IKEY      BDD        750(4,1)
              :
          GET        IDENT=FILEIN,RTNADR=MIST
```

Seeks the record through an indexing scheme. The desired key value is found in the three index blocks and the primary index pointer gives the location of the record on mass storage.

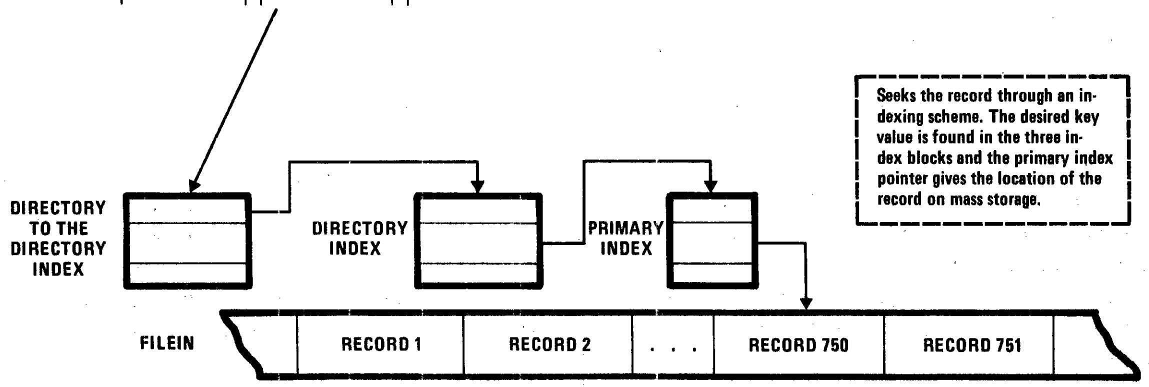


Figure 3-2. Random Access Method

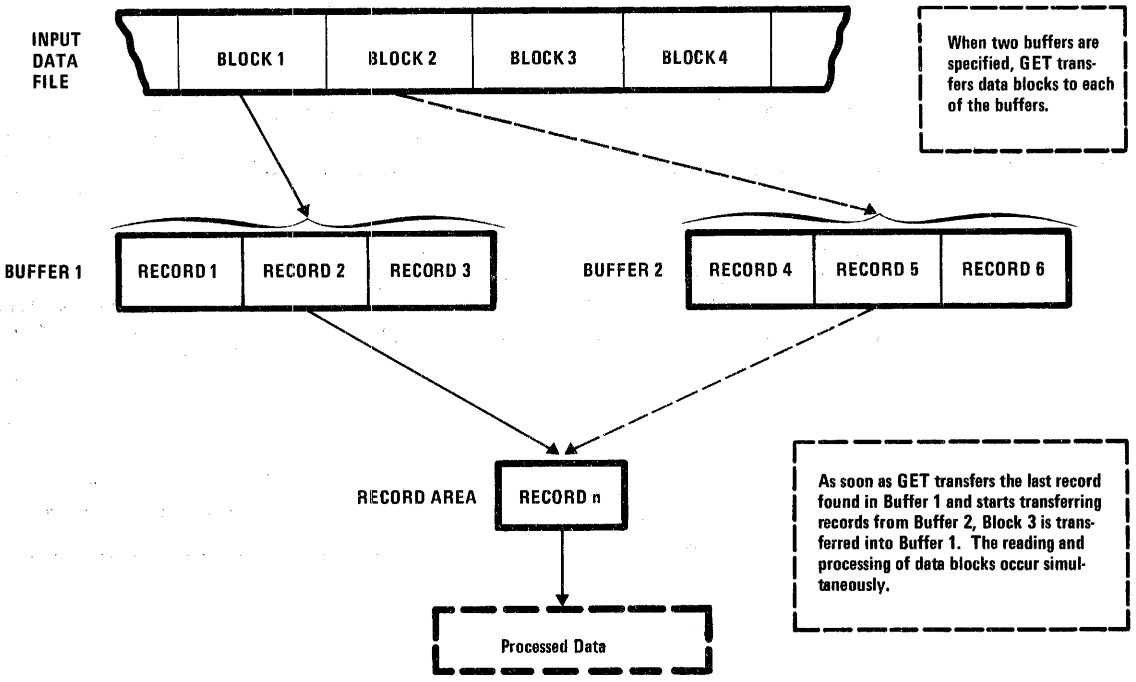

INPUT DATA FILE
BLOCK 1
BLOCK 2
BLOCK 3
BLOCK 4
When two buffers are specified, GET transfers data blocks to each of the buffers.
BUFFER 1
RECORD 1
RECORD 2
RECORD 3
BUFFER 2
RECORD 4
RECORD 5
RECORD 6
RECORD AREA
RECORD n
As soon as GET transfers the last record found in Buffer 1 and starts transferring records from Buffer 2, Block 3 is transferred into Buffer 1. The reading and processing of data blocks occur simultaneously.
Processed Data

Figure 3-3. Buffer Switching with GET Request

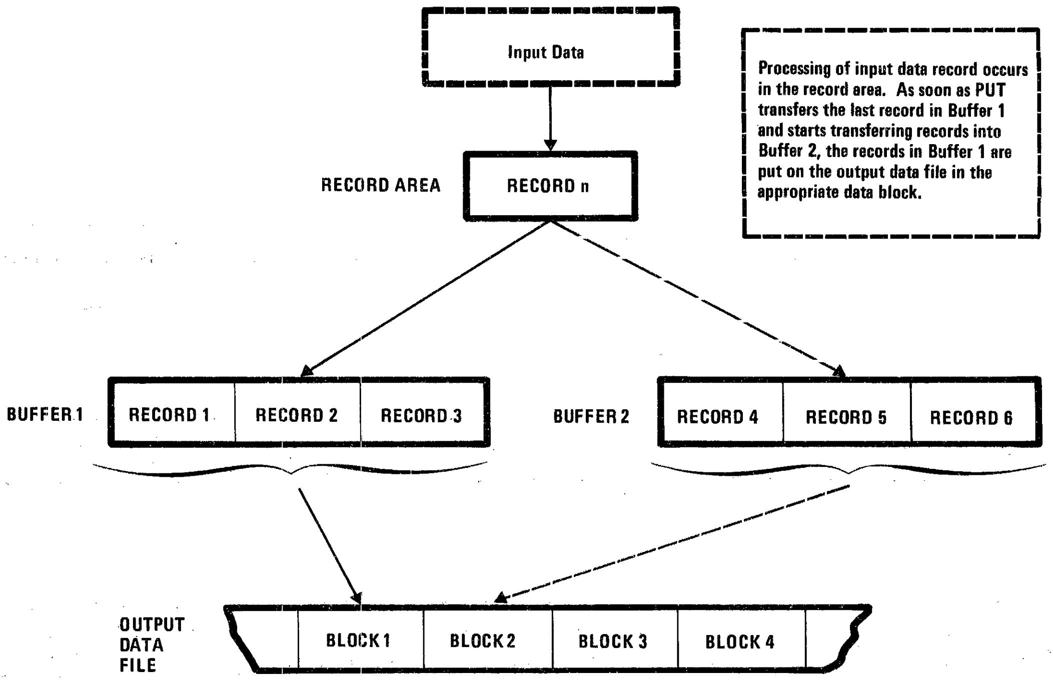

Input Data
Processing of input data record occurs in the record area. As soon as PUT transfers the last record in Buffer 1 and starts transferring records into Buffer 2, the records in Buffer 1 are put on the output data file in the appropriate data block.
RECORD AREA
RECORD n
BUFFER 1
RECORD 1
RECORD 2
RECORD 3
BUFFER 2
RECORD 4
RECORD 5
RECORD 6
OUTPUT DATA FILE
BLOCK 1
BLOCK 2
BLOCK 3
BLOCK 4

Figure 3-4. Buffer Switching with PUT Request

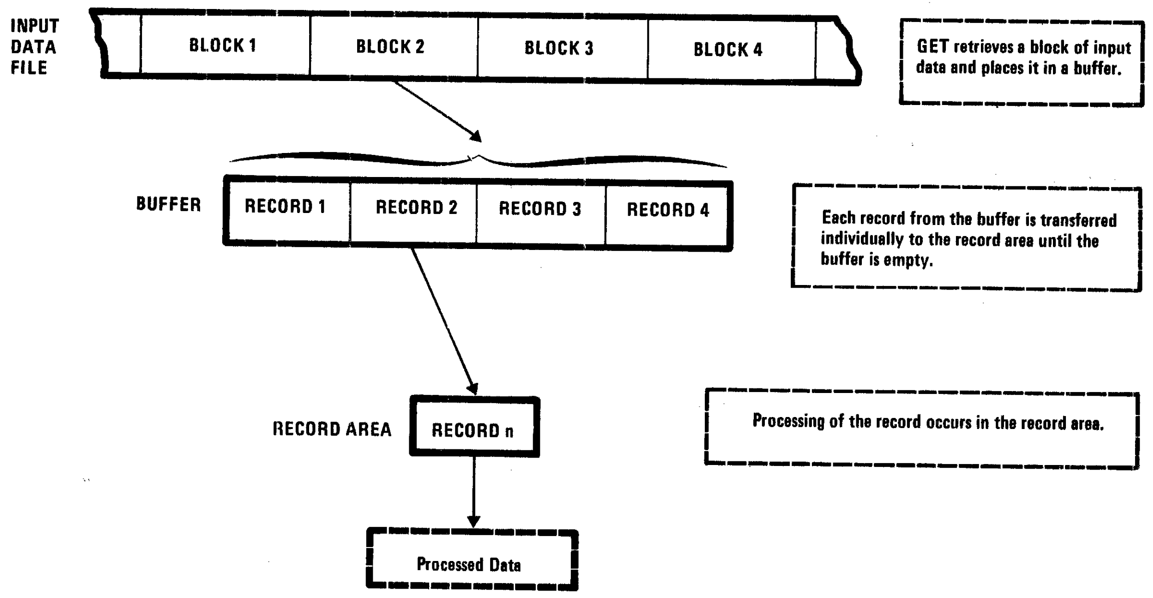

INPUT DATA FILE
BLOCK 1
BLOCK 2
BLOCK 3
BLOCK 4
GET retrieves a block of input data and places it in a buffer.
BUFFER
RECORD 1
RECORD 2
RECORD 3
RECORD 4
Each record from the buffer is transferred individually to the record area until the buffer is empty.
RECORD AREA
RECORD n
Processing of the record occurs in the record area.
Processed Data

Figure 3-5. GET Request with Record Area

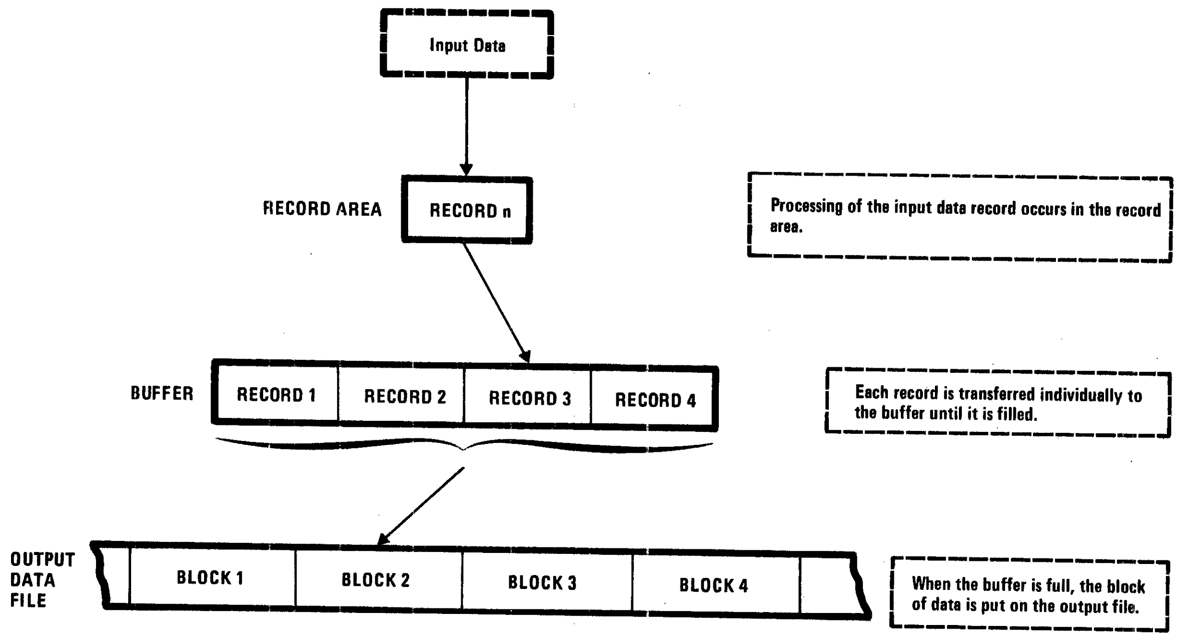

Input Data
RECORD AREA
RECORD n
Processing of the input data record occurs in the record area.
BUFFER
RECORD 1
RECORD 2
RECORD 3
RECORD 4
Each record is transferred individually to the buffer until it is filled.
OUTPUT DATA FILE
BLOCK 1
BLOCK 2
BLOCK 3
BLOCK 4
When the buffer is full, the block of data is put on the output file.

Figure 3-6. PUT Request with Record Area

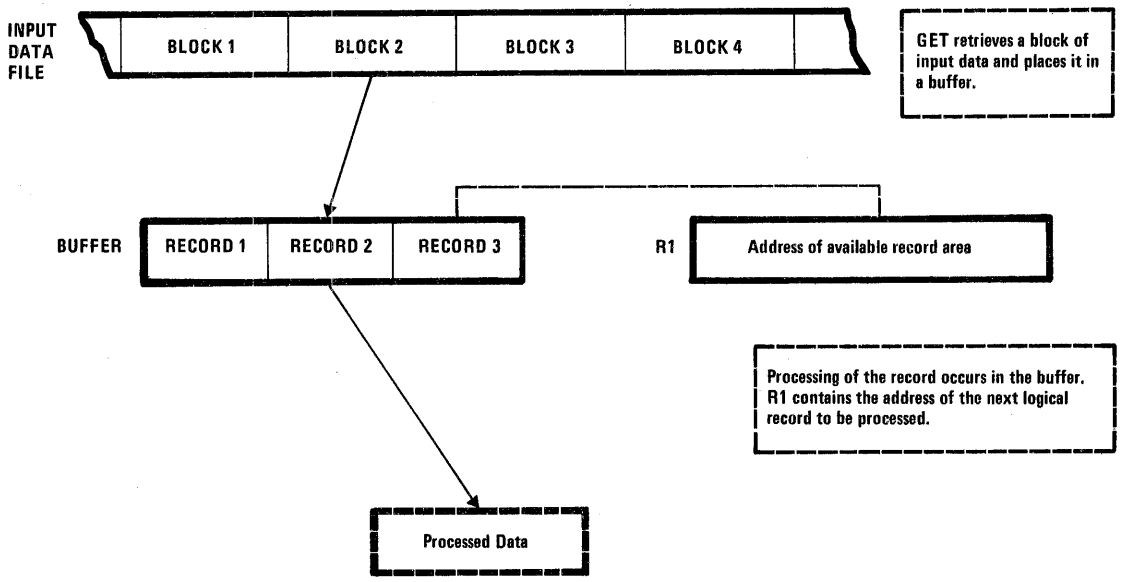

INPUT DATA FILE
BLOCK 1
BLOCK 2
BLOCK 3
BLOCK 4
GET retrieves a block of input data and places it in a buffer.
BUFFER
RECORD 1
RECORD 2
RECORD 3
R1
Address of available record area
Processing of the record occurs in the buffer. R1 contains the address of the next logical record to be processed.
Processed Data

Figure 3-7. GET Request Without Record Area

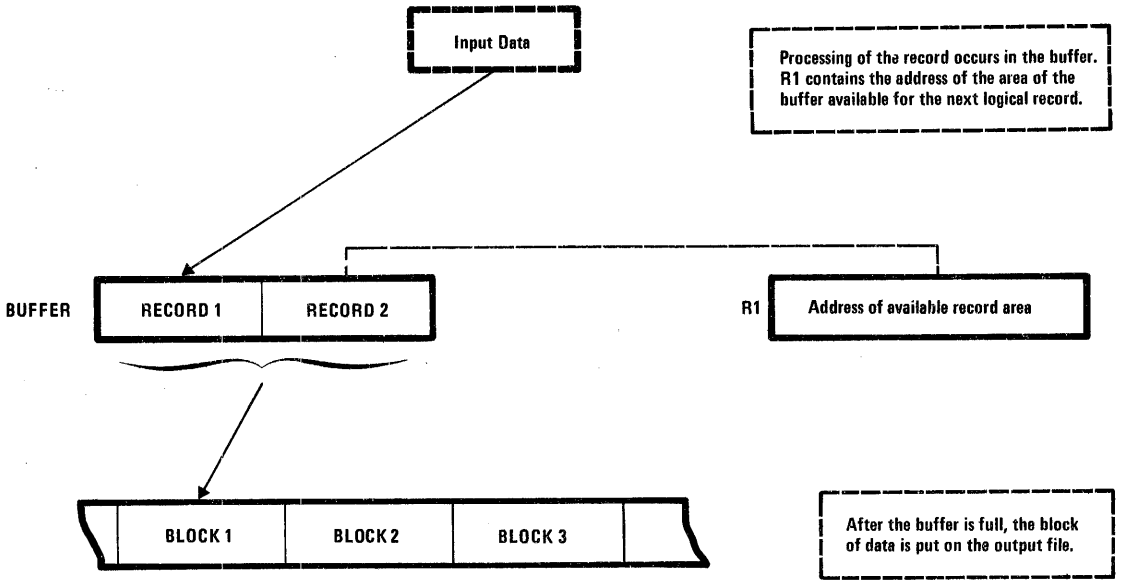

Input Data
Processing of the record occurs in the buffer. R1 contains the address of the area of the buffer available for the next logical record.
BUFFER
RECORD 1
RECORD 2
R1
Address of available record area
BLOCK 1
BLOCK 2
BLOCK 3
After the buffer is full, the block of data is put on the output file.

Figure 3-8. PUT Request Without Record Area

the space left in the current I/O buffer will not contain the next variable length record. If a second I/O buffer is specified, buffer switching is performed before returning control to the user; if a second I/O buffer is not specified with variable length records, the next variable length record is not moved into the buffer until it has been emptied. Figure 3-6 illustrates the PUT request using both a record area and a buffer.

SEQUENTIAL PROCESSING WITHOUT A RECORD AREA

GET returns to the user, in general-purpose register 1 (R1), the address of the first data character of a logical record in the buffer which is available for processing. Buffers are refilled after the GET request for the first logical record of a block is made. If a second I/O buffer is specified, buffer switching is performed; control returns to the user upon switching completion. If there is only one I/O buffer specified, control is retained by Data Management until the buffer has been filled. Figure 3-7 illustrates the GET request using only an I/O buffer for processing.

PUT returns to the user, in R1, the first character address in the buffer which is available for the next logical record. Buffers are emptied after the PUT for the first logical record in the block is made. If a second I/O buffer is specified, buffer switching is performed; control returns to the user upon switching completion. If there is only one I/O buffer, control is retained by Data Management until the buffer has been emptied. Figure 3-8 illustrates the PUT request using only an I/O buffer for processing.

RANDOM PROCESSING

While in the random mode of processing, GET retrieves a specified record. The buffer is filled if the specified record is not currently in the I/O buffer. If a second I/O buffer is specified, it is not used.

When a record area is used, the GET request moves the requested record through the I/O buffer to the record area; however, if only a buffer is available, the first character address of the record in the I/O buffer is returned to the user in R1. Figure 3-9 illustrates the processing of a randomly retrieved record with and without a record area.

When a record area has been specified, the PUT macro empties the buffer if it contains an updated record but does not contain the specified record. The buffer is then filled with the data block that is to contain the new record; the record is moved into the buffer. If this is an update process, the data block of this particular record must be in the buffer before PUT transfers the record. (Thus a GET request is issued first.)

While in random mode, PUT macros are processed only when a record area has been specified. Figure 3-10 illustrates the processing of the PUT macro while in random mode.

MAGNETIC TAPE PROCESSING

MRX/OS supports three types of labels:

- Standard labeled tape
- Nonstandard labeled tape
- Unlabeled tape

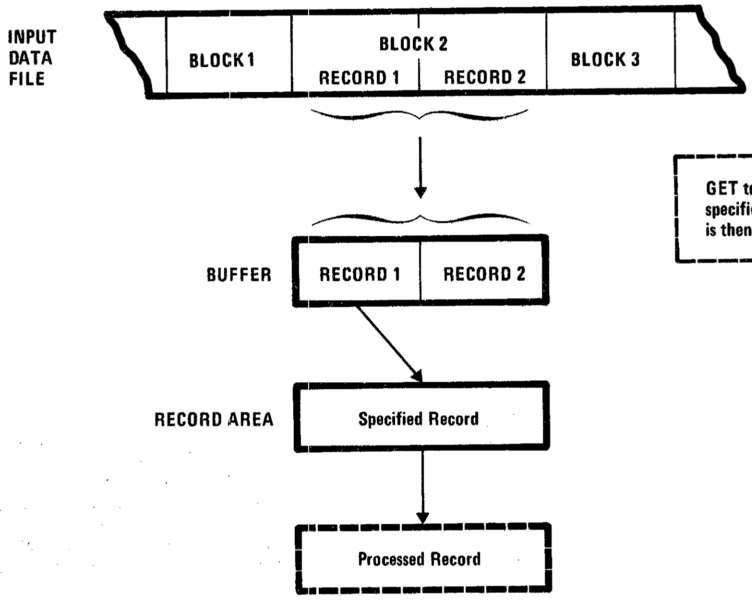
The default for no LABEL specification is standard labels. The nonstandard option is valid only for input files.

The label type may be selected at assembly-time through the optional LABTYP parameter of the DEFSF macro or at run-time through the LABEL parameter of the //DEFINE statement. However, the LABEL parameter of the //DEFINE statement overrides any label specification in the BDT which is created by the DEFSF macro.

On the //DEFINE card, the Control Language specifies the type of magnetic tape labels, plus two additional features - ignore and bypass label processing. The LABEL keyword parameter has the following format:

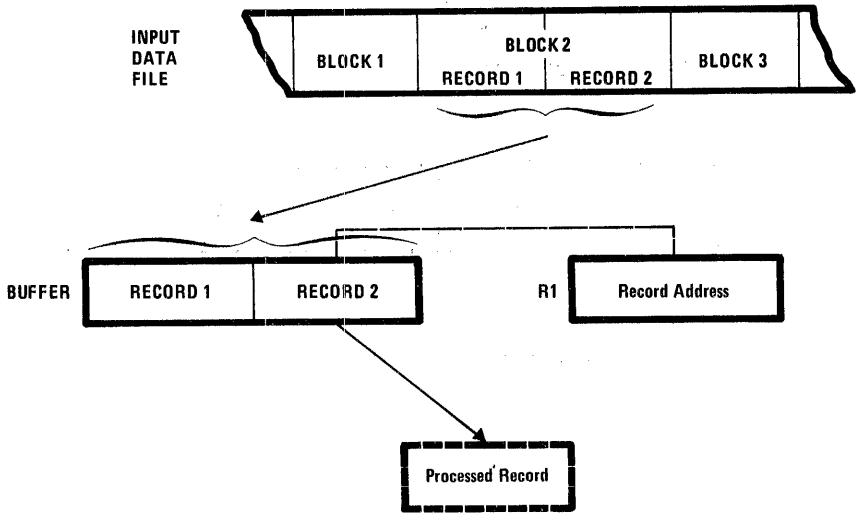
$$\text{LABEL=}\left\{\begin{matrix}\text{S}\\\text{N}\\\text{U}\\\text{B}\\\text{I}\end{matrix}\right\}$$

GET REQUEST WITH RECORD AREA



GET transfers the data block containing the specified record to the I/O buffer. This record is then placed in the record area for processing.

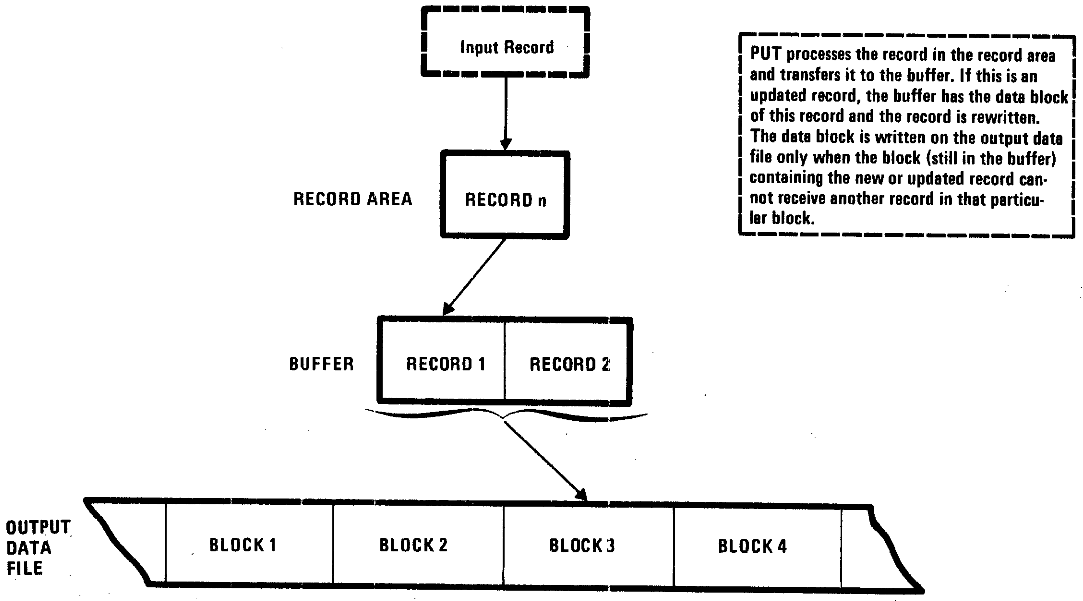
GET REQUEST WITHOUT RECORD AREA



GET retrieves the data block and places it in the I/O buffer. R1 gives the first character address of the specified record in the I/O buffer.

Figure 3-9. Random Access with GET Macro

PUT REQUEST WITH RECORD AREA



PUT processes the record in the record area and transfers it to the buffer. If this is an updated record, the buffer has the data block of this record and the record is rewritten. The data block is written on the output data file only when the block (still in the buffer) containing the new or updated record cannot receive another record in that particular block.

Figure 3-10. Random Access with PUT Macro

The codes are defined as:

Code	Definition
S	Standard tape labels
N	Nonstandard tape labels
U	Unlabeled tapes
B	Bypass label processing
I	Ignore label processing

Chapter 2 of this manual describes the format of standard labels, nonstandard labels and no labels.

OPEN TAPE FILE

Each OPEN(L) request for a tape unit must be supported by a //DEFINE Control Language statement. The SYSIN (system input file) is searched using the IDENT field supplied by the OPEN macro. If no match is found, an error is returned.

Tape units are assigned for exclusive use of a job-step by the Step Initiator. OPEN(L) issues a mount reel request, if necessary, for each unit (1 or 2 per IDENT) assigned by the Step Initiator.* The message includes volume identification, reel sequence number and unit number. If REWIND=NO is specified in the OPEN(L) macro, no rewind is performed before checking labels.

After label checking, an FDT (File Description Table) is built and linked into the FDT string. If the file is open for logical I/O processing, buffers are prefilled, and control returns to the caller.

Bypass and Ignore Tape Labels

The bypass and ignore options of the LABEL parameter are valid for input files only. If ignore (I) is selected, no tape positioning or label checking occurs. If bypass (B) is selected, no label checking between existing label and the Control Language options occurs.

When the bypass option is used, the first tape record is read. If it is a tape mark, the tape is an unlabeled one and is positioned at the first data record and no further positioning occurs. If the first record is a standard tape volume label, it is a standard labeled tape and a forward space file routine positions the tape to the first record beyond the label set. If the first record was neither a tape mark nor a standard tape volume label, an unlabeled tape is assumed and a backspace record routine positions the tape to the beginning. The user is then responsible for correct positioning of the tape with nonstandard labels.

*Step Initiator is a module of Control Language Services.

Input files must conform to the label requirements; this allows no operator override. If a conflict exists, a rewind request is issued and the operator must mount the correct tape reel.

Standard Tape Labels

When the input file has standard labels, the volume serial number of the tape volume label must match the VOLUME parameter of the //DEFINE statement; and the filename of the first header label (standard tape file label) must match the FILENAME parameter of the //DEFINE statement. A forward space file routine then positions the first data record. If the filename in the header does not match the FILENAME parameter, the operator receives a message to retry or abort.

When the standard labels option is selected for an output file, standard labels both volume and header labels must be written on the tape. If the volume label does not exist on the tape, a message is issued to the operator who must respond with RETRY or the information necessary to create a standard volume label. A standard tape file label (HDR1) is created from the Control Language information. A second header label (HDR2) is written with all fields of zero and followed by a tape mark. If the standard volume label does exist, the first header label is read. If HDR1 is valid, the expiration date is checked. For unexpired files, a message is issued; and the operator responds with RETRY or ACCEPT. RETRY starts the sequence again and ACCEPT allows the volume to be used as if expired.

Nonstandard Tape Labels

The nonstandard label option is only valid for input files. A read of the first record on the tape is issued. If the record is not a standard volume label, a forward space file positions the type beyond the header label set.

Unlabeled Tapes

The first record of an unlabeled input file is read. If it is a tape mark, the first data record is in position. If it is not a tape mark, a backspace record routine positions the first data record.

For output files, the first record is read. If the record is a standard volume label, the next record is read. If the next record is not a HDR1 label, the file is backspaced and a tape mark is written. If the second record is a HDR1 label, the expiration date is checked. If the file is expired, the file is backspaced and a tape mark written; a message informs the operator that a standard volume label is being overwritten. If HDR1 is not expired, a message is issued, and the operator may ACCEPT or RETRY. If the record is a tape mark, the correct record is in position. If the first record is not a tape mark nor a volume label, the tape is backspaced and a tape mark written.

CLOSE TAPE FILE

At the end of processing a tape file certain steps are taken according to the option selected.

After any label processing is done, a rewind/unload is issued unless a CLOSE(L) with REWIND=NO was specified. The FDT is taken out of the FDT string unless CLOSE(L) with LOCK=YES was specified. Control is then returned to the caller.

Ignore and Bypass Tape Labels

When the ignore or bypass option is selected, no trailer label checking is performed.

Standard Labels

When processing an input file with standard labels, the last record (which should be an EOF record) is read. If it is not an EOF and the FDT indicates the end of file, a message is issued to the operator, and no comparison of block count (FDT vs. trailer label) is made. If the record is an EOF, the trailer label block count is compared with the block count for the volume; a console message indicates an error when the two block counts are unequal.

After writing the last data record in an output file, a tape mark, an EOF trailer label with block count, and two tape marks are written.

Nonstandard Labels and Unlabeled Tapes

When processing input file with either nonstandard tape labels or no tape labels, no label checking occurs. After completion of writing an unlabeled output file, two tape marks are written.

VOLUME SWITCHING

The CLOVE(L) macro performs volume switching for multivolume tape files. CLOVE(L) performs header and trailer label processing on tapes and alternate unit processing on tapes.

Input Files Without Standard Labels

While processing input files with the ignore, bypass, nonstandard, or unlabeled option, the operator must indicate if there are more volumes to the tape file. If this is the last input volume, EOF is set in the CLOVE packet, and control returns to the caller. If it is not the last volume, label processing for this volume is the same as in CLOSE. A REWIND=UNLOAD and a request to mount the next volume are issued. Label checking for the next volume follows the OPEN procedure.

Output Files With No Labels

After processing an unlabeled output tape, two tape marks are written and a mount the next volume is requested. Label processing occurs for the next volume as in the open routine.

Files with Standard Labels

When the files have standard labels, the current volume processing is the same as CLOSE except the last record is an EOV instead of EOF. A mount message is issued for the next volume, and the processing continues as in OPEN.

After label checking is complete, the FDT is updated if necessary and control returns to the caller.

IBM TAPE PROCESSING

A mode of processing which does not use the common stored data format (described in Chapter 2) will be available for processing of IBM format F and U magnetic tapes. This mode is selected on tape by specifying CSD=NO in the //DEFINE statement and by specifying BLKFAC=n for format F and BLKFAC=1 for format U in the DEFSF macro.

INTERACTION OF DATA MANAGEMENT AND MRX/OS CONTROL LANGUAGE

The Control Language Services, through the //DEFINE statement of the Control Language, provides a run-time interface with some of the Data Management services. Rather than coding into the program all the attributes of each file to be processed, the user supplies certain information about the files in the //DEFINE statement. In this way changes, which might otherwise require re-assembly of the program, only require modification of the Control Language statement. Thus, the program tends to remain independent of such things as the name of the file being processed, what device or volume the file resides on, and for sequential files the type of media used for processing. The Control Language Services provides the additional service of requesting allocation of space for new disc files from the user supplied parameters.

For sequential files, the Control Language can change the file to any device. If files have the same characteristics, the file name specified in DEFLB and referenced by the OPENL macro can be overridden by the Control Language with different file names; that is, files other than the one specified by OPENL can be processed.

At the logical I/O level, the Control Language can handle all the space management (allocation, expansion, and purging). After a control table has been set up by the Control Language, Data Management can use these parameters for OPENL/CLOVEL routines.

When a file is opened (OPENL macro), a File Description Table (FDT) is established. The file attributes described in the table are taken from the System Control Table (SCT) established by the //DEFINE statement and the Buffer Description Table (BDT) established by the define macros. (The Control Language Services, Extended Reference manual provides a detailed discussion of the attributes specified by the //DEFINE statement.)

The CLOVEL macro uses the volume information to switch volumes. The file name of this macro overrides DEFLB macro information.

Example

```
NAME      OPERATION  OPERAND
//DEFINE ID=FILE4, FIL=FILE4PAY,STA=(P,I),;
         VOL=104000,NUM=1000,SIZ=100
    :
FILE4    DEFSF      BLKSIZ=104,DATBUF1=BUFF1,;
                    DATBUF2=BUFF2,;
                    ERROPT=WRONG,RECTYP=F,;
                    RECADR=WORKING,RECSIZ=100
```

The file attributes for FILE4 (cataloged as FILE4PAY) are as follows:

Record size of 100
Block size of 104
Two buffers
Estimated 1000 records
Permanent disc file
Input file

Volume-id	104000

Common stored data format
Sequential organization
Cataloged file
Error routine
Record area

The OPENL macro places the file attributes in the FDT. The Control Program and Data Management, Extended Reference manual contains an exact layout of the FDT.

4. CONTROL PROGRAM SERVICES

All service requests are initially interpreted by the Control Program, but most of them are then transferred to separate routines for processing. There are, however, certain service requests which are processed directly by the Control Program. The two services that fall into this category are program termination and time and date retrieval.

PROGRAM TERMINATION

The program termination macro is necessary for returning control to the system in an orderly fashion on completion of a program. Three Control Program Service macros, HALT, EHALT, and ABEND are provided for this purpose.

These macros differ primarily in their effect upon subsequent steps of the job. HALT (required in all programs) will not affect subsequent steps in any way. EHALT (error halt) and ABEND (abnormal end) will cause any subsequent job steps to be bypassed, going directly to the end of job.

In addition, ABEND provides the ability to log a discrete binary completion code in the Job Control Table which is displayed on SYSOUT. ABEND also forces a dump of the partition, unless a DUMP=NO was coded on the //EXECUTE statement.

The DUMP= keyword operand, when coded explicitly in the //EXECUTE statement, will cause the partition to be dumped or not dumped as specified, without regard to whether termination was via HALT or EHALT. If the DUMP option was not coded on the //EXECUTE statement, termination via HALT or EHALT will not cause a dump; but termination via ABEND will cause the partition to be dumped.

Neither the HALT macro nor the EHALT macro requires any operands. ABEND requires a specification of completion code desired to be logged.

Programs may also be terminated by giving control to an exit routine located in the system linkage area. For this method to work properly, the user program must not disturb (or must save and reload) the contents of general register 7 since this will be loaded by the system with the exit linkage address. To terminate the program a BR @7 should be coded in place of the HALT. This method of termination allows the program to serve as a subroutine which either links back to the caller or terminates depending on the contents of register 7. (Appendix B describes the linkage conventions.)

TIME AND DATE RETRIEVAL

The operating system maintains the current time and date for access by user programs. A user program may read the current time and date by means of the TIME and SDATE macros which will return the time or date to a user specified storage location. Neither the time nor the system date may be altered by a user program (only by operator command), but there is a second date area in the system which is available for storing a date associated with a particular job. The user may set this job date to any date he wishes by means of a //SET statement in the Control Language for his job. To read the job date within the program the JDATE macro must be used. If the job date is not set in the Control Language statements, the system date will be used, and any JDATE macros in the program will then receive the system date.

The use of TIME, SDATE, and JDATE involves specifying a location where the time or date is to be received. TIME gives the time of day in hours, minutes, and seconds while SDATE and JDATE have the option of calendar or Julian formats.

CONSOLE COMMUNICATION

A single macro, CONSOLE, is provided for communication with the operator's console. This macro may be used both for sending messages to be typed on the console and for receiving replies from the console.

A second macro, MESSAGE, is available to generate the actual message format, though this is not essential since the message could also be coded directly if desired (using Assembler data definition statements).

MESSAGES TO CONSOLE

All messages to be sent to the console must conform to a standard format in memory. However, if the MESSAGE macro is used to create the message, the programmer need not be concerned with this format since it will be produced correctly by the macro.

The MESSAGE macro should be coded in a data area of the user program using the parameters specified under the MESSAGE macro description in Section 6; it must be given a label so that it may be referenced by the corresponding CONSOLE macro.

To transmit a message to the console, a CONSOLE macro must be coded using a DATBUF1=symbolic address parameter where the symbolic address is the tag on the MESSAGE macro used to generate the actual message.

The format of the message as it will be typed on the console is as follows:

n ii t messagetext

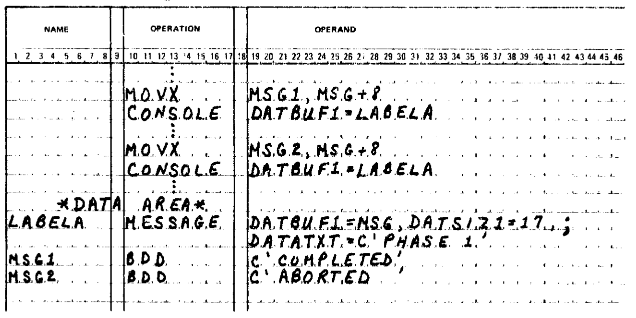
n – the message number assigned by the system. This will be blank unless a reply is required. If a reply is required, this message number must be included by the operator in the reply statement.

ii – a two-character operating system source indicator: P1 means partition 1, P2 means partition 2, # # means operating system.

t – a one-character message type indicator: I means informative, D means directive (requiring operator action). This is obtained from the type specified in the MESSAGE macro.

messagetext – the actual message, up to 100 EBCDIC characters (bytes) long.

Example



```
NAME     OPERATION  OPERAND
            :
         MOVX       MSG1,MSG+8
         CONSOLE    DATBUF1=LABELA
            :
         MOVX       MSG2,MSG+8
         CONSOLE    DATBUF1=LABELA
            :
    *DATA AREA*
LABELA   MESSAGE    DATBUF1=MSG,DATSIZ1=17,                   ;
                    DATATXT=C'PHASE 1'
MSG1     BDD        C'COMPLETED'
MSG2     BDD        C'ABORTED  '
```

At assembly time the MESSAGE macro at LABELA will create a message format with a message text area 17 bytes long, the first byte of which will be labeled MSG. The first seven bytes contained in the message text area will be PHASE 1 and the remaining 10 bytes will be filled with blanks.

During execution of the program, the first MOVX instruction will move the characters COMPLETED into LABELA's message text area displaced eight bytes to the right of the label MSG, thus forming the message PHASE 1 COMPLETED. The CONSOLE macro following this MOVX instruction will then cause the message to be typed out on the console. Similarly, the second MOVX instruction will result in the message PHASE 1 ABORTED, and the CONSOLE macro following the MOVX instruction will send this message to the console.

REPLIES FROM CONSOLE

The link between a given output message and its corresponding reply message is established by a message number which is the first character in both the message and its reply. This number is assigned by the system and typed with the output message if the message requires a reply (otherwise the first character position will be left blank).

When the operator types the reply, the following format is used:

n replytext

n – the number of the output message to which the operator is replying.

replytext – the actual reply, up to 100 EBCDIC characters (bytes) long.

The reply will be received by the same CONSOLE macro that issued the output message. (In addition to the DATBUF1=symbolic address parameter, a CONSOLE macro that is intended to receive a reply must also include a DATBUF2=symbolic address parameter to signal the system that a reply is required and to direct it to the reply buffer area.)

Note that when a CONSOLE macro requires a reply, control will not be returned to the user program until a reply has been received unless RETURN=YES is coded to specify immediate return.

Example

```
NAME      OPERATION  OPERAND
            :
         CONSOLE    DATBUF1=LABELA,;
                    DATBUF2=LABELB
            :
    *DATA AREA*
LABELA   MESSAGE    DATATXT=C'ENTER PASSWORD',;
                    MODE=D
LABELB   MESSAGE    DATSIZ1=10
```

At assembly time the MESSAGE macro at LABELA will create an output message format containing the message text ENTER PASSWORD, and the MESSAGE macro at LABELB will create a reply message format containing a 10-byte field reserved for the reply message text.

During the execution of the program, the CONSOLE macro will cause the message 5 P1 D ENTER PASSWORD to be typed out on the console. The system has included the message number (5 for this example) so that the operator's reply (which will use the same prefix) may be linked to the correct output message. When the operator enters his reply (for example, 5 molybdenum), the reply will be transferred to the 10-byte field reserved for it by LABELB.

5. PROGRAM DEBUGGING

Included in the operating system is a debug package, known as CHECKOUT, for assistance in debugging Assembler language* programs. CHECKOUT enables the programmer, through debug request cards placed in his Control Language deck (as a data file), to call for a selection of register and main-storage dumps to be performed at specified points during the execution of his program. The following dynamic snapshots in hexadecimal format may be obtained.

1. General registers
2. Any specified area of the user program
3. The entire user and system areas
4. Formated system control information

Before debugging requests may be applied to a program, the program must be assembled and linkage edited.

* CHECKOUT may also be used with RPG,COBOL, and FORTRAN programs if an Assembler language listing of the compiler output is available.

CHECKOUT OPERATION

CHECKOUT operates in two phases. The first phase, performed immediately after the program segment to be debugged has been loaded, involves initializing the program by planting special debug service requests at appropriate points within it and building (within the system) the control information necessary for executing the debug request. (The nature and location of the planted service requests is determined by the debugging requirements supplied by the user with the Control Language statements.)

At this point, the directives are written on the job's SYSOUT file, accompanied by directive diagnostic messages, if any.

The second phase, performed after the initialized program has been loaded and goes into execution, involves the processing of the previously planted service requests.

Each debug service request overlays a user program instruction. This instruction is saved by CHECKOUT so that it may be replaced and executed on completion of the service request (to maintain the integrity of the user program). After the replaced instruction has been executed, the Control Program reestablishes the debug service request in preparation for the next time through the program (if any).

REQUESTING DEBUGGING SERVICES

Program debugging services are requested entirely through the use of request cards which are placed in the job deck with the Control Language cards (Figure 5-1). This arrangement enables specific requests to be altered from run to run without requiring repeated reassembly of the program being debugged.

When debugging services are required, the //EXECUTE statement associated with the program to be debugged must always include the parameter DEBUG=YES. This instructs the system to load the CHECKOUT routines immediately after the program itself has been loaded and then to transfer control to CHECKOUT.

The types of dumps required and the circumstances in which they are to be performed are specified in the set of debugging directives coded on the request cards.

BREAK DIRECTIVE

The BREAK directive specifies the type, location, and circumstances of a particular action in a debugging request. There are three types of action that may be requested.

1. Dump
2. Program termination
3. Setting or resetting of the condition flag (an indicator, maintained by CHECKOUT, used to govern the execution of conditional requests*)

The format of the BREAK directive is:

```
BREAK=(breakpoint address,action
        [,type] [,interval]
        [,starting address,ending address] )
```

* For a more detailed discussion, refer to Conditional Request paragraph.

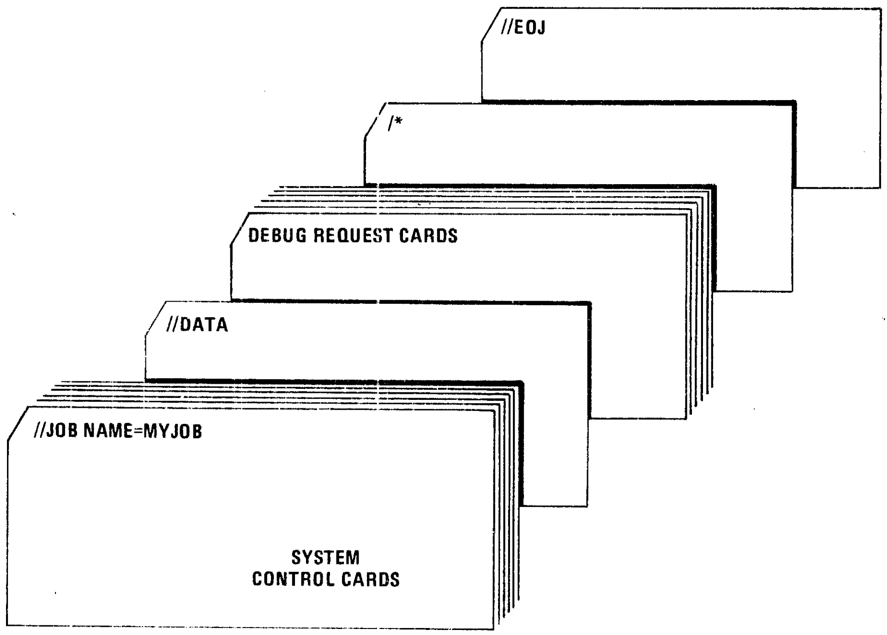


Figure 5-1. Job Deck with Debug Requests

breakpoint address

A 1- to 4-digit hexadecimal code specifying the address within the program at which this particular BREAK directive is to come into operation. The breakpoint address must be an even number (on a word boundary) and must point to the first word of any valid instruction except an SR or a service request macro (for example, READ or WRITE). The breakpoint address must be specified as a displacement within the relevant load module. When a program consists of a single load module which was produced from a single control section (the normal case), this displacement may be obtained directly from the program's assembly listing, but when the program consists of multiple control sections/modules, the Linkage Editor map will also be required to calculate the additional control section displacements.

action

A 1- or 2-character code specifying the action to be performed at the breakpoint address. The following action codes are available.

Code	Action
T	Terminates program and dumps if type parameter is present
TC	Terminates if condition flag is on and dumps if type parameter is present
D	Dumps unconditionally
DC	Dumps if condition flag is on
CS	Turns condition flag on and dumps if type parameter is present
CR	Turns condition flag off and dumps if type parameter is present

type (Optional)

A 1-character code specifying the type of dump to be performed. The following type codes are available.

Code	Type
R	General registers only
S	General registers plus main-storage images (in hexadecimal with EBCDIC translations) of that section of the user partition defined by starting address and ending address
P	General registers, formated system control information, and main-storage images of the entire user partition
A	General registers, formated system control information, and main-storage images of the entire user and system areas

interval (Optional)

A 1- to 3-digit decimal number (0-255) specifying the number of times the breakpoint address must be passed before the selected action is performed. No action will be taken on the first n-1 executions of the instruction at the breakpoint address; action will be taken on execution n, 2n, 3n, etc.

starting address,ending address (Optional)

A pair of 1- to 4-digit hexadecimal codes specifying the starting and ending words of the main-storage dump. Both should be word addresses; if not, starting address and ending address will be rounded down one byte. Starting address and ending address must be specified as displacements within the partition. Starting and ending displacements are relative to the beginning of the user portion of the partition. The linkage editor map will be required to compute the actual offset.

Examples

1. BREAK=(1064,D,R)

   When program execution reaches displacement 1064 (before executing the instruction at 1064), general registers are dumped.

2. BREAK=(1098,CS)

   The condition flag is turned on when program execution reaches displacement 1098.

3. BREAK=(1120,TC,S,,2000,2080)*

   If the condition flag is on when displacement 1120 is reached, the program terminates and the memory from partition displacements 2000 to 2080 is dumped. If the condition flag is off at this time, no action is taken.

4. BREAK=(1250,D,R,40)

   For the first 39 executions of program displacement 1250, no action is taken. At the 40th, 80th, 120th, etc., executions, the general registers are dumped.

PROG DIRECTIVE

The PROG directive establishes the name of the load module to which all succeeding directives apply, up to the point that a new PROG directive is encountered. Until the first PROG directive is reached, all BREAK directives automatically relate to the main module as specified in the //EXECUTE statement. Thus, the PROG directive is only necessary when debugging programs containing more than one load module.

The format of the PROG directive is:

PROG=program name

program name

The name of the load module to which all succeeding directives apply.

*The extra comma indicates that the interval parameter has been omitted.

CONDITIONAL REQUESTS

A conditional request (BREAK directive) is a request which is only performed in the event that the condition flag, maintained by CHECKOUT, is in the on condition at the time that the request is to be implemented. This flag may be dynamically altered during program execution and may thus be used to govern the execution of conditional requests.

The condition flag is always in the off state at the start of a program. If a BREAK directive is coded to set the condition flag to on at a breakpoint address within a given program path, a conditional dump will not be recognized unless the program has followed the given program path. As a result of the condition flag still being off, no conditional requests of the given program path will be honored (Figure 5-2).

EXAMPLE OF JOB CONTROL DECK WITH CHECKOUT DIRECTIVES

Figure 5-3 shows a typical job deck including a CHECKOUT directive set calling for a variety of different dumps. An explanation of each card follows:

Card 1: The //JOB card.

Card 2: The //EXECUTE card includes the parameter DEBUG=YES so that the system will open the CHECKOUT data file and initialize CHECKOUT for each loaded module.

Cards 3, 4, and 5: The //DEFINE cards identify the input file, output file, and CHECKOUT directive file, respectively. (Cards 3 and 4 are not required by CHECKOUT.)

Card 6: The //DATA card informs the system that data destined for the CHECKOUT data file follows immediately.

Card 7: The first of the CHECKOUT directive cards. The BREAK directive on this card directs CHECKOUT to turn the condition flag on when displacement 102A is reached within program COPY. (Since a PROG directive has not yet been issued, the program name given on the //EXECUTE card is assumed.)

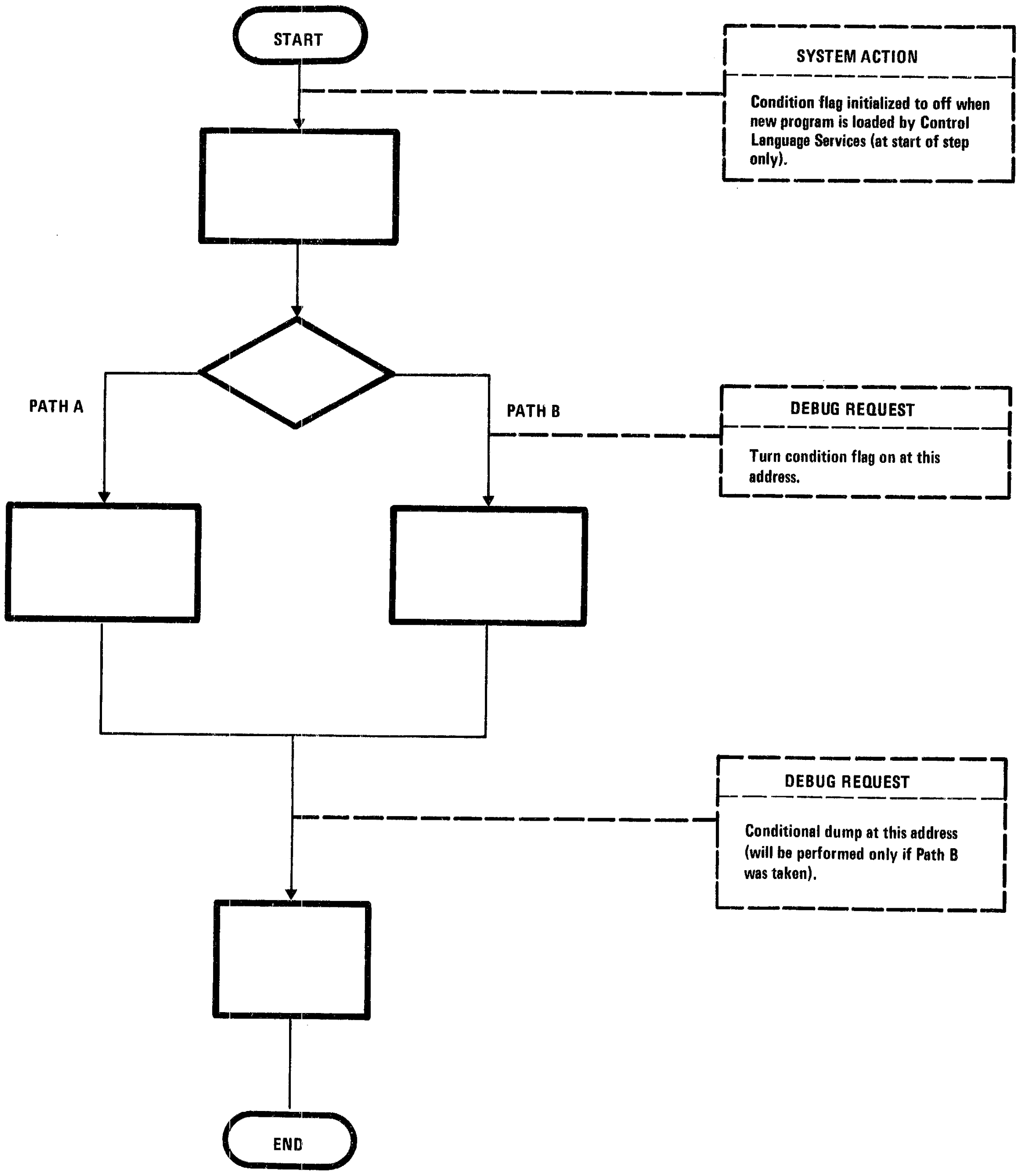
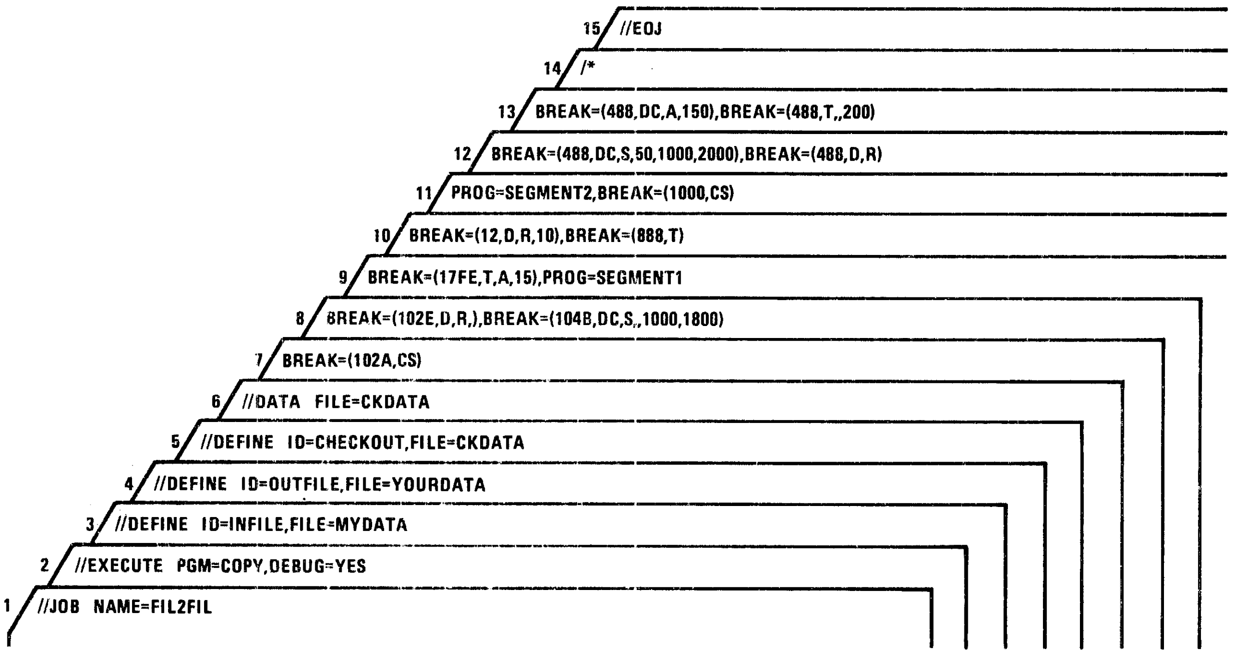

START
SYSTEM ACTION
Condition flag initialized to off when new program is loaded by Control Language Services (at start of step only).
PATH A
PATH B
DEBUG REQUEST
Turn condition flag on at this address.
DEBUG REQUEST
Conditional dump at this address (will be performed only if Path B was taken).
END

Figure 5-2. Example of Use of Conditional Request



```
15 //EOJ
14 /*
13 BREAK=(488,DC,A,150),BREAK=(488,T,,200)
12 BREAK=(488,DC,S,50,1000,2000),BREAK=(488,D,R)
11 PROG=SEGMENT2,BREAK=(1000,CS)
10 BREAK=(12,D,R,10),BREAK=(888,T)
9 BREAK=(17FE,T,A,15),PROG=SEGMENT1
8 BREAK=(102E,D,R,),BREAK=(104B,DC,S,,1000,1800)
7 BREAK=(102A,CS)
6 //DATA  FILE=CKDATA
5 //DEFINE  ID=CHECKOUT,FILE=CKDATA
4 //DEFINE  ID=OUTFILE,FILE=YOURDATA
3 //DEFINE  ID=INFILE,FILE=MYDATA
2 //EXECUTE  PGM=COPY,DEBUG=YES
1 //JOB  NAME=FIL2FIL
```

Figure 5-3. Typical Job Deck with Debugging Directives

Card 8: This card contains two BREAK directives.* The first calls for a register file dump at displacement 102E within COPY. The second BREAK directive specifies a conditional dump of memory between partition displacements 1000 and 1800, to be performed at displacement 104B within COPY only if the condition flag is on at this time.

Card 9: The BREAK directive on this card requests program termination and a type A dump to be performed on every 15th execution of the instruction at program displacement 17FE. The PROG directive indicates that from here on (until another PROG is encountered) all BREAK directives are to apply to SEGMENT1.

Card 10: The first BREAK directive calls for a register dump on every 10th execution of the instruction at displacement 12 within SEGMENT1. The second BREAK directive specifies termination of program at displacement 888 within SEGMENT1.

Card 11: From this point until another PROG directive is issued, all directives apply to SEGMENT2. The BREAK directive calls for the condition flag to be turned on at displacement 1000 within SEGMENT2.

Cards 12 and 13: The four BREAK directives on these cards all apply to displacement 488 within SEGMENT2. The first specifies a conditional dump of memory locations between partition displacement 1000 and 2000, to be performed on every 50th execution of the instruction at the breakpoint address if the condition flag is on. The second calls for a register file dump on each execution. The third is a conditional dump of type A to be performed on the 150th execution. The fourth is an unconditional termination of the whole program on the 200th execution.

Card 14: This is the data delimiter card for the CHECKOUT directive data file.

Card 15: The //EOJ card.

*Note that multiple debugging directives may appear on one card, but no continuation is allowed.

6. MACROS

This section gives the specifications for the logical I/O level functions of Data Management, console communications, 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 identifier. The first six characters must be unique to accommodate the standard suffixes used by the system. These are discussed in the **Control Program** and **Data Management Services, Extended Reference** manual.

The names ident, labadr, and tag are used as identifiers for the software function specified in the operation field.

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,{ }, with the default case being 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 is the 1- to 8-character symbol used to identify a coding statement. Unless otherwise stated all numbers are assumed to be in decimal with no leading zeros.

The linkage conventions for MRX/OS are found in Appendix B.

DATA MANAGEMENT MACROS

In this section only the logical I/O level of Data Management macros are given. This level of macros includes:

- File and Label Definition Macros

  DEFSF
  DEFRF
  DEFIF
  DEFLB

- File Control Macros

  OPENL
  CLOSEL
  CLOVEL

- Data Transmission Macros

  GET
  PUT
  SETL
  LOCRC
  PUTU
  DELR

FILE AND LABEL DEFINITION MACROS

The declarative macros for the logical I/O level define the blocking/deblocking and label information for the three logical I/O level organizations. There must be a file definition macro (DEFSF, DEFRF, or DEFIF) for each file to be opened and processed. The macros generate a main-storage table, the Buffer Description Table (BDT), which is used by the GET/PUT logic.

The name field (ident) of the define macros is a required entry. This 8-character file identifier is the symbolic address of the BDT generated by a macro instruction.

DEFSF – Define Sequential File

The DEFSF macro defines sequential files. The format for DEFSF is as follows:

```
Name    Operation    Operand

ident   DEFSF        [BINARY={YES|NO}]
                     BLKFAC=n
                     BLKSIZ=n
                     [BUFSHR={YES|NO}]
                     [DATBUF1=symbolic address]
                     [DATBUF2={YES|symbolic address}]
                     [ERROPT={symbolic address|S|I}]
                     [LABTYP={STND|NONE|NSTND}]
                     [RECADR=symbolic address]
                     [RECREG={YES|NO}]
                     RECSIZ=n
                     RECTYP={F|V}
                     [VERIFY={YES|NO}]
```

BINARY={YES|NO} (Optional)

BINARY=YES indicates that files assigned to card readers will process binary cards; therefore, the buffer size must be 160 bytes rather than 80 (or in the case of punch, 161 bytes rather than 81). If BINARY=NO or if omitted, the data is in EBCDIC.

BLKFAC=n

Designates the number of records per block (blocking factor). The keyword value gives an integer value of 1 through 255. BLKFAC is required for fixed length records. This parameter should be used for variable length records only when packing is not desired; without it, packing will occur.

BLKSIZ=n

Specifies the block size of the file. The keyword value gives an integer value of 18 bytes through 32K bytes for magnetic tape and 7294 bytes for disc. The four bytes per record for record headers must be included when specifying block size for files which may be assigned to disc or magnetic tape. Buffer size must be greater than or equal to the block size of any file which uses the buffer.

BUFSHR={YES|NO} (Optional)

BUFSHR=YES indicates the I/O buffer(s) is(are) shared with another file. If sharing is specified, the buffer address(es) must be explicitly specified and the user must reserve two bytes (one machine word) immediately before DATBUF1. This word, containing zero, is used by OPENL and CLOSEL to resolve buffer usage conflicts. If BUFSHR=NO or is omitted, I/O buffers are not shared. OPENL and CLOSEL do not check for buffer usage conflicts.

DATBUF1=symbolic address (Optional)

Specifies the address of the first I/O buffer, which must begin on an even byte boundary. When generating buffers, the buffer size must be greater than or equal to the block size of any file which uses the buffer. If the parameter is not specified, DEFSF generates a buffer equal to block size immediately following the BDT.

DATBUF2={YES|symbolic address} (Optional)

Specifies a second I/O buffer, which permits overlapping of data transfer and information processing. The buffer must begin on an even byte boundary. DATBUF2=YES results in generation of a second buffer following DATBUF1. If omitted, processing is from one buffer.

ERROPT={symbolic address|S|I} (Optional)

Specifies one of three conditions for processing errors. The symbolic address gives the user control when errors occur. The S (skip) option and I (ignore) option (only for input files) skips to the next block or ignores errors, respectively, and processing continues. If the parameter is omitted, error conditions abort the job.

LABTYP={STND|NONE|NSTND} (Optional)

Determines the type of tape label processing to be performed for tape files. The parameter is ignored for non-tape files. The default value for files assigned to tape is standard label processing. The code definitions are:

Code	Definition
STND	Check labels for input files. Write standard labels for output files.
NONE	No labels exist and no label checking is done.
NSTND	Nonstandard labels exist and label checking is bypassed. The nonstandard label set must be terminated by a tape mark. The tape is positioned to the record following the tape mark by the OPENL macro.

RECADR=symbolic address (Optional)

Specifies the address of a record area. The size of the record area must be greater than or equal to RECSIZ. Record area address may also be specified at GET/PUT time.

RECREG=$\left\{\begin{matrix}\text{YES}\\ \underline{\text{NO}}\end{matrix}\right\}$ (Optional)

RECREG=YES indicates processing without a record area. When RECREG=YES, R1 contains the address of a logical input record that is available for processing after a GET or the address of the area that is available for construction of the next logical record before a PUT. If this parameter is omitted or if RECREG=NO, a record area (RECADR parameter) must be specified in the DEFSF or at GET/PUT time.

RECSIZ=n

Indicates the record length (in bytes) for fixed length records. The keyword value gives an integer value of 18 bytes through 32K bytes for magnetic tape and 7294 bytes for disc. The maximum record length must be specified for variable length records. Record length does not include the header length.

RECTYP=$\left\{\begin{matrix}\underline{\text{F}}\\ \text{V}\end{matrix}\right\}$

Specifies the record type, fixed (F) or variable (V) length, for the file. If omitted, records will be fixed length. If the file is assigned to cards or printer, the record type must be fixed.

VERIFY=$\left\{\begin{matrix}\text{YES}\\ \underline{\text{NO}}\end{matrix}\right\}$ (Optional)

VERIFY=YES states that a disc write verification will be done. If the file is not on disc, VERIFY is ignored. If omitted or VERIFY=NO, write verification is not used.

Example

NAME	OPERATION	OPERAND
CARDIN	DEFSF	BLKSIZ=80,ERROPT=ERRCD,;
		RECSIZ=80,RECTYP=F,;
		BLKFAC=1

CARDIN is a sequential card file which is fixed length with block and record sizes of 80 bytes each. An error routine, ERRCD, processes any incorrect cards.

DEFRF – Define Relative File

The DEFRF macro defines relative files. The format for DEFRF is as follows:

Name	Operation	Operand
ident	DEFRF	ACCESS=$\left\{\begin{matrix}\underline{\text{S}}\\ \text{R}\end{matrix}\right\}$
		[BLKFAC=n]
		BLKSIZ=n
		$\left[\text{BUFSHR}=\left\{\begin{matrix}\text{YES}\\ \underline{\text{NO}}\end{matrix}\right\}\right]$
		[DATBUF1=symbolic address]
		$\left[\text{DATBUF2}=\left\{\begin{matrix}\text{YES}\\ \text{symbolic address}\end{matrix}\right\}\right]$
		$\left[\text{ERROPT}=\left\{\begin{matrix}\text{symbolic address}\\ \text{S}\\ \text{I}\end{matrix}\right\}\right]$
		KEYADR=symbolic address
		[RECADR=symbolic address]
		$\left[\text{RECREG}=\left\{\begin{matrix}\text{YES}\\ \underline{\text{NO}}\end{matrix}\right\}\right]$
		RECSIZ=n
		RECTYP=$\left\{\begin{matrix}\underline{\text{F}}\\ \text{V}\end{matrix}\right\}$
		[START=X]
		[END=Y]
		$\left[\text{VERIFY}=\left\{\begin{matrix}\text{YES}\\ \underline{\text{NO}}\end{matrix}\right\}\right]$

ACCESS=$\left\{\begin{matrix}\underline{S}\\R\end{matrix}\right\}$

Specifies sequential (S) or random (R) access mode. Together with the usage parameter of OPENL, it determines the processing environment of the file. If omitted, access is sequential.

BLKFAC=n (Optional)

Designates the number of records per block (blocking factor). The keyword value gives an integer value of 1 through 255. When omitted, the blocking factor is determined by block size and record size. When BLKFAC is used, the relationship BLKSIZ≥BLKFAC(RECSIZ+4) must be true.

BLKSIZ=n

Specifies the block size of the file. The keyword value gives an integer value of 18 bytes through 32K bytes for magnetic tape and 7294 bytes for disc. The block size includes record headers for disc or tape files.

BUFSHR=$\left\{\begin{matrix}YES\\\underline{NO}\end{matrix}\right\}$ (Optional)

BUFSHR=YES indicates the I/O buffer(s) is(are) shared with another file. If BUFSHR=YES, the buffer address(es) must be explicitly specified and two bytes (one machine word) must be reserved immediately before DATBUF1. This word, containing zero, is used by OPENL and CLOSEL to resolve buffer usage conflicts. If BUFSHR=NO or if omitted, buffers are not shared, and OPENL and CLOSEL do not check for conflicts.

DATBUF1=symbolic address (Optional)

Specifies the location of the first I/O buffer, which must begin on an even byte boundary. If DATBUF1 is omitted, the buffer immediately follows the BDT. The buffer size must be greater than or equal to the block size of any file which uses the buffer.

DATBUF2=$\left\{\begin{matrix}YES\\\text{symbolic address}\end{matrix}\right\}$ (Optional)

Specifies a second I/O buffer, which allows overlapping of data transfer with information processing for sequential access. DATBUF2 is ignored for random access. If present, DATBUF2 must begin on an even byte boundary. DATBUF2=YES generates the buffer following DATBUF1. If DATBUF2 is omitted, processing if from one buffer.

ERROPT=$\left\{\begin{matrix}\text{symbolic address}\\S\\I\end{matrix}\right\}$ (Optional)

Specifies one of three conditions for processing errors. The symbolic address gives the user control when errors occur. The skip (S) option and ignore (I) option, for input files, skips to the next block or ignores errors, respectively, and processing continues. S is illegal for random access. If ERROPT is omitted, error conditions abort the job.

KEYADR=symbolic address

Gives (for random access) the address of the location that will contain the number of the record to be processed. KEYADR is ignored for sequential access. The symbolic address for KEYADR is a 4-byte location beginning on an even byte boundary.

RECADR=symbolic address (Optional)

Specifies the address of the record area. Record areas may also be specified at GET/PUT time. They are required for random output to a blocked file. The size of the record area must be greater than or equal to RECSIZ.

RECREG=$\left\{\begin{matrix}YES\\\underline{NO}\end{matrix}\right\}$ (Optional)

RECREG=YES defines processing without a record area. When RECREG=YES, R1 contains the address of a logical input record that is available for processing after a GET or the address of the area that is available for constructing the next record before a PUT. If RECREG is omitted or if RECREG=NO, a record area must be defined either in the DEFRF macro at GET/PUT time. If random output is selected with RECREG=YES, the blocking factor must equal one.

RECSIZ=n

Indicates the record length (in bytes) for fixed length records. The keyword value gives an integer value of 18 bytes through 32K bytes for magnetic tape and 7294 bytes for disc. For variable length records, the record length is the maximum record length permitted. Record length does not include the header length.

RECTYP=$\left\{\begin{matrix}\underline{F}\\V\end{matrix}\right\}$

Denotes the record type, fixed (F) or variable (V) length, of the file. The default value is fixed length.

START=X (Optional)

States the logical limit, which may range from 1 to 2^{32}-1 (decimal), for the beginning of processing. At creation, this limit becomes the permanent offset for relative addressing. If START is used, the END parameter must also be used.

END=Y (Optional)

States the logical limit for ending processing, where Y≥X. (X is the START parameter limit.) Limits are decimal numbers.

VERIFY=$\left\{\begin{matrix}YES\\\underline{NO}\end{matrix}\right\}$ (Optional)

VERIFY=YES states that a disc write verification will be done. If omitted or VERIFY=NO, write verification is not used.

Example

NAME (1–8)	9	OPERATION (10–17)	18	OPERAND (19–46)
MPLOY		DEFRF		ACCESS=R, BLKSIZ=104, ;
				KEYADR=KEYIS, RECSIZ=100, ;
				RECTYP=F, VERIFY=YES

MPLOY is a relatively organized file which allows random access through a key name, KEYIS. Although the fixed length record size is 100 bytes, the block size is 104 bytes, which includes the record header. Write verification is specified.

DEFIF – Define Indexed File

The DEFIF macro defines indexed files. The format for DEFIF is as follows:

Name	Operation	Operand
ident	DEFIF	ACCESS=$\left\{\begin{matrix}\underline{S}\\R\end{matrix}\right\}$
		[BLKFAC=n]
		BLKSIZ=n
		$\left[\text{BUFSHR=}\left\{\begin{matrix}\text{YES}\\\underline{\text{NO}}\end{matrix}\right\}\right]$
		[DATBUF1=symbolic address]
		$\left[\text{DATBUF2=}\left\{\begin{matrix}\text{symbolic address}\\\text{YES}\end{matrix}\right\}\right]$
		$\left[\text{ERROPT=}\left\{\begin{matrix}\text{symbolic address}\\\text{I}\end{matrix}\right\}\right]$
		$\left[\text{INDBUF=}\left\{\begin{matrix}\text{YES}\\\text{symbolic address}\end{matrix}\right\}\right]$
		$\left[\text{INDSHR=}\left\{\begin{matrix}\text{YES}\\\underline{\text{NO}}\end{matrix}\right\}\right]$
		INDSIZ=n
		[INMAIN=symbolic address]
		KEYADR1=symbolic address
		[KEYADR2=symbolic address]
		KEYSIZ=n
		[RECADR=symbolic address]
		$\left[\text{RECREG=}\left\{\begin{matrix}\text{YES}\\\underline{\text{NO}}\end{matrix}\right\}\right]$
		RECSIZ=n
		$\left[\text{VERIFY=}\left\{\begin{matrix}\text{YES}\\\underline{\text{NO}}\end{matrix}\right\}\right]$

ACCESS=$\left\{\begin{matrix}\underline{S}\\R\end{matrix}\right\}$

Specifies sequential (S) or random (R) access mode. Together with the usage parameter specified in the OPENL macro, this parameter determines the processing environment of the file. If omitted, processing is sequential.

BLKFAC=n (Optional)

Indicates the number of records per block (blocking factor). The keyword value gives an integer value of 1 through 255. When not specified, the blocking factor is determined by the block size and record size. When BLKFAC is used, the relationship BLKSIZ≥BLKFAC (RECSIZ+4) must be true.

BLKSIZ=n

Specifies the block size of the file. The keyword value is in integer value of 18 bytes through 7294 bytes for disc. BLKSIZ must include the space for record headers.

BUFSHR=$\left\{\begin{matrix}\text{YES}\\ \underline{\text{NO}}\end{matrix}\right\}$ (Optional)

BUFSHR=YES specifies that the I/O buffer is shared with another file. If BUFSHR=YES, the buffer address must have been explicitly specified and one word immediately before DATBUF1 must be reserved. This word contains zero and is used by OPENL and CLOSEL to resolve buffer usage conflicts. If omitted or BUFSHR=NO, buffers are not shared, and OPENL and CLOSEL do not check for conflicts.

DATBUF1=symbolic address (Optional)

Designates the address of the first I/O buffer, which must begin on an even byte boundary. The buffer size must be greater than or equal to BLKSIZ when the user generates his own buffer. If DATBUF1 is omitted, a buffer is generated following the BDT.

DATBUF2=$\left\{\begin{matrix}\text{symbolic address}\\ \text{YES}\end{matrix}\right\}$ (Optional)

Specifies a second I/O buffer, which permits overlapping of data transfer with record processing. If present, DATBUF2 must begin on an even byte boundary. DATBUF2=YES generates the buffer following DATBUF1. If DATBUF2 is omitted, processing is from one buffer.

ERROPT=$\left\{\begin{matrix}\text{symbolic address}\\ \text{I}\end{matrix}\right\}$ (Optional)

Specifies one of the two conditions for processing errors. The symbolic address gives the user control when errors occur. The ignore (I) option, only for input files, ignores errors and processing continues. If ERROPT is not specified, error conditions abort the job.

INDBUF=$\left\{\begin{matrix}\text{YES}\\ \text{symbolic address}\end{matrix}\right\}$ (Optional)

Specifies the address of a buffer to be used by the system for processing of index blocks. If this parameter is omitted for random access, INDSHR=YES must be specified. INDBUF=YES causes automatic buffer generation.

INDSHR=$\left\{\begin{matrix}\text{YES}\\ \underline{\text{NO}}\end{matrix}\right\}$ (Optional)

INDSHR=YES indicates that the index buffer is shared with the data file buffer. Access must be random, DATBUF1 must be specified, and RECREG must not be specified. If omitted or INDSHR=NO, no index buffer sharing occurs.

INDSIZ=n

Specifies the index block size. The value n ranges from 18 bytes to 7294 bytes.

INMAIN=symbolic address (Optional)

Specifies the address in main storage where the directory to the directory index entries are read into from mass storage. This option expedites random processing. For sequential access, this parameter is ignored.

KEYADR1=symbolic name

Specifies the address of the location containing the primary key value of a record to be:

1. Updated, retrieved, added, or deleted when in random mode
2. Updated, added, or deleted when in sequential mode

KEYADR1 is required for record addition in sequential mode, optional for update in sequential mode, and ignored for input only in sequential mode. It is required for access in random mode.

KEYADR2=symbolic address (Optional)

Specifies the address of the location which contains the primary key value of the record after a GET (forward key) that will be obtained on the next GET. When the end of file is reached, the first two bytes of the location specified by KEYADR2 will be set to $FFFF_{16}$ to indicate an EOF. This allows records to be added at the end of the file. KEYADR2 is optional for sequential access and ignored for random access.

KEYSIZ=n

Specifies the length (in bytes) of the primary key. The value n ranges from 2 to 100 bytes.

RECADR=symbolic address (Optional)

Specifies the address of a record area. The record area size must be greater than or equal to RECSIZ. It is required for random output to a blocked file using PUT. Record areas may also be specified at GET/PUT time.

RECREG=$\left\{\begin{matrix}\text{YES}\\ \underline{\text{NO}}\end{matrix}\right\}$ (Optional)

RECREG=YES specifies processing without a record area. When RECREG=YES, R1 contains the address of a logical input record after a GET or the address available for constructing the next record before PUT or PUTU. If omitted or RECREG=NO, a record area must be specified either in the DEFIF macro or at GET/PUT time. Table 6-1 indicates when RECREG may be used.

Table 6-1. RECREG Use

Usage / Access	Input	Update	Output
Sequential	Any BLKFAC	BLKFAC of 1	Any BLKFAC
Random	Any BLKFAC	BLKFAC of 1	Illegal

RECSIZ=n

Specifies the fixed record length in bytes. The value n ranges from 18 bytes to 7294 bytes. The record length does not include the header length.

VERIFY=$\left\{\begin{matrix}\text{YES}\\ \underline{\text{NO}}\end{matrix}\right\}$

VERIFY=YES states that a disc write verification will be done. If omitted or VERIFY=NO, no write verification is done.

Example

```
NAME      OPERATION  OPERAND
PARTLST   DEFIF      ACCESS=R,BLKSIZ=94,
                     ERROPT=MISTAK,VERIFY=YES,
                     KEYADR1=KEYNAM,RECSIZ=90,
                     INDSIZ=250,KEYSIZ=5
```

PARTLST is an indexed file open for random access through KEYNAM. The keysize is five bytes. The block size is 94 which is the record size plus four bytes for the control header. The index block size is 250 bytes. If an error is detected, the error routine MISTAK receives control. A write verification will be done.

Summary of File Definition Parameters

Table 6-2 summarizes the parameters for sequential, relative, and indexed files.

DEFLB – Define Label

The DEFLB macro generates file label data into a main-storage buffer for creating and checking disc labels. The logical I/O level file control macros may use DEFLB, but it is not necessary to define labels at the logical I/O level. The format for DEFLB is as follows:

Name	Operation	Operand
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 $. Imbedded dashes are allowed, but not imbedded blanks. Index file names are created 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 only for work and permanent files. If omitted, blanks are assumed.

Example

```
NAME      OPERATION  OPERAND
TAPE1     DEFLB      FILENAM=PAYROLL
```

TAPE1 gives the label identification for a file named PAYROLL.

Table 6-2. File Definition Parameters

Parameter \ File Type	Sequential	Relative	Indexed
ACCESS	▨	Required; default sequential	Required; default sequential
BINARY	Optional for card devices	▨	▨
BLKFAC	Optional; default is packing	Optional	Optional
BLKSIZ	Required	Required	Required
BUFSHR	Optional	Optional	Optional
DATBUF1	Optional; default follows BDT	Optional; default follows BDT	Optional; default follows BDT
DATBUF2	Optional	Optional	Optional
END	▨	Optional	▨
ERROPT	Optional	Optional	Optional
INDBUF	▨	▨	Required for sequential access
INDSHR	▨	▨	Optional
INDSIZ	▨	▨	Required
INMAIN	▨	▨	Optional
KEYADR(1)	▨	Required for random access	Required for random access, sequential update, sequential output
KEYADR2	▨	▨	Optional
KEYSIZ	▨	▨	Required
LABTYP	Tape files only	▨	▨
RECADR	Optional	Optional	Optional
RECREG	Optional	Optional	Optional
RECSIZ	Required	Required	Required
RECTYP	Required; default fixed	Required; default fixed	Fixed length
START	▨	Optional	▨
VERIFY	Optional	Optional	Optional

LOGICAL I/O FILE CONTROL MACROS

The file control macros (OPENL, CLOSEL, and CLOVEL) direct data transmission at the logical I/O level.

The functions of the OPENL macro are as follows:

- Opens the file for data transmission by establishing an FDT
- Records BDT entries not supplied by DEFSF, DEFRF, or DEFIF
- Catalogs logical level file attributes when files are opened for output for the first time
- Checks file attributes supplied through DEFSF, DEFRF, or DEFIF against permanent cataloged attributes on subsequent opens
- Prefills buffers for sequential access by GET/PUT requests

The CLOSEL macro closes files to data transmission and empties buffers for GET/PUT when required.

The CLOVEL macro closes disc or magnetic tape volumes and switches to the next volume before the end of the allocated disc space or end of tape marker is reached. End of allocated area volume switching is performed by the GET/PUT logic through the CLOVEL macro.

OPENL – Open Logical File

The OPENL macro opens a file for logical I/O processing. The format for OPENL is as follows:

```
Name     Operation     Operand

[tag]    OPENL         IDENT=name
                       [CONTROL={ANS   }]
                                {NATIVE}
                       [LABDEF=symbolic address]
                       [LIST={YES}]
                             {NO }
                       [REWIND={YES}]
                               {NO }
                       [USAGE={I}]
                              {U}
                              {O}
```

IDENT=name

Specifies the file identifier. It must correspond to the name field (file identifier) of a DEFSF, DEFRF, or DEFIF macro.

CONTROL=$\left\{\begin{matrix}\underline{\text{ANS}}\\ \text{NATIVE}\end{matrix}\right\}$ (Optional)

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

LABDEF=symbolic address (Optional)

Specifies the address of the main-storage buffer that contains a file label. The symbolic address should be identical to that specified in the label address name field of a DEFLB macro. For disc files, label information may have been specified by Control Language //DEFINE statements which override LABDEF information if both are present. This parameter may not be used for permanent files or work files with update or output usage. For temporary or scratch files, the filename is concatenated with the job name. It is ignored for files assigned to unit record devices or magnetic tape.

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

If LIST=YES, only the parameter packet is generated. If LIST=NO, only the subroutine linkage is generated and 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 is a user responsibility. If the LIST parameter is omitted, both the subroutine linkage and the parameter packet are generated immediately following the subroutine call; and the user must load R7 with the save area address. A detailed discussion of the LIST parameter is contained in Appendix A.

REWIND=$\left\{\begin{matrix}\underline{\text{YES}}\\ \text{NO}\end{matrix}\right\}$ (Optional)

REWIND=NO specifies that no initial rewind will be performed. This parameter is ignored for non-tape files. If REWIND=YES or if omitted, initial rewind is performed.

USAGE=$\left\{\begin{matrix}\underline{\text{I}}\\ \text{U}\\ \text{O}\end{matrix}\right\}$ (Optional)

Determines input, update, or output 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.

Example

NAME	OPERATION	OPERAND
	OPENL	IDENT=CARDIN, USAGE=I

OPENL opens the input file CARDIN for processing.

CLOSEL – Close Logical File

The CLOSEL macro closes a file to logical I/O processing. The format for CLOSEL is as follows:

Name	Operation	Operand
[tag]	CLOSEL	IDENT=name
		[LIST=$\left\{\begin{matrix}\text{YES}\\ \text{NO}\end{matrix}\right\}$]
		[LOCK=$\left\{\begin{matrix}\text{YES}\\ \underline{\text{NO}}\end{matrix}\right\}$]
		[REWIND=$\left\{\begin{matrix}\underline{\text{YES}}\\ \text{NO}\end{matrix}\right\}$]

IDENT=name

Specifies the file identifier.

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

If LIST=YES, only the parameter packet is generated. If LIST=NO, only the subroutine linkage is generated and loading of R6 with the address of the parameter packet and R7 with the save area address is a user responsibility. If the LIST parameter is omitted, both the subroutine linkage and the parameter packet are generated immediately following the subroutine call; and the user must load R7 with the save area address. A detailed discussion of the LIST parameter is contained in Appendix A.

LOCK=$\left\{\begin{matrix}\text{YES}\\ \underline{\text{NO}}\end{matrix}\right\}$ (Optional)

Indicates the disposition of the file. When the parameter is not specified or LOCK=NO, the file may be reopened from within the same job; tape files are rewound. When LOCK=YES, OPENL may not be executed again during the same job; tape files are unloaded.

REWIND=$\left\{\begin{matrix}\underline{\text{YES}}\\ \text{NO}\end{matrix}\right\}$ (Optional)

REWIND=NO specifies that no rewind will be performed after closing the file. This parameter is ignored for non-tape files. If REWIND=YES or if omitted, initial rewind is performed.

Example

NAME	OPERATION	OPERAND
	CLOSEL	IDENT=MPLOY

CLOSEL closes the file MPLOY to any further processing.

CLOVEL – Volume Switching

The CLOVEL macro performs volume switching. It is used for sequential multivolume files assigned to tape or disc and should be used only to prematurely close a volume and switch to the next sequential volume. Volume switching at EOV is performed automatically by the GET/PUT modules.

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

Name	Operation	Operand
[tag]	CLOVEL	IDENT=name $\left[\text{LIST=}\left\{\begin{matrix}\text{YES}\\ \text{NO}\end{matrix}\right\}\right]$

IDENT=name

Specifies the file identifier.

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

If LIST=YES, only the parameter packet is generated. If LIST=NO, only the subroutine linkage is generated and loading of R6 with the address of the parameter packet and R7 with the save area address is a user responsibility. If the LIST parameter is omitted, both the subroutine linkage and the parameter packet are generated immediately following the subroutine call; and the user must load R7 with the save area address. A detailed discussion of the LIST parameter is contained in Appendix A.

Example

NAME	OPERATION	OPERAND
	CLOVEL	IDENT=TAPFIL

CLOVEL switches volumes in a multivolume file called TAPFIL.

DATA TRANSMISSION MACROS

The data transmission macros – GET, PUT, and associated macros – control the transmission of data to and from the different files.

Table 6-3 lists the GET/PUT macros allowed according to file type and usage.

GET – Get Record

The GET macro makes a logical record available in a record area. When no record area is present, the address of an available logical record in an I/O buffer is provided in R1. The format for GET is as follows:

Name	Operation	Operand
[tag]	GET	IDXSA=YES IDENT=name $\left[\text{LIST=}\left\{\begin{matrix}\text{YES}\\ \text{NO}\end{matrix}\right\}\right]$ [RECADR=symbolic address] [RECSIZ=symbolic address] [RTNADR=symbolic address]

Table 6-3. GET/PUT Macro Usage

Macro / File Type & Usage	GET	PUT	LOCRC	PUTU	SETL	DELR
Sequential						
Input	X					
Update	X	X				
Output		X				
Relative						
Input	X		X			
Update	X	X	X			
Output		X	X			
Indexed						
Input	X				X	
Update	X	X		X	X	X
Output		X				

IDXSA=YES

Specifies sequential processing of indexed files; it is ignored for random processing. If IDXSA is omitted for sequential processing, control is given to the user's error routine or the job aborts.

IDENT=name

Specifies the file identifier.

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

If LIST=YES, only the parameter packet is generated. If LIST=NO, only the subroutine linkage is generated and loading of R6 with the address of the parameter packet and R7 with the save area address is a user responsibility. If the LIST parameter is omitted, both the subroutine linkage and the parameter packet are generated immediately following the subroutine call; and the user must load R7 with the save area address. A detailed discussion of the LIST parameter is contained in Appendix A.

RECADR=symbolic address (Optional)

Specifies the address of a record area. When RECADR is present in the GET macro, it overrides any RECADR specified in a DEFSF, DEFRF, or DEFIF macro.

RECSIZ=symbolic address (Optional)

Specifies the address of a location that will contain the size (in bytes) of the logical record upon return. RECSIZ is ignored for fixed length records. It must specify an even byte boundary.

RTNADR=symbolic address (Optional)

Specifies the address of a user routine which receives control when a sequentially processed file reaches the end of file or a randomly processed file contains an invalid key value in the KEYADR1 address. When RTNADR is omitted, control is given to the error option selected in the ERROPT parameter of the define macro. If no ERROPT was specified, the job will abort.

Example

NAME	OPERATION	OPERAND
	GET	IDENT=CARDIN,RTNADR=DONE

The GET macro reads the CARDIN file until the DONE parameter is satisfied.

PUT – Put Record

The PUT macro moves a logical record from a record area into an I/O buffer or indicates that a logical record has been generated by the user in an I/O buffer. PUT is used to update records for sequential and relative files. The format for PUT is as follows:

```
Name    Operation    Operand

[tag]   PUT          IDXSA=YES
                     IDENT=name
                     [LIST={YES}]
                           {NO }
                     [RECADR=symbolic address]
                     [RECSIZ=symbolic address]
                     [RTNADR=symbolic address]
```

IDXSA=YES

Specifies sequential processing of indexed files; it is ignored for random processing. If IDXSA is omitted for sequential processing, control is given to the user's error routine or the job aborts.

IDENT=name

Specifies the file identifier.

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

If LIST=YES, only the parameter packet is generated. If LIST=NO, only the subroutine linkage is generated and loading of R6 with the address of the parameter packet and R7 with the save area address is a user responsibility. If the LIST parameter is omitted, both the subroutine linkage and the parameter packet are generated immediately following the subroutine call; and the user must load R7 with the save area address. A detailed discussion of the LIST parameter is contained in Appendix A.

RECADR=symbolic address (Optional)

Specifies the address of a record area. When RECADR is present in the PUT macro, it overrides any RECADR specification in a DEFSF, DEFRF, or DEFIF macro.

RECSIZ=symbolic address

Specifies the address of a location that contains the size (in bytes) of the logical record. RECSIZ must specify an even byte boundary. It is ignored for fixed length records. When RECSIZ is absent and the records are of variable length, control is given to ERROPT or the job aborts.

RTNADR=symbolic address (Optional)

Specifies the address of a user routine which receives control when the end of allocation or a key error occurs. When RTNADR is not defined, control goes to the addresses specified in the ERROPT parameter of the define macro. If ERROPT was not specified, the job aborts.

Example

NAME	OPERATION	OPERAND
	PUT	IDENT=RECRD,RECSIZ=SIZE

The PUT macro writes the file RECRD according to the byte size found in SIZE.

SETL – Set Limits

The SETL macro establishes an initial key value for sequential access of indexed files. If SETL is not issued, retrieval starts at the logical beginning of the file. The format for SETL is as follows:

```
Name    Operation    Operand

[tag]   SETL         IDENT=name
                     KEYERR=symbolic address
                     [LIST={YES}]
                           {NO }
                            {KEY }
                     START= {GKEY}
                            {BOF }
```

IDENT=name

Specifies the file identifier.

KEYERR=symbolic address

Specifies the address which receives control when START=KEY and there is no key value specified. This parameter is required for START=KEY; otherwise, it is ignored.

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

If LIST=YES, only the parameter packet is generated. If LIST=NO, only the subroutine linkage is generated and loading of R6 with the address of the parameter packet and R7 with the save area address is a user responsibility. If the LIST parameter is omitted, both the subroutine linkage and the parameter packet are generated immediately following the subroutine call; and the user must load R7 with the save area address. A detailed discussion of the LIST parameter is contained in Appendix A.

START=$\begin{Bmatrix}\text{KEY}\\\text{GKEY}\\\text{BOF}\end{Bmatrix}$

Determines where the retrieval begins according to the following codes.

Code	Option
KEY	Retrieval begins with the record having the key value contained in KEYADR2 of the tagged DEFIF macro. If no data record with this key exists, control returns to KEYERR.
GKEY	Retrieval begins with the record having the key value contained in KEYADR2, or if no record having this value exists, it begins with the record having the next greater value. If no record has a greater key value, KEYADR2 receives $FFFF_{16}$, indicating the end of the file. This permits records to be added to the end of the file.
BOF	Retrieval begins at the beginning of the file.

Example

NAME	OPERATION	OPERAND
	SETL	IDENT=PRTLST, START=BOF

The SETL macro is used with a GET to establish the beginning key value for indexed files. In this case, retrieval will begin at the beginning of the file named PRTLST.

LOCRC – Locate Record

LOCRC is a relative record seek macro. A seek operation is issued to the block containing the record number found in the location specified by the KEYADR parameter of the DEFRF macro. LOCRC is used to overlap seek and processing time for random access. If an error condition arises, the following GET/PUT receives the status by having control returned to the invalid key routine. The format for LOCRC is as follows:

Name	Operation	Operand
[tag]	LOCRC	IDENT=name
		[LIST=$\begin{Bmatrix}\text{YES}\\\text{NO}\end{Bmatrix}$]

IDENT=name

Specifies the file identifier.

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

If LIST=YES, only the parameter packet is generated. If LIST=NO, only the subroutine linkage is generated and loading of R6 with the address of the parameter packet and R7 with the save area address is a user responsibility. If the LIST parameter is omitted, both the subroutine linkage and the parameter packet are generated immediately following the subroutine call; and the user must load R7 with the save area address. A detailed discussion of the LIST parameter is contained in Appendix A.

Example

NAME	OPERATION	OPERAND
	LOCRC	IDENT=MPLOY

The LOCRC macro positions the access mechanism at the location on the disc file which contains the record.

PUTU — Put Update

The PUTU macro updates a record in indexed files and rewrites the record retrieved by the preceding GET. The address specified by KEYADR in the DEFIF macro must contain the key value of the record to be changed. If the key value in KEYADR is not identical to the key value of the record obtained in the previous GET, control goes to the invalid key routine. The format for PUTU is as follows:

Name	Operation	Operand

[tag]	PUTU	IDENT=name
		KEYERR=symbolic address
		$\left[\text{LIST=}\begin{Bmatrix}\text{YES}\\ \text{NO}\end{Bmatrix}\right]$
		[RECADR=symbolic address]

IDENT=name

Specifies the file identifier.

KEYERR=symbolic address

Specifies the address which will receive control when a key error occurs.

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

If LIST=YES, only the parameter packet is generated. If LIST=NO, only the subroutine linkage is generated and loading of R6 with the address of the parameter packet and R7 with the save area address is a user responsibility. If the LIST parameter is omitted, both the subroutine linkage and the parameter packet are generated immediately following the subroutine call; and the user must load R7 with the save area address. A detailed discussion of the LIST parameter is contained in Appendix A.

RECADR=symbolic address (Optional)

Specifies the address of a record area. When RECADR is present in the PUTU macro, it overrides RECADR specified in a DEFIF macro.

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 45 46
		PUTU		IDENT=RECRD,KEYERR=HEREIS

PUTU updates a record in the RECRD file. HEREIS receives control if the key from the file is invalid.

DELR — Delete Record

The DELR macro deletes a record from an indexed file. The indexing information is deleted from the file's associated index blocks (and directory blocks if necessary), and the data record is left unchanged. Processing requirements differ depending upon access.

Sequential access requires that the key value contained in KEYADR of the DEFIF macro be equal to the key value of the record obtained on the previous GET. If it is not equal, control goes to the invalid key routine.

Random access does not require a preceding GET. If a record having the key value contained in KEYADR does not exist, control goes to the invalid key routine. The format for DELR is as follows:

Name	Operation	Operand

[tag]	DELR	IDENT=name
		KEYERR=symbolic address
		$\left[\text{LIST=}\begin{Bmatrix}\text{YES}\\ \text{NO}\end{Bmatrix}\right]$

IDENT=name

Specifies the file identifier.

KEYERR=symbolic address

Specifies the address which will receive control when a key error occurs.

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

If LIST=YES, only the parameter packet is generated. If LIST=NO, only the subroutine linkage is generated and loading of R6 with the address of the parameter packet and R7 with the save area address is a user responsibility.

If the LIST parameter is omitted, both the subroutine linkage and the parameter packet are generated immediately following the subroutine call; and the user must load R7 with the save area address. A detailed discussion of the LIST parameter is contained in Appendix A.

Example

NAME	OPERATION	OPERAND
1 2 3 4 5 6 7 8	10 11 12 13 14 15 16 17	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
	DELR	IDENT=RECRD,KEYERR=WRONG

DELR deletes a given record from the indexed file RECRD. WRONG is the routine that handles key errors.

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 text.

DATBUF2=symbolic address (Optional)

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

Example

NAME	OPERATION	OPERAND
1 2 3 4 5 6 7 8	10 11 12 13 14 15 16 17	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
TYPE	CONSOLE	DATBUF1=LABELA,;
		DATBUF2=LABELB

The message located in the message area, LABELA, is transmitted to the console. When it is received, the reply is stored in the reply area, LABELB.

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={L/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'.

$\text{MODE=}\left\{\begin{array}{c}\text{I}\\\hline\text{D}\end{array}\right\}$ (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.

Example

NAME	OPERATION	OPERAND
LABELA	MESSAGE	DATBUF1=LABELC,;
		DATSIZ1=20,;
		DATATXT=C'FORMAT ERROR'

Creates a message area at assembly time consisting of a message control block, LABELA, followed by a message text area, LABELC. The message text area will be 20 bytes long. The first 12 bytes will contain FORMAT ERROR and the remaining eight bytes will contain blanks.

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.

Example

NAME	OPERATION	OPERAND
LABELB	MESSAGE	DATBUF2=LABELD,;
		DATSIZ2=20

Creates a reply area at assembly time consisting of a reply control block, LABELD, followed by a 20-byte reply text buffer, LABELD.

CONTROL PROGRAM SERVICES MACROS

Control Program Services macros are those macros which are implemented directly by the Control Program itself. These macros include:

- Program Termination

  HALT
  EHALT
  ABEND

- Time and Date

  TIME
  SDATE
  JDATE

PROGRAM TERMINATION MACROS

Three macros are provided for program termination, HALT for normal step termination, EHALT for step and job termination, and ABEND for abnormal termination.

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
[tag]	HALT

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:

Name	Operation
[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:

Name	Operation	Operand
[tag]	ABEND	INFOADR=symbolic address

INFOADR=symbolic address

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

TIME AND DATE MACROS

The time and date of program execution may be obtained from the system with the TIME and SDATE macros. A third macro, JDATE, is available for obtaining a special user-specified job date.

TIME – Retrieve Time of Day*

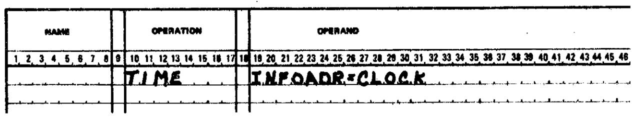
The TIME macro returns the current time of day in the operand specified. The time of day is returned in an unpacked decimal format: hhmmss, where hh is the hour, mm is the minute, and ss is the second. The format for TIME is as follows:

Name	Operation	Operand
[tag]	TIME	INFOADR=symbolic address

INFOADR=symbolic address

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

Example



```
NAME      OPERATION OPERAND
          TIME      INFOADR=CLOCK
```

Transfers the current time (six bytes) to the memory location labeled CLOCK.

SDATE – Retrieve System Date

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:

Name	Operation	Operand
tag	SDATE	INFOADR=symbolic address
		MODE= $\left\{ \begin{matrix} \underline{C} \\ J \end{matrix} \right\}$

INFOADR=symbolic address

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

*Function is available on minimal system (16K storage), but returns zeros.

MODE=$\begin{Bmatrix}C\\J\end{Bmatrix}$

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.

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 45 46
	SDATE	INFOADR=DATE1,MODE=C

Transfers the system date, in calendar format, to the memory location labeled DATE1.

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:

Name	Operation	Operand
tag	JDATE	INFOADR=symbolic address
		MODE=$\begin{Bmatrix}C\\J\end{Bmatrix}$

INFOADR=symbolic address

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

MODE=$\begin{Bmatrix}C\\J\end{Bmatrix}$

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

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 45 46
	JDATE	INFOADR=DATE2,MODE=J

Transfers the job date, in Julian format, to the memory location labeled DATE2.

A. LIST PARAMETER

The LIST parameter, which is used by the logical I/O level file control macros and the GET/PUT macros, allows for separate generation of the parameter packet and the subroutine linkage.

All logical I/O level requests perform a subroutine call to either the GET/PUT module or the Data Management Open Close Control (DMOCC) using the following linkage conventions.

1. Parameter list address set in general-purpose register 6 (R6)

2. Save area address set in general-purpose register 7 (R7)

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

The LIST parameter is used to generate parameter packets and subroutine linkages which make it possible to run program subroutines at execution time. The LIST=YES option generates the parameter packet, and the LIST=NO option generates the subroutine linkage. (Tables A-1 and A-2, at the end of this appendix, provide complete descriptions of the parameter packets for the GET/PUT and OPENL/CLOSEL macros.)

When the LIST=NO option is specified, the IDENT parameter must be specified for the GET/PUT macros. Prior to issuing the LIST=NO option, the user must load R6 with the address of the appropriate parameter packet (generated by the LIST=YES option) to be used and must load R7 with the save area address. Together the address of the parameter packet and the subroutine linkage will take the program to the appropriate subroutine at execution time. When a program subroutine is used several times, the LIST=NO option provides ease and efficiency in coding. The subroutine linkage generated by LIST=NO has the coding format shown in Figure A-1.

LODD	▨	I	R7
Return Address			
B	▨	I	▨
BDT Address (GET/PUT Macros) or DMOCC (OPENL/CLOSEL Macros)			

Figure A-1. Subroutine Linkage Coding Format

Example

NAME	OPERATION	OPERAND
	LODD	SAVEIT,R7
	LODD	TAG4,R6
TAG5	PUT	LIST=NO,IDENT=FILE6
	⋮	
TAG4	PUT	IDENT=FILE6,LIST=YES
	⋮	
SAVEIT	WDD	30

Subroutine Linkage

PUT Parameter List

With the LODD instruction, the user loads the address of a parameter packet which has been specified elsewhere in the program (TAG4) and the address of the save area specified at SAVEIT.

With the PUT macro and a LIST=NO parameter specified, a subroutine linkage is generated (IDENT must be specified).

At address TAG4, the PUT parameter list is generated.

LIST DEFAULT

If the LIST option is omitted, subroutine linkage coding is generated followed by the appropriate parameter list. The user must load R7 with the save area address. Figure A-2 shows the subroutine linkage coding format for the default of the LIST option.

Example

NAME	OPERATION	OPERAND
	LODD	SAVEIT,R7
	GET	IDENT=FILES
	⋮	
SAVEIT	WDD	30

Subroutine Linkage (Default)

GET Parameter List

The LIST default for the GET request results in the generation of the subroutine linkage and the GET parameter list.

LODD	▨	I	R7
Return Address			
BSR	I	R6	
BDT Address (GET/PUT Macros) or DMOCC (OPENL/CLOSEL Macros)			

Figure A-2. LIST Default Subroutine Linkage Coding Format

Table A-1. GET/PUT Parameter List

			Suffix*
0	Length of List		LL
2	FC	▨ | I | RA | S | R | SL	FC/BT
4	Error Code		ER
6	▨	STAG BDT	
8	BDT Address		BD
10	▨	STAG RSA	
12	Record Size Address		SZ
14	▨	STAG EA	
16	End Return Address		EA
18	Index External		
20	▨	STAG RA	
22	Record Address		RA

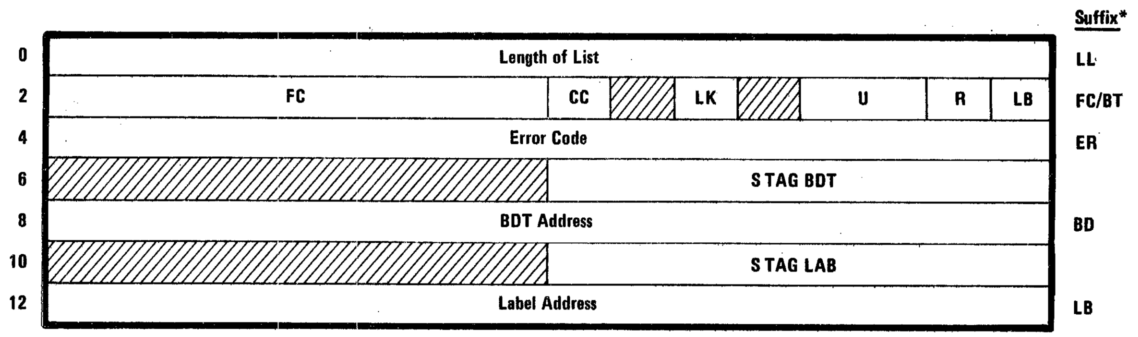
Bytes	Bits	Description
0,1		Length of parameter list. Always set to 4, 6, 8, 9, or 11.
2		Function Codes (FC) 0 GET 1 PUT 2 PUTU 3 LOCRC 7 DELR 8 SETL
3	2	Index external (I) 0 Not indexed function 1 Indexed function
	3	Record address (RA) 0 No record address 1 Record address
	4	Size parameter specified (S) 0 No address specified in word 7 1 Address of variable record size specified in word 7
	5	Return parameter specified (R) 0 No address specified in word 9 1 Return address specified for EOF or invalid key in word 9
3	6,7	SETL flag (SL) 00 Positions to BOF 01 Positions to record equal to KEYADR2 10 Positions to record greater than or equal to KEYADR2
4,5		Error code, word contains the error code returned by the GET/PUT module if an error condition is detected.
7		Segment tag for BDT
8,9		BDT address (must be present)

*The 2-character suffix for unique file identifier.

Table A-1. (Continued)

Bytes	Bits	Description
11		Segment tag for record size address
12,13		Record size address. If S=1 (byte 3, bit 4), this address must be present. This address points to a 2-byte location which contains the size (in binary) of the record. This parameter is required for PUT and PUTU and optional for GET. It is ignored for fixed length records.
15		Segment tag for end return address
16,17		End return address. If R=1 (byte 3, bit 5), this parameter must be present. This address is where control is returned if EOF, EOA, or an invalid key occurs.
18,19		Index external. If I=1 (byte 3, bit 2), this is the external address of the functional module of indexed GET/PUT macros.
21		Segment tag for record address
22,23		Record address, an optional record area that can be specified in each GET or PUT

Table A-2. OPENL/CLOSEL Parameter List



Bytes	Bits	Description
0,1		Length of parameter list, always set to 6
2		Function code (FC) 4 OPENL (Uses CC, U, R, and LB) 5 CLOSEL (Uses LK and R) 6 CLOVEL (Uses none of these)
3	0	Control character (CC) 0 ANSI control characters 1 Device control characters
	2	Lock (LK) 0 No lock 1 Lock
	4,5	Usage (U) 00 Input 01 Input/output 10 Output
3	6	Rewind (R) 0 No rewind 1 Rewind
	7	Label (LB) 0 No label address specified in word 7 1 Label address specified in word 7
4,5		Error code, word contains error code returned by the DMOCC module if an error condition is detected.
7		Segment tag for BDT
8,9		BDT address (must be present)
11		Segment tag for label address
12,13		Label address. If LB=1 (byte 3, bit 7), this is the address of the label packet.

*The 2-character suffix for unique file identifier.

B. LINKAGE CONVENTIONS

Program linkage is the process of linking separately generated object modules into an execution unit. The following two basic functions must be performed in a linkage process:

Matching of external address references

Actual transfer of control during execution

The first function is performed by a linkage editor, and it is not discussed in this manual. The second function is the linkage conventions established for the MRX/OS.

The basic linkage conventions for a calling program include:

1. Using the proper registers to establish linkage
2. Reserving an area that is used by the called program to refer to the parameter list.
3. Reserving an area in which the contents of the registers may be saved.

REGISTER USE

The linkage requires one or two registers, depending upon the passing of the address of the parameter list to the called program.

The user must load the save area address into general-purpose register 7 (R7).

If the parameter list is elsewhere in the program, the user loads the address of the parameter list into general-purpose register 6 (R6).

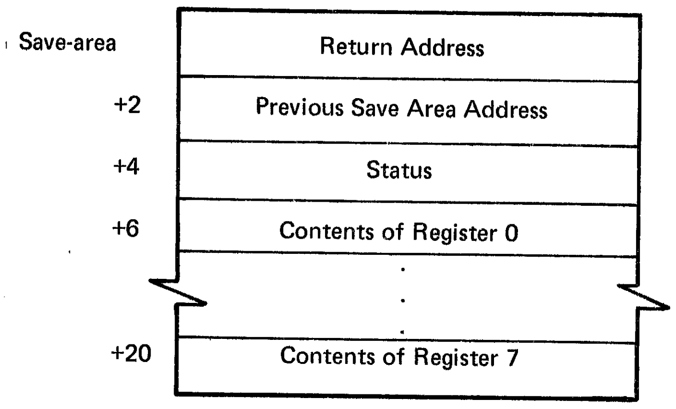
PARAMETER LIST

The parameter list is a list of contiguous words starting on a word boundary. It is the expansion of the macro call. When the parameter list is not found immediately after the macro call, the location of this parameter list must be loaded into R6.

SAVE AREA

The calling program must reserve an area of 22 bytes (beginning on an even-byte boundary) to be used by the called program for saving registers. The calling programs must load the address of the save area into R7.

The following diagram illustrates the layout of the save area.



THE CALLING SEQUENCE

The calling sequence has two forms, depending upon whether the return is in-line or out-of-line.

In-Line

NAME	OPERATION	OPERAND
1 2 3 4 5 6 7 8	10 11 12 13 14 15 16 17	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
	LODD	PARALIST,R6 OPTIONAL
	LODD	SAVING,R7
	BSR	SUBR,@R7

The parameter list address if elsewhere in the program, is loaded into R6. The address of the save area is loaded into R7. A branch and store register instruction saves the address of the next instruction in the return address of the save area.

Out-of-Line

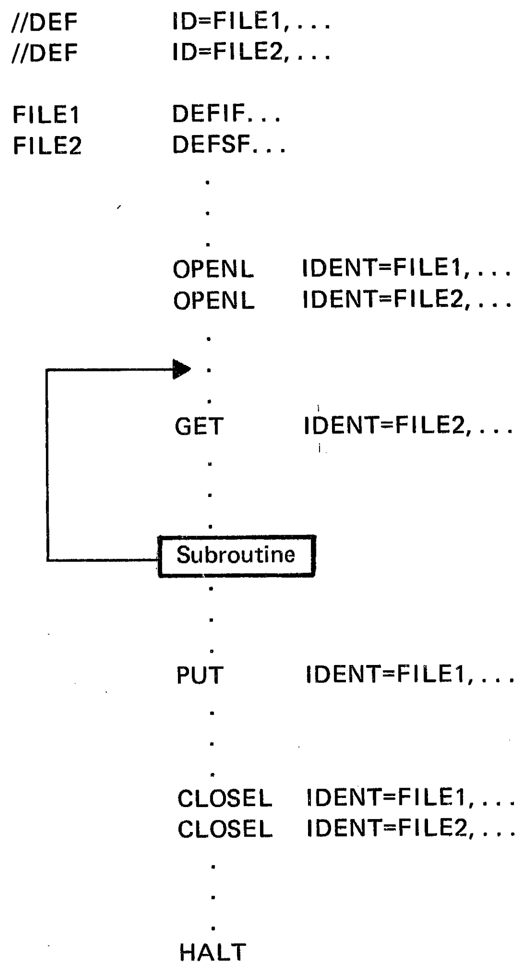
NAME	OPERATION	OPERAND
1 2 3 4 5 6 7 8	10 11 12 13 14 15 16 17	19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46
	LODD	PARALIST,R6 OPTIONAL
	LODD	SAVING,R7
	LODD	MYRTN,@R7
	B	SUBR

The parameter list address, if elsewhere in the program, is loaded into R6. The address of the save area is loaded into R7. The address of the user return (MYRTN) is loaded into the return address of the save area. Then a branch to the subroutine is performed.

C. PROGRAM FLOW

The following figure illustrates the placement of the different macros in the program.

The ID keyword of the Control Language //DEFINE statement and the IDENT keyword of OPENL/CLOSEL and GET/PUT correspond to the identifier entry of the define file macro.



```
//DEF       ID=FILE1,...
//DEF       ID=FILE2,...

FILE1       DEFIF...
FILE2       DEFSF...
              .
              .
              .
            OPENL    IDENT=FILE1,...
            OPENL    IDENT=FILE2,...
              .
              .
              .
            GET      IDENT=FILE2,...
              .
              .
              .
            Subroutine
              .
              .
              .
            PUT      IDENT=FILE1,...
              .
              .
              .
            CLOSEL   IDENT=FILE1,...
            CLOSEL   IDENT=FILE2,...
              .
              .
              .
            HALT
```

D. INDEX PORTION OF INDEXED FILE

The following diagram (Figure D-1) gives the relationship of the different parts of the index portion of an indexed file.

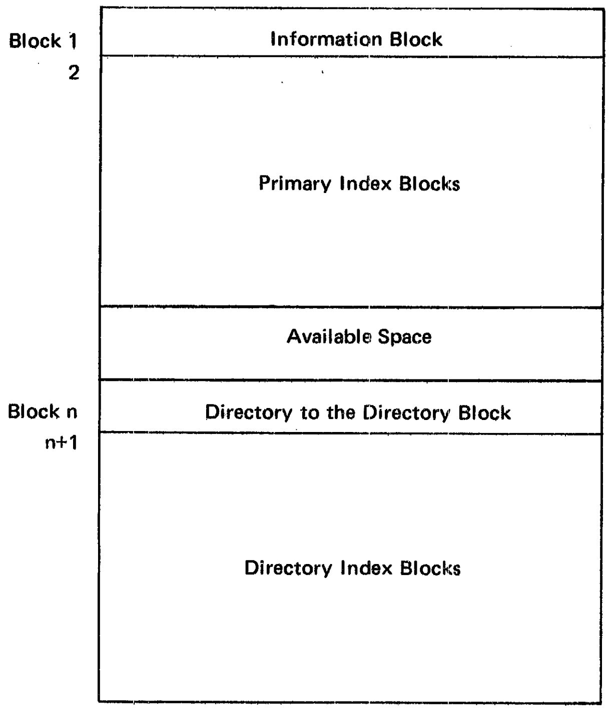
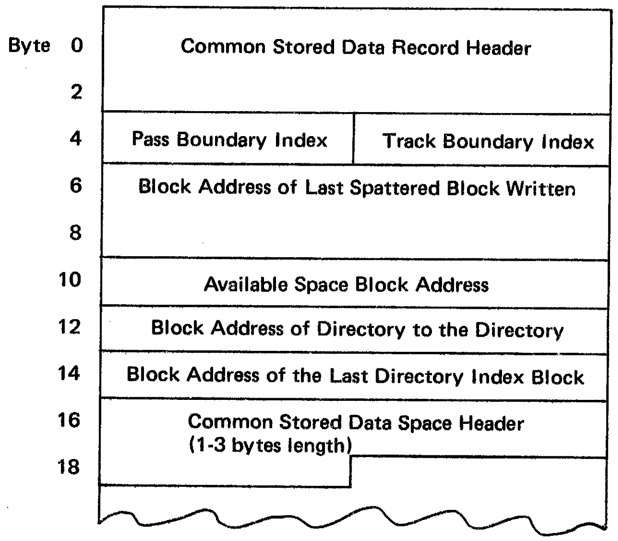


Figure D-1. Index Portion of an Indexed File Layout

The layouts of the individual blocks are found in the succeeding sections of this appendix.

INFORMATION BLOCK

The Information Block of the Index portion of an indexed file gives the addresses of the last blocks of the data portion and index portion of the indexed file that were written. This enables Data Management to make further additions to the file. Figure D-2 gives the fields of the Information Block.



Byte 0
2
4
6
8
10
12
14
16
18
Common Stored Data Record Header
Pass Boundary Index
Track Boundary Index
Block Address of Last Spattered Block Written
Available Space Block Address
Block Address of Directory to the Directory
Block Address of the Last Directory Index Block
Common Stored Data Space Header (1-3 bytes length)

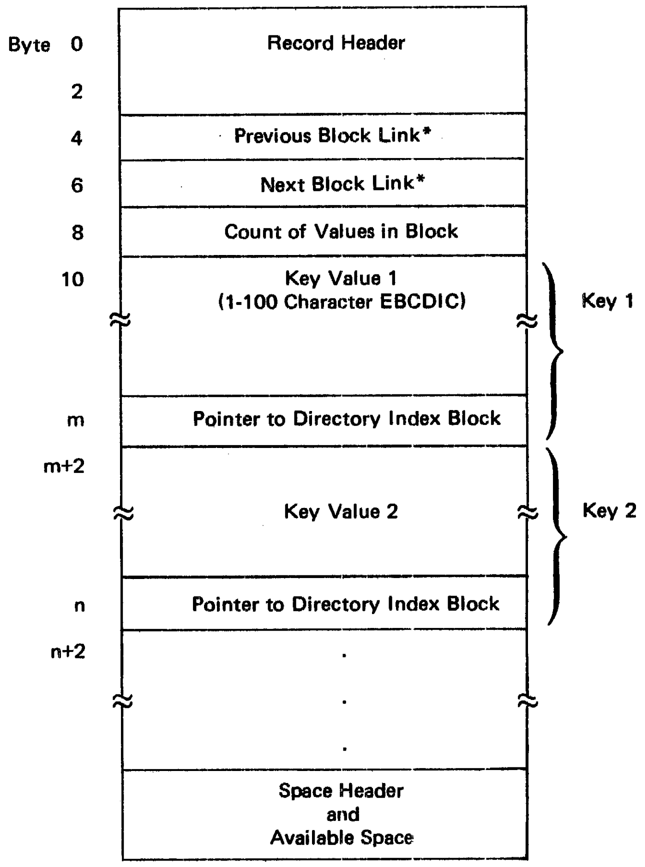
Figure D-2. Information Block

DIRECTORY TO THE DIRECTORY BLOCK

Figure D-3 gives the layout of the Directory to the Directory block. Each key value has an associated directory index block address.

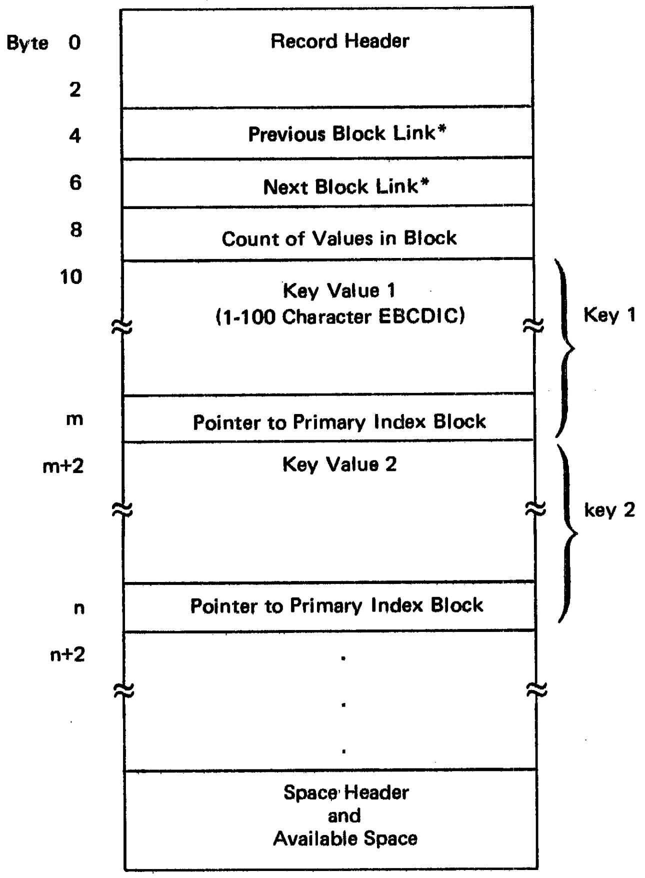
DIRECTORY INDEX BLOCK

Figure D-4 gives the layout of the Directory Index block. Each key value has an associated primary index block address.



Byte 0
2
4
6
8
10
m
m+2
n
n+2
Record Header
Previous Block Link*
Next Block Link*
Count of Values in Block
Key Value 1 (1-100 Character EBCDIC)
Pointer to Directory Index Block
Key Value 2
Pointer to Directory Index Block
Space Header and Available Space
Key 1
Key 2

Figure D-3. Directory to the Directory Block



Byte 0
2
4
6
8
10
m
m+2
n
n+2
Record Header
Previous Block Link*
Next Block Link*
Count of Values in Block
Key Value 1 (1-100 Character EBCDIC)
Pointer to Primary Index Block
Key Value 2
Pointer to Primary Index Block
Space Header and Available Space
Key 1
key 2

Figure D-4. Directory Index Block

*The previous block link and next block link entries are not used in the directory blocks; they are reserved for future use.

PRIMARY INDEX BLOCK

Figure D-5 gives the layout of the Primary Index block. Each key value has an associated block record address.

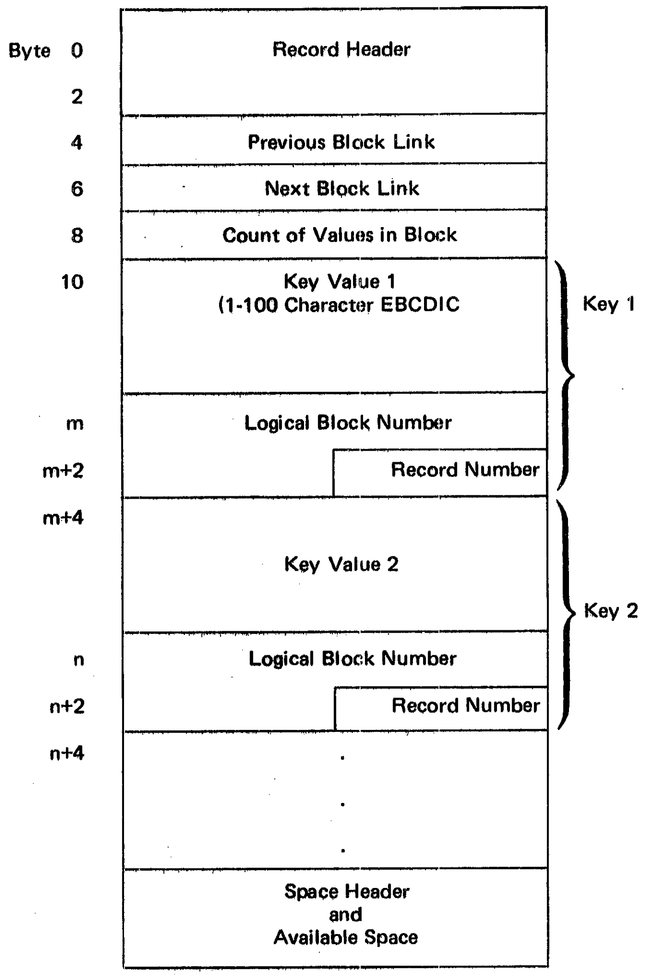


Figure D-5. Primary Index Block

GLOSSARY

Access	Process of obtaining information from or placing information into storage.
Block	A set of words, characters, or records that are recorded as a unit.
Block I/O level	A processing level recognizing no logical records. Data is read or written as a physical data block. Further processing of logical records is a user responsibility.
Buffer switching	The transfer of processing from one buffer to another when two buffers are specified.
Catalog	Ordered compilation of item descriptions and sufficient information to give access to those items.
Close	A function that makes a file unavailable for further processing and does label processing for end of file.
Control Language statement	A statement in a job that is used in identifying the job or describing its requirements to the operating system.
Directive	A control statement (as opposed to data) supplied to a program for the purpose of directing its mode of operation.
Debug	The detection, location, and removal of mistakes from a routine or malfunctions from a computer.
Dump	Copying of contents from internal storage to an external storage.
File	A collection of related records treated as a unit.
File identifier	Identification given by the DEFSF, DEFRF, or DEFIF macro and referenced as the IDENT parameter of the GET/PUT and OPEN/CLOSE macros. IDENT is synonymous with the IDENTIFIER (ID) keyword of the Control Language //DEFINE statement.
File name	Cataloged name of file given by the FILENAME (FIL) keyword of the Control Language //DEFINE statement. This corresponds to the FILENAM parameter of the DEFLB macro.
Global register save area	Storage area used for saving and restoring user registers that can be used by several routines.
Information retrieval	Method and procedure of recovering specific information from stored data.
Job	A specified group of logically related tasks prescribed as a unit of work for a computer.
Job step	The execution of a computer program explicitly identified by a Control Language statement. A job may specify that several job steps be executed.

Key	Characters within a data item that are used to identify it or control its use.
Label	Symbols identifying or describing an item, record, message, or file.
Load point	Preset point (reflector strip) at which magnetic tape is initially positioned under the read/write head to start processing.
Logical I/O level	A processing level that reads and writes logical records. Data Management does the blocking and deblocking.
Macro, action	Macro which generates executable code that results in some action being taken at execution time. Such a macro is always coded in line with the executable part of the program.
Macro, data	Macro which generates nonexecutable code, such as an I/O control table, and should usually be coded in a data area of the program where an attempt to execute it (by mistake) is unlikely. Some macros are inherently data macros in any form while any system macro may become a data macro by including the parameter LIST=YES.
Macro, system	Instructions through which the user avails himself of the various system services, such as I/O processing. They form an extension to the standard machine instruction set and result in the generation of code which provides an interface with the system.
Modification security code	Code that restricts access to a file for security purposes.
Open	Function making file available for processing by checking label and file attributes.
Packing	Contiguous placement of records with no space in between records in a buffer.
Physical I/O level	The lowest level of I/O implementation available to the user. The user may have more responsibility for details, but gains in flexibility. This level of coding is independent of the system file organizations.
Record	Group of related items of data that are treated as a unit.
Record identifier	Identification of a record in terms of block number and relative position within that block.
Relative record number	Number of a record relative to the beginning of the file.
Split cylinder	Denotes the sharing of a cylinder by two files.
Tag	Character(s) attached to record or item for identification.
Tape mark (TM)	Special character written on magnetic tape signaling the physical end of the recording on tape.
Thread	An address pointing to the previous or next item in a series of logically related items.
Unblocked record	Block that contains only one record.

INDEX

ABEND Macro 6-16,17
Access 2-1;3-1;6-3,5;Glossary-1
  random 2-1,4,7;3-1,3,4,8,9
  sequential 2-1,4,7;3-1,2,3,6,8,9
ACCESS parameter 2-4;6-3,4,5,8
Action macro Glossary-2

BDT 6-1
BINARY parameter 6-2,8
BLKFAC parameter 6-2,3,4,5,8
BLKSIZ parameter 6-2,3,4,5,8
Block 2-2;Glossary-1
Block I/O level 3-1;Glossary-1
BREAK 5-2
Buffer description table 6-1
Buffer sharing 3-3;6-2,4,6
Buffer switching 3-3,5;Glossary-1
BUFSHR parameter 6-2,3,4,5,6,8
Bypass tape labels 3-11,12

Catalog 2-8;Glossary-1
  Central 2-10
  disc storage 2-10
  pack 2-10
Central catalog 2-10
CHECKOUT 1-1;5-1,2,4
Close Glossary-1
Close logical file 6-9
Close tape file 3-12
CLOSEL macro 3-3,12;6-1,9
CLOVEL macro 3-12,13;6-1,10
Common stored data format 2-1
Conditional requests 5-4,5
Console communication 4-1
  messages to 4-2
  replies from 4-2
Console communication macros 6-15
  CONSOLE 6-15
  MESSAGE 6-15
CONSOLE macro 4-1,2;6-15
Control characters 2-3
Control language 3-13;Glossary-1
CONTROL parameter 2-3;6-8,9
Control program macros 6-16
  ABEND 6-16,17
  EHALT 6-16,17
  HALT 6-16,17
  JDATE 6-16,17
  SDATE 6-16,17
  TIME 6-16,17
Control program services 1-1;4-1

Data format 2-1
Data macro Glossary-2
Data Management 1-1;3-13
Data Management macros 6-1
  CLOSEL 6-1,9
  CLOVEL 6-1,10
  DEFIF 6-1,5
  DEFLB 6-1,7
  DEFRF 6-1,3
  DEFSF 6-1,2
  DELR 6-1,14
  GET 6-1,10
  LOCRC 6-1,13
  OPENL 6-1,8
  PUT 6-1,12
  PUTU 6-1,14
  SETL 6-1,12
Data portion of indexed files 2-4
Data transmission macros 6-1,10
  DELR 6-1,14
  GET 6-1,10
  LOCRC 6-1,13
  PUT 6-1,12
  PUTU 6-1,14
  SETL 6-1,12
DATATXT parameter 6-15,16
DATBUF1 parameter 6-2,3,4,5,6,8,15
DATBUF2 parameter 6-2,3,4,5,6,8,15,16
DATSIZ1 parameter 6-15
DATSIZ2 parameter 6-16
Debugging 1-1;5-2;Glossary-1
DEFIF macro 2-7;6-1,5
Define indexed file 6-5
Define label 6-7
Define relative file 6-3
Define sequential file 6-2
DEFLB macro 3-13;6-1,7
DEFRF macro 2-4;6-1,3
DEFSF macro 3-8;6-1,2
Delete record 6-14
DELR macro 6-1,14
Device 2-1
  disc 2-1,2,4,7
  printer 2-3
  punch 2-3
  tape 2-1,2,4
  unit record 2-1,2,4
Device usage 2-1
Directive Glossary-1
Directory index 2-6,7;D-2
Directory to the directory index 2-6,7;D-2

- Disc 2-1,2,4,7
- Disc label 2-10
- Disc storage catalog 2-10
- Dump Glossary-1

- EHALT macro 4-1;6-16,17
- END parameter 6-3,5,8
- End-of-file label 2-9
- End-of-volume label 2-9
- ERROPT parameter 6-2,3,4,5,6,8

- FDT 3-11,12,13
- File 2-1;Glossary-1
  - indexed 2-1,3,4,6,8;6-5
  - organization 2-1,3,4,8
  - permanent 2-3,7,8
  - relative 2-1,3,4,5,8;6-3
  - scratch 2-3,7,8
  - sequential 2-1,3,4,5,8;6-2
  - temporary 2-3,7,8
  - type 2-1,3,7,8
  - work 2-3,7,8
- File and label definition macros 6-1
  - DEFIF 6-1,5
  - DEFLB 6-1,7
  - DEFRF 6-1,3
  - DEFSF 6-1,2
- File control macros 6-1,8
  - CLOSEL 6-1,9
  - CLOVEL 6-1,10
  - OPENL 6-1,8
- File description table 3-11,12,13
- File name Glossary-1
- File organization 2-1,4,8
  - indexed 2-1,3,4,6,8
  - relative 2-1,3,4,5,8
  - sequential 2-1,3,4,5,8
- File sharing 3-3
- File type 2-1,3,7,8
  - permanent 2-3,7,8
  - scratch 2-3,7,8
  - temporary 2-3,7,8
  - work 2-3,7,8
- File with standard label 3-12
- FILENAM parameter 6-7
- Fixed length record 2-1,3;6-3,5
- Forward key 6-6

- GET macro 3-3,8;6-1,10
- Get record 6-10
- Global register save area Glossary-1

- HALT macro 4-1;6-16,17
- Header label 2-9

- IBM tape processing 3-12
- IDENT parameter 6-8,9,10,11,12,13,14
- IDXSA parameter 6-10,11,12
- Ignore tape labels 3-11,12
- INDBUF parameter 6-5,6,8
- Indexed file 2-1,3,4,6,8
- Index portion of indexed file 2-7;D-1
- INDSHR parameter 6-5,6,8
- INDSIZ parameter 6-5,6,8
- INFOADR parameter 6-17,18
- Information block D-1
- Information retrieval Glossary-1
- INMAIN parameter 2-7;6-5,6,8
- Input files 3-12
- I/O processing 3-1
  - logical record 3-3
  - magnetic tape 3-8
  - random 3-8,9,10
  - sequential processing with record area 3-3,6
  - sequential processing without record area 3-8,9

- JDATE macro 4-1;6-18
- Job Glossary-1
- Job step Glossary-1

- Key Glossary-2
- KEYADR parameter 6-3,4,8,13,14
- KEYADR1 parameter 6-5,6,8
- KEYADR2 parameter 6-5,6,8
- KEYERR parameter 6-12,14
- KEYSIZ parameter 6-5,6,8

- LABDEF parameter 6-8,9
- Label 2-8;Glossary-1
  - disc 2-10
  - end-of-file 2-9
  - end-of-volume 2-9
  - header 2-9
  - standard tape file 2-9,10;6-2
  - standard tape volume 2-9
  - tape 2-8;6-2
- LABEL parameter 3-8
- Labeled tapes 2-8;3-8,11,12
- LABTYP parameter 3-8;6-2,8
- Linkage conventions B-1
- LIST parameter 6-8,9,10,11,12,13,14;A-1
- Load point Glossary-2
- Locate record 6-13
- LOCRC macro 6-1,13
- Logical I/O level 3-1;Glossary-1
- Logical record 2-2
- Logical record processing 3-3;Glossary-1

Macro expansions A-3,5
Macros 6-1
ABEND 6-16,17
CLOSEL 3-3,12;6-1,9
CLOVEL 3-12,13;6-1,10
CONSOLE 4-1,2;6-15
console communication 6-15
control program 6-16
data management 6-1
data transmission 6-1,10
DEFIF 2-7;6-1,5
DEFLB 3-13;6-1,7
DEFRF 2-4;6-1,3
DEFSF 3-8;6-1,2
DELR 6-1,14
EHALT 4-1;6-16,17
file and label definition 6-1
file control 6-1,8
GET 3-3,8;6-1,10
HALT 4-1;6-16,17
JDATE 4-1;6-18
LOCRC 6-1,13
MESSAGE 4-1,2;6-15,16
OPENL 2-3,4,7,8;3-3,11,13;6-1,8
PUT 3-3,8;6-1,12
PUTU 6-1,14
SDATE 4-1;6-16,17
SETL 3-1;6-1,12
TIME 4-1;6-16,17
Magnetic tape processing 3-8
bypass tape labels 3-11,12
close tape file 3-12
ignore tape labels 3-11,12
open tape file 3-11
MESSAGE macro 4-1,2;6-15,16
Message to console 4-2
MODE parameter 6-15,16,17,18
Modification security code Glossary-2
MSC parameter 6-7

Nonstandard labels 2-8;3-8,11,12

Open Glossary-2
Open logical file 6-8
Open tape file 3-11
OPENL macro 2-3,4,7,8;3-3,11,13;6-1,8
Output files 3-12

Pack catalog 2-10
Packing Glossary-2
Parameters
ACCESS 2-4;6-3,4,5,8
BINARY 6-2,8
BLKFAC 6-2,3,4,5,8
BLKSIZ 6-2,3,4,5,8
BUFSHR 6-2,3,4,5,6,8
CONTROL 2-3;6-8,9
DATATXT 6-15,16
DATBUF1 6-2,3,4,5,6,8,15
DATBUF2 6-2,3,4,5,6,8,15,16
DATSIZ1 6-15
DATSIZ2 6-16
END 6-3,5,8
ERROPT 6-2,3,4,5,6,8
FILENAM 6-7
IDENT 6-8,9,10,11,12,13,14
IDXSA 6-10,11,12
INDBUF 6-5,6,8
INDSHR 6-5,6,8
INDSIZ 6-5,6,8
INFOADR 6-17,18
INMAIN 2-7;6-5,6,8
KEYADR 6-3,4,8,13,14
KEYADR1 6-5,6,8
KEYADR2 6-5,6,8
KEYERR 6-12,14
KEYSIZ 6-5,6,8
LABDEF 6-8,9
LABEL 3-8
LABTYP 3-8;6-2,8
LIST 6-8,9,10,11,12,13,14;A-1
MODE 6-15,16,17,18
MSC 6-7
RECADR 6-2,3,4,5,7,8,10,11,12,14
RECREG 6-2,3,4,5,7,8
RECSIZ 6-2,3,5,7,8,10,11,12
RECTYP 6-2,3,5,8
REWIND 6-8,9,10
RTNADR 6-10,11,12
START 6-3,5,8,12,13
USAGE 6-8,9
VERIFY 6-2,3,5,7,8
Permanent file 2-3,7,8
Physical I/O level 3-1
Primary index 2-6,7;D-3
Printer 2-3
PROG 5-4
Program termination 4-1;6-16
Punch 2-3
PUT macro 3-3,8;6-1,12
Put record 6-12
Put update 6-14
PUTU macro 6-1,14

Random access	2-1,4,7;3-1,3,4,8,9;6-3,5
RECADR parameter	6-2,3,4,5,7,8,10,11,12,14
Record	2-1,2;Glossary-2
Record headers	2-2
Record identifier	Glossary-2
Record size	2-3;6-3,5,7
Record type	2-3;6-3,5
fixed length	2-1,3
variable length	2-1,3
RECREG parameter	6-2,3,4,5,7,8
RECREG use	6-7
RECSIZ parameter	6-2,3,5,7,8,10,11,12
RECTYP parameter	6-2,3,5,8
Relative file	2-1,3,4,5,8;6-3
Relative record number	Glossary-2
Replies from console	4-2
Retrieve job date	6-18
Retrieve system date	6-17
Retrieve time of day	6-17
REWIND parameter	6-8,9,10
RTNADR parameter	6-10,11,12
Save area	B-1
Scratch file	2-3,7,8
SDATE macro	4-1;6-16,17
Sequential access	2-1,4,7;3-1,2,3,6,8,9; 6-3,5
Sequential file	2-1,3,4,5,8;6-2
Sequential processing with record area	3-3,6;6-3
Sequential processing without record area	3-7,8,9
Set limits	6-12
Set up message format	6-15
SETL macro	3-1;6-1,12
Space headers	2-2
Split cylinder	Glossary-2
Standard tape file label	2-9;6-2
Standard tape volume label	2-9
START parameter	6-3,5,8,12,13
System macro	Glossary-2
Tag	Glossary-2
Tape	2-1,2,4
Tape label	2-8;6-2
Tape mark	Glossary-2
Tape organization	2-9
Labeled	2-8;3-8,11,12
nonstandard labels	2-8;3-8,11,12
Unlabeled	2-8;3-8,11,12
Temporary file	2-3,7,8
Terminate program abnormally	6-17
Terminate program normally	6-17
Thread	Glossary-2
Time and date retrieval	4-1
TIME macro	4-1;6-16,17
Transmit message	6-15
Unblocked record	Glossary-2
Unit record device	2-1,2,4
Unlabeled tapes	2-8;3-8,11,12
Update record	6-12,14
Usage conflicts	3-3
USAGE parameter	6-8,9
Variable length record	2-1,3;6-3,5
VERIFY parameter	6-2,3,5,7,8
Volume switching	3-12;6-10
Work file	2-3,7,8

COMMENTS FORM

MRX/OS Control Program and Data Management Services
Basic Reference Manual (2200.001-01)

Please send us your comments, to help us produce better publications. Use the space below to qualify your responses to the following questions, if you wish, or to comment on other aspects of the publication. Please use specific page and paragraph/line references where appropriate. All comments become the property of the Memorex Corporation.

- Is the material:

	Yes	No
Easy to understand?	☐	☐
Conveniently organized?	☐	☐
Complete?	☐	☐
Well illustrated?	☐	☐
Accurate?	☐	☐
Suitable for its intended audience?	☐	☐
Adequately indexed?	☐	☐

- For what purpose did you use this publication? (reference, general interest, etc.)

  ____

- Please state your department's function: ______________________________

  ____

- Please check specific criticism(s), give page number(s), and explain below:

  ☐ Clarification on page(s) ______________________________
  ☐ Addition on page(s) ______________________________
  ☐ Deletion on page(s) ______________________________
  ☐ Error on page(s) ______________________________

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