60499400

CONTROL DATA

CYBER RECORD MANAGER ADVANCED ACCESS METHODS VERSION 2 USER'S GUIDE

CDC OPERATING SYSTEMS:
NOS 1, NOS 2,
NOS/BE 1

ALPHABETIC LIST OF FORTRAN CALL STATEMENTS

INDEXED SEQUENTIAL FILES

DIRECT ACCESS FILES

ACTUAL KEY FILES

60499400

CONTROL DATA

CYBER RECORD MANAGER ADVANCED ACCESS METHODS VERSION 2 USER'S GUIDE

CDC OPERATING SYSTEMS:
NOS 1, NOS 2,
NOS/BE 1

REVISION RECORD

Revision	Description
A (09/01/79)	Original release.
B (05/29/81)	This version reflects CRM Advanced Access Methods Version 2.1 at PSR level 528. Section 4, Multiple-Index Files, has been added, superseding the CRM Version 2 Multiple-Index Processor User's Guide (Publication No. 60480900). All program examples have been updated to FORTRAN Version 5.1. Major organizational changes have been made. This is a complete reprint.
C (03/04/86)	Revised at PSR level 647 to document support of the CYBER 170 800 Series models and the CYBER 180 Computer Systems.

REVISION LETTERS I, O, Q, AND X ARE NOT USED

Address comments concerning this manual to:

CONTROL DATA CORPORATION
Publications and Graphics Division
P. O. Box 3492
SUNNYVALE, CALIFORNIA 94088-3492

or use Comment Sheet in the back of this manual

LIST OF EFFECTIVE PAGES

New features, as well as changes, deletions, and additions to information in this manual are indicated by bars in the margins or by a dot near the page number if the entire page is affected. A bar by the page number indicates pagination rather than content has changed.

Page	Revision
Front Cover	-
Inside Front Cover	B
Title Page	-
ii	C
iii/iv	C
v	C
vi	C
vii	B
viii	B
ix/x	B
xi	B
1-1 thru 1-4	B
2-1 thru 2-8	B
3-1 thru 3-6	B
3-7	C
3-8	C
3-9 thru 3-13	B
3-14	C
3-15 thru 17	B
3-18	C
3-19 thru 3-21	B
4-1	B
4-2	B
4-3	C
4-4	C
4-5 thru 4-34	B
5-1 thru 5-17	B
6-1 thru 6-14	B
7-1	B
7-2	C
7-3 thru 7-6	B
A-1	B
A-2	B
A-3	A
A-4	A
B-1	C
B-2	B
C-1 thru C-3	B
D-1 thru D-9	C
D-10	B
Index-1 thru -5	B
Comment Sheet/Mailer	C
Back Cover	-

PREFACE

CONTROL DATA® CYBER Record Manager Advanced Access Methods (AAM) Version 2.1 operates under control of the following operating systems:

- NOS 1 and NOS 2 for the CONTROL DATA CYBER 180 Series; CYBER 170 Series; CYBER 70 Models 71, 72, 73, 74; and 6000 Series Computer Systems.
- NOS/BE 1 for the CDC® CYBER 180 Series; CYBER 170 Series; CYBER 70 Models 71, 72, 73, 74; and 6000 Series Computer Systems.

AAM handles all input/output processing of files with indexed sequential, actual key, or direct access organization. User programs concerned with any of these file types can communicate with AAM indirectly through a compiler, using the calls supplied by the language; directly through COMPASS macros; or directly through FORTRAN calls. This guide is designed specifically for FORTRAN programmers who are processing files through direct calls to AAM. The material presented, however, can be used to advantage by programmers utilizing COMPASS or any of the languages that provide indirect access to AAM.

If you are already familiar with CYBER Record Manager file organizations, record types, and general file concepts, you can skip sections 1 and 2 of this guide and proceed to the examples of file processing for a specific file organization (beginning with section 3).

Programming examples, written in FORTRAN version 5, emphasize file information table (FIT) field values; specifically, why you set them and how they are interpreted by AAM.

AAM supports two variants of each file organization, initial and extended. This guide is written for users of the extended file organizations. All references to file organizations imply the extended file organizations.

Related material is contained in the listed publications. The publications are listed alphabetically within groupings that indicate relative importance to readers of this manual.

The following publications are of primary interest:

Publication	Publication Number
CYBER Record Manager Advanced Access Methods Version 2 Reference Manual	60499300
CYBER Record Manager Basic Access Methods Version 1.5 Reference Manual	60495700
CYBER Record Manager Basic Access Methods Version 1.5 User's Guide	60495800
FORTRAN Version 5 Reference Manual	60481300
NOS Version 1 Reference Manual, Volume 1 of 2	60435400
NOS Version 1 Reference Manual, Volume 2 of 2	60445300
NOS/BE 1 Reference Manual	60493800

The following publications are of secondary interest:

Publication	Publication Number
Common Memory Manager Version 1 Reference Manual	60499200
FORM Version 1 Reference Manual	60496200
8-Bit Subroutines Version 1 Reference Manual	60495500

CDC manuals can be ordered from Control Data Corporation, Literature and Distribution Services, 308 North Dale Street, St. Paul, Minnesota 55103.

This manual describes a subset of the features documented in the Advanced Access Methods and FORTRAN Version 5 reference manuals. Control Data cannot be responsible for the proper functioning of any features not documented in the Advanced Access Methods or FORTRAN Version 5 reference manuals.

CONTENTS

TABLES

NOTATIONS

The following notations are used throughout the manual with consistent meaning:

UPPERCASE	In language syntax, uppercase indicates a statement keyword or character that is to be written as shown.
lowercase	In language syntax, lowercase indicates a name, number, or symbol that is to be supplied by the programmer.
[ ]	In language syntax, brackets indicate an item that can be used or omitted.
{ }	In language syntax, braces indicate that only one of the vertically stacked items can be used.
...	In language syntax, a horizontal ellipsis indicates that the preceding optional item in brackets can be repeated as necessary.
. . .	In program examples, a vertical ellipsis indicates that statements or parts of the program have not been shown.

Numbers that appear without a subscript are decimal values. Other value formats are denoted as:

n...n	Value is decimal
n...nB	Value is octal
n...nW	Value is decimal, specified in words

INTRODUCTION TO CYBER RECORD MANAGER 1

CONTROL DATA CYBER Record Manager (CRM) is a group of routines that provide an interface between your programs and the operating system routines that read and write files on hardware devices. CRM allows you to group related data into records and to work with organized collections of records (files). The file processing capabilities of CYBER Record Manager are divided into two categories: Basic Access Methods (BAM) and the Advanced Access Methods (AAM). BAM includes the CYBER Record Manager routines that process sequential and word addressable file organizations. AAM includes the Multiple-Index Processor (MIP) and the CYBER Record Manager routines that process indexed sequential, direct access, and actual key files. MIP provides the means to access records by more than one field (that is, alternate paths are available).

Figure 1-1 illustrates the basic CRM components and interfaces.

The CYBER Record Manager routines handle the opening, positioning, and closing of files and perform input/output operations for the files based on information provided by the calling program. This information is conveyed to BAM and AAM through an area in the user's field length called a file information table (FIT). The FIT includes descriptive information that BAM and AAM use to formulate calls for action by the operating system. FIT entries include items such as the method by which the file is to be accessed, the descriptions of record size and type, the current open or close status, and so on.

Your program establishes a FIT for each file that CRM routines reference. This operation is usually performed automatically by the language processor through the language syntax. In most instances, you need not be concerned with the FIT or with the way CRM represents records. As long as you read the file by the same method used to write it, physical representation is irrelevant to you.

However, you must be aware of physical record format to read an existing file from another computer vendor or through a language different from that used in creating the file. After a program or control statement describes the organization and structure of a file, you can access the file with standard read and write statements. You need not refer again to the file description.

As a COBOL programmer, you must use READ and WRITE statements to indirectly access CRM capabilities. As a programmer indirectly accessing CRM, you can:

- Ensure program efficiency by understanding CRM terminology and concepts.
- Supplement, as necessary, input/output processing provided by the COBOL language by using a FILE control statement to override some of the defaults.

As a FORTRAN programmer, you have the following options:

- Using FORTRAN input/output statements for all file manipulations. The fact that CYBER Record Manager becomes involved in execution of these statements can be ignored.

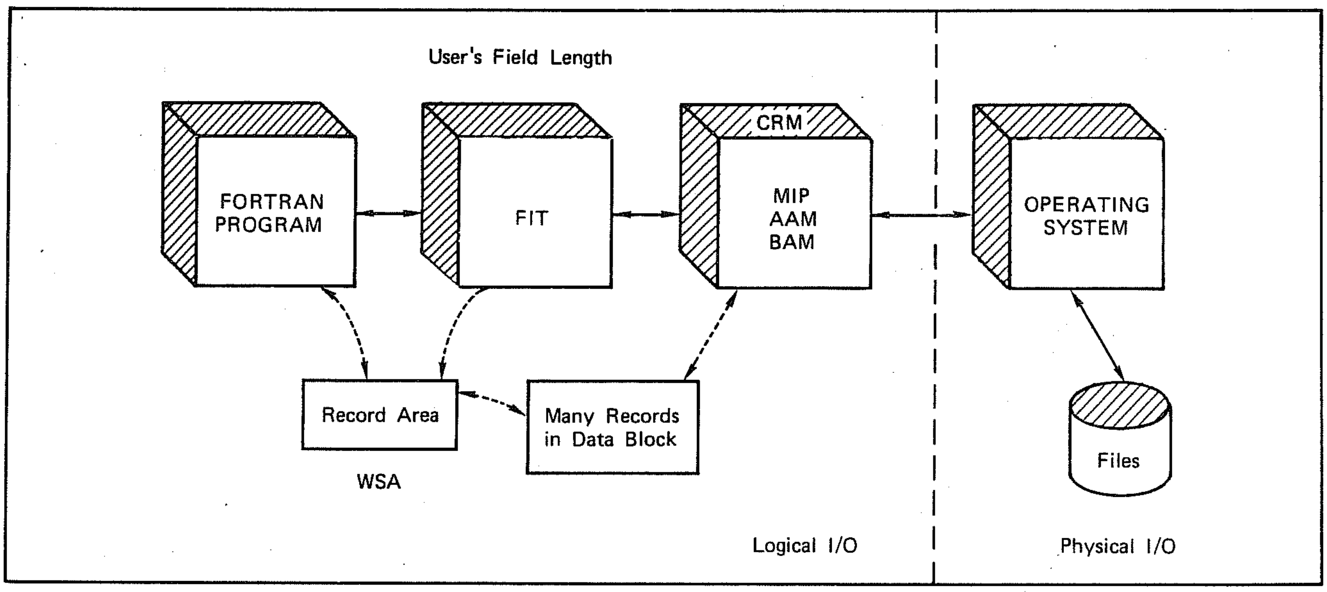


Figure 1-1. CYBER Record Manager Interfaces

- Supplementing input/output processing provided by the FORTRAN compiler by using a FILE control statement to override some of the defaults. Files meeting structure criteria of other systems or languages can be produced in this way. You need only become involved with CYBER Record Manager to the point of properly describing the structure desired.

- Reading and writing files by direct calls to CYBER Record Manager. By using AAM direct calls, your processing options are increased and greater program efficiency can be achieved. CRM must be understood in more detail to successfully issue direct calls.

The file formats created and recognized by direct calls to the CRM routines are independent of the language or processor through which input/output calls are initiated. A file created by one source language can be processed by another. For example, as a FORTRAN programmer with no working knowledge of the COBOL language, you can (by using direct calls) access a file that was created by a COBOL program. Similarly, as a COMPASS programmer with no working knowledge of the FORTRAN language, you can access a file that was created by a FORTRAN program.

Languages that utilize the file management capabilities of BAM and AAM include FORTRAN, COBOL, ALGOL, PL/I, and Query Update. The Sort/Merge and FORM utilities also use CRM. BASIC, UPDATE, and APL do not utilize CRM.

FILE ORGANIZATIONS AVAILABLE

AAM controls the physical processing of indexed sequential (IS), direct access (DA), and actual key (AK) files. Additional processing features are available to all three AAM file organizations when multiple keys are defined. Table 1-1 compares the three AAM organizations.

Some factors you should consider in choosing a file organization include:

- User requirements

  File use - Are all records processed at each use, or are only selected records required? Do you want the best sequential processing, the best random processing, or a good combination?

  File order - Must records be sorted? How often?

  Record keys - Are record key values grouped or scattered? Are keys unique? Must your keys have user meaning? Do you want access by multiple keys?

- Programming effort

  Program creation and maintenance - Do you have programmer expertise for special file creation considerations? Are ease of coding and program maintenance prime considerations?

- File growth

  File size and stability - Are records changed frequently? Is the file expected to grow? Will records be added primarily at the end?

- Performance

  Speed and efficiency - Would your file fit in central memory? How important are processing speed and response-time?

Each file organization offers advantages, depending on the specific file and task at hand. By providing multiple paths to records, MIP improves performance and adds flexibility of access for all AAM file organizations. Often, no file organization will meet all of your needs, and compromises are necessary. A brief overview of each AAM file organization follows.

INDEXED SEQUENTIAL FILES

Indexed sequential (IS) files add the advantages of random processing and updating to those of a sorted sequential file. Records in these files are always logically in ascending order by primary keys. Existing records can be deleted or replaced, and new records can be added, without affecting the remainder of the file. A replacement record can be longer or shorter than the existing record.

Using the character or numeric key associated with each record, CRM creates and maintains an index of the records in the file. You can read any record by specifying its key. In response, CRM searches the index to find the mass storage address of the record and returns to your program the record associated with the key.

Indexed sequential organization is best suited to very large files or files that you need to access both randomly and sequentially. Access times depend upon the number of index levels and the amount of memory allocated to index and data blocks. Typically 0 to 2 disk accesses are required. However, a maximum of 15 accesses might be required in a large, poorly organized file.

Records are grouped into data blocks. Index blocks keep track of data blocks. At creation time, you can plan space for expected growth of indexed sequential files by specifying the size of index and data blocks and by specifying a padding factor. The padding (empty space) can allow for new record insertions without necessarily changing file structure or average retrieval time. Disk storage space must be sacrificed, however. Data block padding should be used with care because the padding is allocated in each data block in the entire file.

Indexed sequential files are generally the easiest to program. Indexed sequential is the best file organization for files processed both sequentially and randomly. IS files are a good compromise between strictly sequential access and strictly random access.

An indexed sequential file becomes a multiple-index file when you define alternate keys for use in retrieving records. An IS file with MIP can be accessed by either its primary key or an alternate key. If you often need to access a file by more than one field, you can improve your program efficiency (in terms of retrieval speed or response time) by using MIP with an IS file.

DIRECT ACCESS FILES

Direct access (DA) files use an access key method involving randomization. The calculation (hashing algorithm) uses the value of the primary key to compute a block number. You can write a routine that performs the necessary calculation or use the system routine provided. The computation should yield block numbers randomly distributed across the range of permissible block numbers. Ideally, a calculated key requires only one access to the mass storage device to find any randomly selected record.

TABLE 1-1. FILE ORGANIZATION COMPARISON

Characteristic	Indexed Sequential	Direct Access	Actual Key	Similarities
Creation Mode	Random by ordered key	Random	Random	
Logical File Structure	Sorted by ascending key value	Random according to hashed key value	In order by block and record slot number	
Physical Structure in Storage	Records in data blocks System maintained index blocks	Records in home blocks Home blocks pre-allocated	Records in data blocks with continuous record slots	
Updating Possible				Delete, add, replace records
User Requirements	Random by key Sequential by position and by ascending key Good hybrid of random and sequential access Major key access	Random by key Serially by position but no logical order Best random processing when alternate keys are not needed	Serially Best random processing when alternate keys are needed	
Programming Effort	Easiest	Harder	Hardest	
File Growth	Moderate	Worst	Best	
Performance	0-15 disk accesses (two or less is typical. Depends on index levels, etc.)	1 disk access is the goal (possible with good hashing algorithm)	1 disk access With MIP, fewest disk accesses and smallest key possible	
Record Types Possible				All
Keys	Symbolic, integer or uncollated keys Embedded key most efficient use of disk	Entire key hashed to home block address Embedded key most efficient use of disk	Key represents storage location (block number and slot number) Non-embedded key most efficient use of disk CRM generates key values	Primary key need not be embedded Alternate keys possible
Storage Medium Possible				Must reside on mass storage during accessing
Utilities	FORM to restructure MIPGEN to add/delete alternate keys FLBLOK to choose best data block size	CREATE to create/restructure file MIPGEN to add/delete alternate keys KYAN for analysis of HMB values and hashing routines	FORM to restructure MIPGEN to add/delete alternate keys	

Records are not maintained in any recognized order, so physical positioning to the next logically sequential record is not possible. However, a direct access file can be processed serially if all records need to be processed. The order of records is not significant in this case.

Direct access file organization is best suited to applications that handle large mass storage files requiring rapid random access (for example, an on-line inventory). A direct access file requires preallocation of disk space.

Direct access file organization is usually the worst choice for files that will change size dramatically (such as a credit check system). DA files require more programming effort than indexed sequential files. Special file creation considerations, such as selection of a hashing algorithm, are required.

Alternate keys can also be defined for a direct access file. A DA file with MIP can be accessed by either its primary key or an alternate key. If you often need to access a file by more than one field, you can improve your program efficiency (in terms of retrieval speed or response time) by using MIP with a DA file.

ACTUAL KEY FILES

Actual key (AK) files use a block number and a record number in that block as the primary key for a given record. Therefore, the unique key for each record represents its storage location. Unlike keys for indexed sequential or direct access files, keys for actual key files do not have an external meaning, such as employee name or number. Logical sequential access by key has no meaning.

CRM generates a key when a record is written to the file. The key value is returned to your program and can be used for future random access by key.

Alternate keys should be defined for most actual key files. AK files with MIP require the fewest disk accesses to obtain a record and the smallest primary key to reduce disk space in the file.

Without MIP, AK files give fast random access to file records, but you are responsible for knowing the storage location (the actual key) of any record. Because you must save the key values generated by AAM for future access by key, AK files require additional effort on your part. You can reduce this effort by combining multiple-index processing with actual key file organization. Actual key values are saved automatically when multiple-index processing is used in conjunction with actual key file organization.

Actual key files might be used for a large data base with many keys to the same record. One file of data records in actual key organization could contain detail information regarding employees. Other files would link the external keys, such as an employee number or department, with the actual key of the data in the first file.

RECORD TYPE SELECTION

The smallest unit of information passed between the user and AAM is a record. The amount of information in a record is determined according to the record type (RT) you specify. Eight different record types are supported by AAM: D, F, R, S, T, U, W, and Z. AAM handles W and S type records as U type records. Each record type is applicable to a specific situation. All records in a given file must be the same record type.

- RT=F

  Use F-type records when each record in your file contains the same number of characters. The following additional FIT field must be defined:

FL	-	fixed record length in 6-bit character bytes.

- RT=Z

  Use Z-type records when you want to eliminate trailing blanks in records. The following additional FIT field must be defined:

FL	-	full record length. This is the maximum number of characters in any record.

- RT=T

  Use T-type records when your records consist of a fixed-length portion (header) followed by a variable number of fixed-length items (trailers). The following additional FIT fields must be defined:

HL	-	Header length in characters
CP	-	Starting character position of count field (numbered from 0)
CL	-	Count field length in characters (one through six)
TL	-	Trailer length in characters
MRL	-	Maximum record length

- RT=W

  W-type records are the default. The following additional FIT fields must be defined:

RL	-	Record length in characters
MRL	-	Maximum record length

More detail on record types (including types D, R, S, and U) can be found in the BAM and AAM reference manuals.

FILE PROCESSING CONCEPTS 2

You process all AAM files in the same general manner:

1. Open the file.

2. Position to a specific record, if desired.

3. Read, write, rewrite, or delete records.

4. Close the file.

This section of the guide presents the basic concepts of file processing to familiarize you with the general principles involved in processing an AAM file. Subsequent sections discuss file processing in detail for each AAM file organization. All examples use FORTRAN direct calls to AAM. COBOL users should refer to the COBOL user's guide.

You might issue direct calls to AAM if you are one of the following:

- A FORTRAN programmer writing applications programs that require the advantages available through direct calls to AAM.

- A FORTRAN or COMPASS programmer writing execution time routines for high level languages.

- A FORTRAN or COMPASS programmer writing input/output modules for use by applications programs.

- A COMPASS programmer writing modules to handle user label processing for applications programs.

As a programmer directly accessing AAM, you are responsible for the following:

- Establishing the FIT to define the file structure and subsequent processing limits.

- Establishing a working storage area (WSA) in a program for passing data records between the program and the file storage device.

- Issuing direct calls to AAM to open the file, to process input and output operations, and to close the file.

When an AAM file processing statement is executed, parameters in the statement are placed in their respective FIT fields. Omitted parameters do not affect the current FIT values.

CONSTRUCTING AND USING THE FIT

You must establish a file information table (FIT) for each file AAM is to process before you can open the file. The contents of the fields in this table define the structure of the file and govern file processing. The FIT is a 35-word table in the user's field length that contains information such as record type, record size, file organization, error count, error flags, and information used by internal system routines to determine file status and position. Some fields are set by you, and other fields are set by AAM.

FIT fields can be set when CRM constructs the FIT, opens the file, executes a processing statement, or executes a STORE macro (in COMPASS) or a CALL STOREF (in FORTRAN). When the file is opened, many fields that have not been set to a specific value are set to default values. A default value is in effect until the field is changed. When a file processing statement is executed, AAM uses the current contents of the applicable FIT fields.

CREATING THE FIT

You must allocate 35 words of central memory for the FIT. The allocated space is then used when your call to the FILExx subroutine is executed. The format of this call is:

```
CALL FILExx (fit, field, value, ...)
```

The mnemonic specified for xx determines the file organization.

- FILEIS specifies an indexed sequential file.

- FILEDA specifies a direct access file.

- FILEAK specifies an actual key file.

The first parameter in the CALL FILExx statement is the name of the array in which the FIT is to be built. This name identifies the FIT and is the first parameter in every direct call to AAM. The remaining parameters are FIT field mnemonics and values for the fields. When the CALL FILExx statement is executed, the FIT is established in the named array, zeros are entered in all fields, and then the specified fields are set to the designated values.

All of the FIT fields required for file processing can be set by the CALL FILExx statement. You identify FIT fields by mnemonics such as RT (record type) and LFN (logical file name). Values for the fields can be program locations, positive integers, and symbolic options. Appendix D lists the FIT fields applicable to each file organization and indicates the fields that can be set by the CALL FILExx statement. More detailed discussions of the individual FIT fields can be found in the sections on the file organizations.

The first parameter in a CALL FILExx statement is the array name that reserves space for the FIT. All subsequent parameters in the statement are paired. The first parameter of a pair specifies the FIT field mnemonic, and the second parameter specifies the value for the field. Some examples of paired parameters are:

- 'LFN','NEWFILE'

  Logical file name NEWFILE

- 'RT','F'

  F type records

- 'FL',42

  Fixed-length records of 42 characters

Pairs of parameters can appear in any order in the parameter list. Figure 2-1 shows some examples of the CALL FILExx statement.

```
CALL FILEIS (ISFIT, 'LFN', 'ISFILE', 'RT', 'F')

     The FIT for the indexed sequential file is constructed
     in the array named ISFIT; the logical file name is
     ISFILE and the record type is F.


CALL FILEAK (AKFIT, 'LFN', 'AKFILE', 'KL', 6,
        'MNR', 80, 'MRL', 100)

     The FIT for the actual key file is constructed in the
     array named AKFIT; the logical file name is AKFILE,
     the key length is 6 characters; the minimum and
     maximum record lengths are 80 and 100 characters,
     respectively.


CALL FILEDA (DAFIT, 'LFN', 'DAFILE', 'HMB', 53,
        'RB', 11)

     The FIT for the direct access file is constructed in
     the array named DAFIT; the logical file name is
     DAFILE, the number of home blocks is 53, and
     the number of records per block is 11.
```

Figure 2-1. CALL FILExx Statement Examples

Many fields are assigned a default value if no value is specified. You should be aware of each default value and the effect it can have on a program (See appendix D).

Any specified value that exceeds the maximum field size is truncated. Misspelled or otherwise unrecognizable FIT mnemonics are noted on the dayfile and ignored. In each of these cases, CRM issues an informative diagnostic. A mnemonic that is valid but not applicable to the specified file organization is ignored providing it produces no conflict with other mnemonics. If a conflict exists, CRM issues an appropriate diagnostic at execution time. FIT field values are also checked for validity and consistency at execution time. If the same value is referenced more than once, the last value specified is used.

The following fields should be considered for the CALL FILExx statement in most programs:

LFN	Logical file name
ORG	Old/New (initial/extended) version of file organization - NEW should be specified and is required to override the installation default
RT	Record type (and additional fields required by the individual record type)
WSA	Working storage area
KA	Key address
EMK	Embedded key
KL	Key length
KP	Key position
RKW	Relative key word
RKP	Relative key position
FL	Fixed/full record length
MRL	Maximum record length
MNR	Minimum record length
MBL	Maximum block length
EFC	Error file control
DFC	Dayfile control

USING THE FIT THROUGH FORTRAN CALLS

FORTRAN routines access BAM and AAM through direct calls to routines in the system library. Appendix B summarizes the call statements and their appropriate parameters.

With FORTRAN direct calls to AAM, you can process files with organizations not available through the standard language statements. By specifying the primary key, you can read or write any record in an indexed sequential, actual key, or direct access file without regard for any other record in the file.

Execution of an AAM file processing statement depends on the current contents of the FIT. When you open an existing file, some values stored in the File Statistics Table (FSTT) are returned to the FIT (such as KL, RKP, RKW, KT, and MRL). You are responsible for setting the contents of appropriate fields at execution time. You can provide a field value in one of several ways:

- Specifying the field and value in the CALL FILExx statement. The value becomes part of the FIT when the statement is executed.
- Omitting the field definition (if permitted) and accepting the default value. The default becomes part of the FIT at the time the file is opened.
- Specifying the field as a parameter in the FILE control statement. The value becomes part of the FIT when the file is opened, and it overrides any previous value set in the field.
- Executing the CALL STOREF statement to store a value directly into the FIT. The field is set at the time the statement is executed and overrides any previous value set in the field.
- Setting the field value in a file processing statement. A value from a CALL GET or CALL PUT statement is set in the FIT when the statement is executed.

The last value set in a FIT field governs an operation using that field. The default value remains in effect until it is changed.

Figure 2-2 illustrates a FORTRAN program with direct calls.

```
      PROGRAM NEWIS
      IMPLICIT INTEGER (A-Z)
      DIMENSION ISFIT (35), ISWSA (5)<----------------- Allocate memory space for FIT and WSA
      CALL FILEIS (ISFIT, 'LFN', 'ISFILE', 'ORG', 'NEW',
     +            'WSA', ISWSA,
     +            'RT', 'F', 'FL', 50,
     +            'MBL', 590, 'NL', 3,                      Identify file named ISFILE and set FIT fields
     +            'KL', 4 'EMK', 'YES',
     +            'RKW', 0, 'RKP', 4,
     +            'EFC', 3, 'DFC', 3)
      CALL OPENM (ISFIT, 'NEW')<----------------------- Open file
   10 READ (*, '(3A10, 2I10)', END = 20) ISWSA
      CALL PUT (ISFIT)                      <---------- Process
      PRINT 100, ISWSA
      GO TO 10
   20 CALL CLOSEM (ISFIT)<----------------------------- Close file
      STOP
  100 FORMAT (3A10, 2I10)
      END
```

Figure 2-2. FORTRAN Program Using AAM Direct Calls

When AAM is called directly, the program must meet the following conditions for each file:

1. You must allocate a 35-word block of central memory for the file information table (FIT).

2. You must identify the file organization by a CALL FILExx statement. Any file defined to AAM through a CALL FILExx statement must not appear in the PROGRAM statement.

   You must issue the CALL FILExx statement before any other AAM call. Parameters in the call to FILExx establish the logical file name and other FIT fields to guide processing.

3. You must open the file by a call to OPENM.

4. You must use AAM calls for all read and write operations on the file identified in the CALL FILExx statement. READ and WRITE or other standard input and output statements, including READMS and WRITMS, must not be used for processing the AAM file.

5. After all processing, you must close the file by a call to CLOSEM to ensure file integrity.

You can write records to the file from an array or character variable identified by the working storage area (WSA) field in the FIT. You can change the WSA field at any time, or you can leave it pointing to the same array or variable.

Table 2-1 summarizes the FORTRAN direct call statements.

CALL STOREF Statement

You can set or change a FIT field value during program execution by the CALL STOREF statement. The format of this statement is as follows:

CALL STOREF (fit, field, value)

You can execute the CALL STOREF statement before or after opening the file. However, the statement must be after execution of the CALL FILExx statement. Some FIT fields cannot be set after opening the file (such as MRL and MNR). Other FIT fields can be set after opening the file (such as WSA). Appendix B indicates fields that can be set by the CALL STOREF statement.

You can set only one field in the FIT by each CALL STOREF statement. The mnemonic for the field and the value to be stored in the field are specified in the statement. The value can be an integer, integer variable, or a symbolic option. Integer values are retained as right-justified integers in the FIT. Symbolic values become bit strings and are fetched as a single bit or an integer, depending on the bit string length.

Figure 2-3 shows some examples of the CALL STOREF statement.

CALL STOREF (ISFIT, 'RL', 80)

> Execution of this statement stores 80 in the RL (record length) field of the FIT in the array named ISFIT.

CALL STOREF (DAFIT, 'ERL', 10)

> Execution of this statement stores 10 in the ERL (trivial error limit) field of the FIT in the array named DAFIT.

Figure 2-3. CALL STOREF Statement Examples

IFETCH Function

You can retrieve the value in a FIT field by using the IFETCH integer function. The format is as follows:

IFETCH (fit, field)

TABLE 2-1. SUMMARY OF FORTRAN CALLS

Function	Call Name	Applicable File Types	Action Taken	Comments
File creation and maintenance	FILExx	IS,AK,DA	Creates a file information table (FIT).	Must be the first call executed. Any file name defined in this statement must not appear in the PROGRAM statement.
	IFETCH	IS,AK,DA	Retrieves the value of a specified field in the FIT.	Can precede an OPENM call.
	STOREF	IS,AK,DA	Sets a value in a field in the FIT.	Can precede an OPENM call.
	FITDMP	IS,AK,DA	Dumps the contents of a FIT to the error file.	Forces the EFC field to 2 or 3.
File initialization and termination	OPENM	IS,AK,DA	Opens a file.	Must be executed before any file access call can be executed.
	CLOSEM	IS,AK,DA	Closes a file.	Only IFETCH, STOREF, and FITDMP calls can follow a CLOSEM call.
Data transfer	GET	†IS,AK,DA	Reads a record randomly by key.	The specified key value identifies the record to be retrieved.
	GETN	†IS,AK,DA	Reads the next record in sequence.	Retrieves records in sequence of position in the file.
	PUT	IS,AK,DA	Writes a record by primary key.	The RL field must be set to the record length for U type records.
File updating	DLTE	IS,AK,DA	Deletes a record.	The specified primary key must identify an existing record.
	REPLC	IS,AK,DA	Replaces a record.	The primary key in the new record must duplicate an existing primary key.
File positioning	REWND	†IS,AK,DA	Rewinds a file.	The file rewinds to beginning-of-information.
	SKIP	†IS,AK	Skips a number of records forward or backward.	Positions the file relative to the current file position.
	STARTM	IS	Positions a file for sequential processing.	Positioning depends on a specified condition.

†Also applies to alternate key index files (see section 4).

You can use IFETCH before or after opening the file. Appendix B indicates fields that can be retrieved by IFETCH.

The format of the value returned depends on the type of field you request, as follows:

- A field that you specify as an integer is returned as a right-justified integer.
- A length field value is returned as a count of 6-bit characters except for the buffer size (BFS) field, which is returned as a count of 60-bit words (as specified).
- A one-bit field value is returned as a positive integer for 0 or a negative integer for 1.
- A symbolic field value that requires more than one bit is returned as a positive integer. For example, the LFN field is returned as a hollerith constant (left-justified and zero filled) in the integer variable. (Refer to appendix D for the integer values.)

Figure 2-4 shows some examples of the IFETCH function.

```
IF (IFETCH (ISFIT, 'ES') .NE. 0) GO TO 50

    This statement causes a branch to statement 50 if the
    ES (error status) field contains a nonzero value.

IF (IFETCH (ISFIT, 'FP') .EQ. O"10") GO TO 20

    This statement causes a branch to statement 20 if the
    FP (file position) field is set to 10.

ICOUNT = IFETCH (AKFIT, 'RL')

    This statement returns the value of the RL (record
    length) field to the variable ICOUNT.

IF (IFETCH (ISFIT, 'LFN') .EQ. L"ISFILE") GO TO 20

    This statement causes a branch to statement 20 if the
    LFN field is set to ISFILE.
```

Figure 2-4. IFETCH Function Examples

USING THE FIT THROUGH COMPASS MACROS

COMPASS routines access AAM through macro calls. The AAM reference manual details the macros and their appropriate parameters.

You should use COMPASS macro calls to AAM if you want to use AAM files. The READ, WRITE, or other macros previously available that used the CPC (central program control) routines cannot access AAM files. You cannot use both CRM and CPC in one program to process a given file. Both sets of macros can be used in the same program only as long as they are processing different files. Existing files created through CPC can be processed by CRM once the file and record structure are properly defined in CRM terminology.

When AAM is called directly, the program must meet the following conditions for each file:

- The FILE macro must appear in a nonexecutable portion of the program. The macro is an assembly-time statement that results in construction of the FIT. The FIT address, rather than the file name, is used in all AAM macro references to the file.
- The file must be opened by an OPENM macro.
- All read and write operations for the file must occur through AAM macros.
- After all processing, the file must be closed by a CLOSEM macro.

THE FILE STATISTICS TABLE

The file statistics table (FSTT) contains file characteristics and access statistics. AAM creates and maintains the FSTT. The FSTT is always the first two physical record units (PRUs) in an AAM file. The FSTT stores information that cannot be changed for the life of the file and statistics about the file.

The FSTT is created when the file is created and is a permanent part of the file. When the file is opened for creation, the file and primary key structures you defined in the FIT are stored in the FSTT. This information is returned to the FIT whenever you subsequently open the file. FSTT information guides processing as long as the file exists. If you set fields in the FIT to change file structure, values that do not conform to the FSTT are rejected. Because you cannot directly access the FSTT, a program cannot inadvertently destroy the file through the FIT.

Statistics such as the number of records inserted or deleted are accumulated with each use of the file and are stored in the FSTT. You can sample the statistical information by using the FLSTAT control statement.

DEFINING THE WORKING STORAGE AREA

The working storage area is a user-defined area in the program where AAM finds or returns one record for a write or read operation. The address of this area is stored in the working storage area (WSA) field in the FIT. The working storage area is typically a one-dimensional, character or integer array. If specified as a character type data structure, WSA must be word-aligned.

Figure 2-5 shows the events that occur in response to your request to write a record. AAM moves the record from the working storage area into the file data block area in central memory. (No input/output is performed at this time.) AAM makes a request to the operating system to transfer the data block contents to a storage device.

In response to your request to read a record, AAM moves one record from the data block area into the working storage area. If the record is not in the data block currently in central memory, the operating system is requested to transfer the proper block of information to central memory so that your request can be carried out. You are aware only that records are moved from and to the working storage area upon request. All other record or block movement is internal to AAM.

The working storage area can be changed anytime during execution of the program. Each record written can originate in a different area of the program. You need not move a record to a specific area for writing. Any program location specified for the WSA field by the write request, or stored in the WSA field before the write request, is used during execution of the write request. Likewise, you need not move a record to a specific area after reading. The program location specified by the WSA field is used during execution of the read request.

AAM does not consider the working storage area to have a specific length. AAM uses the record type (RT) field (and any associated FIT fields) to determine how many characters are to be transferred to or from the working storage area. You must allocate space for the working storage area accordingly.

A full record is returned for each read request. The number of characters in the record is always returned to you in the RL field in the FIT. The working storage area is defined as an integral number of words. However, only the specific number of characters (RL) that are a part of the record are valid.

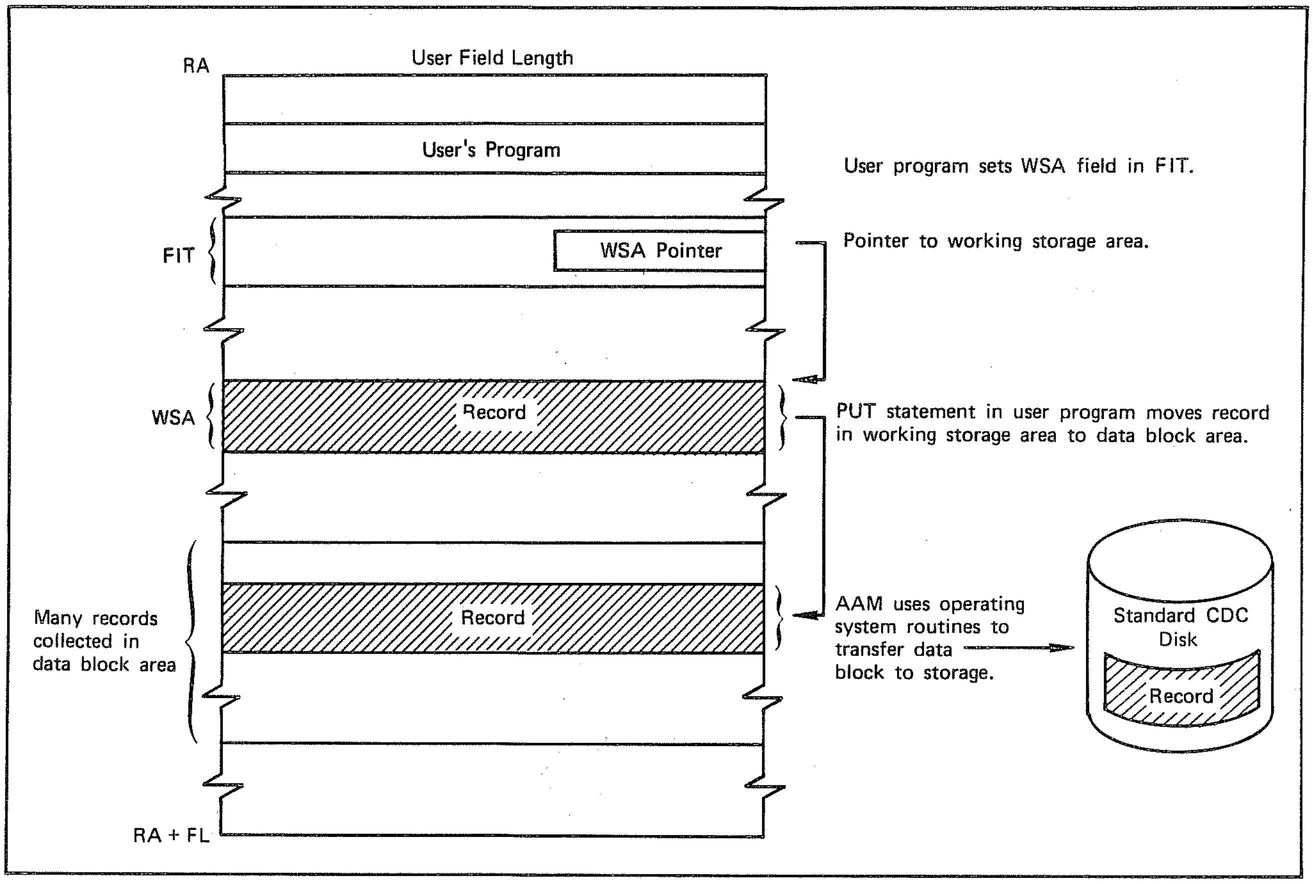

RA
User Field Length
User's Program
User program sets WSA field in FIT.
FIT
WSA Pointer
Pointer to working storage area.
WSA
Record
PUT statement in user program moves record in working storage area to data block area.
Many records collected in data block area
Record
AAM uses operating system routines to transfer data block to storage.
Standard CDC Disk
Record
RA + FL

Figure 2-5. Working Storage Area Use

A write request uses the RL field only for U, W, S, or Z type records. AAM transfers the number of characters set in the field. For record types other than U, W, S, and Z, AAM transfers the number of characters calculated according to the record type.

OPENING THE FILE

You must open the file (with OPENM) before performing any other file processing. Open processing includes the following events in the order listed:

1. FIT fields specified on the OPENM call statement are set.

2. FILE control statement parameters are placed in FIT fields. FILE statement values can override values previously set by the CALL FILExx or the CALL STOREF statement.

3. For an existing file, file descriptions are extracted from the FSTT and stored in the FIT. FSTT values can override previous FIT values.

4. Minimum parameters required by the file organization are checked.

5. FIT fields are checked for consistency in logic.

6. Buffer space is calculated by AAM unless the space has been allocated by the user.

7. The open/close flag (OC) field in the FIT is set to open.

8. For a new file, the file statistics table (FSTT) is constructed.

If AAM detects an error in format or detects an omission or inconsistency in logic in the FIT fields, an error status code is put in the ES field and an appropriate diagnostic is issued (if EFC and/or DFC have been properly set). Errors frequently occur when not all fields required for a particular file organization have been defined correctly, or when one specified field precludes another specified field. An error detected during open processing prevents the open/close flag (OC) field from being set to open. After an OPENM, you should check the ES field.

PROCESSING THE FILE

You can read, write, delete, and replace records in an AAM file. Both sequential and random access are supported for AAM files.

The following is a list of suggested programming practices to use during all AAM file processing (FORTRAN direct calls are assumed).

- Declare the 35-word FIT array INTEGER.

- Declare the WSA field CHARACTER or INTEGER (If CHARACTER is used, the WSA field must be word-aligned.)

- Check the error status (ES) field after every file operation.
- If the ES field is valid (the value is zero), then check the file position (FP) field after every GET or GETN call.
- If the ES and FP fields are both valid, then check the record length (RL) field after every GET or GETN call.
- Use the FILExx statement to set most FIT parameters prior to the file operation.
- Use only the FIT name parameter for most direct calls. (In most cases, the FIT name is the only required parameter.)
- Use the STOREF subroutine to change FIT fields.

The processing direction (PD) field in the FIT determines the operations that can be performed on the file. For reading records, you must set the PD field to either INPUT or I-O. For writing records, PD must be set to either OUTPUT or I-O. Delete and replace operations require the PD field to be set to I-O. NEW specified with the OPENM macro (in COMPASS) or the CALL OPENM call (in FORTRAN) sets the ON field to NEW and the PD field to OUTPUT, as required for file creation.

READING RECORDS

You can read records either sequentially by position or randomly by key value. A sequential read (using GETN) accesses the next record in the file. A random read (using GET) accesses the record associated with the specified key value. When a read operation is performed, AAM returns a record from the buffer to the working storage area defined by the WSA field in the FIT.

At the end of the read operation, the record length (RL) field in the FIT is set by AAM if there were no errors. The RL field reflects the number of characters returned to the working storage area. After a GET or GETN, you should check the ES field, the FP field, and then the RL field.

The number of characters transferred to the working storage area during any read operation is affected by the maximum record length (MRL) field in the FIT as well as by any other control information pertinent to the record. The value in the MRL field sets an absolute limit on data transfer as follows:

- If the MRL field is zero, no data can be transferred to the working storage area.
- If the MRL field is greater than zero, the specified value sets an upper limit on the number of characters that you can read, even if this limit is smaller than a given record.

You are responsible for setting the MRL field before the file is created. The value is stored in the file statistics table and is available during any future access without repeating the specification. It is a good idea to set the MRL field to the size of the working storage area.

WRITING NEW RECORDS

A write operation (using PUT) causes a record to be moved from a working storage area to the buffer (data block area) for the file being processed. The value of the primary key determines the position that a record occupies in the file.

Determine the length of the record written by looking at the record length (RL) field in the FIT or other control information appropriate to the record type. After a PUT, you should check the ES field, the FP field, and then the RL field.

Indexed sequential files can be written sequentially or randomly. On a creation run, however, you should try to write records in sequential order by primary key value for a more efficient run. On subsequent runs, records can be inserted randomly.

Direct access files are always written randomly. The primary key of the record determines the home block in which the record resides.

If you want to create a direct access file with a large number of records, it is more efficient to use the CREATE utility than to put each record into the file as it comes from the source program. The CREATE utility sorts the records to minimize disk accesses to the file.

When actual key files are written, AAM generates the primary key value for each record as blocks are filled.

UPDATING THE FILE

You add records to an existing file by using a write request (PUT). You can also delete records (with DLTE) or replace them with new records (with REPLC). All updating operations require the primary key to identify the affected record.

You delete a record from the file by providing the primary key value for the record. The working storage area is not required. The record is either flagged as deleted or physically deleted from the file.

You can replace any record in an existing file by a new record that has the same primary key value. The new record can be shorter or longer than the original record as long as the record length is within the limits established by the minimum record length (MNR) and maximum record length (MRL) fields in the FIT. The record in the working storage area replaces the record in the file with the specified primary key value.

An error detected during file updating operations is returned to the ES field. After a PUT, DLTE, or REPLC call, you should check the ES field.

CLOSING THE FILE

When file processing is complete, the file must be closed (with CLOSEM). Close processing includes the following events:

- User data records are written to the file storage device from the central memory buffer.
- The file statistics table (FSTT) is updated.
- The open/close flag (OC) field in the FIT is set to closed.

A CLOSEM (in COMPASS) or CALL CLOSEM (in FORTRAN) allows the file to be reopened later. However, the closed file is returned to the operating system when U is specified as the second parameter in the CLOSEM statement.

THE FILE CONTROL STATEMENT

The FILE control statement is used to set values in FIT fields when the file is opened. The specified values can either set fields not previously set or override values specified in the CALL FILExx statement. Values from the FILE control statement are saved by CRM until the file is opened for the first time during program execution. The FILE control statement can appear anywhere in the control statement record before the load sequence that calls for execution of the compiled program.

Many of the FIT fields that can be set by the CALL FILExx statement can also be set by the FILE control statement. Fields that specify a program address, however, (such as WSA or KA) cannot be set by a FILE statement. Appendix D lists the FIT fields applicable to each file organization and indicates the fields that you can set by the FILE control statement.

The first parameter in the FILE control statement must be the logical file name. This is followed by one or more parameter pairs that specify FIT field settings. Each pair consists of a FIT field mnemonic and a value separated by an equal sign. Figure 2-6 shows some examples of the FILE control statement.

You cannot use the FILE control statement to change permanent information about an existing file. For example, the block size cannot be changed after file creation. Any attempt to change such a field is overridden by the value stored in the FSTT. A diagnostic is not issued. Fields specifying permanent information are used by AAM only on a file creation run.

A FILE control statement cannot be continued to a second statement. When you cannot specify all parameters in one statement, you can include additional FILE control statements with the same logical file name. If the same FIT field is referenced in more than one FILE control statement, the last value encountered defines the field. The PREVIOUS FIELD OVERLAPPED message is recorded on the dayfile.

```
FILE(DAFILE,FO=DA,HMB=5,MRL=80,KL=10)
```

The direct access file has the logical file name DAFILE. The number of home blocks (HMB), maximum record length (MRL), and key length (KL) fields are specified for the file.

```
FILE(AKFILE,FO=AK,EFC=1,RB=64,BCK=YES)
```

The actual key file has the logical file name AKFILE. The error file control (EFC), records per block (RB), and block checksum (BCK) fields are specified for the file.

Figure 2-6. FILE Control Statement Examples

The following FILE statements are often useful:

- FILE,lfn.

  Cancels preceding FILE control statements for lfn.

- FILE.

  Cancels all preceding FILE control statements.

An error in the FILE control statement causes the entire statement to be ignored. The error and the parameter in question are printed on the dayfile. Control transfers to any appropriate EXIT control statement.

INDEXED SEQUENTIAL FILE PROCESSING 3

An indexed sequential file is a mass storage file of records that are stored in logically sequential order. You can access records randomly by key value or sequentially by position in the file. This file organization is well suited for applications requiring efficient storage and retrieval of records both randomly and sequentially.

This section discusses the requirements for creating and processing an indexed sequential file by primary key. For alternate key processing, refer to section 4. If alternate keys are defined, the data file must conform to the requirements discussed in this section. The alternate key index is separate from the index associated with the data records.

Each record in an indexed sequential file has a primary key associated with it. As records are written to the file, AAM creates an index using that key. The order of the index entries, as well as the order of the records, depends on the value of the primary key. Keys can be symbolic, integer, or uncollated. However, all keys in one file must be the same type. AAM locates the record for any given primary key by searching the index. The primary key need not be embedded in the record.

An indexed sequential file must reside on a mass storage device. The file can be dumped to tape with a COPYBF or a permanent file dump routine. Because all addresses of index and data blocks are expressed internally as relative locations, the file can be returned later to mass storage without the need to create a new file structure.

Using the FORM utility, you can create an indexed sequential file from a sequential, direct access, or actual key file. You can also re-create an existing indexed sequential file in the same or different file structure.

CONCEPTS OF LOGICAL FILE STRUCTURE

An indexed sequential file consists of data blocks containing a group of records and internal pointers to those records, a file index grouped into index blocks, and a file statistics table (FSTT) created and used by internal routines. You are responsible for defining data block size but you do not manipulate information in the blocks. All index and data blocks are the same size. AAM has sole responsibility for creation and use of the FSTT.

DATA BLOCKS

Records in an indexed sequential file are grouped into data blocks. All data blocks in the file are the same size. Each block contains a two-word header, data records, padding, and record pointers.

Records are always in logical ascending order according to the primary key values. Integer keys are ranked according to numeric value. Symbolic keys are ranked according to a collating sequence value associated with each character. When the collating sequence values are ordered, the symbolic keys are ordered. From your standpoint, the file beginning is the start of the data record with the lowest key value. The end of the file is the data record with the highest key value. Read operations do not return the FSTT, record pointers, index blocks, block headers, or other information created by AAM.

The data records are stored following the block header in ascending sequence of primary key values. When records are added to the file, AAM relocates existing records in the block so that records are always in physical and logical order for sequential access. If the data block cannot accommodate another record, the block is split. Some of the records remain in the existing block; the rest are moved to a newly created data block.

Record pointers are stored at the end of the data block beginning with the last word. Two record pointers are stored in one word. Only one record pointer per block is needed when all records in the block are the same length. Otherwise, one record pointer is required for each record in the block.

Figure 3-1 illustrates the structure of a data block. The two-word block header is at the beginning of the block and is immediately followed by four data records. The four record pointers are in the last two words of the block. Empty space (padding) has been included between the last data record and the record pointers to allow for inserting records at a later time.

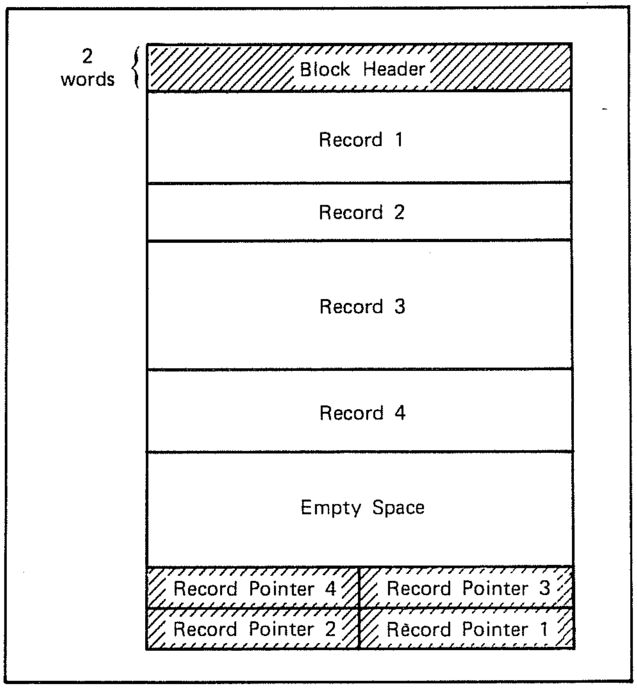


Figure 3-1. Indexed Sequential File Data Block Structure

INDEX BLOCKS

AAM creates and maintains an index that links the primary key of a record to the data block in which the record resides. Entries for the index are grouped into index blocks. A single primary index block, which can lead to lower level index blocks, is always maintained by AAM. A maximum of 15 levels of index blocks is allowed. However, performance is better when no more than three levels exist.

Index blocks are all the same size, which is also the same size as data blocks. The first two words of the index block are the block header. The header is followed by key entry records, empty space (padding), and a record pointer. Only one record pointer per block is needed because the entries are fixed length.

A key entry record contains the primary key value for the first record in a block and the physical record unit (PRU) number of that block. Key entry records are stored in ascending sequence of primary key values. A key entry identifies all records with primary key values between its key and the key identified by the next ordered key entry. The block referenced by the key entry record is either an index block in the next lower level of index blocks or the data block containing the record.

Padding in index blocks is particularly significant because index blocks must contain a key entry record for each data block in the file. If no padding exists in index blocks at the end of file creation, new data blocks cannot be added without forcing another index block. Another index level could result. If the file structure is defined with initial padding that allows for more key entry records, the index levels will probably remain constant.

The structure of an index block is shown in figure 3-2. The six key entry records reference the first record in each of six lower level index blocks or six data blocks. Only one record pointer is needed because key entry records are fixed length.

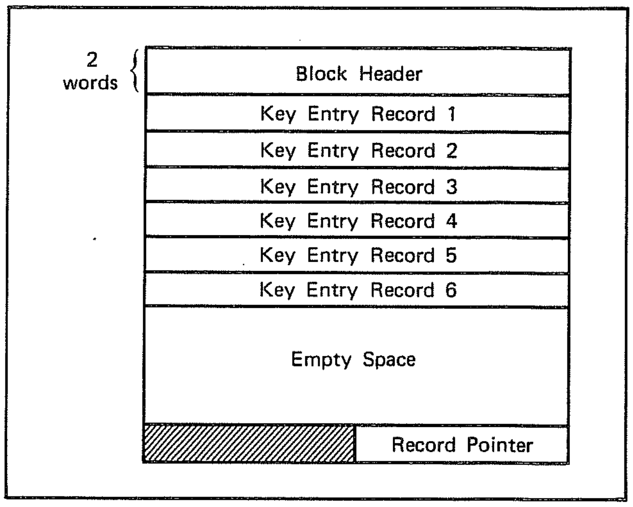


Figure 3-2. Indexed Sequential File Index Block Structure

HIERARCHY OF INDEX AND DATA BLOCKS

When the first data record is written to the file, no index blocks exist. When the need for a second data block arises, AAM creates a primary (or level 0) index block. A key entry record is stored in the index block as each new data block is created. When the index block cannot hold any more records, a second (or level 1) index block is created for data block key entries. At the same time, a new primary index block is created to hold key entries referencing the level 1 index blocks. Additional level 1 index blocks are formed until the primary index block is filled and another level of indexing is needed.

The time required to randomly retrieve a record from the file is related directly to the number of index levels. Each index block accessed and searched adds to retrieval time. As long as the number of index levels remains constant, the random record retrieval time remains the same.

Figure 3-3 illustrates the hierarchy of index blocks and data blocks in an indexed sequential file with two levels of index blocks. The level 0 (primary) index block contains two key entry records that point to the two level 1 (secondary) index blocks. Each level 1 index block contains three key entries that point to three data blocks.

SPECIFYING FILE STRUCTURE

The structure of an indexed sequential file is defined when the file is created. This structure cannot be changed for the life of the file. File structure is defined in terms of:

- Data and index block size
- Record size
- Primary key type, size, and location
- Collating sequence if the file has symbolic keys

RECORD DEFINITION

You must establish the type and size of records before the file is first opened on a creation run. The record type (RT) field in the FIT is either set to one of the six record types or the default of W type records is accepted; however, AAM processes the records as U type records. Other FIT fields are required for record definition depending on the record type selected. Record size specification is also dependent on record type.

- D type records

LP	Length field beginning character position
LL	Length field length
MNR	Minimum record length
MRL	Maximum record length

- F type records

FL	Fixed length

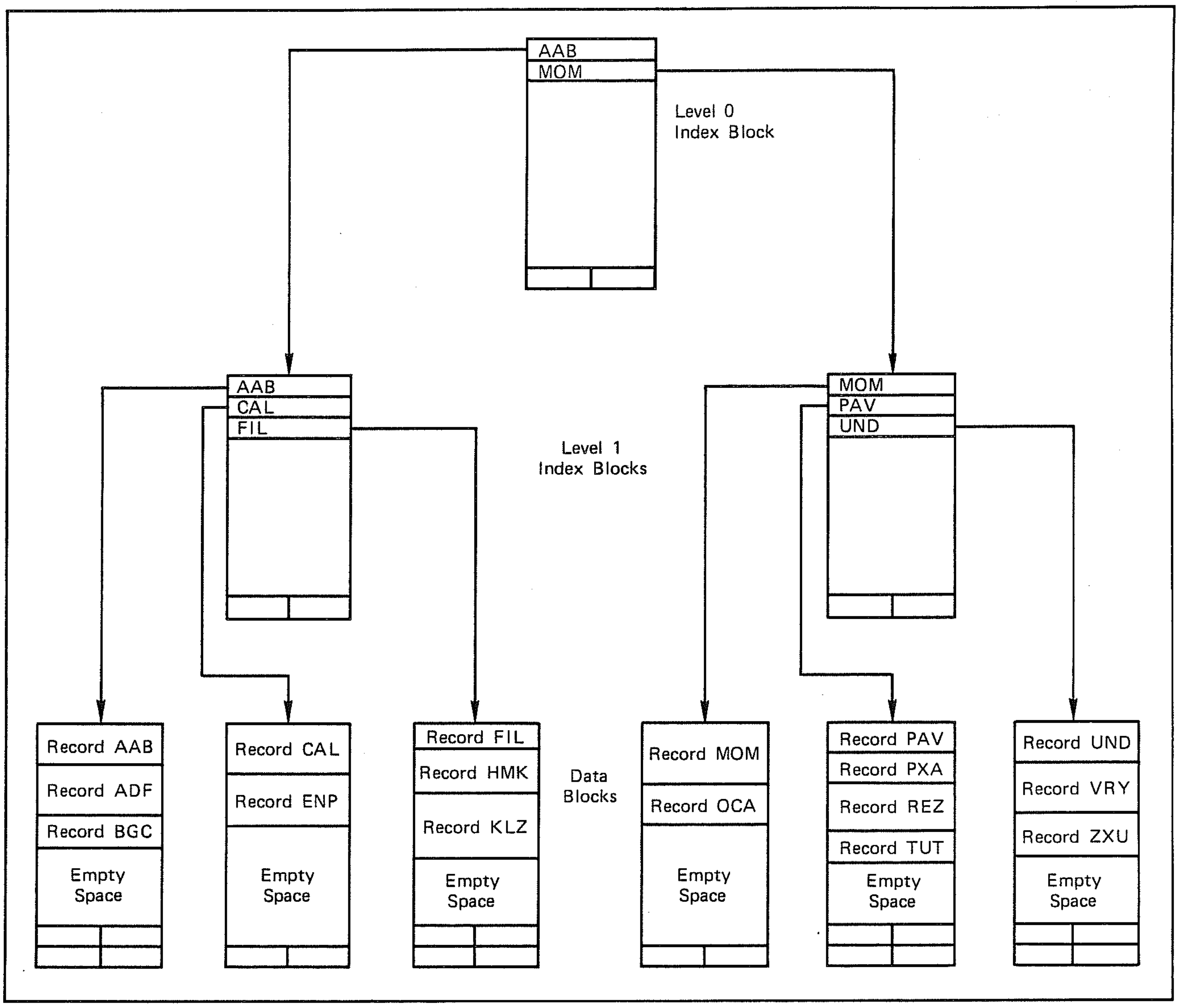


Figure 3-3. Indexed Sequential File Concept

- R type records

RMK	Record mark character
MNR	Minimum record length
MRL	Maximum record length

- T type records

HL	Header length
TL	Trailer length
CP	Trailer count beginning character position
CL	Count field length
MNR	Minimum record length
MRL	Maximum record length

- U type records (includes W and S type records)

RL	Record length
MNR	Minimum record length
MRL	Maximum record length

- Z type records

FL	Full length

The MNR and MRL fields specify the minimum and maximum number of characters in any record in the file. The value of the MNR field cannot be zero and must not be greater than the value of the MRL field. If the primary key is embedded in the record, the minimum record length must include the full primary key. For the life of the file, any record that is larger or smaller is rejected as a trivial error.

PRIMARY KEY DEFINITION

Each record in an indexed sequential file has a primary key associated with it. The value of this key determines the position of the record within the file. The primary key can be embedded or nonembedded. Embedded keys require less space and are, therefore, more efficient.

Two FIT fields are required to define the primary key:

KT	Key type; symbolic, integer, or uncollated; default is symbolic.
KL	Key length; number of characters in the primary key.

The key type and length determine the size of the key entry records in the index blocks. A key entry record consists of a primary key value followed by the relative PRU number of the next level index block or the data block containing the record.

If the primary key is embedded in the record, the following FIT fields are also required:

EMK	Embedded key (set to YES).
RKW	Relative key word; word within the record in which the primary key begins, counting from 0; default is 0.
RKP	Relative key position; character position within the relative key word in which the primary key begins, counting from 0; default is 0.

If the primary key is not embedded in the record, the following FIT fields are required in addition to the KT and KL fields:

KA	Key address; location of the word containing the primary key.
KP	Beginning key position; for a symbolic key, starting character position in the word indicated by the KA field; default is 0.

Symbolic Keys

A symbolic key is a string of alphanumeric characters; maximum key length is 255 characters. To describe a symbolic key, the KT field is set to S and the KL field is set to the number of characters in the key.

Records with symbolic keys are in order according to the sequence indicated by the display code to collating sequence conversion table, which is stored in the file statistics table (FSTT). If you supply a collating sequence table when the file is created, that table is used; otherwise, a default table is used.

Each character is stored internally in its 6-bit display code equivalent, 10 characters per word. The number of characters in the key determines the size of the key entry records in the index blocks. The key length plus four characters for the PRU number, rounded up to the next full word, is the key entry record length.

Integer Keys

Integer keys consist of 60-bit signed binary values. Floating point items can be defined as integer keys. To describe an integer key, the KT field is set to I and the KL field is set to 10. A key entry record in an index block requires two words for an integer key.

When integer keys are defined for a file, records are stored in order of the magnitude of the key values. For example, a record with the key value 50 precedes a record with the key value 55.

Uncollated Symbolic Keys

An uncollated symbolic key is the same as a symbolic key except that characters are ranked according to their binary display code values (0 through 63). To describe an uncollated symbolic key, the KT field is set to U and the KL field is set to the number of characters in the key.

COLLATING SEQUENCE DEFINITION

When symbolic keys are specified, you can either use the default collating sequence or you can provide a collating sequence. Only one collating sequence can exist during the life of the file; it must be defined before opening the file on the creation run. The collating sequence is stored in the FSTT and becomes a permanent part of the file. It need not be referenced again by the user.

To define a collating sequence, you must construct a table for converting display code values to the desired collating sequence. AAM generates the reverse conversion table from the user-supplied table. The display code to collating sequence conversion table (DCT) field in the FIT must be set to the name of the array that holds the table.

The conversion table is an eight-word table that establishes the relationship between 6-bit display codes and collating sequence ranking. Each display code equivalent of a character is assigned a collating sequence value. The eight words, numbered 0 through 7, are divided into ten fields, numbered 0 through 9 from the left. Fields 8 and 9 are not used and should be set to 55_8, representing a blank in display code. The display code value for a character is represented by a combination of the word number and field number:

- The first digit of a display code value indicates the table word number.
- The second digit of a display code value indicates the field number within that word.

The collating sequence value for a character is entered at the intersection of the word and field representing the display code value for the character. For example, the letter S has a display code value of 23_8. The collating sequence value for the letter S is entered in word 2, field 3.

A user-supplied conversion table is not checked for duplicates or inconsistencies; however, duplication and inconsistency should be avoided. Any inadvertent duplication is accepted and deliberate duplication is allowed. Fields 0 through 7 must be defined for all eight words of the table.

Figure 3-4 shows an array set up for the display code to collating sequence conversion table. The DCT field must be set to DCTABLE before the file is opened. This conversion table ranks the numbers 0 through 9 (beginning in word 3, field 3) before the letters A through Z (beginning in word 0, field 1). Fields 8 and 9 for all words are also set to blanks.

```
DIMENSION ISFIT (35)
DIMENSION DCTABLE (8)
DATA DCTABLE / 77 13 14 15 16 17 20 21 55 55 B,
               22 23 24 25 26 27 30 31 55 55 B,
               32 33 34 35 36 37 40 41 55 55 B,
               42 43 44 01 02 03 04 05 55 55 B,
               06 07 10 11 12 63 66 65 55 55 B,
               67 71 61 64 72 00 70 60 55 55 B,
               51 47 76 46 73 50 75 52 55 55 B,
               53 54 74 55 45 56 57 62 55 55 B /
CALL FILEIS (ISFIT,'DCT',DCTABLE, . . .)
```

Figure 3-4. User-Supplied Conversion Table

DATA BLOCK DEFINITION

AAM uses the values in various FIT fields when data blocks are created. Data block definition is specified in one of two ways:

- Specify the block length (MBL) directly
- Accept the default block size calculated by AAM

You specify the data block size directly by defining the maximum block length (MBL). This is the preferred method. MBL is the maximum number of user characters in the data block. The block size specified by the MBL field must be large enough to hold at least one maximum-length record plus enough words to hold the primary key (if not embedded). The maximum record length (MRL) must also be defined.

You should use the FLBLOK utility to help select the appropriate value for the MBL field. FLBLOK is described later in this section.

AAM increases the specified MBL to use mass storage efficiently. The resulting physical data block size will be:

[(Specified MBL + 50 characters) rounded to the next PRU multiple]- 20 characters

Therefore, you should specify an MBL value 50 characters less than the PRU multiple desired.

Alternatively, you can accept the default block size. If an installation default has not been defined, AAM will calculate the default as follows:

1. The mean of MRL and MNR values are used as the average record length. If MNR is not defined, it is assumed to be zero.

2. If the RB field is not specified, it is set to 2.

3. The FLM value is used as the estimated maximum number of records in the file. If FLM is not specified, 100000 is used.

4. The NL value is used as the maximum number of index levels. If the NL field is not defined, 2 is used.

5. Increasing from 8 PRUs, AAM looks for the smallest block size that will:

   Contain RB average records,

   and

   Contain enough index records that when the file has grown to the estimated maximum number of data records (FLM), the number of index levels (NL) will be within the maximum.

Typically 8 PRUs is generated as the default block size. A maximum number of 128 PRUs are allowed in one block.

If a file is usually processed sequentially, large data blocks are most efficient because the next record to be accessed will usually be in the block already in the buffer and transfer of another block into central memory is not required. If a file is usually processed randomly, small data blocks are most efficient because the next record to be accessed usually will not be in the block currently in the buffer in central memory. Therefore, a disk access will almost always be necessary, and time will be saved if the access involves reading a short block rather than a long block.

PADDING

Padding is the unused area in a data block at file creation time. As a data block is created (assuming sequentially ordered records), data records and record pointers are stored in the block until the specified percentage of the block remains empty; then a new block is started. The first time the file is closed, this reserved space ceases to be respected and the space is used as needed.

Later, you can add records to the file without necessitating additional data blocks. If the file is not expected to grow or to grow only at the end, a padding percentage of zero (the default) can be used. Data block padding should be used with care because the padding is allocated in every data block in the data file.

You can specify data block padding directly by defining the following field:

DP	Data block padding. Percentage of data block reserved for padding. Default of zero percent can be used. Values 0 to 99 are valid.

Index blocks are the same size as data blocks. Therefore, index block size is automatically defined when data block size is defined. The percentage of padding, however, need not be the same as data block padding. A small percentage of index block padding is usually recommended. If index block padding is specified at file creation time, index entries can be added without necessitating additional index levels.

In addition to the FIT fields used to create data blocks, AAM uses the following field to create index blocks:

IP	Index block padding. Percentage of index block reserved for padding. Default of zero percent can be used. Values 0 to 99 are valid.

FLBLOK UTILITY

Block size has major effects on both the physical structure and the performance characteristics of the indexed sequential file. The FLBLOK utility assists you in selecting the appropriate value for the maximum block length (MBL) field. The utility computes various file characteristics for different block sizes based on specified file characteristics.

The FLBLOK control statement provides the file description that is used by the utility. The description is specified by the following parameters:

NR	Number of records; total number of records the file is expected to contain; default is 100000.
KL	Key length; number of characters in the primary key; default is 10.
RL	Record length; average number of characters in a record; default is 80.
IP	Index block padding; percentage of index block reserved for padding; default is 0.
DP	Data block padding; percentage of data block reserved for padding; default is 0.
NL	Number of index levels; the highest number of index levels for the file.
MRL	Maximum record length; maximum number of characters in a record; default is RL value.

Subsequent calculations are based on these parameters. For best results, all parameters should be specified in the FLBLOK control statement.

Results of the FLBLOK utility depend heavily on the value of the RL parameter. When the file contains variable-length records, the value for the RL parameter should be determined as follows:

- If most records in the file are of a specific length, the RL parameter should be set to that length.
- If record lengths are well distributed, the RL parameter should be set to the median average. That is, half the records are larger and half are smaller than the RL value.
- If the EMK field is set to NO, AAM appends the key to the record. Therefore, the key length (KL) value must be added to the RL value.

Output from the FLBLOK utility is a listing of file characteristics that are calculated for different values in the MBL and NL fields. The MBL field is increased (stepping by PRU) until the file can be built in the number of index levels indicated by the NL parameter. File characteristics are calculated and listed as output. The utility continues increasing MBL and calculating file characteristics for each lower number of index levels.

The output listing indicates the minimum MBL value for each index level. In batch mode, the listing also indicates the MBL value when disk usage decreases within each index level. This can be important when disk usage is critical.

Figure 3-5 illustrates a job structure for the FLBLOK utility. The file is to contain 100000 records with a primary key length of 20 characters; maximum record length is 1000 characters and average record length is 750 characters. The first control statement specifies no padding for both index and data blocks with one index level. The second control statement specifies five percent padding for both index and data blocks with two index levels. Output generated by these two control statements is shown in figure 3-6.

```
Job statement
USER statement
CHARGE statement
FLBLOK (OUTPUT,NR=100000,KL=20,RL=750,IP=0,
  DP=0,NL=1,MRL=1000)
FLBLOK (OUTPUT,NR=100000,KL=20,RL=750,IP=5,
  DP=5,NL=2,MRL=1000)
- - EOI
```

Figure 3-5. Sample Deck Structure for FLBLOK Utility - NOS

CREATING AN INDEXED SEQUENTIAL FILE

You can create an indexed sequential file through a source program or through the FORM utility. The file statistics table (FSTT) is created on the file creation run and becomes a permanent part of the file.

You must set the old/new file (ON) field to NEW for the file creation run. Other FIT fields that must be defined and cannot be subsequently changed are as follows:

FO	File organization; set to IS by the CALL FILEIS statement.
ORG	Old/new (initial or extended) file organization; must be set to NEW.
KT	Key type; default is symbolic keys.
KL	Key length; number of characters for symbolic keys, 10 for integer keys.
MRL	Maximum record length; maximum number of characters in any record; set by AAM when FL is specified for F or Z type records.
RT	Record type; default is W, which AAM processes as U; fields required by the record type must also be specified.

One additional FIT field that you must define for the file creation run, but you can change on subsequent runs, is as follows:

LFN	Logical file name; one to seven characters, beginning with a letter.

If your program will define alternate keys during file creation, you must specify the following FIT field:

XN	Index file name.

```
/flblok(output,nr=100000,kl=20,rl=750,ip=0,dp=0,nl=1,mrl=1000)

                INDEXED SEQUENTIAL FILE PARAMETER ESTIMATE
          CONTROL CARD : FLBLOK(OUTPUT,NR=100000,KL=20,RL=750,IP=
                         0,NL=1,MRL=1000)

 MAXIMUM RECORD LENGTH(MRL) = 1000   NUMBER OF INDEX LEVELS(NL) =  1
 AVERAGE RECORD LENGTH(RL)  =  750   DATA BLOCK PADDING(DP)     =  0
 KEY LENGTH(CHARACTERS)(KL) =   20   INDEX BLOCK PADDING(IP)    =  0
                TOTAL NUMBER OF RECORDS(NR) =    100000

        BLOCK     844-21 1/2  ACCESSES  NON-POOLED              MAXIMUM
  NO.   LENGTH    TRK. DISK   PER       BUFFER       MIN.       FILE
  NDX.  (MBL)     GET EST.    GET       SIZE(WDS)    BUFFER     CAPACITY
  LVLS  (CHARS/   (SEQ./      (SEQ./    (SEQ./       SIZE       IN
  (NL)  PRUS)     RANDOM)     RANDOM)   RANDOM)      (WORDS)    RECORDS

    1  47950/75    2.0/126    .016/1    14555/14555    9755      100674
      .038 CP SECONDS EXECUTION TIME.

/flblok(output,nr=100000,kl=20,rl=750,ip=5,dp=5,nl=2,mrl=1000)

                INDEXED SEQUENTIAL FILE PARAMETER ESTIMATE
          CONTROL CARD : FLBLOK(OUTPUT,NR=100000,KL=20,RL=750,IP=
                         5,DP=5,NL=2,MRL=1000)

 MAXIMUM RECORD LENGTH(MRL) = 1000   NUMBER OF INDEX LEVELS(NL) =  2
 AVERAGE RECORD LENGTH(RL)  =  750   DATA BLOCK PADDING(DP)     =  5
 KEY LENGTH(CHARACTERS)(KL) =   20   INDEX BLOCK PADDING(IP)    =  5
                TOTAL NUMBER OF RECORDS(NR) =    100000

        BLOCK     844-21 1/2  ACCESSES  NON-POOLED              MAXIMUM
  NO.   LENGTH    TRK. DISK   PER       BUFFER       MIN.       FILE
  NDX.  (MBL)     GET EST.    GET       SIZE(WDS)    BUFFER     CAPACITY
  LVLS  (CHARS/   (SEQ./      (SEQ./    (SEQ./       SIZE       IN
  (NL)  PRUS)     RANDOM)     RANDOM)   RANDOM)      (WORDS)    RECORDS

    2   5070/8     6.5/78     .167/2     1691/2203     1179      171366
    1  50510/79    2.1/131    .016/1    15323/15323   10267      111078
      .045 CP SECONDS EXECUTION TIME.
/
```

Figure 3-6. FLBLOK Utility Output

If you do not specify the following FIT fields on the creation run, the default values are effective for the life of the file:

IP	Index block padding; default is zero percent padding.
MBL	Maximum block length for data and index blocks; default is number of characters required for two average size records.
DP	Data block padding; default is zero percent padding.
DCT	Display code to collating sequence conversion table for collated symbolic keys; default is standard CDC collating sequence (refer to appendix A).
BCK	Block checksum; default is no checksums.
EMK	Embedded key; set to YES if the primary key is embedded within the record; default is NO. (YES is more efficient.)

If primary keys are embedded, two additional FIT fields are required:

RKW	Relative key word in which the key begins; default is word 0.
RKP	Relative key position within the word in which the key begins; default is character position 0.

Other optional FIT fields that you can define on the creation run and you can change on a subsequent run are as follows:

ERL	Trivial error limit; default is no limit.
FLM	File limit; default is no limit imposed by AAM.
DFC	Dayfile control; default is only fatal error messages to the dayfile.
EFC	Error file control; default is no messages to the AAM error file.
FWI	Forced write indicator; default is buffers are written only when space is needed.

FWB	First word address of the buffer; default is the buffer provided by AAM.
BFS	Buffer size; default is buffer size calculated by AAM.
CPA	Compression routine address; default is no compression of records.
DCA	Decompression routine address; default depends on the CPA field (refer to appendix F).
DX	End-of-data exit; default is no exit subroutine.
EX	Error exit; default is no exit subroutine.

FILE CREATION BY A SOURCE PROGRAM

When an indexed sequential file is created through a source program, file structure and key characteristics must be defined by setting applicable fields in the FIT before the file is opened. FIT fields can be specified in the FILE control statement, the CALL FILEIS statement, and the CALL STOREF statement.

The old/new file (ON) field must be set to NEW for a file creation run. This is accomplished by setting the ON field with one of the FIT manipulation statements or by specifying NEW for the processing direction (pd) parameter in the open request. Setting the pd parameter to NEW sets the ON field to NEW and the PD field to OUTPUT. Records can then be inserted into the file with write requests.

After all records have been written, the file should be closed. On a file creation run, the only file processing requests that can be issued are those that establish the FIT, open and close the file, write records, and read and write fields in the FIT.

Establishing the FIT

The first statement referencing the indexed sequential file must be the CALL FILEIS statement. When this statement is executed, the FIT is constructed and the specified values are stored in the FIT. The first parameter in the CALL FILEIS statement is the name of the 35-word array to hold the FIT. The same FIT array name is the first parameter in every statement accessing the indexed sequential file. Refer to section 2 for a more detailed explanation of the FIT and the CALL FILEIS statement.

Opening the File

You must open the indexed sequential file by executing a CALL OPENM statement before any records can be written to the file. The format of this statement is shown in the AAM reference manual.

The following statement opens the file for output:

```
CALL OPENM (ISFIT, 'NEW')
```

This statement performs open processing for the indexed sequential file whose FIT is stored in the array named ISFIT. The second parameter must specify NEW to set the processing direction (PD) field in the FIT to OUTPUT and the old/new file (ON) field to NEW. If the PD and ON fields are set before the file is opened, the second parameter in the open request can be omitted.

Open processing includes storing FILE control statement values in the FIT, processing buffer parameters, supplying default values for FIT fields not set by the user, and checking the FIT for logical consistency and required fields.

Writing Records

Records are written to an indexed sequential file by executing a CALL PUT statement. The format of this statement is shown in appendix B.

The following statement writes a record to the file:

```
CALL PUT (ISFIT)
```

The FIT for the indexed sequential file is in the array named ISFIT.

Fields not specified in the write request default to the current values in the FIT. After the record is written, AAM sets the RL field to the number of characters in the record.

Records should be presented to AAM with primary key values in ascending order. This order results in data and index blocks that preserve the padding specified by the DP and IP fields in the FIT and also shortens the time required for file creation. If records are not written in order by primary key values, the result is inefficient file structure with more data blocks than necessary because of the numerous data block splits. More overhead time is also required.

The record to be written on the file must be established in the working storage area location. If the primary key is not embedded, the location of the key value for the record to be written must also be established. For embedded primary keys, the key location is determined by the RKW and RKP fields.

Closing the File

The last program statement referencing the file must be a CALL CLOSEM statement to ensure file integrity. The format of this statement is shown in appendix B.

The following statement initiates close processing:

```
CALL CLOSEM (ISFIT, 'U')
```

ISFIT is the name of the array containing the FIT. The file is rewound, the OC flag is cleared, and the file is returned to the permanent file manager. If the file is subsequently reopened, FIT verification and FILE control statement processing are repeated.

The second parameter sets a value for the close flag (CF) field. The following values can be specified for the CF fields:

R (rewind)

> The file is rewound to beginning-of-information; this is the default setting. The open/close flag (OC) field is set to closed.

N (no rewind)

> The file is not rewound; it remains at the current position, even when reopened. The OC field is set to closed.

U (unload) or RET (return)

The file is rewound and the OC field is cleared. A permanent file is detached from the job and returned to the permanent file manager. Scratch mass storage space assigned to a nonpermanent file is released.

DET (detach)

The file is not rewound and the OC field is cleared.

When the close request is executed, any data or index blocks in the central memory buffer are written to the mass storage file. The FSTT, which is used to maintain continuity over the life of the file, is also written to the file. File statistics are written to the error file, if requested.

A close request issued for a file that has never been opened or that has been closed but neither unloaded nor reopened results in a trivial error. The file is positioned as specified before the error is issued.

Sample Creation Program

Program NEWIS, shown in figure 3-7, creates an indexed sequential file through direct calls to AAM. The program reads an input file from the file CORP and writes records to the new indexed sequential file ISFILE. Each input record is printed on the file OUTPUT to show the records used to create the file.

Program statements that are related to creation of the indexed sequential file are defined as follows:

- DIMENSION ISFIT(35), ISWSA(5)

  This statement allocates a 35-word array named ISFIT for construction of the FIT and a 5-word array named ISWSA for the working storage area.

- CALL FILEIS (ISFIT, 'LFN', 'ISFILE',...)

  This statement sets fields in the FIT to describe the structure of the indexed sequential file. Required parameters in the statement are:

  FIT array (ISFIT)

  Logical file name (ISFILE)

  Extended file organization (NEW)

  Record type (fixed length)

  Fixed record length (50 characters)

  Primary key type (uncollated symbolic)

  Primary key length (4 characters)

  Embedded key and location (EMK, RKW, and RKP)

  The statement also defines the following:

  Maximum block length (590 characters)

  Index block padding (10 percent)

  Data block padding (5 percent)

  Error file control (3, errors and notes)

- CALL OPENM (ISFIT, 'NEW')

  This statement opens the file, sets the PD field to OUTPUT, and sets the ON field to NEW for a creation run. Records can only be written to the file.

- CALL PUT (ISFIT)

  This statement writes the record in the working storage area ISWSA to the indexed sequential file. The primary key location is not specified because the key is embedded in the record. AAM uses the RKW and RKP fields to locate the primary key.

- CALL CLOSEM (ISFIT)

  This statement initiates close processing, which includes writing the FSTT and any blocks in the buffer to the file.

FILE CREATION THROUGH FORM

You can use the FORM utility to create an indexed sequential file. FORM can also be used to restructure an existing indexed sequential file or to dump an existing file to tape for backup or storage purposes. Refer to the FORM reference manual for more detailed information on the FORM utility.

FORM uses CYBER Record Manager routines to perform input on the input file and output on the indexed sequential file being created. FILE control statements are used to provide descriptions of the input and output files. Parameters in the FILE control statement for the input file depend on the file organization. The FILE control statement describing the new indexed sequential file must specify the logical file name and the following FIT fields:

FO	File organization; must be FO=IS for indexed sequential file organization.
ORG	Old/new (initial or extended) file organization; must be ORG=NEW to specify extended file organization.
KT	Key type; can be S (symbolic), U (uncollated symbolic), or I (integer).
KL	Key length; number of characters for symbolic keys, 10 for integer keys.
MRL	Maximum record length; maximum number of characters in any record; set by AAM when FL is specified for F or Z type records.
MNR	Minimum record length; minimum number of characters in any record; set by AAM when FL is specified for F or Z type records.
RT	Record type; default assigned by FORM is W, which AAM processes as U; fields required by the record type must also be specified.

A. NOS Operating System

```
Job statement
USER statement
CHARGE statement
FTN5.
DEFINE(ISFILE/CT=PU,M=W)
ATTACH(CORP)
LGO.
CRMEP(LO,RU)
```

NOS/BE Operating System

```
Job statement
ACCOUNT statement
FTN5.
REQUEST(ISFILE,PF)
ATTACH(CORP,ID=AAMUG)
LGO.
CATALOG(ISFILE,ID=AAMUG)
CRMEP(LO,RU)
```

B. Input File: CORP

```
B695  ABC DISTRIBUTORS        412568   7880621
E482  CORP SALES INC           89207   1145833
H314  DAY AND NIGHT SALES      52573   2066122
K260  FRIENDLY SALES         1027866   6347410
M857  OAKVILLE CORP           149287   3014192
R500  RETAILERS INC            76854   1255868
T258  SELECTIVE SALES CO      248916   4337092
T289  SMITH AND SON            95625   1528863
V440  WORLD SALES CO          914819   6844272
X179  YOUNG BROTHERS          527638   2428535
```

C. Source Program

```
      PROGRAM NEWIS
C
C  ****************************************************************
C  * THIS PROGRAM CREATES AN IS FILE (ISFILE) FROM A SEQUENTIAL
C  * FILE (CORP).  EACH RECORD IS ALSO PRINTED AS OUTPUT.         *
C  ****************************************************************
C
      IMPLICIT INTEGER (A-Z)
      DIMENSION ISFIT(35), ISWSA(5)
      CALL FILEIS (ISFIT, 'LFN', 'ISFILE', 'ORG', 'NEW',
     +            'WSA', ISWSA,
     +            'RT', 'F', 'FL', 50,
     +            'MBL', 590, 'IP', 10, 'DP', 5,
     +            'KT', 'U', 'KL', 4, 'EMK', 'YES', 'RKW', 0, 'RKP', 4,
     +            'EFC', 3, 'DFC', 3)
      OPEN (2, FILE = 'CORP')
      CALL OPENM (ISFIT, 'NEW')
      IF (IFETCH (ISFIT, 'ES') .NE. 0) GO TO 50
   10 READ (2, '(3A10, 2I10)', END = 30) ISWSA
      CALL PUT (ISFIT)
      IF (IFETCH (ISFIT, 'ES') .NE. 0) GO TO 50
      PRINT 100, ISWSA
      GO TO 10
   30 CALL CLOSEM (ISFIT)
      STOP
   50 PRINT 902, IFETCH (ISFIT, 'ES')
      CALL CLOSEM (ISFIT)
      STOP 'CRM ERROR RETURNED'
  100 FORMAT (3A10, 2I10)
  902 FORMAT ('ES = ', O3)
      END
```

D. Output

(same as input)

Figure 3-7. Indexed Sequential File Creation

If the primary key is embedded in the output record, the following FIT fields are also required:

EMK	Embedded key; use EMK=YES.
RKW	Relative key word in which the key begins; default is word 0.
RKP	Relative key position within the word in which the key begins; default is character position 0.

Unless default values are to be accepted for data and index blocks, the following FIT fields must be specified in the FILE control statement:

MBL	Maximum block length for data and index blocks; default assigned by FORM is 640 characters.
DP	Data block padding; default is zero percent padding.
IP	Index block padding; default is zero percent padding.

When you use FORM, block size is either the length specified for the MBL field or the default assigned by FORM. Specifying the records per block (RB) field is not sufficient for block size calculation.

Other optional FIT fields can be set by the FILE control statement or be set to the default values. The FORM directive OUT can specify values for the following FIT fields:

FLM	File limit; maximum number of records for the indexed sequential file; default is no limit.
CPA	Compression routine address; number or entry point name of the compression routine; default is no compression of records.
EX	Error exit; entry point name of the user-supplied error processing routine; default is no error exit subroutine.

If the input file is a sequential or initial direct access file, each record must contain the primary key. Primary keys of all records must be the same type and must be in the same location within the records in the input file. An existing sequential file with records that do not contain a key can have an integer key added through the FORM directive SEQ. If the input file is an existing indexed sequential, actual key, or extended direct access file, the primary keys need not be within the records.

A job using FORM to create an indexed sequential file must contain the following:

- A FILE control statement describing the input file structure, unless the file is a sequential file with default file structure characteristics.
- A FILE control statement describing the new indexed sequential file structure.
- A FORM control statement.
- FORM directives.

The FORM directives INP and OUT identify the input and output files, respectively. Other FORM directives are available to specify record selection criteria, reformatting, and conversion. Consult the FORM reference manual for details of optional directives that might be useful.

The INP directive specifies the source of input records for the FORM run. The logical file name of the input file is the only required parameter. Optional parameters that can be included in the directive specify the maximum number of records to be processed, rewind action at end of run, and owncode routines for various options. The choice of optional parameters depends on the file structure.

The OUT directive defines an output file to be generated by FORM. The logical file name of the output file is the only required parameter. Optional parameters in this directive include the KEY parameter, which describes the location of the primary key and specifies whether or not the key is to be embedded in the record. The KEY parameter is required under the following conditions:

- The indexed sequential file being created has nonembedded primary keys.
- The input file is an indexed sequential, actual key, or direct access file and the primary key for the new indexed sequential file is not the primary key for the input file.
- The input file is a sequential file.

The format of the KEY parameter is:

KEY=±iTm

+	The primary key is embedded in the output record.
-	The primary key is extracted from the input record and not embedded in the output record.
i	Character position in which the primary key begins, counting by the FORM convention where the first character position is 1.
T	Key type in FORM terminology; must be I for an integer key or X for a symbolic key.
m	Number of 6-bit display code characters in a symbolic key; omitted for an integer key.

Figure 3-8 shows the job structure for creating the file ISFILE through the FORM utility. If only the INP and OUT directives with file names are required, they can be specified in the FORM control statement.

```
NOS Operating System

Job statement
USER statement
CHARGE statement
DEFINE(ISFILE/CT=PU,M=W)
FILE(ISFILE,FO=IS,ORG=NEW,KT=U,KL=4,RT=F,FL=50,
  MBL=590)
FILE(ISFILE,EMK=YES,RKW=0,RKP=4,DP=5,IP=10,NL=3)
FORM(INP=INPUT,OUT=ISFILE)
- - EOR
Input data
- - EOI

NOS/BE Operating System

Job statement
ACCOUNT statement
REQUEST(ISFILE,PF)
FILE(ISFILE,FO=IS,ORG=NEW,KT=U,KL=4,RT=F,FL=50,
  MBL=590)
FILE(ISFILE,EMK=YES,RKW=0,RKP=4,DP=5,IP=10,NL=3)
FORM(INP=INPUT,OUT=ISFILE)
CATALOG(ISFILE,ID=AAMUG)
*EOR
Input data
*EOI
```

Figure 3-8. Indexed Sequential File Creation, FORM Utility

PROCESSING AN EXISTING INDEXED SEQUENTIAL FILE

After the file creation run, an indexed sequential file can be read randomly or sequentially, new records can be inserted into the file, and existing records can be deleted or be replaced by other records with the same primary key values. File processing is governed by many of the FIT fields set on the file creation run. You need not set the following FIT fields before opening an existing file:

MBL	Maximum block length for data and index blocks
NL	Number of index levels
MNR	Minimum record length
MRL	Maximum record length
KT	Key type
KL	Key length
RKW	Relative key word for embedded keys
RKP	Relative key position within RKW for embedded keys
DCT	Display code to collating sequence conversion table

The values for these fields are preserved in the FSTT and are returned to the FIT when the file is opened. Any attempt to change block or record size is ignored. An attempt to change any primary key specification prevents further file access.

You must set the following FIT fields the same as on the file creation run:

RT	Record type; all applicable fields must also be set
ORG	Old/new (initial or extended) file organization

The logical file name (LFN) field must be set to the current logical file name for the file. This name need not be the same as the name used on the file creation run.

If alternate keys have been defined, the index file name must be specified with the XN parameter.

If you do not set the following FIT fields before the file is opened, the default values must be accepted:

FWB	First word address of the buffer; default is buffer location provided by AAM.
BFS	Buffer size; default is buffer size calculated by AAM.
CPA	Compression routine address; default is no compression of records.
DCA	Decompression routine address; default depends on the CPA field.

Optional FIT fields that can be set at any time before being required by a file processing statement are as follows:

DFC	Dayfile control
EFC	Error file control
ERL	Trivial error limit
EX	Error exit
DX	End-of-data exit
FLM	File limit
FWI	Forced write indicator
MKL	Major key length
KA	Key address
KP	Key position

The KA and KP fields are required for nonembedded keys and for random access or deletion of records with embedded keys.

Various FIT fields can be set by the file processing statements. The default value listed for a field used by one of these statements is applicable only if the field has not been set by any other statement. The current value in a FIT field is always used by a file processing statement.

ESTABLISHING THE FIT

The FIT is established for an existing file in the same manner as for a new file during the creation run. The CALL FILEIS statement is the first statement that can reference the indexed sequential file. It specifies the name of the array that is to contain the FIT and the logical file name. Fields defining file structure are not specified, however, because file structure information is saved in the FSTT and is returned to the FIT when the file is opened. Other fields that need to be set for program execution can be specified in the CALL FILEIS statement. The FILE control statement can also be used to set FIT fields or to override values set by the CALL FILEIS statement.

OPENING THE FILE

Before any data records in an existing file can be accessed, you must open the file by executing a CALL OPENM statement. The format of this statement is shown in appendix B.

The following example is a typical open request:

```
     CALL OPENM (ISFIT, 'I-O')
```

The file is opened and positioned to beginning-of-information. Records can be read, written, rewritten or deleted.

The setting of the PD field (second parameter in the call) determines the input/output statements that can be executed. The PD field can be set as follows:

INPUT	Only statements that read or position the file can be executed; this is the default setting.
OUTPUT	Only statements that write new records to the file can be executed.
I-O	Any file processing statement related to input/output can be executed. (If the PD field is set by any other statement, the two-character mnemonic IO must be specified.)

Values from the FSTT are stored in applicable FIT fields during open processing. FILE control statement processing and FIT consistency checking are performed in the same manner as on a file creation run.

The first time you open an indexed sequential file after the creation run, the old/new file (ON) field in the FIT must be changed from NEW to OLD. This can be accomplished by the CALL FILEIS statement, the FILE control statement, or the CALL STOREF statement before the file is opened. If the ON field is not set by one of these statements, it is set to OLD by specifying INPUT, OUTPUT, or I-O for the processing direction parameter in the open request.

A trivial error occurs if you attempt execution of a file processing statement and the PD field is not set to an appropriate value for that statement. A fatal error occurs if the ON field is not set to OLD.

READING THE FILE RANDOMLY

Records in an indexed sequential file can be read randomly by primary key values by executing a CALL GET statement. For a random read, the file must be open for either input or input/output (the PD field is set to INPUT or IO). The format of the CALL GET statement is shown in appendix B.

The following statement reads a record randomly from the file associated with the FIT stored in the array ISFIT:

```
     CALL GET (ISFIT)
```

When this GET statement is executed, the record with the current primary key value indicated by the KA field is returned to the working storage area.

You must set the WSA and KA fields either through the read request or through the CALL FILEIS or CALL STOREF statement. The location indicated by the KA field is set to the primary key value for the record to be read. Execution of the read request then returns the record to the specified working storage area. If a record with the specified primary key value cannot be located, a trivial error occurs; the file is positioned at the point where the record should exist.

At the completion of the read request, the record length (RL) field is set to the number of characters returned to the working storage area. Any user value for the RL field is ignored for the read operation. The number of characters returned is determined by the length of the record written.

You can set the MKL field when a major key is to be used for the random read. A major key is defined to be a number of leading left-hand characters in the key location for a symbolic key. The length of the major key is specified by the MKL field; it must be at least one character but less than the full key length. The major key value for the random read must be set in the location indicated by the KA field and must be aligned so that it begins in the character position indicated by the KP field. For example, a full key with a length (KL) of 6 and starting position (KP) of 4 is aligned as follows:

0	1	2	3	4	5	6	7	8	9
				A	B	C	D	E	F

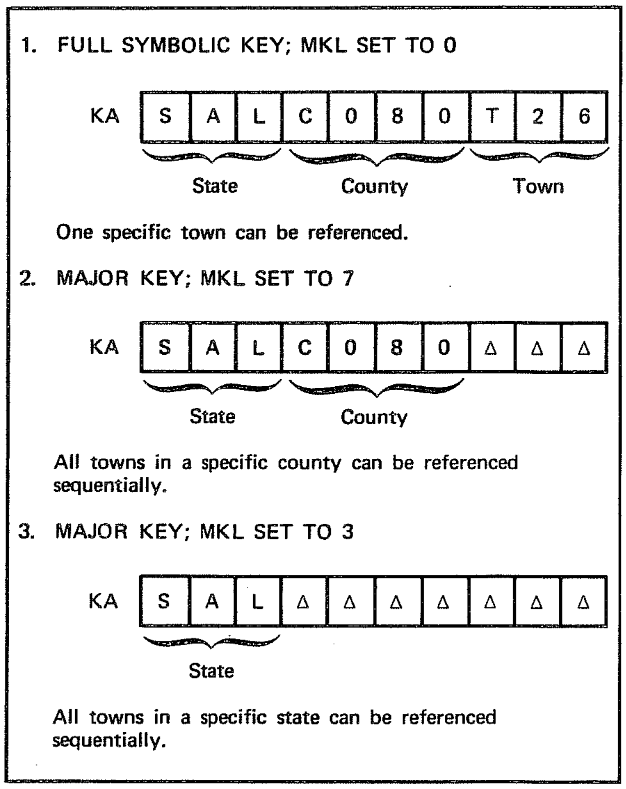
If major key length (MKL) is defined as 3, the value for the random read must be aligned as follows:

0	1	2	3	4	5	6	7	8	9
				A	B	C	Δ	Δ	Δ

The contents of the positions following the major key (7 through 9 in this example) are irrelevant. The MKL field is reset to zero after the read request is executed.

When a record is read randomly by major key, the first record that has a matching major key is returned to the working storage area. The full key of the first record with the specified major key is returned to the location indicated by the KA and KP fields (positions 4 through 9 in the previous example).

Figure 3-9 illustrates major key processing. The 10-character symbolic key consists of three codes: three characters (SAL through SWY) for the state, four characters (C001 through C100) for the county, and three characters (T01 through T99) for the town. The first example shows the primary key value at location KA for accessing town 26 (T26) of county 080 (C080) in Alabama (SAL). Example 2 shows the seven-character major key value at location KA for accessing the first town of county 080 in Alabama. Example 3 shows the three-character major key value at location KA for accessing the first town of the first county in Alabama.



One specific town can be referenced.

All towns in a specific county can be referenced sequentially.

All towns in a specific state can be referenced sequentially.

Figure 3-9. Major Key Processing

READING THE FILE SEQUENTIALLY

Sequential reading of records in an indexed sequential file retrieves the records in ascending primary key order. Records are read sequentially by executing a CALL GETN statement. The file must be open for either input or input/output (the PD field is set to INPUT or IO). The format of the CALL GETN statement is shown in appendix B.

The WSA field must be set before the sequential read request can be executed. When the statement is executed, the next record in primary key sequence is returned to the specified working storage area. If the KA field is set and the EMK field is set to NO, the primary key of the record retrieved is returned to the location indicated by the KA field. If the KA field is specified, its value (location) will remain fixed until changed by using it again. Therefore, if subsequent GETN calls do not specify KA, the primary key will be returned to the same location.

When a sequential read request is issued, the next record in primary key sequence from the current file position is returned to the working storage area. File position is changed by an open, rewind, read (random or sequential), start, or skip request. A replace or delete request does not alter the current file position. File positioning is discussed in more detail later in this section.

The following statement reads the next record in sequence:

```
CALL GETN (ISFIT)
```

This statement reads the next sequential record and moves it to the working storage area.

INSERTING NEW RECORDS

New records are written to an existing indexed sequential file in the same manner as when the file was originally created. The format of the write request is identical to the format for a file creation run. The file must be open for output or input/output (the PD field is set to OUTPUT or IO).

The parameters in the write request are the same as in the write request on a file creation run. (See the AAM reference manual.)

Execution of the following statement adds a new record to the file:

```
CALL PUT (ISFIT)
```

The record to be written is stored in the array defined by WSA. At the completion of the write request, AAM sets the RL field to the number of characters in the record written to the file.

When adding a group of contiguous records, execution is faster if the records are sorted in ascending primary key order and are then presented to AAM in sequential order.

DELETING EXISTING RECORDS

A record in an indexed sequential file can be eliminated from the file by executing a CALL DLTE statement. The file must be open for input/output (the PD field is set to IO) to delete a record. The format of the CALL DLTE statement is shown in appendix B.

Execution of the following statement deletes a record from the file:

```
CALL DLTE (ISFIT)
```

The primary key value for the record to be deleted must be established at the location indicated by the KA field. (The KA field is typically set by the CALL FILEIS statement.) A trivial error results if the specified key value does not match the primary key for any existing record.

The data record is deleted physically as well as logically. Any remaining records in the data block are relocated to eliminate the space previously occupied by the deleted record. When the first record in a data block is deleted, index blocks are updated as applicable.

When the file contains fixed-length records and a series of contiguous records are being deleted, significant saving in execution time results by beginning with the highest primary key value. Deletion of the last record in a data block is performed quickly by changing the block header field pointing to the next available empty word. Deletion of the first record in a data block, however, requires relocation of all remaining records in the data block. By deleting the record with the highest key value first, relocation of records to be deleted subsequently can be avoided.

The primary key for the record to be deleted can be established by a return from a sequential read request. For example, the following sequence of statements is acceptable:

```
CALL GET (ISFIT)
CALL SKIP (ISFIT, +3)
CALL GETN (ISFIT)
CALL DLTE (ISFIT)
```

ISFIT is the name of the FIT array. The GET statement reads a record randomly by primary key value. The SKIP statement then positions the file forward three records. The sequential read statement returns the primary key value for the record to the area indicated by KA. The DLTE statement deletes the record with that primary key value.

If a data block or index block contains no records as a result of the delete request, it is linked into a chain of deleted blocks. These blocks are then used when new blocks are required for file expansion.

REPLACING EXISTING RECORDS

The contents of any record in an indexed sequential file can be modified and the changed record rewritten to the file by executing a CALL REPLC statement. The file must be open for input/output (the PD field is set to IO). The format of the CALL REPLC statement is shown in appendix B.

The following statement replaces an existing record in the file with the record in the working storage area:

```
    CALL REPLC (ISFIT)
```

If the primary key value at location KA does not duplicate the primary key value for any existing record, a trivial error occurs and the replace request is ignored.

The new record can be smaller or larger than the existing record; however, it must be within the minimum and maximum record lengths established in the MNR and MRL fields of the FIT when the file was created.

CLOSING THE FILE

The last reference to the indexed sequential file should be a close request. This ensures that all updated records are written to the file and that the file statistics table contains current information. The format of the CALL CLOSEM statement is shown in appendix B.

Close processing is performed when the following statement is executed:

```
    CALL CLOSEM (ISFIT)
```

This statement initiates close processing for the file associated with the FIT stored in the array ISFIT. The CF field is not set and the current contents of the field determine file positioning.

If the error file control (EFC) field is set to 2 or 3, file statistics are written to the error file. If the dayfile control (DFC) field is set to 2 or 3, file statistics are written to the dayfile.

A close request for a file that has never been opened or that has been closed but neither unloaded nor reopened results in a trivial error. File position does not change.

SAMPLE UPDATING PROGRAM

Program ISUPDAT, shown in figure 3-10, accesses the existing indexed sequential file ISFILE through direct calls to AAM. Input records, which are read from the file UPDAT, contain a code number in the first field (CODE). The code number indicates whether to update (code number 1), add (code number 2), or delete (code number 3) a record. After all input records have been processed, the indexed sequential file is rewound to beginning-of-information and read sequentially.

Program statements related to processing the existing indexed sequential file are defined as follows:

- CALL FILEIS (ISFIT, 'LFN', 'ISFILE',...)

  This statement sets the FIT fields required for processing the existing file:

  FIT array (ISFIT)

  Logical file name (ISFILE)

  Extended file organization (NEW)

  Record type (must be the same as for the creation run)

- CALL OPENM (ISFIT, 'I-O')

  This statement opens the file for input/output processing. The PD field is set to IO and the ON field is set to OLD.

- CALL GET (ISFIT)

  This statement reads a record randomly and returns it to the working storage area ISWSA. The variable ISKEY contains the primary key for the record to be retrieved.

- CALL REPLC (ISFIT)

  This statement rewrites the updated record in the working storage area ISWSA to the file.

- CALL PUT (ISFIT)

  This statement adds the new record in the working storage area ISWSA to the file.

- CALL DLTE (ISFIT)

  This statement deletes a record from the file. The primary key for the record to be deleted is in the variable ISKEY beginning in character position 0.

- CALL GETN (ISFIT)

  This statement reads the next record in sequence and returns it to the working storage area ISWSA.

- IF (IFETCH (ISFIT, 'FP') .NE. 0"20") GO TO 80

  This statement checks the file position (FP) field in the FIT for a valid read. The FP field is set to 20_8 when a record is successfully read from the file.

- CALL CLOSEM (ISFIT)

  This statement writes the updated FSTT and any altered data and index blocks in the buffer to the file, closes the file, and disposes it according to the current value in the CF field.

A. NOS Operating System

```
Job statement
USER statement
CHARGE statement
FTN5.
ATTACH(ISFILE/UN=userno,M=W)
ATTACH(UPDAT)
LGO.
CRMEP(LO,RU)
```

NOS/BE Operating System

```
Job statement
ACCOUNT statement
FTN5.
ATTACH(ISFILE,ID=AAMUG)
ATTACH(UPDAT,ID=AAMUG)
LGO.
CRMEP(LO,RU)
```

B. Input File: UPDAT

```
update { 1  R500   104476
       { 1  B695   382592
add ---> 2  S703                   S703  ROYAL SUPPLY          185722    184722
delete-> 3  H314
```

C. Source Program

```
      PROGRAM ISUPDAT
C
C  ***************************************************************
C  * THIS PROGRAM UPDATES THE IS FILE NAMED ISFILE.  INPUT      *
C  * RECORDS, CONTAINING UPDATE INFORMATION, EXIST ON THE FILE*
C  * NAMED UPDAT.                                               *
C  ***************************************************************
C
      IMPLICIT INTEGER (A-Z)
      DIMENSION ISFIT (35), ISWSA (5)
      CALL FILEIS (ISFIT, 'LFN', 'ISFILE', 'ORG', 'NEW',
     +            'WSA', ISWSA,
     +            'RT', 'F',
     +            'KA', ISKEY,
     +            'EFC', 3, 'DFC', 3)
      OPEN (2, FILE = 'UPDAT')
      CALL OPENM (ISFIT, 'I-O')
      IF (IFETCH (ISFIT, 'ES') .NE. 0) GO TO 90
      PRINT 100
   10 READ (2, 110, END = 60) CODE, ISKEY, ORDER, ISWSA
   20 PRINT 140, CODE, ISKEY, ORDER, ISWSA
C
C UPDATE EXISTING RECORD IN THE FILE
C
      IF (CODE .EQ. 1) THEN
        CALL GET (ISFIT)
        IF (IFETCH (ISFIT, 'ES') .NE. 0) THEN
          PRINT 903, IFETCH (ISFIT, 'ES')
          CALL CLOSEM (ISFIT)
          STOP 'CRM ERROR RETURNED'
          END IF
        ISWSA (4) = ORDER
        ISWSA (5) =   ISWSA (5) + ORDER
        CALL REPLC (ISFIT)
        GO TO 10
        END IF
C
C ADD NEW RECORD TO THE FILE
C
      IF (CODE .EQ. 2) THEN
        CALL PUT (ISFIT)
        IF (IFETCH (ISFIT, 'ES') .NE. 0) GO TO 90
        GO TO 10
        END IF
```

Figure 3-10. Processing an Existing Indexed Sequential File (Sheet 1 of 2)

Source Program (Contd)

```
C
C  DELETE EXISTING RECORD FROM THE FILE
C
      IF (CODE .EQ. 3) THEN
        CALL DLTE (ISFIT)
        IF (IFETCH (ISFIT, 'ES') .NE. 0) GO TO 90
        GO TO 10
        END IF
C
C  REWIND THE FILE FOR SEQUENTIAL READING
C
   60 CALL REWND (ISFIT)
      PRINT 120
   70 CALL GETN (ISFIT)
      IF (IFETCH (ISFIT, 'ES') .NE. 0) GO TO 90
      IF (IFETCH (ISFIT, 'FP') .NE. O"20") GO TO 80
      PRINT 130, ISWSA
      GO TO 70
   80 CALL CLOSEM (ISFIT)
      STOP 'NORMAL TERMINATION'
   90 PRINT 903, IFETCH (ISFIT, 'ES')
      CALL CLOSEM (ISFIT)
      STOP 'CRM ERROR RETURNED'
  100 FORMAT ('1CODE   KEY     ORDER AMT            NEW RECORD')
  110 FORMAT (1X, I1, 2X, A4, 2X, I10, 10X, 3A10, 2I10)
  120 FORMAT ('1    KEY        COMPANY            ORDER    YTD AMT')
  130 FORMAT (3A10, 2I10)
  140 FORMAT (4X, I1, 2X, A4, 2X, I10, 10X, 3A10, 2I10)
  902 FORMAT (5X, 8A10)
  903 FORMAT ('ES = ', O3)
      END
```

D. Output

```
CODE  KEY    ORDER AMT          NEW RECORD
   1  R500     104476                                               0         0
   1  B695     382592                                               0         0
   2  S703          0           S703  ROYAL SUPPLY             185722    184722
   3  H314          0                                               0         0
   KEY        COMPANY          ORDER    YTD AMT
   B695  ABC DISTRIBUTORS      382592   8263213
   E482  CORP SALES INC         89207   1145833
   K260  FRIENDLY SALES       1027866   6347410
   M857  OAKVILLE CORP         149287   3014192
   R500  RETAILERS INC         104476   1360344
   S703  ROYAL SUPPLY          185722    184722
   T258  SELECTIVE SALES CO    248916   4337092
   T289  SMITH AND SON          95625   1528863
   V440  WORLD SALES CO        914819   6844272
   X179  YOUNG BROTHERS        527638   2428535
```

Figure 3-10. Processing an Existing Indexed Sequential File (Sheet 2 of 2)

Figure 3-10 (Part D) shows the printed output from program ISUPDAT. The first part of the listing shows the input records. The last part of the listing shows the contents of the updated file, which can be compared with the original file contents shown in figure 3-7. The updated file reflects the following operations:

- The record for Retailers Inc (primary key R500) was updated with a new order amount and an increased year-to-date amount.
- The record for ABC Distributors (primary key B695) was updated with a new order amount and an increased year-to-date amount.
- The record for Royal Supply (primary key S703) was inserted into the file.
- The record for Day and Night Sales (primary key H314) was deleted from the file.

POSITIONING AN INDEXED SEQUENTIAL FILE

A file with indexed sequential organization can be positioned in several ways. Because data records are in sequential order, sequential access is possible from the beginning of the file or from any record at which the file is positioned. File position is accomplished as follows:

- An open request positions the file either at the beginning of the first record or after the last record in the file.
- A random read request positions the file at the end of the record with the specified primary key and returns that record to the working storage area.
- A start request positions the file at the beginning of the record that meets a specified condition; the record is not returned to the working storage area.
- A skip request positions the file forward or backward to the beginning of a record identified by its sequential position in the file.
- A rewind request positions the file at the beginning of the first record in the file.

When a series of sequential read requests are being issued, the sequential file position is not affected by intervening write, replace, or delete requests.

POSITIONING TO A SPECIFIC RECORD

An indexed sequential file is positioned to a specific record within the file by either a random read request or a start request. In both cases, sequential read requests can then be issued to access the file sequentially from that position. For a random read request, the record is returned to the user. For a start request, the record is not returned.

Random Read Request

A random read request transfers to the working storage area the record associated with the specified primary key value. File position is changed by the record transfer. A sequential read request then returns the next logical file record. The random read alters the current file position to the record returned. Each sequential read increments the current file position by one record.

If the file has symbolic primary key values AKEY through ZKEY, a random read request with the primary key value SKEY changes the current file position to the end of the record with that primary key value and returns the record to the user. A sequential read request then advances the current file position one record and returns the record with the primary key value TKEY.

Start Request

The start request only positions the file; it does not return a record to the working storage area. The file is positioned to the beginning of the record with the primary key value that satisfies a certain primary key relationship. This type of file positioning is accomplished by executing a CALL STARTM statement. The format of this statement is shown in appendix B.

The following statements position the file at the first record in which the primary key is greater than a specified value:

```
     CALL STOREF (ISFIT, 'REL', 'GT')
     CALL STARTM (ISFIT)
```

If the value of the MKL field is zero, the full primary key is used to position the file. If the MKL field is not zero, the file is positioned by major key. The full primary key of the record at which the file is positioned is returned to the location indicated by the KA and KP fields. If GETN is called and the KA field it not specified following a STARTM call, the primary key will be returned to the address specified by the STARTM call.

Setting the key relation (REL) field establishes the relationship between the primary key of the record at which the file is positioned and the primary key at the location indicated by the KA field. The possible relationships are as follows:

- REL field set to EQ

  The data record primary key is equal to the primary key at location KA.

- REL field set to GE

  The data record primary key is equal to the primary key at location KA if an equal key exists, or it is the next greater primary key value.

- REL field set to GT

  The data record primary key is the first key value greater than the primary key at location KA.

If the REL field is set to EQ and an equal primary key does not exist in the file, a trivial error occurs and the file is positioned at the record with the next greater primary key value.

The MKL field is set when the file is to be positioned according to a major key. Major key processing is the same for a start request as it is for a random read request. The number of leading characters in the symbolic key to be used as a major key is specified by the MKL field. The file is then positioned to the first record with a major key that satisfies the relationship, and the full primary key is returned to the user.

Program STARTCK, shown in figure 3-11, issues a start request to position the file for sequential reading. The REL field is set to EQ by the CALL STOREF statement and the file is positioned to the record with the primary key value read interactively. Sequential reading then begins with that record.

SKIPPING RECORDS

You can reposition an indexed sequential file forward or backward a specified number of records by executing a CALL SKIP statement. The format of this statement is shown in appendix B.

The following statement causes a forward skip of five records:

```
    CALL SKIP (ISFIT, +5)
```

A. NOS Operating System

```
Job statement
USER statement
CHARGE statement
FTN5.
ATTACH(ISFILE/UN=userno)
LGO.
CRMEP(LO,RU)
```

NOS/BE Operating System

```
Job statement
ACCOUNT statement
FTN5.
ATTACH(ISFILE,ID=AAMUG)
LOG.
CRMEP(LO,RU)
```

B. Input File: ISFILE

```
B695  ABC DISTRIBUTORS        382592   8263213
E482  CORP SALES INC           89207   1145833
K260  FRIENDLY SALES         1027866   6347410
M857  OAKVILLE CORP           149287   3014192
R500  RETAILERS INC           104476   1360344
S703  ROYAL SUPPLY            185722    184722
T258  SELECTIVE SALES CO      248916   4337092
T289  SMITH AND SON            95625   1528863
V440  WORLD SALES CO          914819   6844272
X179  YOUNG BROTHERS          527638   2428535
```

C. Source Program

```
      PROGRAM STARTCK
C
C ****************************************************************
C * THIS PROGRAM ILLUSTRATES THE POSITIONING OF AN IS FILE      *
C * (ISFILE) BY SETTING THE REL FIELD AND THEN USING THE        *
C * THE CALL STARTM STATEMENT.                                  *
C ****************************************************************
C
      IMPLICIT INTEGER (A-Z)
      DIMENSION ISFIT (35), ISWSA (5)
      CALL FILEIS (ISFIT, 'LFN', 'ISFILE', 'ORG', 'NEW',
     +            'WSA', ISWSA,
     +            'RT', 'F',
     +            'EFC', 3, 'DFC', 3)
      CALL OPENM (ISFIT)
      IF (IFETCH (ISFIT, 'ES') .NE. 0) GO TO 50
    5 READ (*, '(A4)') ISKEY
      CALL STOREF (ISFIT, 'REL', 'EQ')
      CALL STARTM (ISFIT, ISKEY)
C
C TEST FOR KEY NOT FOUND
C
      IF (IFETCH (ISFIT, 'ES') .EQ. O"445") THEN
        PRINT *, 'KEY NOT FOUND'
        GO TO 5
        END IF
C
C TEST FOR ANY OTHER CRM ERROR
C
      IF (IFETCH (ISFIT, 'ES') .NE. 0) THEN
        PRINT 902, IFETCH (ISFIT, 'ES')
        CALL CLOSEM (ISFIT)
        STOP 'CRM ERROR RETURNED'
        END IF
   10 CALL GETN (ISFIT)
      IF (IFETCH (ISFIT, 'ES') .NE. 0) GO TO 50
      IF (IFETCH (ISFIT, 'FP') .NE. O"20") GO TO 20
      PRINT 110, ISWSA
      GO TO 10
   20 CALL CLOSEM (ISFIT)
      STOP
```

Figure 3-11. Positioning an Existing Indexed Sequential File (Sheet 1 of 2)

Source Program (Contd)

```
 50  PRINT 902, IFETCH (ISFIT, 'ES')
     CALL CLOSEM (ISFIT)
     STOP 'CRM ERROR RETURNED'
110  FORMAT (1X, 3A10, 2I10)
902  FORMAT ('ES = ', O3)
     END
```

D. Sample Output

```
/LGO
? m857
     M857  OAKVILLE CORP            149287   3014192
     R500  RETAILERS INC            104476   1360344
     S703  ROYAL SUPPLY             185722    184722
     T258  SELECTIVE SALES CO       248916   4337092
     T289  SMITH AND SON             95625   1528863
     V440  WORLD SALES CO           914819   6844272
     X179  YOUNG BROTHERS           527638   2428535
      .028 CP SECONDS EXECUTION TIME.
/REWIND,*
  13 FILE(S) PROCESSED.
/LGO
? s703
     S703  ROYAL SUPPLY             185722    184722
     T258  SELECTIVE SALES CO       248916   4337092
     T289  SMITH AND SON             95625   1528863
     V440  WORLD SALES CO           914819   6844272
     X179  YOUNG BROTHERS           527638   2428535
      .026 CP SECONDS EXECUTION TIME.
/REWIND,*
  13 FILE(S) PROCESSED.
/LGO
? bad1
 KEY NOT FOUND
? t289
     T289  SMITH AND SON             95625   1528863
     V440  WORLD SALES CO           914819   6844272
     X179  YOUNG BROTHERS           527638   2428535
      .027 CP SECONDS EXECUTION TIME.
```

Figure 3-11. Positioning an Existing Indexed Sequential File (Sheet 2 of 2)

The skip request specifies the name of the FIT array, the direction of the skip, and the number of records to be skipped. The skip direction and count parameter is specified as follows:

+count	Skip forward the specified number of records.
-count	Skip backward the specified number of records.

No FIT fields are set by the skip request. Skipping stops if beginning-of-information or end-of-information is reached before the specified number of records have been skipped. If the end-of-data exit (DX) field has been set to a subroutine name, the subroutine is executed. The file position (FP) field is set to 1 for beginning-of-information or 100_8 for end-of-information.

Skipping either direction always positions the file to the beginning of another record. The skip request should be used only for skipping a small number of records. This is because AAM reads and counts each intervening record and thus increases execution time. A random read request takes less time than a lengthy skip request.

REWINDING THE FILE

Execution of a CALL REWND statement positions the file to beginning-of-information, which is the start of the record with the lowest primary key value. The format of this statement is shown in appendix B.

Execution of the following statement rewinds the file to beginning-of-information.

```
CALL REWND (ISFIT)
```

The file must be open when the rewind request is issued. Rewinding the file is preferable to extensive backward skipping of records. The only parameter in the rewind request is the name of the array containing the FIT.

The CALL REWND statement must not be used during an indexed sequential file creation run.

OBTAINING FILE STATISTICS

Various statistics are accumulated in the file statistics table for an indexed sequential file (as for all AAM files). These statistics can be accessed by the user through the FLSTAT utility. This utility can be called into execution following program execution or it can be run in a separate job after attaching the permanent indexed sequential file. Output from the FLSTAT utility includes current file structure, primary key information, and total file transaction counts.

Information in the output provides clues to file efficiency. An increase in the number of index levels over the last use of the file indicates an increase in the time required to retrieve a given record by primary key.

The number of empty blocks reveals unused area in the mass storage allocated to the file. Empty blocks, which are created by the deletion of all records in the blocks, are used when new blocks are required for file expansion. Empty blocks are assigned to the current file and are not released to the operating system for reallocation to other jobs. Therefore, it might be desirable to re-create the file in a more compact area of storage.

If the utility is executed both before and after the file is processed, the difference in the statistics reflects the processing performed during program execution.

The FLSTAT control statement specifies the logical file names of the indexed sequential file and the output file for the statistics. Only one indexed sequential file can be named in the control statement. Multiple statements are needed to obtain information about several indexed sequential files. The output file need not be named if statistics are to be written to the system file OUTPUT.

The control statement used to generate the output shown in figure 3-12 is as follows:

```
    FLSTAT (ISFILE)
```

The output file is not specified in the control statement; it defaults to the system file OUTPUT.

```
 STATISTICS FOR FILE ISFILE
 ORGANIZATION------- IS
 CREATION DATE------ 81/02/26.
 DATE OF LAST CLOSE- 81/02/26.
 TIME OF LAST CLOSE- 13.10.02.

 FILE IS NOT MIPPED
 COLLATION IS STANDARD

 PRIMARY KEY INFORMATION
   STARTING WORD POSITION ------ 0
   STARTING CHARACTER POSITION - 4
   TYPE -- UNSIGNED
   LENGTH IN CHARACTERS -------- 4

 MAXIMUM RECORD SIZE  50
 MINIMUM RECORD SIZE  50

 TOTAL TRANSACTIONS
   NUMBER OF PUTS ------ 11
   NUMBER OF GETS ------ 2
   NUMBER OF DELETES --- 1
   NUMBER OF REPLACES -- 2
   NUMBER OF GETNEXTS -- 29

 CIO CALLS FOR FILE
   NUMBER OF READS ----- 8
   NUMBER OF WRITES ---- 2
   NUMBER OF RECALLS --- 1
   NUMBER OF REWRITES -- 7

 NUMBER OF BLOCKS------- 1
 NUMBER OF EMPTY BLOCKS- 0
 BLOCK SIZE IN PRUS----- 1
 NUMBER OF DATA RECORDS- 10

 FILE LENGTH IN PRUS  3
 NUMBER OF INDEX LEVELS IN USE    0
FLSTAT,ISFILE.
```

Figure 3-12. FLSTAT Utility Output

MULTIPLE INDEX FILES 4

The multiple-index capability provides added features to the AAM file structuring and management facilities for indexed sequential, direct access, and actual key file organizations. The term multiple-index file refers to an indexed sequential, direct access, or actual key file for which additional keys, called altenate keys, have been defined. (The terms multiple-index file and alternate key file are interchangeable.) AAM can locate a record in a multiple-index file by the primary key or by one of the alternate keys. You can create multiple-index files through COBOL, FORTRAN, COMPASS, CDCS, and Query Update. This section illustrates the usage of the Multiple-Index Processor (MIP) through FORTRAN 5.

WHY USE MIP?

Consider the data base shown in figure 4-1. The file is indexed sequential file organization and NAME is the key. Suppose you want a list of all persons who live in DETROIT. Each record in the file must be read and each CITY field must be examined for DETROIT. This would result in poor performance, especially for a large data base file. In a time-sharing environment, response time would be unacceptable.

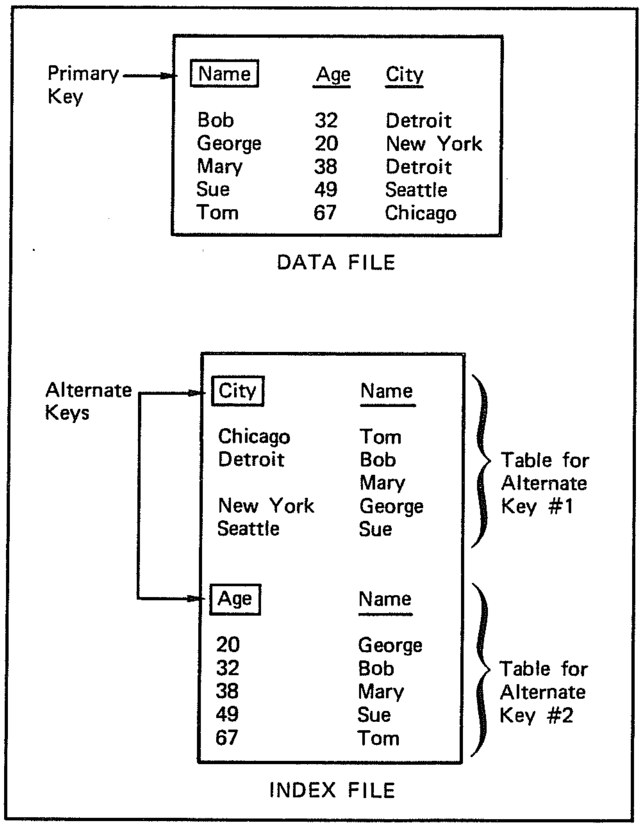


Figure 4-1. A Multiple-Index File

To improve performance, CITY can be defined as an alternate key and a table can be constructed to list all NAMES for each CITY. The table can then be scanned for DETROIT. Random accesses of the data file return all the names for Detroit.

Similarly, if AGE is defined as an alternate key, a table can be constructed to list all NAMES for each AGE. Locating all persons within a given age range would be easy.

Alternate key reads are more efficient (requiring fewer disk accesses) than a sequential pass through the data file. When you define a field as an alternate key, MIP automatically constructs a table such as the tables shown in figure 4-1. The table scan for a particular value and the random accesses of the data file are also automatic. Other MIP facilities include:

- Automatic updating of the index file when the data file is changed.
- Counting the number of record occurrences for a given alternate key value or for a given range of alternate key values.
- Retrieving only primary keys of records for a given alternate key value or for a given range of alternate key values.

Although more alternate keys add more flexibility, there is a cost involved. If a file is to be updated frequently, you should avoid unnecessary alternate keys since each key would generate extra disk accesses per update of a record.

MULTIPLE KEY CONCEPTS

Figure 4-2 illustrates the flow of control within AAM for alternate key processing. Control passes to MIP if an alternate key is specified on a read or if an alternate key file is being updated. On a read, MIP obtains a primary key from the index file and returns control to the appropriate AAM module to actually retrieve a record from the file and return it to your working storage area. On a write or update of the file, MIP updates the index file to reflect any changes.

When a primary key is specified on a read of a multiple-index file, control does not pass to MIP. The record is retrieved exactly the same way as it would be for a file that did not have alternate keys defined for it.

The basic principles of multiple-index file structure and processing with which you should be familiar include the following:

- The indexed sequential, direct access, or actual key data file structure is not affected by the definition of alternate keys.
- One index file exists for each data file. Within the index file, MIP creates a separate index for each alternate key defined for the data file. Both the data file and the index file must be present for the file to be accessed by alternate keys.

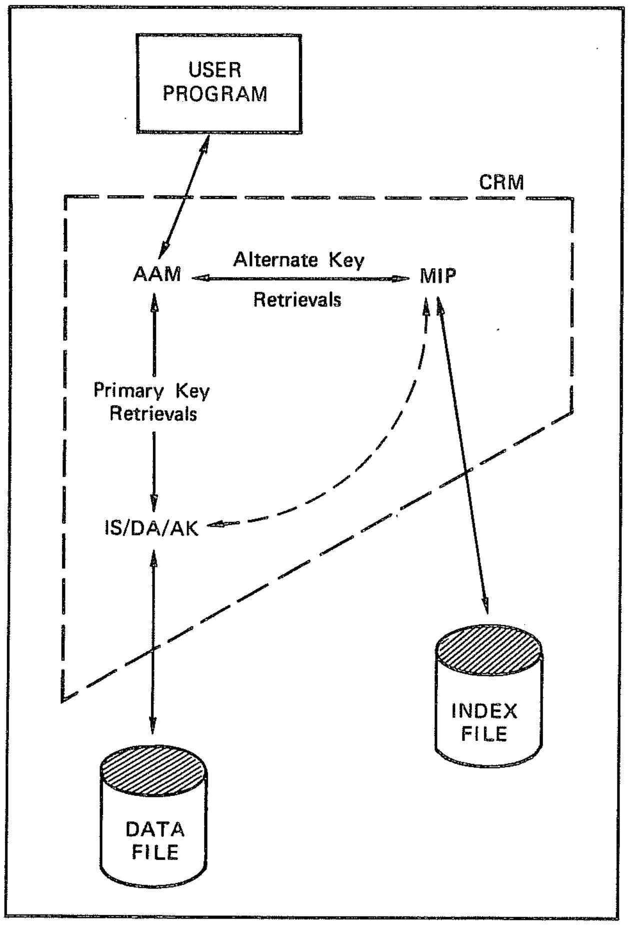


Figure 4-2. Multiple-Index File Processing

- The MIPGEN utility is used to define alternate keys for an existing data file, to delete alternate keys from an existing index file, and to add new alternate keys to an existing index file.
- Each alternate key is defined by the RKW, RKP, and KL values. (There is no name associated with an alternate key.)
- Reading of records can be by either primary or alternate key values, but the data file is updated by primary key values only.
- Information from the index file can be retrieved through FORTRAN (or COMPASS) programs without accessing the data file.

PRIMARY KEYS

A primary key is a field whose contents determine the physical location of the record within the data file. The location of the primary key in a data record is established when the file is created; it remains the means by which the system accesses each record for the life of the file. A primary key can be located within the record (embedded) or outside of the record (non-embedded). Primary key values must be unique.

A program can always access a record by its primary key value. All write or rewrite operations must be according to primary key value. Read access, however, can be by primary key value or by the value of any alternate key defined for the record. MIP translates alternate keys into the primary record key. The program can specify an alternate key value, but the system request is for the proper primary key value.

ALTERNATE KEYS

An alternate key is a field within a record, the contents of which can be used to access the record. An alternate key position can be either defined through a source program at the time a new data file and its index file are created, or defined through the MIPGEN utility if the data file already exists. (MIPGEN is more efficient.) Up to 255 alternate keys can be defined.

You cannot use the alternate key value to update a record directly. You can, however, use the alternate key value to locate the primary key value and then perform a delete or rewrite by primary key. Each read by alternate key value can return the value of the primary key to the program, depending on the file organization and key handling.

Alternate key values can be unique (such as a social security number for each name) or can identify a group of data records (such as one city associated with many names). Further, an alternate key can be defined as a field within a group that repeats within a record, so that a single definition might result in several alternate key values for that record (such as license numbers of cars owned by one person).

Shorter keys are preferable for alternate keys because of space considerations in the index file. An index block can hold twice as many 10-character keys as 20-character keys. For alternate keys with unique values, the number of characters in the alternate key and the primary key can affect how much storage space is needed. For example, because round-up to a full word boundary is performed, an alternate key with 6 characters and a primary key with 15 characters would require three words of storage space, while an alternate key with 5 characters and a primary key with 15 characters would require only two words. (Selection of actual key file organization and a small key length minimizes many of these problems.)

Alternate Key Location

You define an alternate key by its length and position within the data record. For MIPGEN purposes, and for key definition through COMPASS and FORTRAN, the position is described by the starting location and length in words and characters. You must also specify the format of the key and whether or not it is within a repeating group.

Alternate keys defined for a file must be contained within the data record (embedded). All alternate keys must be within the minimum record length specified by the MNR field of the FIT (except for T-type records, when MNR=HL). Alternate key positions can overlap. One alternate key can begin at the same location as another alternate key or at the same location as the primary key; however, their lengths must differ if they begin at the same location. An example of overlapping definition might be employee name, where two alternate keys could be defined to provide retrieval on the last name and whole name.

- Words in the data record are numbered from 0, with ten 6-bit characters per word.
- Characters within a word are numbered 0 through 9, left to right, with six bits per character.
- The length of an alternate key is specified in characters, whether the key contains character or numeric data.

The content of an alternate key field is interpreted according to the key format specification.

Alternate Key Format

Use the format of the alternate key to rank the alternate key contents. (Within an index for a single alternate key field, alternate key values are always in ascending order.) Each alternate key field can be a different format, since each field appears in its own index within the index file. The following formats can be defined:

- Symbolic (KT=S)

  The content of the key field is a string of letters, digits, or special characters. These keys are sorted according to the collating sequence in effect. (For an indexed sequential file, a user-defined or a default collating sequence is used for both the data file and the index when the key is symbolic format. For a direct access or actual key file, any user-defined collating sequence specified by the DCT/CDT parameters is used for the index only.) Key length is 1 to 255 characters. If you do not supply a collating sequence, the standard collating sequence is used (see appendix A).

- Integer (KT=I)

  The content of the key field is interpreted as an integer; the left-most bit of the field indicates the sign (0 for positive, 1 for negative), and the remainder of the field indicates the absolute value of the number (complemented, if negative). These keys are sorted by magnitude. You must specify an integer key as 10 characters in length; the key must begin on a word boundary. Floating point numbers are stored with KT=I.

- Uncollated (KT=U)

  The content of the key field is interpreted as a positive integer, even though the content might be a character string. These keys are sorted by magnitude. Key length must be a multiple of six bits, 1 to 255 characters.

THE INDEX FILE

The index file is separate from the data file. It is created in response to one of the following:

- A FORTRAN or COMPASS program defining at least one alternate key. The RMKDEF subroutine is used to define alternate keys after the file is opened and before the first record is created.
- The MIPGEN utility (RMKDEF directive) when the named index file is not attached to the job and the data file FSTT indicates that an index file does not exist. (This is the most efficient method of creating an index file.)

You now have two ways to define alternate keys: using the RMKDEF subroutine in a FORTRAN program or using the RMKDEF directive with the MIPGEN utility. The first parameter of the RMKDEF directive is lfn; however, the first parameter of the RMKDEF subroutine is fit. The parameters differ because within a FORTRAN program all references to an AAM file are made through the FIT.

You are responsible for permanent file operations with the index file and for making it available to any job that accesses the data file by alternate keys or that updates records in the data file. The correct permissions must be specified. The XN field of the FIT for the data file specifies the index file name.

Once an index file has been associated with a data file, you must attach that index file to any job that either:

- Attaches and updates the data file.

or

- Retrieves records by alternate key.

Within the index file is the logical structure shown in figure 4-3. All programmer operations are concerned only with the logical structure.

Notice that the index file contains only key information. A read by alternate key value requires a search of the index file, then access to the data file using the primary key value obtained from the index for a particular alternate key field.

Figure 4-3 illustrates the two following words, index and keylist, that are used in describing index files:

- An index is a part of an index file that pertains to a single alternate key field. As many indexes exist as there are alternate keys defined.
- A keylist is a part of an index that pertains to a single alternate key value. One keylist contains the primary keys of all data records with a particular value of an alternate key. As many keylists exist as there are alternate key values. A keylist can take one of three structures: indexed sequential, unique, and first-in-first-out (FIFO). These structures are specified on the RMKDEF call by I, U, or F, respectively.

Consider the data base for a telephone directory shown in figure 4-4. Assume the file was created with records in the order listed in part A of the figure. NAME is the primary key. Three alternate keys are defined: AREA (the 3-digit area code), PHONE (the complete 10-digit phone number), and CITY (the 7-character city code).

The index file, shown in part B, has three indexes. The index for AREA can have multiple NAME values for each area code. You can select either the indexed or FIFO structure for the keylist. Assuming that you have defined the key with indexed structure (by specifying I on the RMKDEF directive), the index for AREA appears as shown in part B of the figure. Notice that FERON precedes WADE and ANTON precedes JONES.

The index for PHONE has a unique NAME for each value of the alternate key. Therefore, you must define the key as unique (by specifying U on the RMKDEF directive).

The index for CITY can have multiple NAME values for each city. Assuming that you have defined the key with FIFO structure (by specifying F on the RMKDEF directive) the index for CITY appears as shown in part B of

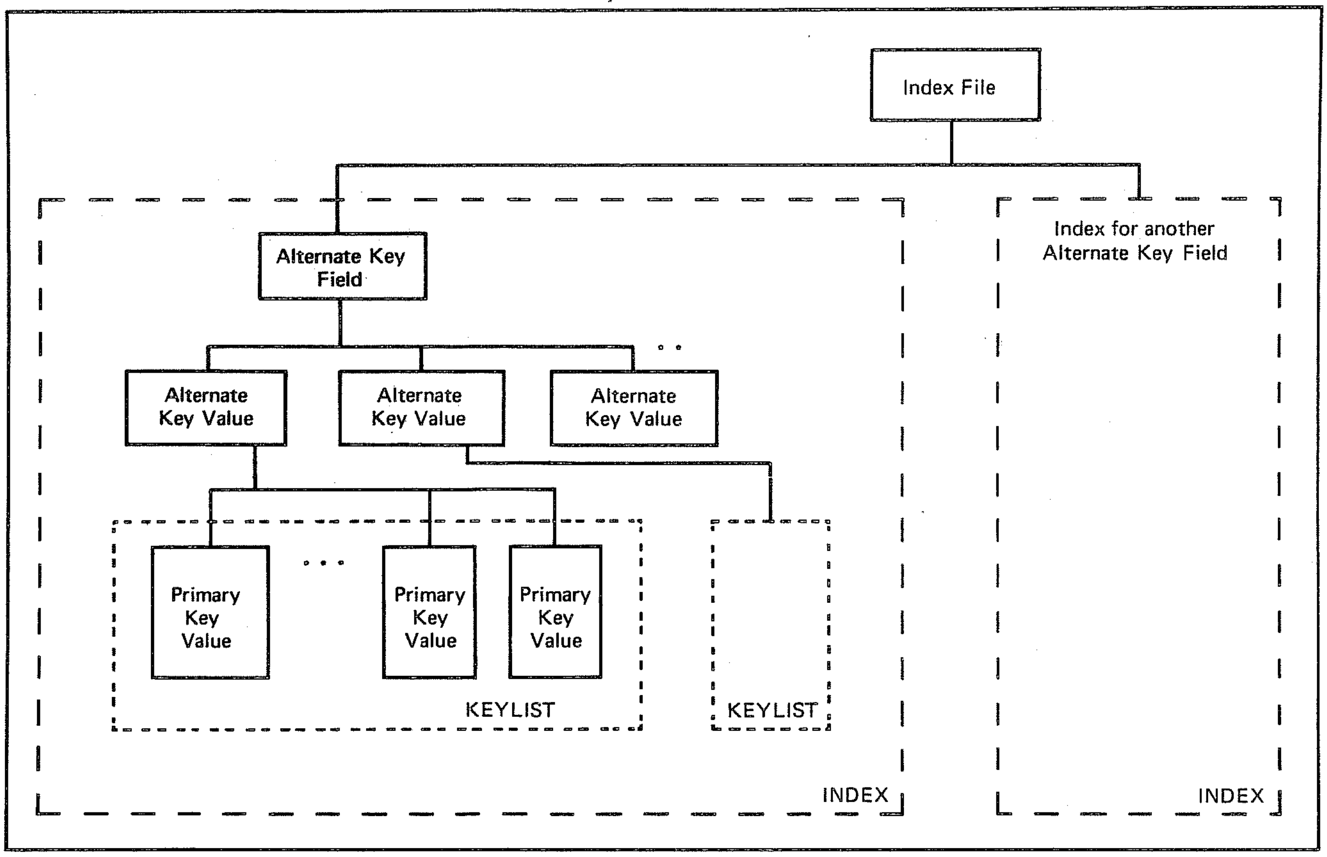


Figure 4-3. Logical Structure of Index File with Two Alternate Keys and Multiple Alternate Key Values

the figure. Notice that JONES comes before ANTON. (This method is not recommended.)

The keylist structure for a particular index is selected when the alternate key is defined originally or is added to the index file. One of the following structures can be specified for the keylist:

- Indexed

  The primary keys are stored in the keylist in sorted order. Numeric keys (integers) are stored in ascending order. Symbolic keys are stored in sorted order, with the order depending on the file organization and the means used to create the index. Uncollated keys are stored in order of magnitude of their display code values.

  Keys for a file in indexed sequential organization always are stored in the order established by the collating sequence in effect for the data file.

  Keys for a file in direct access or actual key organization are stored in one of two ways:

    If the MIPGEN utility creates the index, the keys are in the default collating sequence order.

    If the index is created at the same time the data file is created, the order is affected by your use of the DCT/CDT fields of the FIT. If you establish a table to convert display code to a collating sequence, that sequence is used for the index; otherwise, the default collating sequence is used.

  The indexed keylist structure is preferable to FIFO because it gives superior performance when an alternate key file is accessed.

- Unique

  Each alternate key value can be contained in only one record in the file; the keylist for each alternate key value has only one primary key.

- First-in-first-out (FIFO)

  The primary keys are stored in the keylist in the order in which their corresponding records are written to the data file. If MIPGEN is used to define alternate keys for an existing data file, the primary keys are stored in the order in which the records are read by MIPGEN.

  FIFO structure cannot be used for repeating group keys.

THE MIPGEN UTILITY

The MIPGEN utility performs the following operations:

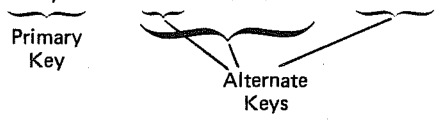
- Defines alternate keys and an index file for an existing file that does not have them.

- Adds or removes alternate key indexes to/from an existing multiple-index file.

The MIPGEN utility is the only method by which you can change alternate key definitions after an index file has been created. You cannot add, replace, or delete alternate keys through source language directives (that is, through FORTRAN or COMPASS calls to RMKDEF). Figure 4-5 illustrates an overview of MIPGEN processing.

A. Order in which records were written:

NAME	PHONE		CITY
	AREA	LOCAL	
Jones, P	612	853-5523	Denver
Wade, D	415	393-7204	Boise
Anton, R	612	482-3933	Denver
Feron, A	415	297-4841	Boise
Tabb, W	404	465-3207	Houston



B. Structure of index file

AREA	NAME
404	Tabb, W
415	Feron, A
	Wade, D
612	Anton, R
	Jones, P

Alternate Key Index
(Indexed)

PHONE	NAME
404-465-3207	Tabb, W
415-297-4841	Feron, A
415-393-7204	Wade, D
612-482-3933	Anton, R
612-853-5523	Jones, P

Alternate Key Index #2
(Unique)

CITY	NAME
Boise	Wade, D
	Feron, A
Denver	Jones, P
	Anton, R
Houston	Tabb, W

Alternate Key Index #3
(FIFO)

Figure 4-4. Telephone Directory Data Base Illustrating 3 Index File Structures

The MIPGEN utility is used to generate a new index file or to add or remove an entire index in an existing index file. It is more efficient for most applications to create an index file through MIPGEN rather than to create the index file and the data file at the same time through the source language. If the data file is created in a separate job step, MIP can then sort the records and build a compact index file. Minimum input/output operations are involved. If the data file and the index file are created at the same time, records are randomly written to the index file and a large number of input/output operations are required. This method is always inefficient for large files, especially if a file has more than one alternate key.

The format of the MIPGEN control statement is shown in figure 4-6. The logical file name of the existing data file is a required parameter. The logical file names of the directive file and the output listing file are optional when the defaults (INPUT and OUTPUT) are applicable.

One or more RMKDEF directives are required to define the alternate keys to be incorporated in or removed from an index file. MIPGEN obtains primary key information from the data file FSTT.

Consider the records shown in the data base in part A of figure 4-7. NAME is the primary key. Suppose you want to use MIPGEN to add AUTO as an alternate key. Part B of the figure shows the statements you need. Part C of the figure shows the output and the dayfile from the MIPGEN utility execution.

MIPGEN JOB STRUCTURE

The MIPGEN utility is called by a control statement. During execution, the utility reads user-specified directives from a directive file and then creates or modifies an index file according to the instructions on those directives.

A job that calls MIPGEN must contain control statements to describe the files, to ensure files are properly handled by the permanent file part of the operating system, and to call the utility itself. The directives for the utility can exist as part of a job deck or as a separate file. The data file must be in existence before the utility is called.

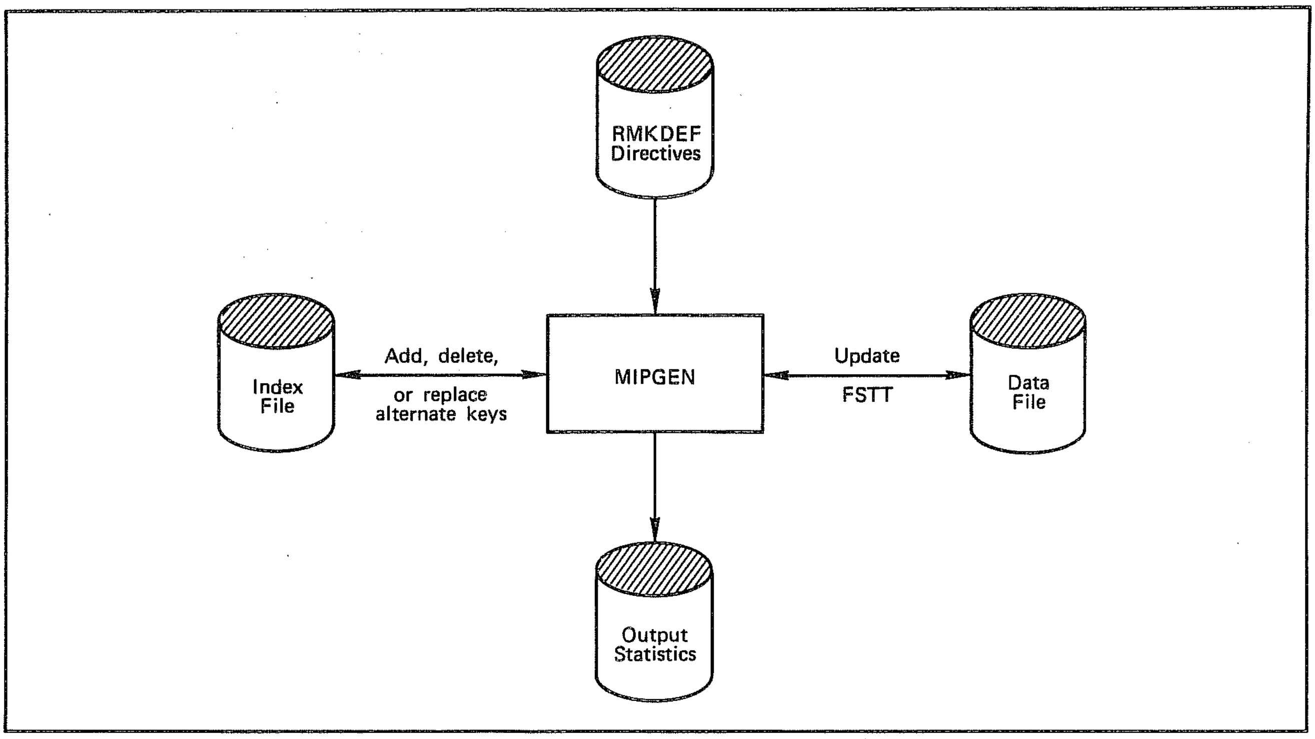


Figure 4-5. MIPGEN Utility Overview

MIPGEN(prifile,directs,lfile)

prifile	Logical file name of the existing indexed sequential, direct access, or actual key file.
directs	Name of the file containing the RMKDEF directives; optional; default is INPUT.
lfile	Name of the file that contains the output listing from MIPGEN; optional; default is OUTPUT.

Figure 4-6. MIPGEN Control Statement Format

The control statements that you need for a MIPGEN job include:

- An ATTACH control statement to attach the existing data file. Write permission is required since the data file FSTT on disk will be updated to indicate the presence of the MIP file.

- Permanent file control statements for the index file: for NOS, the DEFINE control statement; for NOS/BE, the REQUEST and CATALOG control statements. These are required if no index file exists prior to the run. For subsequent runs, an ATTACH control statement (with write permission) is required to attach the existing index file.

- The FILE control statement to communicate to AAM. The logical file name of the index file must be specified via the XN parameter, and the organization of the data file must be specified via the FO parameter.

- The record type (RT) must be specified for T-type records, if the alternate key is in the trailer portion of the record. The associated parameters (HL, CP, CL, TL, and MRL) must also be defined.

- If the file named by the XN parameter is not attached to the job, and if the data file FSTT indicates that no index file exists, the utility creates such a file. If the file exists, the utility modifies it. In either case, the alternate key definitions are read from the directive file.

- The default index file block size is the data file block size. Accepting the default is more efficient. However, the default can be overridden by specifying a value in the XBS field of the FIT.

- The MIPGEN control statement to identify the data file and the source of directives and to initiate loading and execution of the utility program.

- An RMKDEF directive for each alternate key. A directive need not be included for the primary key.

  If the directives are on a separate file, the file must have CRM sequential organization with C-type blocking and Z-type records.

DIRECTIVE FOR DEFINING ALTERNATE KEYS - RMKDEF

When an index file is being created, each RMKDEF directive defines the location of one alternate key within the record. Figure 4-8 shows the directive format for defining an alternate key for the index file.

A. CARS File

```
NAME        AUTO

Bill        Saab
Jim         Fiat
John        Ford
Paul        Saab
↑           ↑
column 1    column 11
```

B. Define Auto as an Alternate Key

```
Job statement
USER statement
CHARGE statement
ATTACH,CARS/M=W ◄──────────────────────────── Attach CARS with write permission
FILE,CARS,FO=IS,ORG=NEW,RT=F,FL=20,XN=XCARS ◄── Specify XCARS as index file
DEFINE,XCARS ◄─────────────────────────────── Allocate permanent file space for index file
MIPGEN,CARS ◄──────────────────────────────── Call utility to add key
- - EOR
RMKDEF(CARS,1,0,10,0,S,I) ◄────────────────── Directive to define the AUTO field
- - EOI
            Key location   Key length   Key format   Keylist structure
```

C. MIPGEN Output and Job Dayfile

```
MIPGEN DIRECTIVES            80/10/03. 15.37.13.


        RMKDEF(CARS,1,0,10,0,S,I) ◄──────────── Defines Key added


 ACMQCLE. 80/10/03.(22) SVL SN112 NOS.

15.37.03.ADDKEY.
15.37.03.UCCR, 7641,       0.009KCDS.
15.37.04.USER(AAMUG)
15.37.04.CHARGE(XXXX,XXXXXX)
15.37.06.ATTACH,CARS/M=W.
15.37.07.FILE,CARS,FO=IS,ORG=NEW,XN=XCARS.
15.37.07.DEFINE,XCARS.
15.37.08.MIPGEN,CARS.
15.37.16.***KEY COMPARISON USED
15.37.18.MIPGEN COMPLETE ◄────────────────── You now have a multiple-index file CARS
15.37.19.UEAD.       0.002KUNS.               and an index file XCARS
15.37.19.UEPF,       0.021KUNS.
15.37.19.UEMS,       0.815KUNS.
15.37.19.UECP,       0.059SECS.
15.37.19.AESR,       2.285UNTS.
15.37.24.UCLP, 7645,       0.192KLNS.
```

Figure 4-7. MIPGEN Example - NOS

RMKDEF(prifile,rkw,rkp,kl,0,kf,ks,kg,kc,nl,ie,ch)

prifile	Logical file name of the existing indexed sequential, direct access, or actual key file; required.	
rkw	Relative word in the record in which the alternate key begins, counting from 0; required.	
rkp	Relative beginning character position within the relative key word (rkw), counting from 0; required.	
kl	Number of characters in the key, 1 to 255; required.	
0	Required to mark position for the reserved field.	
kf	Key format, required:	
	0 or S	Symbolic
	1 or I	Integer
	2 or U	Uncollated
	3 or P	Purge alternate key definition from the index
ks	Substructure for each primary key list in the index; optional:	
	U	Unique (default)
	I	Indexed sequential: recommended for efficiency in processing
	F	First-in first-out
kg	Length in characters in the entire repeating group in which the key resides.	
kc	Maximum number of occurrences of the repeating group; zero for T type records, and a number for a repeating group embedded within the record.	
nl	Null suppression; a null value is all spaces (symbolic key) or all zeros (integer key):	
	0	Null values are included in the index (default)
	N	Null values are not included in the index
ie	Include/exclude sparse control character:	
	I	Include alternate key value if the record contains a sparse control character
	E	Exclude alternate key value if the record contains a sparse control character
ch	Characters that qualify as sparse control characters; up to 36 letters and digits can be specified as a character string.	

Figure 4-8. RMKDEF Directive Format

Nonrepeating Keys

All RMKDEF directive parameters, except kg and kc, are applicable to an alternate key that does not repeat within the record. An example of a nonrepeating field was shown in figure 4-7.

Repeating Group Keys

A repeating group key is a single alternate key that can have several values within one data record because the key field can occur more than once within the record. MIP treats each occurrence as a separate value for the same alternate key field. Any of the values can be used to access the data record.

Consider the SPORTS file shown in Part A of figure 4-9. From one to three repetitions of the alternate key ACTIVITY can occur in the record. Part B of the figure shows the usage of the MIPGEN utility to add the alternate key ACTIVITY.

In the RMKDEF directive, the rkw, rkp, kl, kg, and kc parameters are all needed to define the group. The word and position for rkw and rkp must be that of the first occurrence of the key. The kl parameter specifies the length of a single occurrence of the key; the kg parameter specifies the number of characters in the entire group in which the key field resides. The repeating group can appear a fixed number of times or a variable number of times, depending on record format. The kc value either equals zero for a variable number or equals the fixed number.

In order to minimize mass storage required for each data record, you can establish a variable number of occurrences of the key field by using a repeating group at the end of the record. (Select T-type records.) Selection of T-type records, however, trades processing time for mass storage space.

The RMKDEF directive that describes the alternate key for the records in figure 4-9 is:

```
RMKDEF(SPORTS,3,4,6,0,S,I,6,0)
```

Notice that the last parameter is 0 to indicate that the content of the KOUNT field establishes the number of repetitions.

The keylist structure cannot be FIFO for a repeating group key. If F is specified in the directive, AAM changes it to I without a message.

Adding a New Index

When a new alternate key index is to be added to an existing index file an RMKDEF directive defines the key to be added. The format of the RMKDEF directive is the same as that when an alternate key is being defined for a new index file. The format shown in figure 4-8 should be used with parameters, depending on the characteristics of the alternate key field.

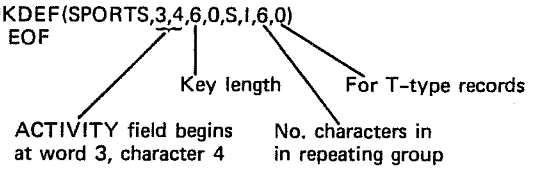
A. SPORTS File

	CLASS	NAME	KOUNT		ACTIVITY		
	SOPH	DIANE	2		GYMN	SKI	
	FRESHMAN	MIKE	1		SKI		
	SENIOR	RICH	2		TRACK	SOCCER	
	JUNIOR	RON	3		SWIM	SKI	TENNIS
	JUNIOR	SUSAN	1		TENNIS		
Column	11	21	31		35		52

3 Repeating Groups
6 chars each

B. Add ACTIVITY as an Alternate Key

```
JOB
USER
CHARGE
ATTACH,SPORTS/M=W
FILE,SPORTS,RT=T,HL=34,CP=30,CL=1,TL=6,MRL=52
FILE,SPORTS,FO=IS,ORG=NEW,XN=XSPORT
DEFINE,XSPORT
MIPGEN,SPORTS
- - EOR
RMKDEF(SPORTS,3,4,6,0,S,I,6,0)
- - EOF
```



Key length
For T-type records
ACTIVITY field begins at word 3, character 4
No. characters in in repeating group

Figure 4-9. Using the MIPGEN Utility to Add a Repeating Group Key - NOS

Deleting an Index

When an existing alternate key index is being removed from an index file, use the first six parameters of the RMKDEF directive format shown in figure 4-8. The key to be removed is identified by its starting position and length, as it is for all MIPGEN operations. The ks parameter, which under other circumstances establishes the structure of the keylist for each alternate key value, must be set to P to indicate that an index is to be removed from the index file.

ALTERNATE KEY PROCESSING EXAMPLES

This section describes the creation and processing of indexed sequential, direct access, and actual key files that have alternate keys defined.

Concepts are illustrated through FORTRAN 5 programs that show:

- Creating an indexed sequential file with an embedded primary key and two alternate keys.
- Reading an indexed sequential file by alternate key values.
- Creating an actual key file with a nonembedded primary key and two alternate keys.

  Reading an actual key file by alternate key values.
- Reading an actual key file through alternate key values, using relations other than equality.

The processing described in the examples is, in general, applicable to all AAM file organizations. Differences in processing arise from the underlying differences in multiple-index processing.

GENERAL CONSIDERATIONS

The following steps are needed in the creation of a multiple-index file:

1. Define the logical file names for both the data file (LFN parameter) and the index file (XN parameter)
2. Set additional FIT fields as required.
3. Create the data file and index file in one of two ways:

   Create both at the same time through the source program

   or

   Create only the data file through the source program and then create the index file through MIPGEN (generally more efficient).

The following steps are needed in the processing of a multiple-index file, when accessing by alternate key:

1. Attach the data file and the index file.
2. Open the data file. (The index file is opened automatically by AAM.)
3. Establish the alternate key field to be used for access. (Set values for RKW, RKP, and KL).
4. Establish search criteria. (Set values for KA, KP, and REL.)
5. Position the index file to a specific key value. (STARTM can be used).
6. Deliver a record to the working storage area. (GETN can be used.)
7. Check for file position (Use IFETCH (FIT, 'FP').)
8. Close the data file.

In addition, when processing a multiple-index file, you should be aware of the following:

- The current values of the RKW, RKP, and KL fields determine whether a primary key or alternate key controls the read. (The key currently being used to access a record is sometimes called the key of reference.) The RKW, RKP, and KL fields are set to the primary key when a CALL OPENM statement is executed.
- The REL setting for STARTM positions to the keylist for the first alternate key value that satisfies the REL condition.
- The FP field reflects a file position in relation to an index when an alternate key position is identified in the FIT. The proper interpretation of FP is:

10_8	A record returned to the working storage area from the current call to GET or GETN. The record is the last in the keylist.
20_8	A record was successfully returned to the working storage area.
100_8	The last GETN attempt encountered the end of the index file.

Table 4-1 contains a summary of FORTRAN calls for use with AAM files with alternate keys defined.

INDEXED SEQUENTIAL FILES WITH MIP

The programs that follow illustrate the creation of an indexed sequential file and the reading of the file by alternate key. The file processing shown is, in general, applicable to all AAM files with MIP.

Creating a Multiple-Index File

The program MAKXIS, shown in figure 4-10, creates an indexed sequential file (FLRECS) from a sequential file (MOVIES). The program establishes the FIT for the data file, specifies the index file name, defines two alternate keys, reads a file of input records, and writes the records to a new file. Any job that executes program MAKXIS is responsible for preserving both the data file and the index file as permanent files.

A FILE control statement is used to specify the maximum block length (MBL). (It could be included in the FILEIS call instead.) If MBL is not specified, a default value is calculated by AAM.

Define File Names

The indexed sequential file to be created is given the logical file name FLRECS. Its FIT is to be constructed in the array XISFIT, which is dimensioned to 35 words as required for all FITs.

The index file to be created is given the logical file name FLRXIP. You must specify the XN field when creating a file with alternate keys. Notice that this is the only reference you need for the index file. The file is created automatically by MIP.

The first line of the FILEIS call specifies the FIT name, data file name, and index file name.

```
CALL FILEIS (XISFIT, 'LFN', 'FLRECS', 'XN',
            'FLRXIP',
```

The next line of the FILEIS call specifies extended AAM file organization, as required.

```
'ORG', 'NEW'
```

Set Additional FIT Fields

Input data records exist on a file MOVIES (shown in part B of figure 4-10). The records have a maximum length of 80 characters and contain character data giving the name, date, and actors in a motion picture. The input file has CZ format (Z-type records and C-type blocking). Any other structure would require a FILE statement specifying the BT and RT fields.

The third line of the FILEIS call specifies record structure for the output file by setting the following FIT field values:

```
'RT', 'F', 'FL', 80
```

Each record in the data file contains the primary key (name of movie) in positions 4 through 31. The fourth line of the FILEIS call defines the primary key as 28 characters long, embedded, and located in word 0, character 3.

```
'KL', 28, 'EMK', 'YES', 'RKW', 0, 'RKP', 3,
```

The EMK field is set to YES for greater efficiency. Because the key is located within the record, AAM does not need to use additional space for the key value that is prefixed to each record when EMK=NO. Alternate keys are defined in the same manner regardless of whether the primary key is embedded in the record.

The working storage area from which AAM writes the record to the file is defined to be the eight words of REC.

```
'WSA', REC
```

TABLE 4-1. SUMMARY OF FORTRAN CALLS FOR AAM FILES WITH MIP

Call Name[†]	Function for Data File	Corresponding Operation on Index File (NDX=NO)[††]	Function for Index File (NDX=YES)[†††]
CLOSEM	Close data file	Close index file	Close index file only
DLTE	Delete record with identified primary key	Index is updated	None
FILEIS	Establish FIT	FIT for data and index files	FIT for index file only
GET	Read record identified by key at RKW, RKP, KL	None	Read keylist for alternate key value
GETN	Read next sequential record in data file	Read next record indicated in keylist	Read more of keylist
OPENM	Open data file	Open index file	Open index file only
PUT	Write record identified by primary key	Index is updated	None
REPLC	Replace record identified by primary key	Index is updated	None
REWND	Rewind data file	Rewind index file	Rewind index file
RMKDEF	None	Establish alternate key	None
SKIP	Skip records on IS or AK file (forward only)	None	Skip keys in keylist for IS, DA, or AK file (forward only)
STARTM	Establish key of reference for data file	Position index file to first value in keylist	Same
STOREF	Set value in FIT field	None	None
IFETCH	Retrieve FIT field value	None	None

[†]Associated parameters are shown in appendix B.

[††]Function performed through references to the FIT for the data file.

[†††]NDX must be reset to NO before data file is closed.

The last line of the FILEIS call controls the listing of error messages and file statistics. The EFC field is set to 3; therefore error messages, file statistics and processing notes are written to the error file. The CRMEP control statement is used to list the information from the error file. The DFC field is also set to 3; therefore error messages, file statistics and notes are written to the dayfile. The NOS DAYFILE control statement can be used in an interactive or batch environment to list the information from the dayfile.

'EFC', 3, 'DFC', 3

Because values are not otherwise specified, the system uses default values for these file characteristics: no padding for the data blocks and index blocks, no block checksums, no compression of records, no trivial error limit, and no limit on the number of records in the file. The primary key and alternate keys are sorted according to the default collating sequence, as no other sequence is specified.

Characteristics of the index file can be specified both through the FIT and through the RMKDEF calls that specify alternate key positions. The file name is indicated through the XN field of the FIT. In the absence of an XBS field setting, the size of the blocks in the index file is the same as the size of the blocks of the data file.

The program requires two RMKDEF calls to define two alternate keys. RMKDEF calls must be executed after the file is opened but before any records are written to the file. These calls have the same parameters as the RMKDEF directives of the MIPGEN utility. (Refer to figure 4-8).

The first RMKDEF directive defines the year-movie-made as an alternate key:

CALL RMKDEF (XISFIT, 0, 0, 2, 0, 'S', 'I')

A. NOS Control Statements

```
Job statement
USER statement
CHARGE statement
FTN5.
DEFINE,FLRECS
DEFINE,FLRXIP
FILE,FLRECS,MBL=3790
LGO.
CRMEP(LO,RU)
FLSTAT,FLRECS
FLSTAT,FLRXIP
```

NOS/BE Control Statements

```
Job statement
ACCOUNT statement
FTN5.
REQUEST,FLRECS,PF
REQUEST,FLRXIP,PF
FILE,FLRECS,MBL=3790
LGO.
CATALOG,FLRECS,ID=AAMUG
CATALOG,FLRXIP,ID=AAMUG
CRMEP,LO,RU
FLSTAT,FLRECS
FLSTAT,FLRXIP
```

B. Input File: MOVIES

Maximum Record Length 80 characters

```
38 THE ADVENTURES OF ROBIN HOOD4 FLYNN      RAINS      DEHAVILLANDRATHBONE
51 THE AFRICAN QUEEN           2 BOGART     HEPBURN
44 IN OUR TIME                 2 LUPINO     HENREID
40 THE PHILADELPHIA STORY      3 HEPBURN    GRANT      STEWART
42 CASABLANCA                  4 HENREID    BERGMAN    RAINS      LORRE
49 ADAM'S RIB                  2 TRACY      HEPBURN
41 JOAN OF PARIS               1 HENREID
38 BRINGING UP BABY            2 HEPBURN    GRANT
41 THE LITTLE FOXES            2 DAVIS      MARSHALL
42 NOW VOYAGER                 3 HENREID    DAVIS      RAINS
46 DECEPTION                   3 DAVIS      HENREID    RAINS
34 THE LITTLE MINISTER         1 HEPBURN
33 THE INVISIBLE MAN           1 RAINS
39 MR SMITH GOES TO WASHINGTON 2 RAINS      STEWART
39 ELIZABETH AND ESSEX         3 FLYNN      DAVIS      DEHAVILLAND
34 OF HUMAN BONDAGE            4 DAVIS      HOWARD     PARKER     HENREID
39 GOODBYE MR CHIPS            1 HENREID
49 ROPE OF SAND                3 LORRE      RAINS      HENREID
33 SHE DONE HIM WRONG          2 GRANT      WEST
44 BETWEEN TWO WORLDS          1 HENREID
44 ARSENIC AND OLD LACE        3 LORRE      GRANT      MASSEY
35 CAPTAIN BLOOD               2 FLYNN      DEHAVILLAND
44 THE CONSPIRATORS            1 HENREID
39 DODGE CITY                  2 FLYNN      DEHAVILLAND
43 ACTION IN THE NORTH ATLANTIC2 MASSEY     BOGART
40 NIGHT TRAIN TO MUNICH       1 HENREID
45 THE SPANISH MAIN            1 HENREID
46 DEVOTION                    1 HENREID
47 SONG OF LOVE                1 HENREID
48 THE SCAR                    1 HENREID
```

2-character year made — 28 character movie title — Col 34 — Actor's Name

Figure 4-10. CREATING an Indexed Sequential File With Multiple Keys (Sheet 1 of 2)

C. Source Program

```
      PROGRAM MAKXIS
      IMPLICIT INTEGER (A-Z)
C
C  **************************************************************
C  * THIS PROGRAM ILLUSTRATES THE CREATION OF AN IS FILE       *
C  * (FLRECS) FROM A SEQUENTIAL FILE (MOVIES).  ALL RECORDS    *
C  * ARE THEN LISTED.  TWO ALTERNATE KEYS ARE DEFINED (THE     *
C  * 2-CHARACTER YEAR-MOVIE-MADE AND THE ACTOR'S NAME).        *
C  * THE INDEX FILE IS NAMED FLRXIP.                           *
C  **************************************************************
C
      DIMENSION XISFIT(35), REC(8)
      CALL FILEIS (XISFIT, 'LFN', 'FLRECS', 'XN', 'FLRXIP',   )
     +           'ORG', 'NEW',                                 }—— Required
     +           'RT', 'F', 'FL', 80,                          }
     +           'KL', 28, 'EMK', 'YES', 'RKW', 0, 'RKP', 3,  )
     +           'WSA', REC,
     +           'EFC', 3, 'DFC', 3)◄——————————————————————————— Optional (but recommended)
C
C  OPEN FILE BEFORE ALTERNATE KEYS DEFINED
C
      CALL OPENM (XISFIT, 'NEW') ◄———————————————————————————— Open data file
      IF (IFETCH (XISFIT, 'ES') .NE. 0) GO TO 850
      CALL RMKDEF (XISFIT, 0, 0, 2, 0, 'S', 'I')          }
      CALL RMKDEF (XISFIT,3, 3, 11, 0, 'S', 'I', 11, 4)   }—————— Define 2 alternate keys
      OPEN (2, FILE = 'MOVIES')
   50 CONTINUE
      READ (2, '(8A10)', END = 800)  REC
      CALL PUT (XISFIT) ◄————————————————————————————————————— Write a record to FLRECS
      IF (IFETCH (XISFIT, 'ES') .NE. 0) THEN  )
        PRINT *, 'ES = ', ES                  }
        CALL CLOSEM (XISFIT)                  }——————————————— Check for non-zero error status
        STOP ' CRM ERROR RETURNED'            }
        END IF                                )
      PRINT 900, REC
      GO TO 50
  800 CALL CLOSEM (XISFIT) ◄——————————————————————————————————— Close data file
      STOP
  850 PRINT 902, IFETCH (XISFIT, 'ES')
      CALL CLOSEM (XISFIT)
      STOP 'CRM ERROR RETURNED'
  900 FORMAT (1X, 8A10)
  902 FORMAT ('ES = ', O3)
      END
```

Notice that the program does not (and must not) contain OPEN, PUT, CLOSE, etc. for the index file FLRXIP. Only XN=FLRXIP is required. CRM automatically opens the index file.

D. Output files:

FLRECS - data file (same as input shown in Part A.)
FLRXIP - index file

Figure 4-10. CREATING an Indexed Sequential File With Multiple Keys (Sheet 2 of 2)

This call specifies word 0, character 0 as the beginning location of the key. Key length is defined as two characters. The key format is character ('S'). The keylist structure is indexed ('I'). The key is not a repeating group key.

The second RMKDEF directive defines the actor's name as an alternate key:

```
     CALL RMKDEF (XISFIT, 3, 3, 11, 0, 'S', 'I', 11, 4)
```

This call specifies word 3, character 3 (column 34) as the beginning location of the key. Key length is defined as 11 characters. The key format is character. The keylist structure is indexed. The key is a repeating group key occurring four times (kc=4) at intervals of 11 characters (kg=11).

Create Data and Index Files

Execution of the program in figure 4-10 produces a data file FLRECS and an index file FLRXIP. The CALL PUT (XISFIT) statement causes records to be written to FLRECS. As records are written to the data file, the index file is automatically created also. All references in CRM calls identify the file by the array in which the FIT for the data file exists. The XN value is the only direct reference to the index file.

Obtain File Statistics

Because the EFC field is set to 3, error messages and file statistics/notes are written to the error file. The CRMEP control statement (shown in part A of figure 4-11) lists the information that was written to the error file during the creation of the file FLRECS.

File statistics from the FSTT for the data file FLRECS and the FSTT for the index file FLRXIP are shown in parts B and C of figure 4-11. The following statements call the FLSTAT utility, which reflects the file processing performed during the execution of program MAKXIS:

- FLSTAT,FLRECS
- FLSTAT,FLRXIP

Accessing the Multiple-Index File

The program SEEFILM (shown in figure 4-12) reads an indexed sequential file by alternate key values. The program sets specific values for the alternate key (actor's name) and lists all records with that value.

The data file FLRECS and its index file FLRXIP are those created through the program in figure 4-10. Since the data file exists, the file structure is determined from the FSTT.

Attach the Data and Index Files

Both the data file FLRECS and the index file FLRXIP must be attached, since the data file is to be accessed by alternate key. The index file is a mass storage permanent file that must be preserved between jobs. It must be made available to a job that updates the data file or reads the data file by alternate key.

Open the Data File

The OPENM (FIT, 'I-O') opens the data file FLRECS and its associated index file FLRXIP. The XN field must be set at the time the program executes a call to open the data file. Any valid method (FILE statement, FILExx call, or STOREF call) can be used to set XN.

Establish the Alternate Key to be Used

The choice of which alternate key index is to be read depends on the current value of the RKW, RKP, and KL fields. When a file is opened, these fields are set from FSTT information for the primary key. A change in values for RKW, RKP, and KL fields changes the index associated with the key of variable KA during subsequent STARTM operations.

In this example, records are to be read by specific actor name values. The RKW, RKP, and KL fields are immediately reset to the location of that alternate key by STOREF.

Alternatively, if you wanted to read the entire file (with GETN operations), you must set RKW, RKP, and KL, and then issue a REWND call.

Establish Search Criteria

The index position is at the alternate key index that satisfies the conditions established by the value in variable KA and the relationship indicated by the current setting of the REL field of the FIT. The REL field can be set to EQ (equal to), GT (greater than), or GE (greater than or equal to) to indicate the relationship between variable KA and the index position. Since REL is not specified in the example, EQ is used as the default value. The REL field is reset to EQ at the end of each operation. If REL is set to GT with a STOREF call, and then a CALL GET is executed, the GT value is used.

Position the Index

STARTM execution positions the index to the alternate key value identified in the call. It does not return a record to the working storage area.

```
     CALL STARTM (FIT)
```

Note that the first operation after new values of RKW, RKP, and KL are stored must position to that index. Positioning can be performed by any of the following calls:

- REWND

  Rewind to the beginning of the index indicated by the key fields.

- STARTM

  Position to the start of the keylist for the value specified in the location indicated by the KA field of the FIT.

- GET

  Position to the start of the keylist for the value specified in the location indicated by the KA field of the FIT. Return data record to the working storage area.

A. CRMEP Output

```
        CRMEP,LO,RU.
RM NOTE  1001 ON LFN FLRECS   FILE OPENED
RM NOTE  1002 ON LFN FLRECS   FILE CLOSED
RM NOTE  1003 ON LFN FLRECS   NUMBER OF INDEX LEVELS          0
RM NOTE  1004 ON LFN FLRECS   ***NUMBER OF GETS THIS OPEN     0
RM NOTE  1005 ON LFN FLRECS   ***NUMBER OF PUTS THIS OPEN    30        30 records were written on
RM NOTE  1006 ON LFN FLRECS   ***NUMBER OF REPLACES THIS OPEN     0    FLRECS
RM NOTE  1007 ON LFN FLRECS   ***NUMBER OF DELETES THIS OPEN      0
RM NOTE  1033 ON LFN FLRECS   ***NUMBER OF GET NEXTS THIS OPEN    0
RM NOTE  1010 ON LFN FLRECS       ***TOTAL DISKAREA***          512  WORDS
```

B. FLSTAT Output for Data File

```
STATISTICS FOR FILE FLRECS
ORGANIZATION------- IS                                          FLRECS has IS file organization
CREATION DATE------ 81/02/27.
DATE OF LAST CLOSE- 81/02/27.
TIME OF LAST CLOSE- 09.37.47.

FILE IS MIPPED
COLLATION IS STANDARD

PRIMARY KEY INFORMATION
  STARTING WORD POSITION ------ 0                               Information regarding the
  STARTING CHARACTER POSITION - 3                               28-character movie title
  TYPE -- COLLATED SYMBOLIC
  LENGTH IN CHARACTERS -------- 28

MAXIMUM RECORD SIZE  80
MINIMUM RECORD SIZE  80

TOTAL TRANSACTIONS
  NUMBER OF PUTS ------ 30                                      Thirty records written
  NUMBER OF GETS ------ 0
  NUMBER OF DELETES --- 0
  NUMBER OF REPLACES -- 0
  NUMBER OF GETNEXTS -- 0

CIO CALLS FOR FILE
  NUMBER OF READS ----- 0
  NUMBER OF WRITES ---- 2
  NUMBER OF RECALLS --- 0
  NUMBER OF REWRITES -- 1

NUMBER OF BLOCKS------- 1
NUMBER OF EMPTY BLOCKS- 0
BLOCK SIZE IN PRUS----- 6                                       Block size is 6 PRU because
NUMBER OF DATA RECORDS- 30                                      MBL was set to 3790 characters
                                                                ((6 x 640) - 50)
FILE LENGTH IN PRUS  8
NUMBER OF INDEX LEVELS IN USE   0                               Block size x No. Blocks + 2 FSTT
                                                                (6 x 1 + 2)
```

Figure 4-11. Obtaining File Statistics (Sheet 1 of 2)

C. FLSTAT Output for Index File

```
*STATISTICS FOR FILE FLRXIP
ORGANIZATION------- MIP ◄──────────────────── FLRXIP is an index file
CREATION DATE------ 81/02/27.
DATE OF LAST CLOSE- 81/02/27.
TIME OF LAST CLOSE- 09.37.47.

PRIMARY KEY INFORMATION
  STARTING WORD POSITION ------ 0
  STARTING CHARACTER POSITION - 3
  TYPE -- COLLATED SYMBOLIC
  LENGTH IN CHARACTERS -------- 28

ALTERNATE KEY INFORMATION
  CHARACTERS IN LARGEST KEY-- 11 ◄──────────── Actor's Name is 11 characters

PRIMARY KEY INFORMATION
  NUMBER OF UNIQUE -- 0
  NUMBER OF -IS-   -- 2 ◄──────────────────── Two alternate key indexes with
  NUMBER OF FIFO   -- 0                        Indexed Keylist structure

CIO CALLS FOR FILE
  NUMBER OF READS ----- 0
  NUMBER OF WRITES ---- 4
  NUMBER OF RECALLS --- 2
  NUMBER OF REWRITES -- 2

NUMBER OF BLOCKS------- 3
NUMBER OF EMPTY BLOCKS- 0
BLOCK SIZE IN PRUS----- 6 ◄──────────────── Same as data file (default)
NUMBER OF DATA RECORDS- 2 ◄──────────────── Two alternate key indexes

FILE LENGTH IN PRUS  20 ◄─────────────────── Block size x No. Blocks + 2 FSTT
MAX NUMBER OF LEVEL 2 INDEX LEVELS   1       (6 x 3 + 2)
MAX NUMBER OF LEVEL 3 INDEX LEVELS   1
```

Figure 4-11. Obtaining File Statistics (Sheet 2 of 2)

Deliver a Record to WSA

Program SEEFILM in figure 4-12 assumes interactive program execution. Part D shows typical output. The terminal operator is instructed to enter an 11-character alternate key value of an actor name. That value is then used in a call to STARTM, which positions to a keylist within an index. A GETN call is required to return that record to the working storage area.

During STARTM execution, the following FIT field is set:

RC	Total number of records with this same alternate key value.
KNE	Key-not-equal indicator

The program SEEFILM prints the RC value for each key value entered.

Without a check for the end of the keylist, successive GETN calls would read through all subsequent alternate key values in the index until the end of the index occurred. (An attempt to read past the end of the index is an error.) GETN can be used to read all of one index for a particular alternate key position. GETN cannot be used to read all indexes in the index file unless the index file is repositioned to the second index by a call to GET, STARTM, or REWND.

Program SEEFILM continues to ask the operator for an alternate key value and processes the resulting data records in a keylist until an interactive end-of-file is encountered.

ACTUAL KEY FILES WITH MIP

The programs that follow illustrate the creation of an actual key file and the reading of the file by alternate key. The file processing shown is, in general, applicable to all AAM files with MIP.

Creating A Multiple-Index File

The program MAKEIT, shown in figure 4-13, creates an actual key file AKRECS from a sequential file INRECS. The program establishes the FIT for the data file, specifies the index file name, defines two alternate keys, reads a file of input records, and writes the records to a new file. Any job that executes MAKEIT is responsible for preserving both the data file and the index file as permanent files.

A. NOS Control Statements

Job statement
USER statement
CHARGE statement
FTN5.
ATTACH,FLRECS
ATTACH,FLRXIP
LGO.
CRMEP,LO,RU

NOS/BE Control Statements

Job statement
ACCOUNT statement
FTN5.
ATTACH,FLRECS,ID=AAMUG
ATTACH,FLRXIP,ID=AAMUG
LGO.
CRMEP,LO,RU

B. Input Files:

FLRECS - data file (created in program MAKXIS, figure 4-10)
FLRXIP - index file

C. Source Program

```
      PROGRAM SEEFILM
C
C  *************************************************************
C  * THIS PROGRAM ILLUSTRATES ACCESSING AN INDEXED SEQUENTIAL *
C  * FILE (FLRECS) BY A SPECIFIC ALTERNATE KEY (ACTORS NAME). *
C  * A VALUE FOR ACTOR NAME IS ENTERED INTERACTIVELY AND ALL  *
C  * RECORDS WITH THAT VALUE ARE LISTED.                      *
C  *************************************************************
C
      IMPLICIT INTEGER (A-Z)
      DIMENSION FIT(35), WSA(8), ACTORKA(2)
      CALL FILEIS (FIT, 'LFN', 'FLRECS', 'XN', 'FLRXIP', 'ORG', 'NEW',<--Required
     +            'WSA', WSA,
     +            'EFC', 3, 'DFC', 3) <------------------------------Optional (but recommended)
C
C  SET ACTOR NAME AS THE ALTERNATE KEY TO BE USED
C  AND POSITION TO BEGINNING OF ACTOR INDEX
C
      CALL OPENM (FIT, 'I-O') <-------------------------------------Open FLRECS and FLRXIP
      IF (IFETCH (FIT, 'ES') .NE. 0) GO TO 300
      CALL STOREF (FIT, 'RKW', 3) )
      CALL STOREF (FIT, 'RKP', 3) }---------------------------------The first actors-name keylist it to be
      CALL STOREF (FIT, 'KL', 11) )                                 associated with subsequent STARTM
      CALL REWND (FIT) <----------------------                      operations
C                                             \
C  CLEAR AND ACCEPT ACTOR'S NAME                 -------------------Rewind FLRECS and FLRXIP and
C                                                                   postion to first value in the actor index
   50 DO 60 I = 1, 10
   60 WSA (I) = 10H
      PRINT *, ' ENTER ACTOR NAME OR EOF (RETURN KEY) '
      READ (*, '(A10, A2)', END = 200) ACTORKA <--------------------Accept a value for actors-name
      PRINT 900, ACTORKA
      CALL STARTM (FIT, ACTORKA) <----------------------------------Position index file to actor's name
C                                                                   entered
C  TEST FOR KEY NOT FOUND
C
      IF (IFETCH (FIT, 'ES') .EQ. O"506") THEN
        PRINT *, 'KEY NOT FOUND'
        GO TO 50
        END IF
C
C TEST FOR ANY OTHER CRM ERROR
C
      IF (IFETCH (FIT, 'ES') .NE. 0) GO TO 300
C
C  PRINT THE NUMBER OF RECORDS FOR SPECIFIED ACTOR NAME
C
```

Figure 4-12. Reading an Indexed Sequential File by Alternate Key (Sheet 1 of 2)

Source Program (Contd)

```
          PRINT *, ' ENTRIES IN KEYLIST ', IFETCH (FIT, 'RC')
      70  CALL GETN (FIT)◄──────────────── Next sequential record indicated in
          IF (IFETCH (FIT, 'ES') .NE. 0) GO TO 300    keylist is placed in WSA
          PRINT 902, (WSA (J), J = 1, 8)
          IF (IFETCH (FIT, 'FP') .EQ. O"100") GO TO 50
          IF (IFETCH (FIT, 'FP') .EQ. O"10") GO TO 50
          DO 80 I = 1, 10
      80  WSA (I) = 10H
          GO TO 70
     200  CALL CLOSEM (FIT)
          STOP 'NORMAL TERMINATION'
     300  PRINT 903, IFETCH (FIT, 'ES')
          CALL CLOSEM (FIT)
          STOP 'CRM ERROR RETURNED'
     900  FORMAT (4X,  'NAME ENTERED WAS  ',  A10, A2)
     902  FORMAT (5X, 8A10)
     903  FORMAT ('ES = ', O3)
          END
```

D. Typical Program Output

```
   ENTER ACTOR NAME OR EOF (RETURN KEY)
 ? grant
     NAME ENTERED WAS  GRANT
   ENTRIES IN KEYLIST 4
      44 ARSENIC AND OLD LACE         3 LORRE     GRANT      MASSEY
      38 BRINGING UP BABY             2 HEPBURN   GRANT
      33 SHE DONE HIM WRONG           2 GRANT     WEST
      40 THE PHILADELPHIA STORY       3 HEPBURN   GRANT      STEWART
   ENTER ACTOR NAME OR EOF (RETURN KEY)
 ? dehavilland
     NAME ENTERED WAS  DEHAVILLAND
   ENTRIES IN KEYLIST 4
      35 CAPTAIN BLOOD                2 FLYNN     DEHAVILLAND
      39 DODGE CITY                   2 FLYNN     DEHAVILLAND
      39 ELIZABETH AND ESSEX          3 FLYNN     DAVIS      DEHAVILLAND
      38 THE ADVENTURES OF ROBIN HOOD4 FLYNN      RAINS      DEHAVILLANDRATHBONE
   ENTER ACTOR NAME OR EOF (RETURN KEY)
 ? bad one
     NAME ENTERED WAS  BAD ONE
  KEY NOT FOUND
   ENTER ACTOR NAME OR EOF (RETURN KEY)
 ? massey
     NAME ENTERED WAS  MASSEY
   ENTRIES IN KEYLIST 2
      43 ACTION IN THE NORTH ATLANTIC2 MASSEY     BOGART
      44 ARSENIC AND OLD LACE         3 LORRE     GRANT      MASSEY
   ENTER ACTOR NAME OR EOF (RETURN KEY)
 ? bergman
     NAME ENTERED WAS  BERGMAN
   ENTRIES IN KEYLIST 1
      42 CASABLANCA                   4 HENREID   BERGMAN    RAINS      LORRE
   ENTER ACTOR NAME OR EOF (RETURN KEY)
 ? west
     NAME ENTERED WAS  WEST
   ENTRIES IN KEYLIST 1
      33 SHE DONE HIM WRONG           2 GRANT     WEST
   ENTER ACTOR NAME OR EOF (RETURN KEY)
 ? davis
     NAME ENTERED WAS  DAVIS
   ENTRIES IN KEYLIST 5
      46 DECEPTION                    3 DAVIS     HENREID    RAINS
      39 ELIZABETH AND ESSEX          3 FLYNN     DAVIS      DEHAVILLAND
      42 NOW VOYAGER                  3 HENREID   DAVIS      RAINS
      34 OF HUMAN BONDAGE             4 DAVIS     HOWARD     PARKER     HENREID
      41 THE LITTLE FOXES             2 DAVIS     MARSHALL
   ENTER ACTOR NAME OR EOF (RETURN KEY)
 ?
       .043 CP SECONDS EXECUTION TIME.
```

Figure 4-12. Reading an Indexed Sequential File by Alternate Key (Sheet 2 of 2)

A. NOS Control Statements

```
Job statement
USER statement
CHARGE statement
DEFINE,AKRECS
DEFINE,MIPAK
LGO.
CRMEP(LO,RU)
FLSTAT,AKRECS
FLSTAT,MIPAK
```

NOS/BE Control Statements

```
Job statement
ACCOUNT statement
FTN5.
REQUEST,AKRECS,PF
REQUEST,MIPAK,PF
LGO.
CATALOG,AKRECS,ID=AAMUG
CATALOG,MIPAK,ID=AAMUG
CRMEP,LO,RU
FLSTAT,AKRECS
FLSTAT,MIPAK
```

B. Input File: INRECS

```
         Record Length 40 Characters
CAPT   NEMO     WRITTEN AS RECORD 01.
SUSAN  JONES    WRITTEN AS RECORD 02.
CHARLESKANE     WRITTEN AS REOCRD 03.
CANDY  KANE     WRITTEN AS RECORD 04.
DAVY   JONES    WRITTEN AS RECORD 05.
MRS.   JONES    WRITTEN AS RECORD 06.
ROSE   BUD      WRITTEN AS RECORD 07.
BIG    CASTLE   WRITTEN AS RECORD 08.
MRS.   NEMO     WRITTEN AS RECORD 09.
SUB    MARINE   WRITTEN AS RECORD 10.
XANA   DU       WRITTEN AS RECORD 11.
SUB    ROSA     WRITTEN AS RECORD 12.
TOUGH  SLEDDING WRITTEN AS RECORD 13.
SUB    STANDARD WRITTEN AS RECORD 14.
MRS.   KANE     WRITTEN AS RECORD 15.
SUB    STITUTE  WRITTEN AS RECORD 16.
FON    DU       WRITTEN AS RECORD 17.

7 character   8 character
First name    Second name
```

C. Source Program

```
      PROGRAM MAKEIT
      IMPLICIT INTEGER (A-Z)
      DIMENSION AKFIT (35), AKWSA (4)
      COMMON /AKLAB/ KEY ◄──────────── Key must be in common area for
C                                      OPT=2 processing
C  ****************************************************************
C  * THIS PROGRAM CREATES AN ACTUAL KEY FILE (AKRECS) FROM       *
C  * A SEQUENTIAL FILE (INRECS).  ALL RECORDS ARE THEN LISTED.   *
C  * TWO ALTERNATE KEYS ARE DEFINED (FIRST NAME AND LAST NAME).  *
C  * THE INDEX FILE IS NAMED MIPAK.                              *

C  ****************************************************************
C
      CALL FILEAK (AKFIT, 'LFN', 'AKRECS', 'XN', 'MIPAK', 'ORG', 'NEW',
     +           'RT', 'F', 'FL', 40, 'MBL', 400, 'RB', 10,           ── Required
     +           'KA', KEY, 'KL', 3, 'EMK', 'NO',
     +           'WSA', AKWSA,
     +           'EFC', 3, 'DFC', 3) ◄──────────── Optional (but recommended)
```

Figure 4-13. Creating an Actual Key File With Multiple Keys (Sheet 1 of 2)

Source Program (Contd)

```
C   OPEN FILE BEFORE ALTERNATE KEYS ARE DEFINED
C
      CALL OPENM (AKFIT,'NEW')
      IF (IFETCH (AKFIT, 'ES') .NE. 0) GO TO 300
      CALL RMKDEF (AKFIT, 0, 0, 7, 0, 'S', 'I') }
      CALL RMKDEF (AKFIT, 0, 7, 8, 0, 'S', 'F') }---------- Define 2 alternate keys
      OPEN (2, FILE = 'INRECS')
  100 KEY = 0 <-------------------------------------------- CRM is to generate primary keys
      READ (2, '(4A10)', END = 200) AKWSA
      CALL PUT (AKFIT) <----------------------------------- Write a record to AKRECS
      IF (IFETCH (AKFIT, 'ES') .NE. 0) GO TO 300
      GO TO 100
 200  CALL CLOSEM (AKFIT)
C
C   REOPEN FILE TO SEE WHAT IS THERE
C
      CALL OPENM (AKFIT,'INPUT')
      IF (IFETCH (AKFIT, 'ES') .NE. 0) GO TO 300
      CALL REWND(AKFIT)
  250 CALL GETN (AKFIT)
      IF (IFETCH (AKFIT, 'ES') .NE. 0) THEN   }
        PRINT  902, IFETCH (AKFIT, 'ES')      }
        PRINT 900, (AKWSA (J), J = 1, 4), KEY }---------- Check for non-zero error status
        CALL CLOSEM (AKFIT)                   }
        STOP 'CRM ERROR RETURNED'
        END IF
      IF (IFETCH (AKFIT, 'FP') .NE. O"20") THEN }
        CALL CLOSEM (AKFIT)                     }
        STOP                                    }-------- Check for unsuccessful return of
        END IF                                  }         record to WSA
      PRINT 901, (AKWSA (J), J= 1, 4)
      GO TO 250
 300  PRINT 902, IFETCH (AKFIT, 'ES')
      CALL CLOSEM (AKFIT)
      STOP 'CRM ERROR RETURNED'
  900 FORMAT ('RECORD IS ', 4A10, 'KEY IS ', O10)
  901 FORMAT (5X, 4A10)
  902 FORMAT (' ES = ', O3)
      END
```

D. Output Files:

AKRECS - data file (same as input file)
MIPAK - index file

Figure 4-13. Creating an Actual Key File With Multiple Keys (Sheet 2 of 2)

Define File Names

The actual key file to be created is given the logical file name AKRECS. Its FIT is to be constructed in the array AKFIT, which is dimensioned to 35 words as required for all FITs.

The index file to be created is given the logical file name MIPAK. You must specify the XN field when creating a file with alternate keys. Notice that XN is the only direct reference you need for the index file. The file is created automatically by MIP.

The first line of the FILEAK call specifies the FIT name (AKFIT), data file name (AKRECS), the index file name (MIPAK), and extended AAM file organizatin (NEW).

```
CALL FILEAK (AKFIT, 'LFN', 'AKRECS', 'XN',
             'MIPAK', 'ORG', 'NEW',
```

Set Additional FIT Fields

Input data records exist on a file INRECS (shown in part B of figure 4-13). The records have a fixed length of 40 characters. The file has CZ format (Z-type records and C-type blocking). Any other structure would require a FILE statement specifying the BT and RT fields.

The second line of the FILEAK call specifies the record structure for the output file. If MBL and RB are not specified, default values would be calculated by AAM.

```
'RT', 'F', 'FL', 40, 'MBL', 400, 'RB', 10,
```

The third line of the FILEAK call defines a nonembedded primary key with the name KEY and length 3. Since the primary key value is set to zero, AAM generates the key values when the file is created.

```
'KA', KEY, 'KL', 3, 'EMK', 'NO'
```

The working storage area from which AAM writes the record to the file is defined in the FILEAK call to be the four words of AKWSA.

'WSA', AKWSA

Characteristics of the index file can be specified through the FIT and through the RMKDEF calls that specify alternate key positions. The file name is indicated through the XN field of the FIT. In the absence of an XBS field setting, the size of the blocks in the index file and in the data file is the same.

The program shown in figure 4-13 requires two RMKDEF calls to define two alternate keys. RMKDEF calls must be executed after the file is opened but before any records are written to the file. These calls have the same parameters as the RMKDEF directives of the MIPGEN utility (Refer to figure 4-8).

The first RMKDEF directive defines the first name in the record as an alternate key:

```
CALL RMKDEF (AKFIT, 0, 0, 7, 0, 'S', 'I')
```

This call specifies word 0, character 0 as the beginning location of the key. Key length is defined as seven characters. The key format is character ('S'). The keylist structure is indexed ('I').

The second RMKDEF directive defines the second name in the record as an alternate key:

```
CALL RMKDEF (AKFIT, 0, 7, 8, 0, 'S', 'F')
```

This call specifies word 0, character 7 as the beginning location of the key. Key length is defined as eight characters. The key format is character ('S'). The keylist structure is first-in-first-out ('F').

Create Data and Index Files

Execution of the program in figure 4-13 produces a data file AKRECS and an index file MIPAK. The CALL PUT (AKFIT) statement causes records to be written to AKRECS. As records are written to the data file, the index file is automatically created. All references in CRM calls identify the file by the array in which the FIT for the data file exists. The XN value is the only direct reference to the index file.

Obtain File Statistics

File statistics from the FSTT for the data file AKRECS and the FSTT for the index file MIPAK are shown in figure 4-14. The following statements call the FLSTAT utility, which reflects the file processing performed during the execution of the program MAKEIT.

- FLSTAT,AKRECS
- FLSTAT,MIPAK

Accessing the Multiple-Index File

The program CINDX (shown in figure 4-15) reads an actual key file first by its primary key and then by each of its alternate keys. Three passes are made through the file and all records are listed each time.

The data file AKRECS and its index file MIPAK are those created through the program in figure 4-13. Since the data file exists, the file structure can be determined from the FSTT.

Attach the Data and Index Files

Both the data file AKRECS and the index file MIPAK must be attached, since the data file is to be accessed by alternate key. The index file is a mass storage permanent file that must be preserved between jobs. It must be made available to a job that updates the data file or reads the data file by alternate key.

Reading The File by Primary Key

Since AAM initially sets the RKW, RKP, and KL values to those of the primary key (and because no GET, STARTM, or REWND has intervened since the OPENM), the first GETN call in the program (figure 4-15) reads records by primary key.

Output from the first pass through the file is shown in part D of the figure. Notice that the value of the FP field is 20_8 after each GETN. This value indicates that a record has been successfully returned to the working storage area. When FP equals 100_8, the end-of-information on AKRECS has been reached.

Reading The File by Alternate Keys

Resetting RKW, RKP, and KL values, establishes first name as the alternate key to be used. A REWND is necessary to record this change of key and to position the index file MIPAK to its beginning. The first entry in the first-name index becomes the key of reference. Records are then read sequentially (as positioned in the index) by first name with the GETN call.

Part E of figure 4-15 shows the order in which records exist in the first-name key index. Notice that the value of the FP field after each GETN is either 10_8 or 20_8. A value of 20_8 indicates that a record has been sucessfully read and returned to the working storage area. A value of 10_8 indicates that, in addition to a successful read, the next first name value is different from the current first name value (that is, the end of the keylist has been reached).

For duplicate first name values (such as MRS.), entries appear in sorted order according to primary key value (record number). This is due to the I value, indicating indexed, of the ks parameter on the RMKDEF call when the file was created.

The second name is then established as the key to be used next by setting new RKW, RKP, and KL values and issuing a REWND.

Part F of figure 4-15 shows the order in which records exist in the second name key index. For duplicate key values, entries appear in the order in which data records were written to the file. This is due to the F value (indicating first-in-first-out) of the ks parameter of the RMKDEF call when the file was created.

```
/FLSTAT,AKRECS

 STATISTICS FOR FILE AKRECS
 ORGANIZATION------- AK
 CREATION DATE------ 81/03/06.
 DATE OF LAST CLOSE- 81/03/06.
 TIME OF LAST CLOSE- 11.07.31.

 FILE IS MIPPED

 PRIMARY KEY INFORMATION
   STARTING WORD POSITION ------ 0
   STARTING CHARACTER POSITION - 10
   LENGTH IN CHARACTERS -------- 3

 MAXIMUM RECORD SIZE  40
 MINIMUM RECORD SIZE  40

 TOTAL TRANSACTIONS
   NUMBER OF PUTS ------ 17
   NUMBER OF GETS ------ 0
   NUMBER OF DELETES --- 0
   NUMBER OF REPLACES -- 0
   NUMBER OF GETNEXTS -- 17

 CIO CALLS FOR FILE
   NUMBER OF READS ----- 3
   NUMBER OF WRITES ---- 3
   NUMBER OF RECALLS --- 0
   NUMBER OF REWRITES -- 2

 NUMBER OF BLOCKS------- 2
 NUMBER OF EMPTY BLOCKS- 0
 BLOCK SIZE IN PRUS----- 1
 NUMBER OF DATA RECORDS- 17
 NUMBER OF RECORDS PER BLOCK ----- 10
 NUMBER OF OVERFLOW RECORDS ------ 0

 FILE LENGTH IN PRUS  4
```

```
/FLSTAT,MIPAK

 STATISTICS FOR FILE MIPAK
 ORGANIZATION------- MIP
 CREATION DATE------ 81/03/06.
 DATE OF LAST CLOSE- 81/03/06.
 TIME OF LAST CLOSE- 11.07.31.

 PRIMARY KEY INFORMATION
   KEY IS NOT EMBEDDED
   TYPE -- UNSIGNED
   LENGTH IN CHARACTERS -------- 3

 ALTERNATE KEY INFORMATION
   CHARACTERS IN LARGEST KEY-- 8

 PRIMARY KEY INFORMATION
   NUMBER OF UNIQUE -- 0
   NUMBER OF -IS-   -- 1
   NUMBER OF FIFO   -- 1

 CIO CALLS FOR FILE
   NUMBER OF READS ----- 1
   NUMBER OF WRITES ---- 4
   NUMBER OF RECALLS --- 2
   NUMBER OF REWRITES -- 3

 NUMBER OF BLOCKS------- 3
 NUMBER OF EMPTY BLOCKS- 0
 BLOCK SIZE IN PRUS----- 1
 NUMBER OF DATA RECORDS- 2

 FILE LENGTH IN PRUS  5
 MAX NUMBER OF LEVEL 2 INDEX LEVELS  1
 MAX NUMBER OF LEVEL 3 INDEX LEVELS  1
```

Figure 4-14. Obtaining File Statistics

Reading the File by Relative Alternate Key Value

In program ALTREL (shown in figure 4-16) interactive program execution is assumed. You are instructed to enter a 7-character alternate key value (first name). That value is then used in a STARTM call which positions the index file for the subsequent GETN call.

Search criteria are established by setting the REL field values to EQ, GT, or GE. When REL is set to EQ (the default), all read operations access records whose alternate key value is equal to the value at the location specified by KA. Also, if the specified value cannot be found in the alternate key index, an error results.

When REL is set to a value other than EQ, however, the record referenced by a STARTM call is one that satisfies the condition established by REL. Any value in the location established by KA that does not exactly correspond to an alternate key value normally does not produce an error. Rather, the value in the key location and the REL field establish a relationship, and the first record in the index that satisfies the relationship is returned to the working storage area. A key-not-found error exists only if no record in the index satisfies the relationship (and not simply the value) indicated through KA.

The REL field, unlike most fields, does not retain its value across file operations. It is reset to EQ at the end of each operation.

Program ALTREL continues to ask for an alternate key value and an REL value until an interactive end-of-file is encountered.

A. NOS Control Statements

```
Job statement
USER statement
CHARGE statement
FTN5.
ATTACH,AKRECS
ATTACH,MIPAK
LGO.
CRMEP,LO,RU
```

NOS/BE Control Statements

```
Job statement
ACCOUNT statement
FTN5.
ATTACH,AKRECS,ID=AAMUG
ATTACH,MIPAK,ID=AAMUG
LGO.
CRMEP,LO,RU
```

B. Input Files:

AKRECS - data file (created in program MAKEIT, figure 4-13)
MIPAK - index file

C. Source Program

```
      PROGRAM CINDX
C
C  ***************************************************************
C  * THIS PROGRAM ILLUSTRATES ACCESSING AN ACTUAL KEY FILE       *
C  * (AKRECS) BY EACH OF ITS THREE KEYS:                         *
C   *          PRIMARY KEY = CRM-ASSIGNED RECORD NUMBER          *
C  *           1ST ALTERNATE KEY = FIRST NAME                    *
C  *           2ND ALTERNATE KEY = LAST NAME                     *
C  ***************************************************************
C
      IMPLICIT INTEGER (A-Z)
      DIMENSION FIT(35), WSA(5)
      CALL FILEAK (FIT, 'LFN', 'AKRECS', 'XN', 'MIPAK', 'ORG', 'NEW',
     +           'WSA', WSA,
     +           'KA', KEY,
     +           'EFC', 3, 'DFC', 3)
      CALL OPENM (FIT, 'INPUT')
      IF (IFETCH (FIT, 'ES') .NE. 0) GO TO 500
      PRINT *, ' +++++++ RECORDS FROM PRIMARY KEY ACCESS +++++++ '
 100  CALL GETN (FIT)
      IF (IFETCH (FIT, 'ES') .NE. 0) THEN
        PRINT 902, IFETCH (FIT, 'ES')
        CALL CLOSEM (FIT)
        STOP 'CRM ERROR RETURNED'
        END IF
      IF (IFETCH (FIT, 'FP') .EQ. O"100") GO TO 200
      PRINT 901, (WSA (J), J = 1, 4), IFETCH (FIT, 'FP')
      GO TO 100
 200  CONTINUE
C
C   ESTABLISH FIRST NAME AS THE KEY TO BE USED
C
      CALL STOREF (FIT, 'RKW', 0)
      CALL STOREF (FIT, 'RKP', 0)
      CALL STOREF (FIT, 'KL', 7)
      PRINT *, ' +++++++ RECORDS FROM FIRST NAME KEY ACCESS +++++++'
      CALL REWND (FIT)
 250  CALL GETN (FIT)
      IF (IFETCH (FIT, 'ES') .NE. 0) THEN
        PRINT 902, IFETCH (FIT, 'ES')
        CALL CLOSEM (FIT)
        STOP 'CRM ERROR RETURNED'
        END IF
      IF (IFETCH (FIT, 'FP') .EQ. O"100") GO TO 300
      PRINT 901, (WSA (J), J = 1, 4), IFETCH (FIT, 'FP')
      GO TO 250
 300  CONTINUE
```

Figure 4-15. Reading an Actual Key File by Primary and Alternate Keys (Sheet 1 of 3)

Source Program (Contd)

```
C   ESTABLISH SECOND NAME AS THE KEY TO BE USED
C
      CALL STOREF (FIT, 'RKW', 0)
      CALL STOREF (FIT, 'RKP', 7)
      CALL STOREF (FIT, 'KL', 8)
      PRINT *, ' +++++++ RECORDS FROM SECOND NAME KEY ACCESS +++++++'
      CALL REWND (FIT)
350   CALL GETN (FIT)
      IF (IFETCH (FIT, 'ES') .NE. 0) THEN
        PRINT 902, IFETCH (FIT, 'ES')
        CALL CLOSEM (FIT)
        STOP 'CRM ERROR RETURNED'
        END IF
      IF (IFETCH (FIT, 'FP') .EQ. O"100") GO TO 400
      PRINT 901, (WSA (M), M = 1, 4), IFETCH (FIT, 'FP')
      GO TO 350
400   CALL CLOSEM (FIT)
      STOP
901   FORMAT (' RECORD = ', 4A10, '   FP = ', O3)
500   PRINT 902, ES
      CALL CLOSEM (FIT)
      STOP 'CRM ERROR RETURNED'
902   FORMAT ('ES = ', O3)
      END
```

D. Output

```
 +++++++ RECORDS FROM PRIMARY KEY ACCESS +++++++
RECORD = CAPT      NEMO      WRITTEN AS RECORD 01.     FP = 020
RECORD = SUSAN     JONES     WRITTEN AS RECORD 02.     FP = 020
RECORD = CHARLESKANE         WRITTEN AS REOCRD 03.     FP = 020
RECORD = CANDY     KANE      WRITTEN AS RECORD 04.     FP = 020
RECORD = DAVY      JONES     WRITTEN AS RECORD 05.     FP = 020
RECORD = MRS.      JONES     WRITTEN AS RECORD 06.     FP = 020
RECORD = ROSE      BUD       WRITTEN AS RECORD 07.     FP = 020
RECORD = BIG       CASTLE    WRITTEN AS RECORD 08.     FP = 020
RECORD = MRS.      NEMO      WRITTEN AS RECORD 09.     FP = 020
RECORD = SUB       MARINE    WRITTEN AS RECORD 10.     FP = 020
RECORD = XANA      DU        WRITTEN AS RECORD 11.     FP = 020
RECORD = SUB       ROSA      WRITTEN AS RECORD 12.     FP = 020
RECORD = TOUGH     SLEDDING  WRITTEN AS RECORD 13.     FP = 020
RECORD = SUB       STANDARD  WRITTEN AS RECORD 14.     FP = 020
RECORD = MRS.      KANE      WRITTEN AS RECORD 15.     FP = 020
RECORD = SUB       STITUTE   WRITTEN AS RECORD 16.     FP = 020
RECORD = FON       DU        WRITTEN AS RECORD 17.     FP = 020
```

020 indicates a record successfully returned to WSA

E. Output (Contd)

```
 +++++++ RECORDS FROM FIRST NAME KEY ACCESS +++++++
RECORD = BIG       CASTLE    WRITTEN AS RECORD 08.     FP = 010
RECORD = CANDY     KANE      WRITTEN AS RECORD 04.     FP = 010
RECORD = CAPT      NEMO      WRITTEN AS RECORD 01.     FP = 010
RECORD = CHARLESKANE         WRITTEN AS REOCRD 03.     FP = 010
RECORD = DAVY      JONES     WRITTEN AS RECORD 05.     FP = 010
RECORD = FON       DU        WRITTEN AS RECORD 17.     FP = 010
RECORD = MRS.      JONES     WRITTEN AS RECORD 06.     FP = 020
RECORD = MRS.      NEMO      WRITTEN AS RECORD 09.     FP = 020
RECORD = MRS.      KANE      WRITTEN AS RECORD 15.     FP = 010
RECORD = ROSE      BUD       WRITTEN AS RECORD 07.     FP = 010
RECORD = SUB       MARINE    WRITTEN AS RECORD 10.     FP = 020
RECORD = SUB       ROSA      WRITTEN AS RECORD 12.     FP = 020
RECORD = SUB       STANDARD  WRITTEN AS RECORD 14.     FP = 020
RECORD = SUB       STITUTE   WRITTEN AS RECORD 16.     FP = 010
RECORD = SUSAN     JONES     WRITTEN AS RECORD 02.     FP = 010
RECORD = TOUGH     SLEDDING  WRITTEN AS RECORD 13.     FP = 010
RECORD = XANA      DU        WRITTEN AS RECORD 11.     FP = 010
```

010 indicates a successful read and the last record in keylist

Notice that these records are not the last in the keylist

Figure 4-15. Reading an Actual Key File by Primary and Alternate Keys (Sheet 2 of 3)

F. Output (Contd)

```
+++++++ RECORDS FROM SECOND NAME KEY ACCESS +++++++
RECORD = ROSE    BUD      WRITTEN AS RECORD 07.      FP = 010
RECORD = BIG     CASTLE   WRITTEN AS RECORD 08.      FP = 010
RECORD = XANA    DU       WRITTEN AS RECORD 11.      FP = 020
RECORD = FON     DU       WRITTEN AS RECORD 17.      FP = 010
RECORD = SUSAN   JONES    WRITTEN AS RECORD 02.      FP = 020
RECORD = DAVY    JONES    WRITTEN AS RECORD 05.      FP = 020
RECORD = MRS.    JONES    WRITTEN AS RECORD 06.      FP = 010     Not last record in keylist
RECORD = CHARLESKANE      WRITTEN AS REOCRD 03.      FP = 020
RECORD = CANDY   KANE     WRITTEN AS RECORD 04.      FP = 020
RECORD = MRS.    KANE     WRITTEN AS RECORD 15.      FP = 010
RECORD = SUB     MARINE   WRITTEN AS RECORD 10.      FP = 010
RECORD = CAPT    NEMO     WRITTEN AS RECORD 01.      FP = 020
RECORD = MRS.    NEMO     WRITTEN AS RECORD 09.      FP = 010
RECORD = SUB     ROSA     WRITTEN AS RECORD 12.      FP = 010
RECORD = TOUGH   SLEDDING WRITTEN AS RECORD 13.      FP = 010
RECORD = SUB     STANDARD WRITTEN AS RECORD 14.      FP = 010
RECORD = SUB     STITUTE  WRITTEN AS RECORD 16.      FP = 010
```

Figure 4-15. Reading an Actual Key File by Primary and Alternate Keys (Sheet 3 of 3)

INDEX-FILE-ONLY OPERATIONS

As a FORTRAN or COMPASS programmer, you can access the index file independent of the data file. Operations on the data file return records to the working storage area. Index-file-only operations, on the other hand, ignore the data file and return only index information to the working storage area. (During data file operations, the index file might be updated. Although this is an internal operation on the index file, it is not included when the term index file operation is used in this section.)

Two types of information can be obtained from the index file.

- Count Information

  Count information includes the following:

  The number of records with a particular alternate key value.

  The number of records within a range of alternate key values.

- Primary Key Values

  Either of the following can be returned to the working storage area:

  The primary keys of records having a particular alternate key value.

  The primary keys of records having a range of alternate key values.

For example (referring to the data file shown in figure 4-1), you could obtain the following type of information using only index file operations:

- The number of people in New York
- The number of people between the ages of 21 and 65
- A list of names of all people in Seattle
- A list of names of all people between the ages of 60 and 70

NDX SETTINGS

The NDX field of the data file FIT controls whether the data file or the index file is accessed in response to a CRM call, as follows:

- When NDX is set to NO, all information returned to the program references the data file. This is the normal mode of operation.
- When NDX is set to YES, the program references only the index file.

Data file and index file operations can be intermixed by changing the value in the NDX field between CRM calls, if NDX was set to NO when OPENM was issued.

You can set the NDX field of the FIT through a call to STOREF, through a FILExx call, or through a FILE control statement.

If NDX is set to YES before a call to OPENM, the system opens only the index file. In this instance, the data file need not be attached to the job; and if it is attached, it is not opened. NDX must not be switched to NO after such an OPENM. If NDX=NO at the open call, the index file must be present, and the system opens both files. Subsequent setting of NDX=YES restricts access to the index file only.

Whenever the data file has been opened, NDX should be set to NO before the call to CLOSEM. In the absence of NDX=NO, only the index file is closed normally.

A. NOS Control Statements

NOS Control Statements	NOS/BE Control Statements
Job statement	Job statement
USER statement	ACCOUNT statement
CHARGE statement	FTN5.
FTN5.	ATTACH,AKRECS,ID=AAMUG
ATTACH,AKRECS	ATTACH,MIPAK,ID=AAMUG
ATTACH,MIPAK	LGO.
LGO.	CRMEP,LO,RU
CRMEP,LO,RU	

B. Input Files

AKRECS - data file (created in program MAKEIT, figure 4-13)
MIPAK - index file

C. Source Program

```
      PROGRAM ALTREL
C
C  *****************************************************************
C  * THIS PROGRAM ILLUSTRATES ACCESSING AN AK FILE (AKRECS)        *
C  * BY A SPECIFIC KEY (FIRST NAME).  A VALUE FOR THE FIRST        *
C  * NAME IS ENTERED INTERACTIVELY; A REL VALUE OF EQ, GE, OR GT*
C  * IS ENTERED.  ALL RECORDS SATISFYING THE RELATIONSHIP          *
C  * ARE LISTED.                                                   *
C  *****************************************************************
C
      IMPLICIT INTEGER (A-Z)
      DIMENSION FIT(35), WSA(5)
      CALL FILEAK (FIT, 'LFN', 'AKRECS', 'XN', 'MIPAK', 'ORG', 'NEW',
     +            'KA', KEY,
     +            'WSA', WSA,
     +            'EFC', 3, 'DFC', 3)
      CALL OPENM (FIT, 'INPUT')
      IF (IFETCH (FIT, 'ES') .NE. 0) GO TO 850
C
C    ESTABLISH FIRST NAME AS THE KEY TO BE USED
C
      CALL STOREF (FIT, 'RKW', 0)}
      CALL STOREF (FIT, 'RKP', 0)}------------------------  First name index will be used for
      CALL STOREF (FIT, 'KL', 7) }                           subsequent GETN
  100 PRINT *, ' ENTER 7 CHARACTER ALTERNATE KEY'
      READ (*, '(A7)', END = 800) KEY <------------------  Accept a first name value
      PRINT *, ' ENTER REL VALUE EQ, GE OR GT'
      READ (*, '(A2)') REL <-----------------------------  Accept a REL value
      PRINT 903, KEY, REL
C
C    SET RELATION AND KEY VALUE
C
      CALL STOREF (FIT, 'REL', REL) <--------------------  Set REL field
      CALL STARTM (FIT) <--------------------------------  Position index to the first name
C                                                          value entered
C    TEST FOR KEY NOT FOUND
C
      IF (IFETCH (FIT, 'ES') .NE. O"506") THEN
        PRINT *, ' KEY NOT FOUND'
        GO TO 100
        END IF
C
C    TEST FOR ANY OTHER CRM ERROR
C
      IF (IFETCH (FIT, 'ES') .NE. 0) GO TO 850
```

Figure 4-16. Reading an Actual Key File by Relative Alternate Key Value (Sheet 1 of 2)

Source Program (Contd)

```
C   PRINT THE NUMBER OF RECORDS SATISFYING THE RELATIONSHIP
C
      RC = IFETCH (FIT, 'RC')
      PRINT *, ' RC FROM STARTM IS ', RC
      DO 200 I = 1, RC
      CALL GETN (FIT)◄──────────────────────────────────── Return a record
      IF (IFETCH (FIT, 'ES') .NE. 0) GO TO 850
      IF (IFETCH (FIT, 'FP') .EQ. O"100") GO TO 100 ◄─────── Test for end-of-information
      PRINT 901, (WSA (J), J = 1, 4)
  200 CONTINUE
      GO TO 100
  800 CONTINUE
C
C            SET ANOTHER ALTERNATE KEY  IN RKW, RKP, AND KL
C            POSITION INDEX BY GET, REWND, OR STARTM
C            PROCESS AS DESIRED
C
      CALL CLOSEM (FIT)
      STOP 'NORMAL TERMINATION'
 850  PRINT 902, IFETCH (FIT, 'ES')
      CALL CLOSEM (FIT)
      STOP 'CRM ERROR RETURNED'
  901 FORMAT (5X, 4A10)
  902 FORMAT (' ES = ', O3)
  903 FORMAT (' KEY ENTERED WAS   ', A7, ' REL WAS  ', A2)
      END
```

D. Typical Output From Execution

```
  ENTER 7 CHARACTER ALTERNATE KEY
? capt
  ENTER REL VALUE EQ, GE OR GT
? eq
 KEY ENTERED WAS   CAPT    REL WAS  EQ
  RC FROM STARTM IS 1
     CAPT   NEMO     WRITTEN AS RECORD 01.
  ENTER 7 CHARACTER ALTERNATE KEY
? looooo
  ENTER REL VALUE EQ, GE OR GT
? ge
 KEY ENTERED WAS   LOOOOOO REL WAS  GE
  RC FROM STARTM IS 3
     MRS.   JONES    WRITTEN AS RECORD 06.
     MRS.   NEMO     WRITTEN AS RECORD 09.
     MRS.   KANE     WRITTEN AS RECORD 15.
  ENTER 7 CHARACTER ALTERNATE KEY
? tough
  ENTER REL VALUE EQ, GE OR GT
? gt
 KEY ENTERED WAS   TOUGH   REL WAS  GT
  RC FROM STARTM IS 1
     XANA   DU       WRITTEN AS RECORD 11.
  ENTER 7 CHARACTER ALTERNATE KEY
? eopopop
  ENTER REL VALUE EQ, GE OR GT
? eq
 KEY ENTERED WAS   EOPOPOP REL WAS  EQ
  KEY NOT FOUND
  ENTER 7 CHARACTER ALTERNATE KEY
? fon
  ENTER REL VALUE EQ, GE OR GT
? ge
 KEY ENTERED WAS   FON     REL WAS  GE
  RC FROM STARTM IS 1
     FON    DU       WRITTEN AS RECORD 17.
  ENTER 7 CHARACTER ALTERNATE KEY
?
      .042 CP SECONDS EXECUTION TIME.
/
```

Figure 4-16. Reading an Actual Key File by Relative Alternate Key Value (Sheet 2 of 2)

When the NDX field is set to YES and only the index file is accessed:

- Keylists, rather than data records, are returned to the working storage area.
- File updating operations are not possible, since the data file is not necessarily attached to the job.
- The FP field has these meanings:

0	The working storage area has been filled, but the end of the keylist has not been encountered.
10_8	The entire keylist has been delivered to the working storage area.
100_8	The end of the index has been encountered.

  Notice that these FP values are not the same as those returned when NDX=NO.

- GETN and SKIP point to key values and use the REL field. (These calls function differently when NDX=NO.)

Generally, subroutines have the following uses for index file operations:

- For count retrieval operations

  STARTM retrieves the count of the number of records with the specified alternate key value. The FIT field that shows the count is RC.

  STARTM/REWND followed by SKIP retrieves the count of the number of records in a range of alternate key values. You can set the low end of the range to the start of the index, or to either an exclusive or inclusive alternate key value. The high end of the range can be set to the end of the index, or to either an inclusive or exclusive alternate key value. The FIT field that shows the count in a range is RL.

- For keylist retrieval operations

  STARTM/REWND followed by GETN retrieves the keylist for a range of alternate key values. GETN terminates on the KA and REL condition established. No keylist entries are transferred to the working storage area if KA and REL are the same for a GETN as for the preceding STARTM, for this would indicate a range whose end coincides with its beginning.

Count information and primary key values returned are discussed separately in this section. One operation can retrieve both types of information.

Table 4-2 summarizes subroutine usage and the meaningful FIT fields.

The RKW, RKP, and KL field values establish the particular alternate or primary key on which the operation occurs.

RETRIEVING RANGE COUNT INFORMATION

The number of records with a particular alternate key value, or range of values, is one of the two types of information that can be retrieved from the index file. The NDX field must first be set to YES.

The subroutines to be called and the use of FIT fields differ depending on whether the information to be retrieved is for a single alternate key value or a range of values.

Single Alternate Key Count

The STARTM call retrieves the number of records with a particular alternate key value. At the time of the STARTM call, the REL field of the FIT normally is set to EQ. The value of the alternate key should be in the variable established by the KA and KP fields of the FIT.

For count retrieval operations, major keys are not applicable.

In response to STARTM, MIP positions the index and sets the RC and KNE fields to reflect that position. Since REL is EQ, the KA variable should contain the value of an existing alternate key. In this instance, the RC field of the FIT reflects the number of entries in the keylist for the alternate key value at KA.

If the alternate key value at KA does not exist within the index, however, an error situation exists.

At the conclusion of STARTM, MIP sets the KNE (when REL is GE) and RC fields:

KNE	Key-not-equal indicator
	0 The value of the key at KA matches that of an alternate key in the index, and the index is positioned to that keylist.
	1 The value of the key at KA could not be found within the index. The index position is at the next higher alternate key value (might also be the end of the index).
RC	Count of number of entries for the keylist at the current file position. This is the desired count only when KNE is 0.

Range of Alternate Key Count

The number of records containing any of a range of alternate key values is obtained by the following procedure:

- Establish the low end of the range in one of two ways.

  REWND call to position to the beginning of the index.

  STARTM call with an alternate key value in variable KA and the REL field of the FIT set to GE, GT, or EQ.

- Establish the high end of the range in one of two ways.

  Reset value in KA and set REL to GE, GT, or EQ.

  Indicate end of index by setting KA to 0.

- Call SKIP with a skip count of 0.

```
CALL SKIP(fit,0)
```

In response to the SKIP call, the index position changes until the condition established by REL and KA is met. The position change must always be forward. As with any index file operation, you must be familiar with key values.

TABLE 4-2. INDEX FILE OPERATIONS WHEN NDX=YES

Subroutine	USE	Possible REL Values	WSA Effect	FIT Field Effect				
				RC	KNE	RL	PTL	FP
STARTM	Single count retrieval	EQ	None	Number of entries for specified key	0 EQ found 1 EQ not found			
	Set low end range for count retrieval or key-list retrieval	GT,GE			If REL=EQ, KNE always is 1	0		
GET	Retrieve single keylist	EQ	Fill to MRL limit or until keylist transfer is complete	Number of entries for specified key if FP=0 (i.e., not all keys delivered)		Number of entries transferred this GET. If FP=10 (i.e., all primary keys delivered) then RL=RC	Same as RL	Nonzero if all entries are transferred; otherwise 0
GETN	Continue keylist read after GET	GT				RL=RL+PTL	Number of entries transferred to WSA on this GETN	
	Continue keylist read after STARTM and new KA/REL condition	GT,GE	Fill to MRL limit or until new KA/REL is satisfied					
SKIP	Set high end for count retrieval and perform count	GT,GE	None			Number of entries in range		

At the end of this sequence, the RL field of the FIT reflects the number of entries that has been skipped. (In contrast, the count for a single key value is returned to RC.) The RL count excludes the number of entries in which REL and KA terminate the skip.

Other FIT fields at the end of the skip operation reflect the current index position, which is after the range indicated.

Both the low end and the high end of the range can be set by KA and REL. It is important to note that the REL and KA relationship at the beginning of the range is not interpreted the same as at the end of the range.

At the low end of the range, index position is as follows:

GE	Index position is at the beginning of the keylist for any value specified for KA. (RL includes KA value keylist if it exists.)
GT	Index position is at the beginning of the keylist after that specified by KA. (RL excludes KA value keylist.)

At the high end of the range, the SKIP operation terminates when the next alternate key in the index satisfies the condition specified by KA and REL. At the high end of the range, index position is as follows:

GE	Index position is at the beginning of the keylist for any alternate key value at KA. (RL does not include the primary keys for the value at KA.)
GT	The index position is at the beginning of the keylist for the next higher alternate key value than is specified by KA. (RL includes the primary keys for the value at KA if it exists.)

If KA is set to 0 for the high end of the range, all keylists through the end of the index are included in the RL count.

The alternate key value for the high end of the range must be greater than the value for the beginning of the range. A wrap-around search does not occur. Rather, when the values are not in relative order, RL reflects the number of primary keys between the first KA/REL specified and the end of the index. No error condition is indicated. You have full responsibility for knowing which of two alternate key values is higher.

Example of Range Count Retrieval

Program RGREAD in figure 4-17 illustrates the obtaining of index information for a range of key values. The program uses index file FLRXIP (created in figure 4-10).

READING KEYLISTS

The keylist read operations return primary key values to the working storage area. The operation requires the NDX field of the FIT to be set to YES.

Format of the primary key entries returned to the working storage area depends on the key type specified on an RMKDEF directive or call. If the key type is symbolic or uncollated, the primary key is returned left-justified with blank fill in as many words as required by key length. If the key type is integer, a 60-bit integer is returned.

As with all other multiple index file operations, the current values of RKW, RKP, and KL establish the particular index within the index file.

You must control the working storage area length through the MRL field of the FIT. It must be at least long enough to contain one primary key value. The value is rounded up to an integral number of words.

The operations required to read keylists depend on whether you are retrieving a single keylist or a range of keylists.

Single Keylist Read

A STARTM call sets an alternate key value to be used in the retrieval of one keylist. The keylist associated with a single alternate key value is returned to the working storage area through a call to GETN. Parameters of the call are the same as for read of a data record, except that neither the major key nor the record length parameter is applicable. At the time of the GETN call, the REL field of the FIT should be set to EQ.

In response to GETN, MIP returns the keylist for the alternate key value in the variable indicated by the KA and KP fields of the FIT. Since REL is EQ, the value of an existing alternate key should have been specified.

If no alternate key value in the index matches that in the variable KA, an error situation exists. In this instance, MIP sets the KNE field of the FIT to 1 to indicate that a key match was not found.

For a GETN operation that reads a keylist, transfer of entries to the working storage area ends when one of the following conditions occurs:

- One entire keylist has been transferred.
- The working storage area is filled, as established by the MRL field.

If the entire keylist has been transferred, the FP field has a value of 10_8. (The FP might have a value of 100_8 if the end of the keylist is also the end of the index.)

If the entire keylist has not been transferred, the FP field has a value of 0. In this instance, you must set REL to GT and call GETN to read the remaining entries in the keylist.

Transfer of entries from the keylist to the working storage area begins at the current position within the keylist. This position must always be within a keylist, as indicated by a 0 value in the FP field. Transfer continues until one of the following occurs:

- The working storage area is filled to the limit established by MRL.
- The entire keylist begun with the previous GETN has been transferred to the working storage area.

If GETN completes transfer of the keylist, the FP field has a nonzero value. A value of 0 in the FP field indicates that you must call GETN again to transfer additional entries.

The actual number of entries transferred to the working storage area for GETN is reflected in the PTL field or the RL field of the FIT. See table 4-2.

A. NOS Control Statements

```
Job statement
USER statement
CHARGE statement
FTN5.
ATTACH,FLRECS
ATTACH,FLRXIP
LGO.
```

NOS/BE Control Statements

```
Job statement
ACCOUNT statement
FTN5.
ATTACH,FLRECS,ID=AAMUG
ATTACH,FLRXIP,ID=AAMUG
LGO.
```

B. Source Program

```
      PROGRAM RGREAD
      IMPLICIT INTEGER (A-Z)
      DIMENSION FIT(35), KEY(2)
C
C  ***************************************************************
C  * THIS PROGRAM ILLUSTRATES THE FOLLOWING OPERATIONS ON THE *
C  * INDEX FILE FLRXIP:                                        *
C  *     INTERACTIVELY SET A LOW KEY VALUE FOR ACTOR'S NAME   *
C  *     INTERACTIVELY SET A HIGH KEY VALUE FOR ACTOR'S NAME  *
C  *     OBTAIN THE COUNT OF RECORDS  WITHIN THE RANGE        *
C  ***************************************************************
C
      CALL FILEIS (FIT, 'LFN', 'FLRECS', 'XN', 'FLRXIP', 'NDX', 'YES',
     +            'ORG', 'NEW', 'KA', KEY)
C
C  SELECT ACTOR NAME INDEX
C
      CALL OPENM (FIT, 'INPUT')
      IF (IFETCH (FIT, 'ES') .NE. 0) GO TO 950
      CALL STOREF (FIT, 'RKW', 3)
      CALL STOREF (FIT, 'RKP', 3)
      CALL STOREF (FIT, 'KL', 11)
C
C  SET HIGH AND LOW KEY VALUES
C
    6 PRINT *, ' ENTER GT OR GE -- OR EOF (RETURN KEY)'
      READ (*, '(A2)', END = 800) REL
      CALL STOREF (FIT, 'REL', REL)
      PRINT *, 'ENTER LOW END OF RANGE '
      READ (*, '(2A10)', END = 800) KEY
      PRINT 75, KEY, REL
      CALL STARTM (FIT)
      IF (IFETCH (FIT, 'ES') .NE. 0) GO TO 950
      PRINT *, 'ENTER HIGH END OF RANGE '
      READ (*, '(2A10)', END = 800) KEY
      PRINT *, ' ENTER GE OR GT '
      READ (*, '(A2)', END = 800) REL
      CALL STOREF (FIT, 'REL', REL)
      PRINT 75,  KEY, REL
      CALL SKIP (FIT,  0)
      IF (IFETCH (FIT, 'ES') .NE. 0) GO TO 950
      PRINT *, ' NUMBER OF RECORDS IN RANGE IS  ', IFETCH (FIT, 'RL')
      GO TO 6
  800 CONTINUE
      CALL REWND (FIT)
      DO 801 I = 1, 2
  801 KEY (I) = 0
      CALL SKIP (FIT, 0)
      IF (IFETCH (FIT, 'ES') .NE. 0) GO TO 950
      PRINT *, ' RL AFTER REWIND AND SKIP IS  ', IFETCH (FIT, 'RL')
  900 CALL CLOSEM (FIT)
      STOP
  950 PRINT 76, IFETCH (FIT, 'ES')
      CALL CLOSEM (FIT)
      STOP 'CRM ERROR RETURNED'
   75 FORMAT (4X, ' KEY IS ', 2A10, ' REL IS ', A2)
   76 FORMAT ('ES = ', O3)
      END
```

Figure 4-17. Using Only the Index File (Sheet 1 of 2)

C. Typical Program Output

```
/LGO
  ENTER GT OR GE -- OR EOF (RETURN KEY)
? eq
 ENTER LOW END OF RANGE
? bogart
     KEY IS BOGART              REL IS EQ
 ENTER HIGH END OF RANGE
? lorre
  ENTER GE OR GT
? eq
     KEY IS LORRE               REL IS EQ
  NUMBER OF RECORDS IN RANGE IS  40
  ENTER GT OR GE -- OR EOF (RETURN KEY)
? gt
 ENTER LOW END OF RANGE
? bogart
     KEY IS BOGART              REL IS GT
 ENTER HIGH END OF RANGE
? henreid
  ENTER GE OR GT
? gt
     KEY IS HENREID             REL IS GT
  NUMBER OF RECORDS IN RANGE IS  32
  ENTER GT OR GE -- OR EOF (RETURN KEY)
?
  RL AFTER REWIND AND SKIP IS  88
      .036 CP SECONDS EXECUTION TIME.
/
```

Figure 4-17. Using Only the Index File (Sheet 2 of 2)

You must monitor the FP field to determine when the complete keylist has been transferred. A zero value indicates that additional entries exist in the keylist. You should not try to compare the RC field with the total in RL to determine the end of the keylist; at the time control is returned to the program, the RC value has no meaning for the previous GETN.

Range of Keylists

The keylists associated with a range of alternate key values are returned to the working storage area through a three-step operation similar to that for retrieving the count of a range of keylist entries.

- Establish the low end of the range by either of the following:

  REWND call to position to the beginning of the index.

  STARTM call with alternate key value in variable KA and with the REL field of the FIT set to GE or GT.

- Establish the high end of the range in one of two ways:

  Reset value in KA and set REL to GE or GT

  Indicate end of index by setting KA to 0

- Call GETN repetitively until FP is nonzero.

In response, GETN operates as it does for retrieving additional entries for a single value of an alternate key. GETN transfers entries from the keylists to the working storage area until one of the following conditions occurs:

- The working storage area is filled to the limit established by the MRL field of the FIT. In this instance, GETN must be recalled to complete the retrieval.

- All entries that conform to the condition established by KA and REL are transferred to the working storage area.

- The end of the index is reached.

The series of GETN calls does not necessarily fill the working storage area, but terminates when the current index position corresponds to the condition established by the KA and REL values. (Consequently, a STARTM call that positions the index also establishes the conditions to terminate a subsequent GETN; and no entries are transferred for a GETN call after STARTM unless the KA value is changed.)

When the end of the keylist has been reached as a result of GETN, FP is set to 10_8 or 100_8. No entries for the subsequent key value are moved to the working storage area even if space exists for them.

You must monitor the FP field of the FIT to determine when all the desired entries have been transferred. FP has a zero value until transfer is complete.

The conditions for establishing the range are determined by the contents of the KA variable and the REL field of the FIT. As with count retrieval, GE or GT can be established for REL. Interpretation of REL at the beginning of the range is not the same as at the end of the range.

At the beginning of the range:

- GE results in transfer of keylist entries for the value in variable KA if it exists.
- GT excludes the entries for the alternate key value of KA.

At the end of the range:

- GE excludes the entries for the alternate key value of KA.
- GT includes the entries for the alternate key value of KA.

You must know the relative order of key values and specify the low end of the range before the high end of the range. A wrap-around search does not occur. FP is set to 100_8 if the end of the index is reached before the condition for the high end of the range is encountered.

You can use the entries returned to the working storage area to access the data file. To do this, however, you must change the NDX field to NO before GET (or any subroutine) is called to read the data file. Execution of subroutines always depends on the NDX setting. Therefore, data file and index file operations can be intermixed.

ADDITIONAL MIP FEATURES

Several features exist for the experienced AAM/MIP user. Null value suppression or sparse key suppression can improve program efficiency. The MIPDIS utility is used for file maintenance.

SPARSE KEY VALUE SUPPRESSION

In the normal use of alternate keys, every alternate key value is entered in the index file according to the contents of every defined alternate key. If a data file has 100000 records and 3 alternate keys, for example, each of the three indexes in the index file contains 100000 primary keys.

MIP provides the ability to designate certain records to be excluded from the index file if they are generally of no interest or if they will never be used for retrieval. For example, a delinquent field on a credit file might be of interest only when it is non-blank or contains a particular value. The features of null value suppression and sparse key suppression allow you to control whether the value of the field appears in the index.

A sparse key control field is a one-character field within a data record. The content of this field determines whether the record is to be included in the index file. The control field can be anywhere in the record, including within the primary key field or the alternate key field. To specify the sparse control field, use an RMKDEF call with KL=0.

Only one sparse key control field can be defined for each file. It can be applied to more than one alternate key, however. And, multiple values can be specified for exclusion. At the time the alternate key is defined, you must specify whether the sparse key field is to be considered in relation to the alternate key field; if so, you must also specify a string of 1 through 36 letters or digits. If one of the specified letters or digits appears in the control field of a record, that record is excluded from the index according to RMKDEF call (assuming E is specified in the ie parameter).

For example, the following RMKDEF calls could have been used in the MIPGEN illustration (figure 4-9) to define ACTIVITY as an alternate key. Assume that you have no interest in the SPORTS file for members of the freshman class.

- RMKDEF(SPORTS,1,0,0)
- RMKDEF(SPORTS,3,6,6,0,S,I,6,0,N,E,F)

NULL VALUE SUPPRESSION

A null value consists of all spaces (blanks) in an alternate key field described as symbolic type. A null value consists of all zeros in an alternate key field described as integer or uncollated symbolic.

You can include or exclude null values in an index at your option. To suppress the null values, specify N in the nl parameter on the RMKDEF call. (Refer to figure 4-8.)

For example, the following RMKDEF call could have been used in the example in figure 4-9 to define ACTIVITY as an alternate key.

RMKDEF(SPORTS,3,6,6,0,S,I,6,0,N)

Since the nl parameter is set to N, any record with all spaces in the ACTIVITY field is omitted from the alternate key index.

The first RMKDEF call defines the first character of the CLASS field as the sparse key control field. (Word 1, character 0, KL=0). The second RMKDEF call specifies E for the ie parameter, and F for the specific value to be excluded. That is, all records with an F in the first position of the CLASS field will be excluded from the alternate key index file.

Instead of excluding all freshman records, you could include only sophomore, junior, and senior records. You would replace E,F (exclude F's) by I,SJ (include S and J) in the second RMKDEF call.

Sparse key control can be specified together with null values. The null suppression takes precedence and suppresses an index entry even if the control field would otherwise cause it to appear in the index.

THE MIPDIS UTILITY

The MIPDIS utility exists to disassociate the index and data files. This allows, for example, the data file to be restructured through FORM so the size of data blocks can be changed or unused space eliminated, without the need to recreate the index file.

The MIPDIS utility is the only method by which you can disassociate an index file from its data file. The utility can also be used to reassociate an index file with its data file, after a temporary disassociation. The number and content of the records in the data file must not have changed during the disassociation.

The MIPDIS utility temporarily or permanently disassociates an index file from its data file. The index file can be reassociated with the data file if the primary and alternate key fields have not been updated during the disassociation. The format of the MIPDIS control statement is shown in the AAM reference manual.

You might find MIPDIS useful in the following circumstances:

- A data file is no longer being accessed by alternate key and, therefore, the index file is no longer needed.
- A data file is to be reorganized (through FORM or through a user program) to redistribute extraneous padding in data blocks. The index file can be temporarily disassociated with the data file during reorganization. MIPDIS can be used for both the disassociation and the reassociation of index file with the data file.

DIRECT ACCESS FILE PROCESSING 5

A direct access file is composed of records that are stored randomly in home blocks on mass storage. Home blocks are fixed-length areas preallocated for the file at the time the file is created. Records are usually accessed randomly by primary key. Serial access is also possible, but the key of a record has no necessary relation to the key of the preceding record. Direct access file organization is particularly well suited to applications where random access speed is of paramount importance and symbolic primary keys are used.

This section discusses the requirements for creating and processing a direct access file by primary key. Refer to section 4 for alternate key processing. If alternate keys are defined, the data file must conform to the requirements discussed in this section.

Each record in a direct access file must have a unique primary key associated with it. The primary key value is converted to a number signifying a home block. The process of converting the primary key value to a home block number is called hashing. It can be considered a mapping technique that uses an algorithm to calculate a value to be equated with a home block. Only one hashing routine can be used for the life of the file. It can be the routine supplied with the release system or a routine written by you.

Mass storage residence is necessary for a direct access file. The file can be dumped to tape for backup or storage purposes with a COPYBF or a permanent file dump routine. At a later time, the file can be restored to mass storage for processing.

You can use the CREATE utility to create a direct access file. For most files, file creation through this utility is more efficient than through source program statements. The CREATE utility is called into execution by statements in a source program written in COMPASS, FORTRAN, or COBOL.

You can use the FORM utility to create a direct access file from a sequential, actual key, or indexed sequential file (not the most efficient method). FORM can also re-create an existing direct access file in the same or different file structure.

CONCEPTS OF LOGICAL FILE STRUCTURE

A direct access file consists of an internal file statistics table (FSTT), a user-defined number of home blocks, and any overflow blocks established by AAM. The creation and use of the FSTT is controlled by AAM. You are responsible for defining the size and number of home blocks.

HOME BLOCKS

Records in a direct access file are grouped into home blocks. The specific home block in which a record resides is determined by hashing the primary key value to a home block number. The size and the number of home blocks are defined at file creation time; mass storage equivalent to all home blocks is reserved for the file the first time a record is written to the file. Only direct access files have preallocated storage.

Each home block contains a two-word block header, a number of data records, and record pointers. All home blocks in the file are the same size. If the block checksum option has been selected, the checksum is stored in the block header.

Varying numbers of records appear in home blocks, dependent entirely on the results of primary key hashing. When variable-length records are stored in the home blocks, the blocks often have unused storage space. Data records are stored following the block header in the order they are written to the file.

Record pointers are stored at the end of the home block beginning with the last word. Two record pointers are stored in one word. Only one record pointer per block is needed when all records in the block are the same size; otherwise, one record pointer is required for each record in the block.

Figure 5-1 illustrates the structure of a home block in a direct access file. The block header at the beginning of the block is followed by four variable-length records. The primary keys of the records have no logical relationship to each other. The four record pointers are stored in the last two words of the block.

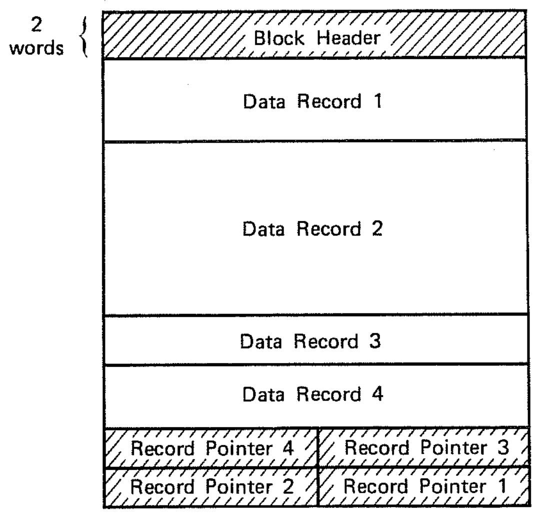


Figure 5-1. Direct Access File Home Block Structure

OVERFLOW BLOCKS

Home blocks are all the same fixed-length size. When the primary key of a record to be written hashes to a block that cannot accommodate the record, AAM creates an overflow block to hold the record. Overflow blocks are the same size as home blocks. A pointer in the block header of the home block specifies the location of the overflow block. If an overflow block becomes full, another overflow block is created. Records that are stored in overflow blocks require at least two mass storage accesses, thus resulting in an increase in execution time. When the file has no overflow blocks, execution time is faster.

SPECIFYING FILE STRUCTURE

On the file creation run, the structure of the direct access file is defined. This structure remains the same for the life of the file. File structure is defined in terms of:

- Record size
- Primary key size and location
- Home block size
- Hashing routine

RECORD DEFINITION

Before the file is first opened on the creation run, you must establish the record structure. If you do not set the record type (RT) field in the FIT, the default of W type records is set in the RT field. However, AAM processes the records as U type records. Other FIT fields required for record definition depend on the record type selected. Record size specification is also dependent on record type.

The FIT fields unique to a record type are described in the section 3 discussion of record definition. The MNR and MRL fields specify the minimum and maximum number of characters in any record in the file. The value of the MNR field cannot be zero and cannot exceed the value of the MRL field. If the primary key is embedded in the record, the minimum record length must include the full primary key. For the life of the file, any record larger or smaller than the limits established by the MNR and MRL fields is rejected as a trivial error.

PRIMARY KEY DEFINITION

Each record in a direct access file is identified by a unique primary key value. This value is hashed to indicate the home block in which the record is stored. Any key suitable to your program is acceptable as long as it is an integral number of characters less than the minimum record length.

Two FIT fields are required to define the primary key:

EMK	Embedded key; primary key is within the record; default is YES. (YES is more efficient.)
KL	Key length; number of characters in the primary key.

If primary keys are embedded (the EMK field is set to YES), the following FIT fields are also required:

RKW	Relative key word; word within the record in which the primary key begins, counting from 0; default is 0.
RKP	Relative key position; character position within the relative key word in which the primary key begins, counting from 0; default is 0.

If primary keys are not embedded (the EMK field is set to NO), two different FIT fields are required. These two fields, which are also required to read the direct access file randomly, are as follows:

KA	Key address; location of the word containing the primary key.
KP	Beginning key position; starting character position in the word indicated by the KA field; default is 0.

HOME BLOCK DEFINITION

You must define the number and size of home blocks in the file before the file is opened on the file creation run. The number of home blocks must be specified by setting the following field in the FIT:

HMB	Number of home blocks; should be a prime number for the default hashing routine.

A more even distribution is often obtained by selecting a prime number for the HMB field. Different values for the HMB field can be tested by the key analysis utility. This utility shows the distribution of input records for different numbers of home blocks. Refer to the AAM reference manual.

Home block size is determined by the maximum block length (MBL) field in the FIT, which is set in one of three ways:

Specify the block size directly.

Accept the default calculated by AAM.

Specify the blocking factor and accept the block size calculated by AAM.

The default calculated by AAM is based on two average size records per home block. The average record size is determined from the following FIT fields set by the user:

MNR	Minimum record length; minimum number of characters in any record; cannot be zero and must not exceed the maximum record length.
MRL	Maximum record length; maximum number of characters in any record.

A blocking factor other than two average size records per block can be used by AAM to calculate block size. In addition to the MNR and MRL fields, you must specify the following FIT field:

RB	Records per block; number of average size records per home block; cannot be zero; does not necessarily affect the actual number of records in a block.

When you directly specify the block size, the MNR and MRL fields are not involved with block size specification; however, the fields must still be specified for file creation. The following FIT field is set to specify block size directly:

MBL	Maximum block length; maximum number of characters in a home block.

The setting of the MBL field, in all three cases, must be large enough to hold at least one maximum length record (specified by the MRL field), the system-supplied two-word block header, and one record pointer. AAM increases the value of the MBL field, if necessary, to an integral multiple of physical record unit (PRU) size less two words.

You should specify an MBL value 50 characters less than the PRU multiple desired. The resulting data block size will be:

[(Specified MBL + 50 characters) rounded to the next PRU multiple] - 20 characters

HASHING ROUTINE

You have the option of writing a hashing routine or of using the default routine released with AAM. The hashing routine is required to manipulate the primary key to determine the block where the record resides.

Extensive analysis of the record key structure, key range, and key distribution is necessary to implement a randomly organized file in an optimum manner. An ideal hashing algorithm distributes records uniformly across all home blocks.

The hashing routine used to determine the home block in which a record is stored depends on the setting of the following FIT field:

HRL	Hashing routine location; name of the routine for hashing primary key values; default is the system-supplied routine.

When a user-supplied hashing routine is specified, the routine must be made available to the program. The HRL field must be set before the file is opened on the file creation run and the same hashing routine must be used for the life of the file. A user-supplied hashing routine does not become part of the file. You need not call the routine. AAM calls the hashing routine each time it is required. AAM passes three parameters to the hashing routine. These parameters are the values in the following FIT fields:

KL	Key length; number of characters in the primary key.
KA	Key address; location of the word containing the primary key.
HMB	Number of home blocks; total number of home blocks allocated at file creation.

When a user-supplied hashing routine is written in FORTRAN, it should begin in the following manner:

```
SUBROUTINE HASHING (I, J, K, L)
DIMENSION J(26)
```

In this example, I and K represent the KL and HMB fields, respectively. L is an integer variable in which the result must be stored before returning. J represents an array containing the primary key with a maximum length of 255 characters. The key is left-justified in this array. If the key length is not a multiple of 10 characters, the last word of the key has binary zero padding.

The user hashing routine must return an integer hashed result. AAM divides the returned integer by the value of the HMB field to ensure a value in the range of home block numbers. Only the lower 48 bits of the returned integer are used for the division.

During open processing for an existing file, AAM uses the hashing routine indicated by the HRL field. The hashed value of the first record in the file is checked to verify that no change has occurred in the hashing routine. Any change in the hashing routine would deny access to existing records.

KEY ANALYSIS UTILITY

The key analysis utility assists you in selecting the most effective hashing routine or the number of home blocks that best suits a particular file. Ideally, a good hashing routine results in a uniform distribution of records in all home blocks, with no overflow blocks. When the file has no overflow blocks, execution time is faster. However, mass storage requirements for the file could be greater.

Information about hypothetical record distribution for the file is provided by the key analysis utility. Changing the number of home blocks or the routine that distributes records among home blocks allows the user to balance mass storage requirements and access time considerations before the file is actually created. For each record in the file, the key analysis utility reads the primary key and determines the home block where the record would reside. After all primary keys have been examined, statistics are output by the utility.

Up to five different hashing routines, including the system-supplied routine, can be tested during execution of the key analysis utility. The same hashing routine can also be tested with different values for the number of home blocks. The results of the utility are output to a file named KEYLIST. If the file is to be printed, it must be rewound and copied to the file OUTPUT.

User Hashing Routines

Any user-supplied hashing routine to be run with the key analysis utility must have the following characteristics:

- It must be in relocatable binary format on the file identified by the LFN parameter in the KYAN directive.
- It must have an entry point identified by the H parameter in the KYAN directive.
- Execution of the hashing routine must result in an integer value in the lower 48 bits of a word.

KYAN Directive

The KYAN directive must be supplied to describe the proposed file parameters. When the key analysis utility is invoked, the KYAN directive must be the next unexecuted record on the file INPUT.

You must specify a minimum of six parameters in the KYAN directive. These parameters, which must be specified in the order listed, are as follows:

- LFN=lfn

  Name of the file containing the hashing routine in relocatable binary format; LFN=0 when only the system hashing routine is used; LFN=LGO when the user hashing routine is submitted with the job as source code.

- MRL=mrl

  Maximum record length; must be defined in the directive, but it is not used.

- KL=kl

  Number of characters in the primary key.

- RKP=rkp

  Relative character position (counting from 0) in which the key begins within the word addressed by the parameter when SDAKEYH is called.

- RKW=0

  Relative key word; must be defined in the directive, but it is not used.

- H1=entry,hmb,option

  Name of the entry point for the hashing routine (SDAHASH for the system routine), number of home blocks, and type of statistics to be output; available output options are:

S	Number of keys (synonyms) that hash to each block
D	Standard deviation of records in all blocks
B	Both synonyms and standard deviation

The last parameter can be specified as many as five times using H2 through H5 for the additional parameters. This allows the key analysis utility to perform up to five different tests for one file. Any combination of hashing routine entry points and number of home blocks can be specified. When more than one hashing routine is tested, any user hashing routines must reside on the file identified by the LFN parameter.

The parameter list in the KYAN directive must be enclosed in parentheses. Blanks can appear after commas but must not be embedded within a parameter. When all parameters cannot be contained within 80 columns, a slash (/) must be placed in column 80 to indicate that the directive is continued. The directive can be continued as many as six times.

Source Program Call

You call the key analysis utility into execution by using two entry points, SDAKEYH and SDAENDH. The entry points are referenced as subroutines by a source program written in COMPASS, FORTRAN, or COBOL. The source program must provide the key analyzer with key values one by one. The values are usually taken from records on an existing sequential file; however, the source program could get them from a simple list of keys or even generate the key values in some way. In any case, the source program has to locate each new key value and then call SDAKEYH with one parameter, which gives the address of the first word of the key value.

After the last key value has been analyzed (this is usually when the end of the sequential file has been reached), a call to SDAENDH must be executed to write the results of the key analysis utility to the file KEYLIST. The file KEYLIST can then be rewound and copied to the file OUTPUT for printing.

The first time the key analysis utility is entered, the next unexecuted record on the file INPUT must contain the KYAN directive. This prevents the job from reading input records directly from the file INPUT. The utility reads this and calls for loading the file that contains the hashing routines. You should ensure that the file is available to the job but should not load the hashing routines.

If the KYAN directive specifies more than one hashing routine or more than one blocking factor to be tested, each hashing routine and blocking factor is used for each primary key value before control is returned to your program. All user-supplied hashing routines specified in the KYAN directive must reside on the same file.

Sample Program, Key Analysis Utility

Figure 5-2 shows a program that calls the key analysis utility into execution. Program KEYHASH reads an input record and then issues a call to SDAKEYH. After all records in the input file have been read, a call to SDAENDH is issued. Subroutine MYHASH is a user routine for hashing primary key values. The input file contains the records for the direct access file to be subsequently created. The KYAN directive provides the following information:

- User hashing routine location (file HASHBIN)
- Maximum record length (50 characters)
- Primary key length (6 characters)
- Primary key beginning position (first character of first word)
- Hashing routine entry points (user routine MYHASH and system routine SDAHASH)
- Number of home blocks (5 for both routines)
- Type of statistics (synonyms and standard deviation for both routines)

The figure also shows the statistics output by the key analysis utility. The last column is the system routine statistics, which indicate a more even distribution of records in the five home blocks.

```
NOS Operating System                    NOS/BE Operating System

Job statement                           Job statement
USER statement                          ACCOUNT statement
CHARGE statement                        FTN5.
FTN5.                                   FTN5,B=HASHBIN.
FTN5,B=HASHBIN.                         LDSET(LIB=AAMLIB)
LDSET(LIB=AAMLIB)                       ATTACH(ACCOUNT,ID=AAMUG)
ATTACH(ACCOUNT)                         LGO.
LGO.                                    REWIND(KEYLIST)
REWIND(KEYLIST)                         COPYBF(KEYLIST,OUTPUT)
COPYBF(KEYLIST,OUTPUT)
- - EOR

    PROGRAM KEYHASH
    IMPLICIT INTEGER (A-Z)
    DIMENSION DATA(5)
    OPEN (2, FILE = 'ACCOUNT')
10  READ (2, '(5A10)', END=40) DATA
    CALL SDAKEYH (DATA)
    GO TO 10
40  CALL SDAENDH
    STOP
    END

- - EOR

    SUBROUTINE MYHASH (KLTH, KEYADD, HBLKS, HKEY)
    INTEGER KEYADD(1), HBLKS, HKEY, KLTH, KW
    KW = (KLTH + 9) / 10
    HKEY = 1
    DO 20 I = 1, KW
20  HKEY = HKEY + KEYADD (I)
    END

- - EOR

KYAN(LFN=HASHBIN,MRL=50,KL=6,RKP=0,RKW=0,H1=MYHASH,5,B,H2=SDAHASH,5,B)

- - EOI
```

OUTPUT

```
  HOME BLOCK

     0              7        7
     1              3        2
     2              0        2
     3              4        3
     4              5        5

STANDARD DEVIATION

                  2.57      2.15
```

Figure 5-2. Calling the Key Analysis Utility

CREATING A DIRECT ACCESS FILE

You can use a source program, the CREATE utility, or the FORM utility to create a direct access file. On the file creation run, the file statistics table (FSTT) is created by AAM and becomes a permanent part of the file.

Various fields in the FIT must be defined on the file creation run. You must set the old/new file (ON) field to NEW. The following FIT fields are defined when the file is created and cannot be changed for the life of the file:

FO	File organization; set to DA by the CALL FILEDA statement.
ORG	Old/new file organization; must be set to NEW.
HMB	Home blocks; number of home blocks to be allocated for the file.
KL	Key length; number of characters in the primary key.

MRL	Maximum record length; maximum number of characters in any record; set by AAM when FL is specified for F or Z type records.
MNR	Minimum record length; minimum number of characters in any record; cannot be zero; set by AAM when FL is specified for F or Z type records.
RT	Record type; default is W, which AAM processes as U; fields required by the record type must also be specified.

One additional FIT field must be defined for the file creation run; however, this field can be changed on subsequent runs. The following FIT field must be set:

LFN	Logical file name; one to seven characters, beginning with a letter.

If alternate keys are to be defined during the file creation run, you must specify the following FIT field:

XN	Index file name

You must define the following FIT fields on the file creation run or the default values are effective for the life of the file:

MBL	Maximum block length; default is number of words required for two average size records per home block.
BCK	Block checksum; default is no checksums.
HRL	Hashing routine location; name of the entry point for the hashing routine; default is the system-supplied routine.
EMK	Embedded key; must be set to NO if the primary key is not contained within the record; default is YES.

If the primary key is embedded, you must define two additional FIT fields:

RKW	Relative key word in which the key begins; default is word 0.
RKP	Relative key position within the word in which the key begins; default is character position 0.

Other optional FIT fields that you can define on the creation run and can change on a subsequent run are as follows:

DFC	Dayfile control; default is only fatal error messages to the dayfile.
EFC	Error file control; default is no messages to the error file.
ERL	Trivial error limit; default is no limit.
FLM	File limit; default is no limit.
FWI	Forced write indicator; default is buffers written only when space is needed.
FWB	First word address of the buffer; default is buffer address provided by AAM.
BFS	Buffer size; default is buffer size calculated by AAM.
CPA	Compression routine address; default is no compression of records.
DCA	Decompression routine address; default depends on the CPA field (refer to appendix F).
DX	End-of-data exit; default is no exit subroutine.
EX	Error exit; default is no exit subroutine.

FILE CREATION THROUGH CREATE

You can use the CREATE utility through a COBOL, COMPASS, or FORTRAN source program to create a direct access file. The system-supplied hashing routine or a user-supplied hashing routine can be used with the CREATE utility. The SORT library must be available in order to use this utility.

When you are creating a large file (more than 1000 records), you should use the CREATE utility. A large file can be created more efficiently by using this utility because all records whose primary keys hash to a given home block are written in one mass storage access. When the file is created by other methods, a home block must be transferred from mass storage to the central memory buffer for each record to be written.

The CREATE utility hashes the primary key in the input record and prefixes it to the record. After all records have been read, the utility calls the SORT routines to sort the records according to the hashed primary key values. When the sort operation is complete, the CREATE utility calls AAM to create the direct access file. The hashed key values are removed from the records; they are only used by the sort routines.

A job using the CREATE utility must contain the following:

- A FILE control statement describing the new direct access file, including the key position within the record if the key is embedded
- A source program call that reads a record and calls the CREATE utility
- A CREATE directive that is in a separate record in the job input file

The FILE control statement for the direct access file is required because you cannot describe the file within the program calling the CREATE utility. You must set the following FIT fields by the FILE control statement:

FO	File organization; must be FO=DA for direct access file organization.
ORG	Old/new file organization; must be ORG=NEW.
HMB	Home blocks; number of home blocks to be allocated for the file.
MBL	Maximum block length; number of characters in a home block.
MNR	Minimum record length; minimum number of characters in any record; cannot be zero; set by AAM when FL is specified for F or Z type records.
MRL	Maximum record length; maximum number of characters in any record; set by AAM when FL is specified for F or Z type records.

KL	Key length; number of characters in the primary key.
RT	Record type; default is W, which AAM processes as U; fields required by the record type must also be specified.
EMK	Embedded key; must be set to NO if the primary key is not embedded in the record.

If the primary key is contained within the direct access file records, you must also set the following fields in the FIT.

RKW	Relative key word in which the key begins; default is word 0.
RKP	Relative key position within the word in which the key begins; default is character position 0.

CREATE Directive

You must supply a CREATE directive as the next unexecuted record in the file INPUT when the CREATE utility is called. The format of the CREATE directive is shown in the AAM reference manual.

The following directive example indicates that the file named DAFILE is to be created:

```
CREATE (DAFILE)
```

Only one CREATE directive can be input to the CREATE utility. If you supply the hashing routine, it must be on the specified file in relocatable binary form.

Source Program Call

The CREATE utility is called into execution through a source program written in COMPASS, FORTRAN, or COBOL. When you use this utility, the source program must make calls to two entry points. For each record read, a CALL SDACRTU statement must be executed to hash the primary key and affix the hashed value to the record. The format of this statement is as follows:

```
CALL SDACRTU (wsa,ka,rl)
```

Parameters in this statement identify the location of the record, the primary key, and the number of characters in the record. After all input records have been read and processed, a CALL SDAENDC statement must be executed to complete creating the direct access file. The format of this statement is as follows:

```
CALL SDAENDC
```

The first time the CALL SDACRTU statement is executed, the CREATE directive must be the next unexecuted record in the input file. This prevents the job from reading input records directly from the file INPUT. The parameters specified in the call cannot override the primary key specifications existing at the time the file is opened. That is, key specifications designated in the FILE control statement are effective for file creation and for the life of the file.

When the CALL SDAENDC statement is executed, the records are sorted by the hashed primary key values. AAM is then called by the CREATE utility to create the direct access file. The hashed primary key values are removed from the records. Hashed values do not become part of the records in the direct access file.

Sample Program, CREATE Utility

Figure 5-3 shows a program that calls the CREATE utility to create a new direct access file. Program CREATDA reads a record from the input file and then issues a call to SDACRTU. Three parameters are passed to the CREATE utility:

- Working storage area (the array DAWSA)
- Key address (first word of the array DAWSA)
- Record length (50 characters)

After all records in the input file have been read, the program issues a call to SDAENDC to produce the direct access file.

FILE CREATION BY A SOURCE PROGRAM

When a direct access file is created through a source program, file structure and key characteristics must be defined by setting applicable fields in the FIT before the file is opened. FIT fields can be specified in the FILE control statement, the CALL FILEDA statement, and the CALL STOREF statement.

You must set the old/new file (ON) field to NEW for a file creation run. This is accomplished by setting the ON field with one of the FIT manipulation statements or by opening the file with the processing direction (pd) parameter in the open request set to NEW. Setting the pd parameter to NEW actually sets two FIT fields; the ON field is set to NEW and the processing direction (PD) field is set to OUTPUT. You can then insert records into the file with write requests.

After all records have been written to the file, you should close the file. The only statements that can be issued on a file creation run are those that establish the FIT, open and close the file, write records, and read and write fields in the FIT.

Establishing the FIT

The first statement referencing the direct access file must be the CALL FILEDA statement. Execution of this statement causes the FIT to be constructed and the specified values to be stored in the FIT. The first parameter in the CALL FILEDA statement is the name of the 35-word array to hold the FIT. The same FIT array name is the first parameter in every statement accessing the direct access file. Refer to section 2 for a more detailed explanation of the FIT and the CALL FILEDA statement.

Opening the File

You must open the direct access file by executing a CALL OPENM statement to prepare the file for processing before you can write any records to the file. The format of this statement is shown in appendix B.

The following statement initiates open processing:

```
CALL OPENM (DAFIT, 'NEW')
```

The array named DAFIT contains the FIT for the direct access file being opened. The second parameter indicates a file creation run and sets the PD field to OUTPUT and the ON field to NEW.

A. NOS Operating System

```
Job statement
USER statement
CHARGE statement
FTN5.
ATTACH(EQUIP/UN=userno)
DEFINE(DAFILE/CT=PU,M=W)
FILE(DAFILE,FO=DA,HMB=5,ORG=NEW,
  RT=F,FL=50,KL=6,RB=10)
LGO.
```

NOS/BE Operating System

```
Job statement
ACCOUNT statement
FTN5.
ATTACH(EQUIP,ID=AAMUG)
REQUEST(DAFILE,PF)
FILE(DAFILE,FO=DA,HMB=5,ORG=NEW,
  RT=F,FL=50,KL=6,RB=10)
LGO.
CATALOG(DAFILE,ID=AAMUG)
```

B. Input File: EQUIP

```
AB5972    METAL DESK          038995    004     1
AB5973    OAK DESK            128250    002     2
AB5975    WALNUT DESK         130000    002     3
BB0013    BULLETIN BOARD      001500    010     4
CB0168    CHALK BOARD         001952    007     5
CB1001    FILE CAB, 1 DRAWER  004500    004     6
CB1003    FILE CAB, 3 DRAWER  006000    010     7
CB1005    FILE CAB, 5 DRAWER  009000    010     8
CH0059    ARM CHAIR           029500    007     9
CH0060    DESK CHAIR          014995    005    10
CH0080    SWIVEL CHAIR        009600    009    11
CM0575    LETTER RACK         000398    048    12
SH0011    BOOK CASE           003995    010    13
ST0592    STOOL               001620    011    14
TY5015    TYPEWRITER          036900    002    15
XM6158    COFFEE TABLE        006500    008    16
YB0020    DESK LAMP           001995    495    17
YB0059    FLOOR LAMP          006995    025    18
YB0060    TABLE LAMP          003995    020    19
```

C. Source Program

```
      PROGRAM CREATDA
      DIMENSION DAWSA (5)
      OPEN (2, FILE = 'EQUIP')
   10 READ (2, '(5A10)', END = 30) DAWSA
      PRINT 50, DAWSA
   20 CALL SDACRTU (DAWSA (1), 50)
      GO TO 10
   30 CALL SDAENDC
      STOP
   50 FORMAT (1X, 5A10)
      END
--EOR
   CREATE(DAFILE)
--EOI
```

D. Output

(same as input)

Figure 5-3. Direct Access File Creation, CREATE Utility

Writing Records

You can write records to a direct access file by executing a CALL PUT statement. The format of this statement is shown in appendix B.

The following statement writes a record to the file associated with the FIT in the array named DAFIT:

```
    CALL PUT (DAFIT)
```

The record to be written is stored in the location defined by WSA.

The KA and KP fields are not required when the primary key is embedded in the record. The relative key word (RKW) and relative key position (RKP) fields define an embedded key.

FIT fields not set by the write request default to the current values in the FIT. After the record has been written to the file, AAM sets the RL field to the number of characters in the record.

Records written to a direct access file are stored randomly within the file. The order in which records are written, therefore, is not significant. The record must be established in the working storage area when the write request is issued. If the primary key is not embedded, the location of the key value for the record to be written must also be established. For embedded primary keys, the key location is determined by the RKW and RKP fields.

Closing the File

When all records have been written to the file, a CALL CLOSEM statement must be executed to ensure file integrity. The format of this statement is shown in appendix B.

The following statement illustrates close processing:

```
    CALL CLOSEM (DAFIT)
```

This statement specifies that the file associated with the FIT in the array named DAFIT is to be closed and rewound.

Any home blocks in the central memory buffer are written to the file. The FSTT, which is used to maintain continuity over the life of the file, is also written to the file. If requested, file statistics are written to the error file. A close request issued for a file that has never been opened or that has been closed but neither unloaded nor reopened results in a trivial error.

Sample Creation Program

Program NEWDA, shown in figure 5-4, creates a direct access file through direct calls to AAM. The program reads an input file from the file EQUIP and writes records to the new direct access file DAFILE. For illustration purposes, the program also prints each input record as it is read.

Program statements related to creation of the direct access file are defined as follows:

- DIMENSION DAFIT(35), DAWSA(5)

  This statement allocates a 35-word array named DAFIT for construction of the FIT and a 5-word array named DAWSA for the working storage area.

- CALL FILEDA (DAFIT, 'LFN', 'DAFILE',...)

  This statement sets fields in the FIT to describe the structure of the direct access file. Required parameters include:

  FIT array (DAFIT)

  Logical file name (DAFILE)

  Extended file organization (NEW)

  Record type (fixed length)

  Fixed record length (50 characters)

  Primary key length (6 characters)

  Number of home blocks (5)

  Optional parameters include:

  Records per block (10)

  Embedded primary key (YES)

  Error file control (3, errors and notes)

  Dayfile control (3, errors and notes)

- CALL OPENM (DAFIT, 'NEW')

  This statement opens the file for a creation run. Records can only be written to the file.

- CALL PUT (DAFIT)

  This statement writes the record in the working storage area DAWSA to the direct access file.

- CALL CLOSEM (DAFIT)

  This statement writes the FSTT and any home blocks in the buffer to the file.

FILE CREATION THROUGH FORM

The CREATE utility is generally the most efficient means of creating or restructuring a direct access file. However, you can use the FORM utility to create a direct access file. FORM can also be used to restructure an existing direct access file or to dump an existing file to tape for backup or storage purposes. The following paragraphs briefly discuss using FORM to create a direct access file. Refer to the FORM reference manual for more detailed information.

A. NOS Operating System

```
Job statement
USER statement
CHARGE statement
FTN5.
ATTACH(EQUIP/UN=userno)
DEFINE(DAFILE/CT=PU,M=W)
LGO.
CRMEP(LO,RU)
```

NOS/BE Operating System

```
Job statement
ACCOUNT statement
FTN5.
REQUEST(DAFILE,PF)
ATTACH(EQUIP,ID=AAMUG)
LGO.
CATALOG(DAFILE,ID=AAMUG)
CRMEP(LO,RU)
```

B. Input File: EQUIP

```
AB5972   METAL DESK          038995   004    1
AB5973   OAK DESK            128250   002    2
AB5975   WALNUT DESK         130000   002    3
BB0013   BULLETIN BOARD      001500   010    4
CB0168   CHALK BOARD         001952   007    5
CB1001   FILE CAB, 1 DRAWER  004500   004    6
CB1003   FILE CAB, 3 DRAWER  006000   010    7
CB1005   FILE CAB, 5 DRAWER  009000   010    8
CH0059   ARM CHAIR           029500   007    9
CH0060   DESK CHAIR          014995   005   10
CH0080   SWIVEL CHAIR        009600   009   11
CM0575   LETTER RACK         000398   048   12
SH0011   BOOK CASE           003995   010   13
ST0592   STOOL               001620   011   14
TY5015   TYPEWRITER          036900   002   15
XM6158   COFFEE TABLE        006500   008   16
YB0020   DESK LAMP           001995   495   17
YB0059   FLOOR LAMP          006995   025   18
YB0060   TABLE LAMP          003995   020   19
```

C. Source Program

```
      PROGRAM NEWDA
      IMPLICIT INTEGER (A-Z)
      DIMENSION DAFIT (35), DAWSA (5)
      CALL FILEDA (DAFIT, 'LFN', 'DAFILE', 'ORG', 'NEW',
     +           'RT', 'F', 'FL', 50, 'RB', 10, 'HMB', 5,
     +           'KL', 6, 'EMK', 'YES',
     +           'WSA', DAWSA,
     +           'EFC', 3, 'DFC', 3)
      OPEN (2, FILE = 'EQUIP')
      CALL OPENM (DAFIT, 'NEW')
10    READ (2, '(5A10)', END = 30) DAWSA
20    CALL PUT (DAFIT)
      PRINT 50, DAWSA
      GO TO 10
30    CALL CLOSEM (DAFIT)
      STOP
50    FORMAT (1X, 5A10)
      END
```

D. Output

(same as input)

Figure 5-4. DA File Creation, Source Program

FORM uses CYBER Record Manager to perform input on the input file and output on the direct access file being created. Descriptions of the input and output files are provided in FILE control statements. Parameters in the FILE control statement for the input file depend on the file organization. For the direct access file being created, the FILE control statement must specify the logical file name and the following FIT fields:

FO	File organization; must be FO=DA for direct access file organization.
ORG	Old/new file organization; must be ORG=NEW.
HMB	Home blocks; number of home blocks in the file.
KL	Key length; number of characters in the primary key.
MRL	Maximum record length; maximum number of characters in any record; set by AAM when FL is specified for F or Z type records.
MNR	Minimum record length; minimum number of characters in any record; set by AAM when FL is specified for F or Z type records.
RT	Record type; FORM default is W, which AAM processes as U; fields required by the record type must also be specified.

If the primary key is embedded in the output record, the following FIT fields are also required:

RKW	Relative key word in which the key begins; default is word 0.
RKP	Relative key position within the word in which the key begins; default is character position 0.

If the primary key is not contained within the record, the embedded key (EMK) field must be set to NO.

The FILE control statement can also specify optional FIT fields. Fields not specified are set to the default values. The FORM directive OUT can specify values for the following FIT fields:

FLM	File limit; maximum number of records for the direct access file.
CPA	Compression routine address; number or entry point name of the compression routine; default is no compression of records.
HRL	Hashing routine location; entry point name of the user-supplied hashing routine; default is system-supplied hashing routine.
EX	Error exit; entry point name of the user-supplied error processing routine; default is no error exit subroutine.

A job using FORM to create a direct access file must contain the following:

- A FILE control statement describing the input file structure unless the file is a sequential file with default structure characteristics
- A FILE control statement describing the new direct access file structure
- A FORM control statement
- FORM directives

The input and output files are identified by the FORM directives INP and OUT, respectively. Other FORM directives are available to specify record selection criteria, reformatting, and conversion. Refer to the FORM reference manual for details of optional directives that might be useful.

The INP directive specifies the source of input records for the FORM run. The logical file name is the only required parameter in the directive. Optional parameters can be specified to indicate the maximum number of records to be processed, rewind action at end of run, and owncode exits for various options.

The output file is declared by the OUT directive. Only the logical file name need be specified. Other parameters that can be specified are similar to those in the INP directive. In addition, the KEY parameter indicates the location of the primary key and whether or not the key is embedded in the record. The KEY parameter is required when one of the following conditions exists:

- The direct access file being created has nonembedded primary keys.
- The input file is an indexed sequential, actual key, or direct access file and the primary key for the new direct access file is not the primary key for the input file.

The format of the KEY parameter is:

- KEY=±iTm

+	The primary key is embedded in the output record.
-	The primary key is extracted from the input record and is not embedded in the output record.
i	Character position in which the primary key begins, counting by the FORM convention where the first character position is 1.
T	Key type in FORM terminology; must be X.
m	Number of characters in the primary key.

The job structure for creating the file DAFILE through the FORM utility is shown in figure 5-5.

```
NOS Operating System

Job statement
USER statement
CHARGE statement
DEFINE(DAFILE/CT=PU,M=W)
FILE(DAFILE,FO=DA,ORG=NEW,RT=F,FL=50,KL=6
  HMB=5,RB=10,EFC=3,ERL=1)
FORM(INP=INPUT,OUT=DAFILE)
- - EOR
Input data
- - EOI

NOS/BE Operating System

Job statement
ACCOUNT statement
REQUEST(DAFILE,PF)
FILE(DAFILE,FO=DA,ORG=NEW,RT=F,FL=50,KL=6,
  HMB=5,RB=10,EFC=3,ERL=1)
FORM(INP=INPUT,OUT=DAFILE)
CATALOG(DAFILE,ID=AAMUG)
*EOR
Input data
*EOI
```

Figure 5-5. Direct Access File Creation, FORM Utility

PROCESSING AN EXISTING DIRECT ACCESS FILE

After the file creation run, a direct access file can be read randomly by primary key or sequentially by order of the records in home blocks. New records can be inserted into the file and existing records can be deleted or be replaced by other records with the same primary key values. Processing of an existing file is governed by many of the FIT fields set on the file creation run. You cannot change the following FIT fields:

HMB	Number of home blocks
MBL	Maximum home block length
MNR	Minimum record length
MRL	Maximum record length
KL	Key length
RKW	Relative key word for embedded keys
RKP	Relative key position within RKW for embedded keys

The values for these fields are preserved in the FSTT and are reset in the FIT whenever the existing file is opened. Any attempt to change these fields is ignored by AAM. After the file has been opened, the MRL field can be reset to the size of the working storage area.

FIT fields that must be set the same as on the file creation run are as follows:

RT	Record type; all applicable fields must also be set.
ORG	Old/new file organization.
HRL	Hashing routine location (the address might be different, but it must be the same routine).

The logical file name (LFN) field must be set to the current logical file name for the file. This name need not be the same as the name used on the file creation run.

If you do not set the following FIT fields before opening the file, the default values are set in the fields:

FWB	First word address of the buffer; default is buffer location provided by AAM.
BFS	Buffer size; default is buffer size calculated by AAM.
CPA	Compression routine address; default is no compression of records.
DCA	Decompression routine address; default depends on the CPA field.
BCK	Block checksum; applies only if this option was selected on the file creation run; default is no checksums for read processing.

Optional FIT fields that you can set at any time before being required by a file processing statement are as follows:

DFC	Dayfile control
EFC	Error file control
ERL	Trivial error limit
KA	Key address
KP	Beginning key position
FLM	File limit
FWI	Forced write indicator
EX	Error exit
DX	End-of-data exit

The KA and KP fields are required for nonembedded keys and for random access or deletion of records with embedded keys.

Various FIT fields can be set by the file processing statements. The default value listed for a field used by one of these statements is applicable only if the field has not been set by any other statement. The current value in a FIT field is always used by a file processing statement.

ESTABLISHING THE FIT

Before an existing direct access file can be opened for processing, the FIT must be established. The CALL FILEDA statement, which is the first statement that can reference the file, causes construction of the FIT. The first parameter in this statement specifies the name of the 35-word array in which the FIT is constructed. Additional parameters specify FIT fields and values to be set in the fields. You can also use the FILE control statement to set FIT fields or to override values set by the CALL FILEDA statement.

OPENING THE FILE

The direct access file must be opened by executing a CALL OPENM statement before any data records can be accessed. The format of this statement is shown in appendix B.

Open processing is initiated by the following statement:

```
CALL OPENM (DAFIT, 'I-O')
```

Execution of this statement opens the file for both input and output processing. Specifying I-O for the second parameter sets the ON field to OLD and the PD field to IO.

The setting of the PD field determines the input/output statements that can be executed. The PD field can be set by the open request as follows:

INPUT	Only statements that read records or position the file can be executed; this is the default setting.
OUTPUT	Only statements that write new records to the file can be executed.
I-O	Any input/output statement can be executed. (If the PD field is set by any other statement, the two-character mnemonic IO must be specified.)

Values from the FSTT are stored in applicable FIT fields during open processing. FILE control statement processing and FIT consistency checking are performed. Open processing also includes verification that the existing hashing routine produces the same results for the first record in the file. When the file is opened, it is positioned at the first record in the first home block.

The first time a direct access file is opened after the creation run, the NEW setting in the old/new file (ON) field must be changed to OLD. This change can be accomplished through the FILE control statement, the CALL FILEDA statement, or the CALL STOREF statement before the file is opened. If the ON field is not changed by one of these statements, it is set to OLD by specifying INPUT, OUTPUT, or I-O for the processing direction (pd) parameter in the open request.

A nonfatal error occurs if a file access statement is attempted before the file has been opened or if the attempted statement is not consistent with the setting of the PD field. A fatal error occurs if the ON field is not set to OLD.

READING THE FILE RANDOMLY

You can read a direct access file randomly by primary key values by executing a CALL GET statement. For a random read, the file must be open for either input or input/output (the PD field is set to INPUT or IO). The format of the CALL GET statement is shown in appendix B.

When the following statement is executed, a record is read randomly from the file associated with the FIT stored in the array named DAFIT:

```
CALL GET (DAFIT)
```

The record is returned to the working storage area, which is the array defined by WSA.

You must set the WSA and KA fields prior to the GET call. The CALL FILEDA or CALL STOREF statement can be used to set the fields. The primary key value for the record to be read is stored at the location indicated by the KA field. Execution of the read request (GET call) then returns the record to the specified working storage area. If the primary key value at the location indicated by the KA field does not match any primary key value in the file, a trivial error occurs.

When a record is read from the file, the number of characters returned to the working storage area is the actual length of the record as it exists on the file. Any value set in the record length (RL) field is ignored for a read operation. After the record has been read, AAM sets the RL field to indicate the length of the record returned to the working storage area.

READING THE FILE SERIALLY

You can read records serially in a direct access file by executing a CALL GETN statement. Primary key values have no logical relationship to the order in which records are retrieved. For serial reading, the file must be open for either input or input/output (the PD field is set to INPUT or IO). The format of the CALL GETN statement is shown in appendix B.

The following statement retrieves the next record in sequence:

```
CALL GETN (DAFIT)
```

This statement returns the next physically sequential record to the working storage area, which is the array defined by WSA.

The WSA field must be set before the serial read request can be executed. If the KA field is set, the primary key of the record retrieved is returned to the location indicated by the KA field.

File position in a direct access file changes only by a sequential read request or a rewind request. When the file is first opened, or after a rewind operation, the file is positioned at the beginning of the first record in the first home block. A serial read request returns that record to the working storage area and advances the file position to the next record in the home block. Subsequent serial read requests return each data record, in order, through the relative order of the home blocks. Any overflow blocks are read after all home blocks have been read.

During serial reading of the file, intervening statements that read the file randomly, delete records, or replace records (unless the new record is a different length from the original record) do not affect the current file position. A random read request does not position the file for subsequent sequential read requests. If a record is replaced by a longer or shorter record, home block contents are disturbed and the serial read position is interrupted. If records are compressed, it is the record length after compression that must be equal in order to preserve the serial read position.

INSERTING NEW RECORDS

New records are added to an existing direct access file with the same write request format used to originally create the file. The file must be open for output or input/output (the PD field is set to OUTPUT or IO).

A new record is inserted into the file by the following statement:

```
CALL PUT (DAFIT)
```

When this statement is executed, the record in the working storage area is added to the existing file.

At the completion of the write operation, AAM sets the RL field to the number of characters in the record written to the file.

DELETING EXISTING RECORDS

A record in an existing direct access file can be logically removed from the file by executing a CALL DLTE statement. The file must be open for input/output (the PD field is set to IO). The format of this statement is shown in appendix B.

A record is deleted from the file by the following statement:

```
    CALL DLTE (DAFIT)
```

Execution of this statement deletes the record with the primary key value defined by KA.

The primary key value for the record to be deleted must be established at the location indicated by the KA field. If the primary key value does not match the primary key value of any existing record, a trivial error results.

REPLACING EXISTING RECORDS

The contents of any existing record in a direct access file can be modified and the changed record rewritten to the file by executing a CALL REPLC statement. The file must be open for input/output (the PD field is set to IO). The format of the CALL REPLC statement is shown in appendix B.

The primary key value must duplicate the primary key value for an existing record. If it does not, a trivial error occurs.

The new record, which can be larger or smaller than the existing record, must be within the minimum and maximum record lengths defined by the MNR and MRL fields in the FIT. A larger or smaller record, however, invalidates the file position if serial reading is in progress.

The following statement replaces an existing record with a new record:

```
    CALL REPLC (DAFIT)
```

The modified record is in the array defined by WSA.

CLOSING THE FILE

The last statement referencing a direct access file should be a CLOSEM. This ensures that the file statistics table contains current information and that all updated records are written to the file. The format of the CALL CLOSEM statement is shown in appendix B.

The following statement initiates close processing:

```
    CALL CLOSEM (DAFIT)
```

When this statement is executed, close processing is performed for the file whose FIT is in the array named DAFIT. The file is rewound (default setting for the CF field), and the FSTT is updated with current information.

If the error file control (EFC) field is set to 2 or 3, file statistics are written to the error file. If the dayfile control (DFC) field is set to 2 or 3, file statistics are output to the dayfile.

A trivial error occurs when a close request is issued for a file that has never been opened or for one that has been closed but neither unloaded nor reopened. File position does not change and the error exit, if specified, is taken.

SAMPLE UPDATING PROGRAM

Program DAUPDAT, shown in figure 5-6, accesses the existing direct access file DAFILE through direct calls to AAM. Input records, which are read interactively from a terminal, contain three fields: the primary key, a code identifying the transaction to be performed, and data to be read into the working storage area. Only the transaction code is required input data. Code numbers are interpreted as follows:

Code 1	Find quantity on hand for a given part number.
Code 2	Insert new record into the file.
Code 3	Delete an existing record from the file.
Code 4	Replace an existing record in the file with new data.

After all input records have been read, the direct access file is rewound to beginning-of-information and read serially.

Program statements related to processing the existing direct access file are defined as follows:

- CALL FILEDA (DAFIT, 'LFN', 'DAFILE', 'ORG', . . .)

  This statement sets FIT fields required for processing the existing file:

  FIT array (DAFIT)

  Logical file name (DAFILE)

  Extended file organization (NEW)

- CALL OPENM (DAFIT, 'I-O')

  This statement opens the file for input/output processing; the ON field is set to OLD and the PD field is set to IO.

- CALL GET (DAFIT)

  This statement reads a record randomly and returns it to the working storage area DAWSA. The variable DAKEY contains the primary key for the record to be read; the key begins in character position 4 of DAKEY.

- CALL PUT (DAFIT)

  This statement inserts into the file the new record in the working storage area DAWSA. The primary key is embedded in the record.

- CALL DLTE (DAFIT)

  This statement deletes from the file the record identified by the primary key value in the variable DAKEY beginning in character position 4.

- CALL REPLC (DAFIT)

  This statement replaces an existing record with the updated record in the working storage area DAWSA. The primary key for the record to be replaced is embedded in the new record.

- CALL GETN (DAFIT)

  This statement reads records serially. Each record retrieved is returned to the working storage area DAWSA.

- IF (IFETCH (DAFIT, 'FP') .NE. 0"20") GO TO 90

  This statement checks the file position (FP) field in the FIT for a valid read. The FP field is set to 20_8 when a record is read from the file.

- CALL CLOSEM (DAFIT)

  This statement writes the updated FSTT and any home blocks in the buffer to the file.

Figure 5-6 also shows the printed output from program DAUPDAT. The first part of the output lists the input records. The second part of the output lists all records serially by position within the file. The last field in the records contains numbers indicating the order in which the records were written to the file. The following transactions were performed on file DAFILE:

- A random read for part number CB0168 returned record number 5 to the working storage area; the quantity on hand and the item description were printed on the output listing.
- A new record for part number AB5976 was added to the file (record number 20).
- The record for part number ST0592 was deleted from the file (record number 14).
- The record for part number CH0060 was updated to reflect a price change (record number 10).

```
A.  NOS Operating System                NOS/BE Operating System

    Job statement                       Job statement
    USER statement                      ACCOUNT statement
    CHARGE statement                    FTN,R.
    FTN,R.                              ATTACH(DAFILE,ID=AAMUG)
    ATTACH(DAFILE/UN=userno,M=W)        LGO.
    LGO.                                CRMEP(LO,RU)
    CRMEP(LO,RU)

B.  Input File:  DAFILE

    AB5972   METAL DESK           038995   004    1
    AB5973   OAK DESK             128250   002    2
    AB5975   WALNUT DESK          130000   002    3
    BB0013   BULLETIN BOARD       001500   010    4
    CB0168   CHALK BOARD          001952   007    5
    CB1001   FILE CAB, 1 DRAWER   004500   004    6
    CB1003   FILE CAB, 3 DRAWER   006000   010    7
    CB1005   FILE CAB, 5 DRAWER   009000   010    8
    CH0059   ARM CHAIR            029500   007    9
    CH0060   DESK CHAIR           014995   005   10
    CH0080   SWIVEL CHAIR         009600   009   11
    CM0575   LETTER RACK          000398   048   12
    SH0011   BOOK CASE            003995   010   13
    ST0592   STOOL                001620   011   14
    TY5015   TYPEWRITER           036900   002   15
    XM6158   COFFEE TABLE         006500   008   16
    YB0020   DESK LAMP            001995   495   17
    YB0059   FLOOR LAMP           006995   025   18
    YB0060   TABLE LAMP           003995   020   19
```

Figure 5-6. Processing an Existing Direct Access File (Sheet 1 of 3)

C. <u>Source Program</u>

```
      PROGRAM DAUPDAT
      IMPLICIT INTEGER (A-Z)
      DIMENSION DAFIT (35), DAWSA (5)
      CALL FILEDA (DAFIT, 'LFN', 'DAFILE', 'ORG', 'NEW',
     +             'WSA', DAWSA,
     +             'KA', DAKEY, 'KP', 4,
     +             'EFC', 3, 'DFC', 3)
      CALL OPENM (DAFIT, 'I-O')
   10 READ (*, 100, END = 70) DAKEY, CODE, DAWSA
      PRINT 100, DAKEY, CODE, DAWSA
C
C  READ EXISTING RECORD FROM THE FILE
C
      IF (CODE .EQ. 1) THEN
        CALL GET (DAFIT)
        PRINT 110, DAWSA (5), DAWSA (2), DAWSA (3)
        GO TO 10
      END IF
C
C  ADD NEW RECORD TO THE FILE
C
      IF (CODE .EQ. 2) THEN
        CALL PUT (DAFIT)
        GO TO 10
      END IF
C
C  DELETE EXISTING RECORD FROM THE FILE
C
      IF (CODE .EQ. 3) THEN
        CALL DLTE (DAFIT)
        GO TO 10
      END IF

C
C  REPLACE EXISTING RECORD WITH NEW RECORD
C
      IF (CODE .EQ. 4) THEN
        CALL REPLC (DAFIT)
        GO TO 10
      END IF
C
C  REWIND FILE FOR SEQUENTIAL READING
C
   70 CALL REWND (DAFIT)
      PRINT 130
   80 CALL GETN (DAFIT)
      IF (IFETCH (DAFIT, 'FP') .NE. O"20") GO TO 90
      PRINT 120, DAWSA
      GO TO 80
   90 CALL CLOSEM (DAFIT)
      STOP
  100 FORMAT (A10, 3X, I1, 6X, 5A10)
  110 FORMAT (' REQUESTED QUANTITY IS ', A3, ' FOR ', 2A10)
  120 FORMAT (1X, 5A10)
  130 FORMAT ('PART NO      DESCRIPTION     PRICE     QTY   REC NO ')
      END
```

Figure 5-6. Processing an Existing Direct Access File (Sheet 2 of 3)

D. Sample Output

```
/
?
   CB0168   1 <------------------------------------------------------ Read this record
 REQUESTED QUANTITY IS 007 FOR CHALK BOARD
?
            2       AB5976    MAPLE DESK         119500    003       20 <-- Add record
?
   ST0592   3 <------------------------------------------------------ Delete this record
?
   CH0060   4       CH0060    DESK CHAIR         016550    005       10 <-- Replace this record
?
PART NO        DESCRIPTION       PRICE     QTY   REC NO  \
 AB5973     OAK DESK             128250    002      2     |
 CB0168     CHALK BOARD          001952    007      5     |
 CH0060     DESK CHAIR           016550    005     10     |
 CH0080     SWIVEL CHAIR         009600    009     11     |
 SH0011     BOOK CASE            003995    010     13     |
 TY5015     TYPEWRITER           036900    002     15     |
 YB0060     TABLE LAMP           003995    020     19     |
 AB5975     WALNUT DESK          130000    002      3     |
 CB1001     FILE CAB, 1 DRAWER   004500    004      6      > ----------------- Updated File
 AB5972     METAL DESK           038995    004      1     |
 CB1005     FILE CAB, 5 DRAWER   009000    010      8     |
 CH0059     ARM CHAIR            029500    007      9     |
 CM0575     LETTER RACK          000398    048     12     |
 AB5976     MAPLE DESK           119500    003     20     |
 BB0013     BULLETIN BOARD       001500    010      4     |
 CB1003     FILE CAB, 3 DRAWER   006000    010      7     |
 XM6158     COFFEE TABLE         006500    008     16     |
 YB0020     DESK LAMP            001995    495     17     |
 YB0059     FLOOR LAMP           006995    025     18    /
```

Figure 5-6. Processing an Existing Direct Access File (Sheet 3 of 3)

ACTUAL KEY FILE PROCESSING 6

An actual key file is a mass storage file in which each record is stored in a location determined from the value of the primary key associated with that record. Record access is typically random by key value. Actual key file organization provides fast random access to records in the file. This file organization is particularly useful for applications with alternate key access, especially those involving large data bases in which many alternate keys exist.

The requirements for creating and processing an actual key file by primary key are discussed in this section. Refer to section 4 for alternate key processing of an actual key file. If alternate keys are defined, the data file must conform to the requirements discussed in this section.

Records in an actual key file are stored in data blocks. Within a data block, records are stored serially in record slots. The block number and slot number for a particular record determine the primary key value of that record. The primary key is a record number that is converted by AAM to a block and slot number. Primary keys need not be contained within the records. Key values generated by AAM are returned to your program. If you do not use alternate keys, random access is possible only through primary keys and you must preserve key values.

Mass storage residence is required for creating and processing an actual key file. For backup purposes, the file can be copied to tape through the COPYBF utility or a permanent file dump routine. Because all locations of blocks and records within blocks are maintained by relative addresses, the file can later be restored on mass storage.

The FORM utility can be used to create an actual key file from a sequential, direct access, or indexed sequential file. It can also re-create an existing actual key file in the same or different file structure.

CONCEPTS OF LOGICAL FILE STRUCTURE

An actual key file consists of an internal file statistics table (FSTT) and data blocks containing records. The FSTT is created and used by AAM; you need be aware only that the FSTT exists and that it contains certain information related to the file. Data block size is defined by you; however, you cannot manipulate information in the blocks. Because the primary key is a record number, the primary key length determines the maximum number of records that the file can contain. For example, a key length of two characters limits the file to 4095 records.

DATA BLOCKS

Records are grouped into data blocks for efficiency in processing. All data blocks are the same size and contain the same number of record slots. Records can be fixed or variable in length. Each block contains a two-word header, data records, padding, and record pointers.

Data records are stored in data blocks according to the block number and record slot number determined from the primary key value. Data blocks are created beginning with block 0 and continuing in consecutive order. All record slots in a data block need not be filled before the next block can be created; however, blocks must be created in order.

Padding for the data block can be specified at file creation time. This is useful when variable-length records are defined for the file. Padding allows for an increase in the average record size.

Record pointers are stored at the end of the data block beginning with the last word in the block. Two record pointers are stored in one word. One record pointer is required for each record in the block.

Figure 6-1 illustrates the structure of an actual key data block. The two-word block header is followed by two records in slot numbers 1 and 2, an overflow pointer record in slot number 3, and two records in slot numbers 4 and 5. The record for slot 3 had to be written in another block because sufficient empty space did not exist in the block. Record pointers are stored in the last three words of the block.

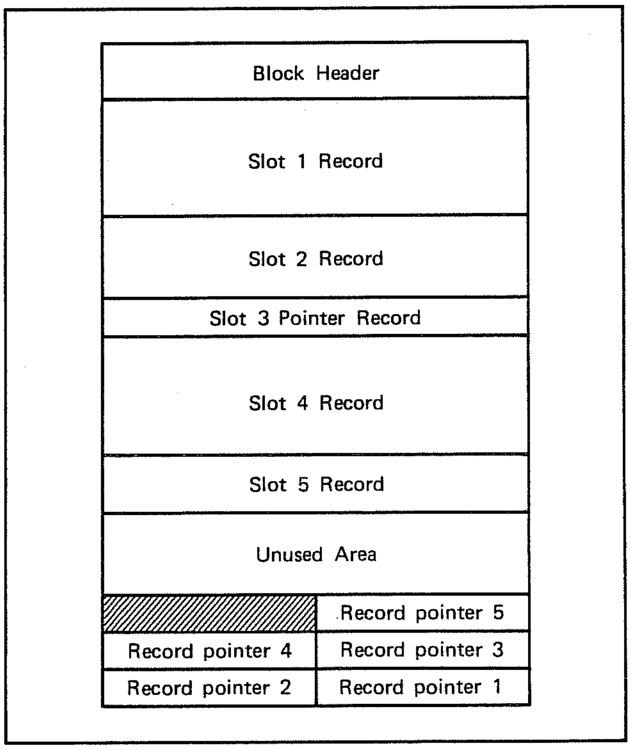


Figure 6-1. Actual Key File Data Block Structure

OVERFLOW RECORDS

When a data block does not have enough empty space for a record, an overflow pointer record is written in the designated record slot. The actual record is written in another data block that has sufficient empty space. The record slot containing the actual record is still logically empty; a read request for that slot does not return a record. A write request for that slot stores the record in its designated slot and the overflow record is moved to its data block, if possible, or to another empty record slot. The overflow pointer record is either replaced by the record or updated to point to the new location of the record.

SPECIFYING FILE STRUCTURE

The structure of an actual key file is defined when the file is created and cannot be changed for the life of the file. File structure is defined in terms of:

Record size

Primary key size

Blocking factor

Data block size

RECORD DEFINITION

Record structure must be established before the file is opened on the creation run. If you set the record type (RT) field in the FIT, the default of W type records is set in the RT field. However, AAM processes the records as U type records. Other FIT fields required for record definition depend on the record type selected.

The FIT fields unique to a record type are described in the discussion of record definition in section 3. The minimum and maximum number of characters in any record in the file are specified by the MNR and MRL fields. The value of the MNR field cannot be zero and cannot exceed the value of the MRL field. For the life of the file, any record larger or smaller than these limits is rejected as a trivial error.

PRIMARY KEY DEFINITION

The primary key for a record in an actual key file is a record number that AAM converts to the block and record slot number in which the record is stored. The key need not be contained within the record.

Fields in the FIT needed for primary key definition depend on whether the key is contained within the record. If it is not, four FIT fields are required for processing by primary key:

EMK	Embedded key. Set to NO (default).
KL	Key length; number of characters in the primary key, 1 through 8.
KA	Key address; location of the word containing the primary key.
KP	Beginning key position; starting character position of the key within the word indicated by the KA field; default is 0.

If the primary key is embedded in the record, the FIT fields required for primary key definition are as follows:

EMK	Embedded key; must be set to YES.
RKW	Relative key word; word within the record in which the primary key begins, counting from 0; default is 0.
RKP	Relative key position; character position within the relative key word in which the primary key begins, counting from 0; default is 0.
KL	Key length; number of characters in the primary key, 1 through 8.

DATA BLOCK DEFINITION

You can either specify the size of the data blocks directly or accept the default block size.

To specify the block size directly, set the maximum block length (MBL) field and the records per block (RB) field as follows:

1. Set the maximum block length by multiplying the average number of characters per record (RL) by an estimated number of records per block (RB). Start with 3 to 10 records per block to favor random processing and keep central memory usage low. Round RL x RB up to a PRU multiple (640 characters) and subtract 50 characters that are needed by AAM. The resulting MBL characters is the physical limitation on the size of data blocks.

2. Adjust the number of records per block used in step 1 so that RB times RL is as close to MBL as possible. By specifying a value for RB, you set the maximum number of records that can be stored logically within each data block that has MBL characters.

For example:

Suppose you have records that each consist of 300 characters (RL = 300). With four records per block, you need data blocks that are 1870 characters or 3 PRUs in length.

[(300 x 4) rounded to PRU multiple]
- 50 characters = 1870

Set MBL to 1870.

Adjust the number of records per block (RB) to six, because six 300 character records (1800 characters) is closer to the MBL value specified (1870).

The interaction of the RB field with the key length (KL) field determines the maximum number of data blocks and records in the file. The KL field sets a limit on the number of records in the file. This limit divided by the RB field is the theoretical maximum number of data blocks in the file. Any remainder from the division is dropped.

Key length values produce theoretical file limits, as follows:

KL=1	record maximum is 63
KL=2	record maximum is 4095
KL=3	record maximum is 262143
KL=4	record maximum is 16777215

NOTE

If you set RB too small, data block space is wasted. If you set RB too large, logical key values are wasted (that is, not all available slot numbers are used). You must then increase the key length to accommodate the same number of records. Figure 6-2 illustrates the concept.

Alternatively, you can accept the default block size or blocking factor. If an installation default has not been defined, AAM calculates defaults for MBL and RB.

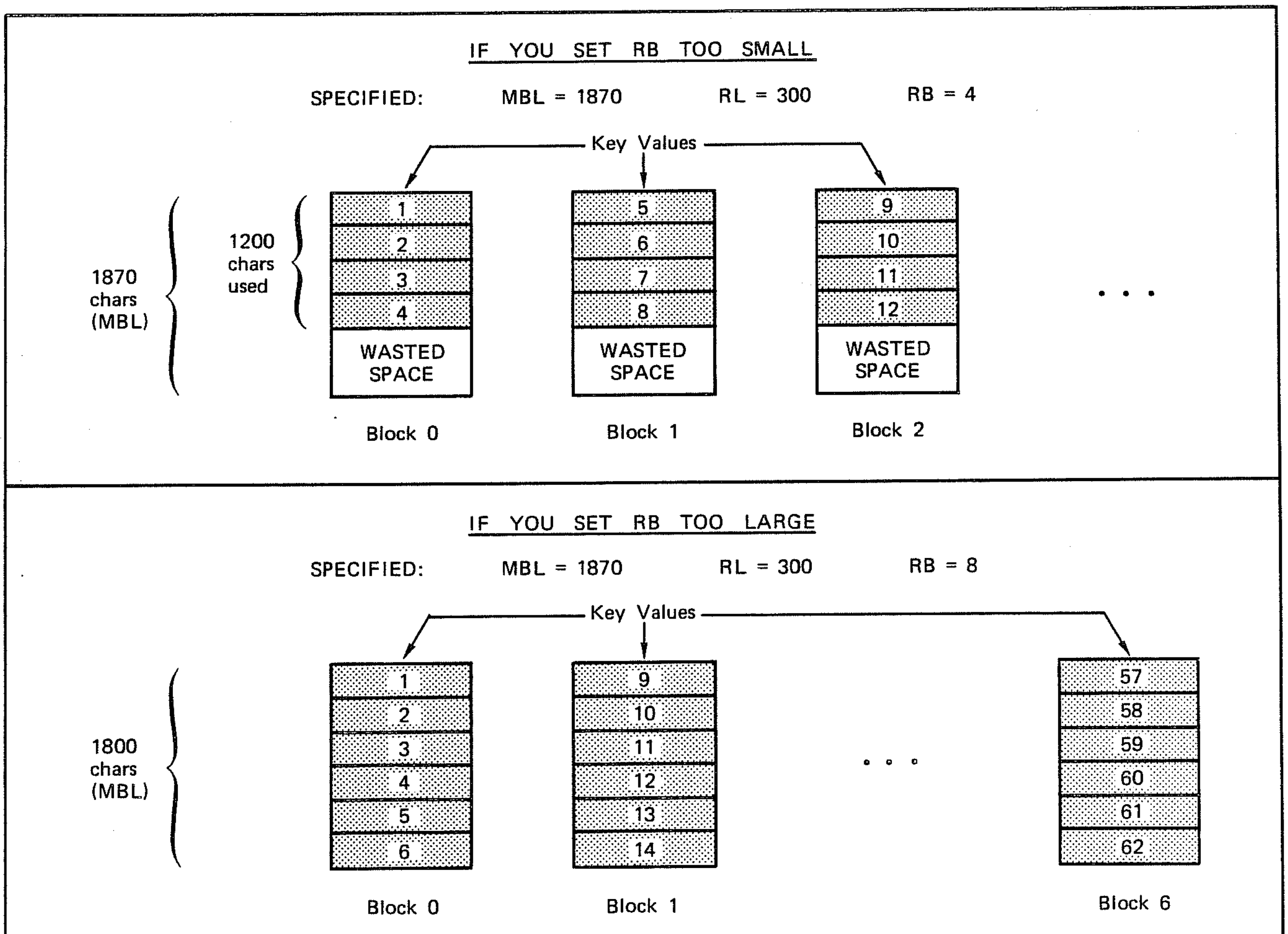
If you specify MBL but do not specify RB (or set it to zero), AAM decides on the appropriate blocking factor. AAM sets RB to one plus the number of average records that would fit into MBL characters.

If you do not specify the MBL field or the RB field (or set them to zero), AAM sets RB to the larger of the following:

- The number of average records that would fit into a 4-PRU block
- Eight average size records

If you do not specify MBL and you do specify RB, AAM calulates the default block size as follows:

1. The mean of MRL and MNR values are used as the average record length. If MNR is not defined, it is assumed to be zero.
2. If the RB field equals one, AAM sets MBL to either 2 average size records or the smallest number of PRUs that would contain RB average records.
3. If RB is specified as anything except one, AAM sets the MBL value to the number of characters in the smallest number of PRUs that would contain RB average records.



Only the first 6 slot numbers in each block can be assigned. The last two logical key values in each block are wasted. In other words, slot numbers 7, 8, 15, 16, . . . cannot be used.

If KL = 1, the theoretical file limits would be 63, records and 7 data blocks (63 ÷ 8 = 7).

However, with RB = 8 only 6 x 7 records can be put in this file. The actual file limit would be 42 records. KL would have to be increased to 2 to accomodate 63 records. Since this would require more space, it would be better to decrease RB to 6.

Figure 6-2. Setting the Data Block Size

You can also specify padding for the data blocks. This allows for an increase in the average record size. After block size is determined by AAM, padding is added to the value in the MBL field. You specify padding by the DP field.

The final block size must be large enough to hold the record pointers (two per word), two words for the block header, and the number of average size records per block specified by the RB field. AAM increases the MBL value, if necessary to use mass storage efficiently. Resulting blocks are always an integral multiple of PRU size (640 characters) minus two words:

[(Specified MBL + 50 characters) rounded to the next PRU multiple]- 20 characters

If you want to set MBL to specify a certain number of PRUs per block, you should use the values 590, 1230, 1870, 2510, and so on. That is, you should set MBL to a PRU multiple less 50 characters.

CREATING AN ACTUAL KEY FILE

An actual key file must be created on a separate file creation run. The file can be created by a source program or by the FORM utility. The file statistics table (FSTT), which becomes a permanent part of the file, is created when the actual key file is created.

For the file creation run, you must set the old/new file (ON) field in the FIT to NEW. The following FIT fields must also be defined and cannot be changed after file creation:

FO	File organization; set to AK by the CALL FILEAK statement.
ORG	Old/new file organization; must be set to NEW.
KL	Key length; number of characters in the primary key, 1 through 8.
MRL	Maximum record length; maximum number of characters in any record; set by AAM when FL is specified for F or Z type records.
MNR	Minimum record length; minimum number of characters in any record; cannot be zero; set by AAM when FL is specified for F or Z type records.
RT	Record type; default is W, which AAM processes as U; fields required by the record type must also be specified.

You must define one additional FIT field for the file creation run; however, this field can be changed on subsequent runs. The following field must be set:

LFN	Logical file name; one to seven characters, beginning with a letter.

If alternate keys are to be defined during the file creation run, you must specify the following FIT field:

XN	Index file name

Other FIT fields that are effective for the life of the file can be specified by you or you can accept default values. These fields are as follows:

RB	Records per block; number of average size records in a block for block size calculation; default is eight records per block.
MBL	Maximum block length; default is number of words required for eight average size records.
DP	Data block padding; default is zero percent padding.
BCK	Block checksum; default is no checksums.
EMK	Embedded key; must be set to YES if the primary key is embedded within the record; default is NO. (NO is more efficient.)

If primary keys are embedded, two additional FIT fields are required:

RKW	Relative key word in which the primary key begins; default is word 0.
RKP	Relative key position within the word in which the key begins, default is character position 0.

Optional FIT fields that can be supplied on the creation run and can be changed at any time during subsequent file access runs are as follows:

ERL	Trivial error limit; default is no limit.
FLM	File limit; default is no limit.
DFC	Dayfile control; default is only fatal error messages to the dayfile.
EFC	Error file control; default is no messages to the error file.
FWI	Forced write indicator; default is buffers are written only when space is needed.
FWB	First word address of the buffer; default is buffer address provided by AAM.
BFS	Buffer size; default is buffer size calculated by AAM.
CPA	Compression routine address; default is no compression of records.
DCA	Decompression routine address; default depends on the CPA field (refer to appendix F).
DX	End-of-data exit; default is no exit subroutine.
EX	Error exit; default is no exit subroutine.

FILE CREATION BY A SOURCE PROGRAM

When you create an actual key file by a source program, you must define file structure and key characteristics by setting applicable fields in the FIT before the file is opened. You can use the FILE control statement, the CALL FILEAK statement, and the CALL STOREF statement to set fields in the FIT.

The old/new file (ON) field must be set to NEW by one of the FIT manipulation statements or by opening the file with the processing direction (pd) parameter in the open request set to NEW. Setting the pd parameter to NEW actually sets two fields in the FIT. The ON field is set to NEW and the processing direction (PD) field is set to OUTPUT. The program can then insert records into the file with write requests.

You should close the file after all records have been written to the file. On a file creation run, only those statements that establish the FIT, open and close the file, write records, and read and write fields in the FIT can be issued.

Establishing the FIT

The CALL FILEAK statement must be the first statement that references the actual key file. When it is executed, the FIT is constructed and FIT field values specified in the statement are stored in the FIT. The first parameter in the CALL FILEAK statement is the name of the 35-word array to hold the FIT. The same FIT array name is the first parameter in every statement accessing the actual key file. Refer to section 2 for a more detailed description of the FIT and the CALL FILEAK statement.

Opening the File

Before any records can be written, you must open the actual key file by executing a CALL OPENM statement to prepare the file for processing. The format of this statement is as follows:

```
CALL OPENM (fit,pd)
```

The name of the array containing the FIT is the first parameter in the open request and is a required parameter. The second parameter sets the processing direction (PD) and old/new file (ON) fields in the FIT. If the PD and ON fields are set before the file is opened, the second parameter can be omitted; otherwise, it must specify NEW.

Values from the FILE control statement are stored in the appropriate fields in the FIT overriding existing values. Default values are supplied for fields not set by the user, buffer fields are processed, and the FIT is checked for consistency in logic and for required fields.

The following statement initiates open processing:

```
CALL OPENM (AKFIT, 'NEW')
```

In this statement, the array named AKFIT contains the FIT for the actual key file being opened. The second parameter causes the PD field to be set to OUTPUT and the ON field to be set to NEW.

Writing Records

Records are written to an actual key file by executing a CALL PUT statement. The format of this statement is shown in appendix B.

The following statement writes a record to the file associated with the FIT in the array named AKFIT:

```
CALL PUT (AKFIT)
```

Any of the FIT fields not set by the write request default to the current value in the FIT. When the record has been written to the file, AAM sets the RL field to the number of characters in the record.

Records written to an actual key file are positioned within the file according to the value of the primary key supplied by the user or, if the primary key value has been set to zero, in the location determined by AAM; in the latter case, the corresponding primary key value is returned to the user. If the file has nonembedded keys, the key address (KA) and key position (KP) fields inform AAM where to find the user-supplied key or where to return the system-supplied key. If the file has embedded keys, the location is indicated by the relative key word (RKW) and relative key position (RKP) fields as specified when the file was first opened.

The recommended method is to allow AAM to provide the key values. Before issuing a write request, set the key location, pointed to by the KA and KP fields or the RKW and RKP fields, to zero. The write request then causes AAM to find a suitable position in the file, store the corresponding key value in the key location, and then copy the record to that position in the file. You are responsible for recording the key value if the file is to be accessed by primary key at any subsequent time.

When you supply key values, the value at the key location must be a unique record number. Data blocks must be created in ascending numeric order. For example, blocks 0 through 16 must already exist before block 17 can be created. Record positions in each block need not be filled in ascending position value; however, processing is more efficient when the records are in primary key order. An error occurs if the user-supplied key value is not unique or if it specifies an invalid block number.

The record to be written to the file must be established in the working storage area. If the primary key is not embedded, the location to provide or receive the primary key must also be established. For embedded keys, the key location is determined by the RKW and RKP fields.

Closing the File

After all records have been written to the file, a CALL CLOSEM statement must be executed to ensure file integrity. The format of this statement is shown in appendix B.

The following statement initiates close processing:

```
CALL CLOSEM (AKFIT)
```

When this statement is executed, the file whose FIT is in the array named AKFIT is closed and rewound to beginning-to-information. The OC flag is set to closed. FIT verification and FILE control statement processing are not repeated if the file is subsequently reopened.

If a close request is not executed, the FSTT, which is used to maintain continuity over the life of the file, might be impaired. This could interfere with system checks that prevent inadvertent destruction of the file.

When the file is closed, any blocks in the central memory buffer are written to the file, the FSTT is updated, and the FSTT is written to the file. File statistics are written to the error file, if requested, and to the dayfile.

A trivial error results if the close request is issued for a file that has never been opened or that has been closed but neither unloaded nor reopened. The file is positioned as specified before the error is issued.

Sample Creation Program

Program NEWAK, shown in figure 6-3, creates an actual key file through direct calls to AAM. Records are read from the input file ZOO and written to the new actual key file AKFILE. The program prints each input record on the file OUTPUT to show the records used to create the file.

Figure 6-3 also shows the control statements used to create the actual key file. The CRMEP control statement, which is described in detail in section 7, is used to write the error file to the file OUTPUT.

Program statements that are related to creation of the actual key file are defined as follows:

- DIMENSION AKFIT(35), AKWSA(8)

  This statement sets up two arrays for the direct calls. The first array, AKFIT, is allocated 35 words for the FIT. The second array is allocated 8 words for the working storage area.

- CALL FILEAK (AKFIT, 'LFN', 'AKFILE',...)

  This statement sets fields in the FIT to describe file structure and sets other file processing parameters:

  Fit array (AKFIT)

  Logical file name (AKFILE)

  Extended file organization (NEW)

  Working storage area (AKWSA)

  Record type (trailer) and other related characteristics of T type records (HL, TL, CP, CL)

  Minimum and maximum record lengths (MNR and MRL fields)

  Key address (KEY)

  Primary key length (2 characters)

  Embedded key and location (EMK, RKW, and RKP fields)

  Average size records per block (16 records)

  Optional parameters include:

  Error file control (3, errors and notes)

  Dayfile control (3, errors and notes)

- CALL OPENM (AKFIT, 'NEW')

  This statement opens the file, sets the PD field to OUTPUT and sets the ON field to NEW for a file creation run. Records can only be written to the file.

- CALL PUT (AKFIT)

  This statement writes the record in the working storage area AKWSA to the actual key file. The KA and KP parameters are omitted because keys are embedded.

- CALL CLOSEM (AKFIT)

  This statement initiates close processing, which includes writing the FSTT and any data blocks in the buffer to the file.

FILE CREATION THROUGH FORM

You can create an actual key file through the FORM utility. An existing actual key file can be restructured or dumped to tape for storage or backup purposes by the FORM utility. The following paragraphs briefly discuss using FORM to create an actual key file. Refer to the FORM reference manual for more detailed information.

FORM uses CYBER Record Manager routines to perform input on the input file and output on the actual key file being created. FILE control statements are used to provide descriptions of the input and output files. The input file can be any file organization except word addressable. FILE control statement parameters for the input file depend on the file organization. For the actual key file being created, you must specify on the FILE control statement the logical file name and the following FIT fields:

FO	File organization; must be FO=AK for actual key file organization.
ORG	Old/new (initial or extended) file organization; must be ORG=NEW.
KL	Key length; number of characters in the primary key.
MRL	Maximum record length; maximum number of characters in any record; set by AAM when FL is specified for F or Z type records.
MNR	Minimum record length; minimum number of characters in any record; set by AAM when FL is specified for F or Z type records.
RT	Record type; default is W, which AAM processes as U; fields required by the record type must also be specified.

You cannot supply primary key values for an actual key file being created through FORM. If the system-supplied primary key is to be embedded in the output record, the following FIT fields are also required:

EMK	Embedded key; must be EMK=YES.
RKW	Relative key word in which the key begins; default is word 0.
RKP	Relative key position within the word in which the key begins; default is character position 0.

A. NOS Operating System

```
Job statement
USER statement
CHARGE statement
FTN5.
ATTACH(ZOO/UN=userno)
DEFINE(AKFILE/CT=PU,M=W)
LGO.
CRMEP(LO,RU)
```

NOS/BE Operating System

```
Job statement
ACCOUNT statement
FTN5.
ATTACH(ZOO,ID=AAMUG)
REQUEST(AKFILE,PF)
LGO.
CATALOG(AKFILE,ID=AAMUG)
CRMEP(LO,RU)
```

B. Input File: ZOO

```
 1MONKEY    BANABA MIX6 JOE   BIMBA TOMMY SUSIE SALLY JIMBO
 5POLAR BEARFISH      2 SAM   PEARL
12BABOON    BANANA MIX5 JANEY BOBBY BUCKO MANDY CARRIE
18GOAT      MASH      4 BILLY BETTY BUTTS BECKY
24SHEEP     MASH      1 CURLY
32RACCOON   MEAT MIX  3 RINGO KINKY KELLY
41PENGUIN   FISH MIX  2 PENNY MR MAC
62BROWNBEAR MEAT MIX  2 ROCKY TESSA
```

C. Source Program

```
      PROGRAM NEWAK
C
C  ************************************************************
C  * THIS PROGRAM ILLUSTRATES THE CREATION OF AN AK FILE      *
C  * (AKFILE) FROM A SEQUENTIAL FILE (ZOO).  EACH RECORD      *
C  * WRITTEN IS ALSO PRINTED AS OUTPUT.                       *
C  ************************************************************
C
      IMPLICIT INTEGER (A-Z)
      DIMENSION AKFIT (35), AKWSA(8)
      CALL FILEAK (AKFIT, 'LFN', 'AKFILE', 'ORG', 'NEW',
     +            'WSA', AKWSA,
     +            'RT', 'T', 'HL', 32, 'TL', 6, 'CP', 30, 'CL', 1,
     +            'MNR', 32, 'MRL', 80,
     +            'KA', KEY, 'KL', 2, 'EMK', 'YES', 'RKW', 0, 'RKP', 8,
     +            'EFC', 3, 'DFC', 3)
      CALL OPENM (AKFIT, 'NEW')
      IF (IFETCH (AKFIT, 'ES') .NE. 0) GO TO 50
      OPEN (2, FILE= 'ZOO')
      KEY = 0
   10 READ (2, '(I10, 7A10)', END = 30) AKWSA
      CALL PUT (AKFIT)
      IF (IFETCH (AKFIT, 'ES') .NE. 0) GO TO 50
      PRINT 100, AKWSA
      GO TO 10
   30 CALL CLOSEM (AKFIT)
      STOP
   50 PRINT 902, IFETCH (AKFIT, 'ES')
      CALL CLOSEM (AKFIT)
      STOP 'CRM ERROR RETURNED'
  100 FORMAT (I10, 7A10)
  902 FORMAT ('ES = ', O3)
      END
```

D. Output

(same as input)

Figure 6-3. Actual Key File Creation, Source Program

Other FIT fields not specified in the FILE control statement are set to the default values. The FORM directive OUT can specify values for the following FIT fields:

FLM	File limit; maximum number of records for the actual key file; default is no limit.
CPA	Compression routine address; number or entry point name of the compression routine; default is no compression of records.
EX	Error exit; entry point name of the user-supplied error processing routine; default is no error exit subroutine.

A job using the FORM utility to create an actual key file must contain the following:

- A FILE control statement describing the input file structure unless the file is a sequential file with Z type records and C type blocks
- A FILE control statement describing the new actual key file structure .
- A FORM control statement
- FORM directives

The FORM directives INP and OUT identify the input and output files, respectively. If primary keys are embedded and you use only the INP and OUT directives, they can be specified in the FORM control statement. Other FORM directives are available to specify record selection criteria, reformatting, and conversion. Consult the FORM reference manual for details of optional directives that might be useful.

The source of input records is specified in the INP directive. The logical file name is the only required parameter. Optional parameters specify the maximum number of records to be processed, rewind action at end of run, and owncode routines for various options. The input file structure determines the parameters that are used.

The OUT directive specifies the logical file name of the actual key file to be generated by FORM. Optional parameters similar to those in the INP directive can also be specified. The KEY parameter is required only when the system-supplied primary key values are to be inserted in the actual key file records. The format of the KEY parameter is:

KEY+iT

+	The primary key is embedded in the output record.
i	Character position in which the primary key begins, counting by the FORM convention where the first character position is 1.
T	Key type in FORM terminology; must be I for an integer key.

When you access an actual key file randomly by primary key values, you must preserve the keys. As each record is written, FORM returns the key value to location KEYA. A second file can be created to hold the key values. The REF directive can be used to reformat the data for the second file. Because FORM creates the files in the order the directives are encountered, the directives for the file of primary key values must follow the directives for the actual key file.

Figure 6-4 shows the control statements and FORM directives used to create an actual key file.

NOS Operating System

```
Job statement
USER statement
CHARGE statement
DEFINE(AKFILE/CT=PU,M=W)
DEFINE(KEYFILE/CT=PU,M=W)
FILE(AKFILE,FO=AK,RT=T,KL=2,MNR=32,MRL=80,
  DP=15,HL=32,TL=6)
FILE(AKFILE,CP=30,CL=1,RB=16,ORG=NEW)
FILE(KEYFILE,FO=IS,RT=F,KT=U,KL=10,FL=20,
  ORG=NEW,EMK=YES)
FORM.
- - EOR
INP(INPUT)
OUT(AKFILE)
OUT(KEYFILE)
REF(KEYFILE,1X10=11X10,11I=KEYA)
- - EOR
Input data
- - EOI
```

NOS/BE Operating System

```
Job statement
ACCOUNT statement
REQUEST(AKFILE,PF)
REQUEST(KEYFILE,PF)
FILE(AKFILE,FO=AK,RT=T,KL=2,MNR=32,MRL=80,
  DP=15,HL=32,TL=6)
FILE(AKFILE,CP=30,CL=1,RB=16,ORG=NEW)
FILE(KEYFILE,FO=IS,RT=F,KT=U,KL=10,FL=20,
  ORG=NEW,EMK=YES)
FORM.
*EOR
INP(INPUT)
OUT(AKFILE)
OUT(KEYFILE)
REF(KEYFILE,1X10=11X10,11I=KEYA)
*EOR
Input data
*EOI
```

Figure 6-4. Actual Key File Creation, FORM Utility

PROCESSING AN EXISTING ACTUAL KEY FILE

An existing actual key file can be read randomly by primary key or sequentially by the order of records and blocks. New records can be added and existing records can be deleted or be replaced by other records with the same primary key values. File processing is governed by many of the FIT fields set before the file was opened on the creation run. The following FIT fields cannot be changed:

MBL	Maximum block length
RB	Records per block
MNR	Minimum record length
MRL	Maximum record length
KL	Key length

RKW	Relative key word for embedded keys
RKP	Relative key position within RKW for embedded keys

When the file is opened, values for these fields are returned from the FSTT to the FIT. Any attempt to change key, record, or block length is ignored by AAM.

FIT fields that must be set the same as on the file creation run are as follows:

RT	Record type; all applicable fields must also be set.
ORG	Old/new (initial or extended) file organization.

The logical file name (LFN) field must be set to the current logical file name for the file. This name need not be the same as the name used on the file creation run.

If alternate keys have been defined, you must specify the following FIT field:

XN	Index file name

If you do not set the following FIT fields before the file is opened, the fields are set to default values:

FWB	First word address of the buffer; default is buffer location provided by AAM.
BFS	Buffer size; default is buffer size calculated by AAM.
CPA	Compression routine address; default is no compression of records.
DCA	Decompression routine address; default depends on the CPA field.
BCK	Block checksum; applies only if this option was selected on the file creation run; default is no checksums for read processing.

The following optional fields can be set at any time before being required by a file processing statement:

DFC	Dayfile control
EFC	Error file control
ERL	Trivial error limit
KA	Key address
KP	Key position
FLM	File limit
FWI	Forced write indicator
EX	Error exit
DX	End-of-data exit

The KA and KP fields are required for nonembedded keys and for random access or deletion of records with embedded keys.

Various FIT fields can be set by the file processing statements. The default value listed for a field used by one of these statements is applicable only if the field has not been set by any other statement. The current value in a FIT field is always used by a file processing statement.

ESTABLISHING THE FIT

The first statement referencing the actual key file must be the CALL FILEAK statement. This statement causes construction of the FIT. The first parameter in the CALL FILEAK statement specifies the name of the 35-word array in which the FIT is constructed. Additional parameters set FIT fields; the logical file name (LFN) field must be set to the current logical file name by which the file is attached. Fields related to file structure are not specified because this information is saved in the FSTT and returned to the FIT when the file is opened. The FILE control statement can also be used to set FIT fields or to override values set by the CALL FILEAK statement.

OPENING THE FILE

You must open the file by executing a CALL OPENM statement before any file processing can occur. The format of this statement is shown in appendix B.

Open processing is initiated by the following statement:

```
CALL OPENM (AKFIT, 'I-O')
```

When this statement is executed, the file is opened for both input and output. Any file processing statement can be issued for the file. The ON field is set to OLD. AKFIT is the name of the array containing the FIT. In addition, the following FIT field can be set:

PD	Processing direction; type of processing to be performed; default is input processing.

The setting of this field determines the type of processing that can be performed. The PD field can be set as follows:

INPUT	Only statements that read records or position the file can be executed. This is the default setting.
OUTPUT	Only statements that write new records to the file can be executed.
I-O	Any input/output statement can be executed. (If the PD field is set by any other statement, the two-character mnemonic IO must be specified.)

When the open request is executed, values from the FSTT are stored in FIT fields, FIT control statement processing occurs, and FIT consistency checking is performed. The file is positioned at the first record in the first block.

The first time an existing file is opened, the old/new file (ON) field must be changed from NEW to OLD. The ON field can be set to OLD by the FILE control statement, the CALL FILEAK statement, or the CALL STOREF statement. The ON field is automatically set to OLD when INPUT, OUTPUT, or I-O is specified for the processing direction (pd) parameter in the open request.

If an open request is not issued, error procedures are initiated when any other file access statement for the actual key file is processed. A trivial error also occurs if a file processing statement is issued and the PD field is not set to an appropriate value for that statement.

READING THE FILE RANDOMLY

An actual key file can be read randomly by primary key values by executing a CALL GET statement. The file must be open for input or for input/output (the PD field is set to INPUT or IO). The format of the CALL GET statement is shown in appendix B.

The following statement specifies a random read of the file associated with the FIT stored in the array named AKFIT:

CALL GET (AKFIT)

The WSA and KA fields must be set either through the random read request or through the CALL FILEAK or CALL STOREF statement. The primary key value for the record to be read is stored at the location indicated by the KA field. The record is then returned to the location designated by the WSA field. If no record is stored in the data block and record slot calculated from the primary key value, a trivial error occurs and, if defined, the exit is taken.

At the completion of the read request, the record length (RL) field is set to the number of characters returned to the working storage area. Any value set in the RL field before the read request is issued is ignored; the number of characters returned is determined by the length of the record written.

READING THE FILE SEQUENTIALLY

A sequential read request returns the next record in sequence to the working storage area. Records are read sequentially by executing a CALL GETN statement. Sequencing of records in an actual key file is by block and record slot within the block. The file must be open for input or input/output (the PD field is set to INPUT or IO). The format of the CALL GETN statement is shown in appendix B.

Execution of the following statement reads the next record in sequence:

CALL GETN (AKFIT)

The WSA field must be set before the sequential read request can be executed. The KA field is an optional field; if it is specified, the primary key of the record retrieved from the file is returned to the designated location if primary keys are not embedded.

If the sequential read request is the first statement issued after the file is opened or after a rewind request, the first record in the file is retrieved. Additional sequential read requests then retrieve records in the order they are stored in the file. Empty record slots are ignored. Overflow records, which are records too large to be stored in the position indicated by the user-supplied primary key, are retrieved according to key value, not according to the position where they are actually stored.

A random read, a replace, or a delete request issued during a series of sequential read requests does not affect the sequential position of the file. If end-of-information is encountered when a sequential read request is issued, the file position (FP) field is set to 100_8. If another sequential read is executed, a trivial error results and the error exit, if specified, is taken.

INSERTING NEW RECORDS

New records are written to an existing actual key file with the same write request format used to create the file. The file must be open for output or input/output (the PD field is set to OUTPUT or IO).

Execution of the following statement inserts a new record in the file:

CALL PUT (AKFIT)

When the primary key value for the record is greater than zero, AAM converts the value to a block and record slot number in which the record can be written. The key value must be unique and must not extend the file by more than one block. If the key is not valid, the request is rejected and the error exit, if specified, is taken.

If the primary key value is zero, the record is inserted in an available record slot and the key value calculated by AAM is returned to the primary key location. You must preserve the key for future random access to the record.

DELETING EXISTING RECORDS

Records are eliminated from an actual key file by executing a CALL DLTE statement. The file must be open for input/output (the PD field is set to IO). The format of the CALL DLTE statement is shown in appendix B.

A record is deleted from the file by the following statement:

CALL DLTE (AKFIT)

The location indicated by the KA field must contain the primary key value of an existing record. If it does not, a trivial error results and any defined error exit routine is executed.

When a delete request is issued, the record header at the end of the block is physically deleted from the block. The record image, however, remains in the data portion of the block until the space is needed as a result of a write or replace request.

REPLACING EXISTING RECORDS

Any existing record in an actual key file can be modified and rewritten to the file by executing a CALL REPLC statement. The file must be open for input/output (PD field is set to IO). The format of the CALL REPLC statement is shown in appendix B.

The following statement replaces a record in the file with the record in the working storage area:

CALL REPLC (AKFIT)

The primary key value must indicate an existing record in the file. The primary key cannot be set to zero for a replace request.

The new record can be smaller or larger than the existing record; however, the record size must be within the minimum and maximum record lengths established by the MNR and MRL fields in the FIT.

CLOSING THE FILE

After completion of all file accesses, you must issue a CALL CLOSEM statement to ensure file integrity. The format of this statement is shown in appendix B.

Close processing is initiated by the following statement:

CALL CLOSEM (AKFIT)

The file is rewound and closed. If the file is reopened in the same job, FIT verification and FILE control statement processing are not repeated.

Any modified blocks in the central memory buffer are written to the file and the FSTT is updated.

A trivial error occurs when a close request is issued for a file that has never been opened or for one that has been closed but neither unloaded nor reopened. File position does not change and the error exit, if specified, is taken.

SAMPLE UPDATING PROGRAM

Program AKUPDAT, shown in figure 6-5, accesses the existing actual key file AKFILE through direct calls to AAM. Input records are read from the file NEWZOO. A transaction code (TRANS) in the input record indicates whether to delete, replace, or add a record. For replacing or adding a record, a second input record is read into the working storage area. When all input records have been read, the actual key file is rewound to beginning-of-information and read sequentially.

Program statements related to processing the existing actual key file are defined as follows:

- CALL FILEAK (AKFIT, 'LFN', 'AKFILE',...)

  This statement sets the FIT fields required for existing file processing:

  FIT array (AKFIT)

  Logical file name (AKFILE)

  Record type and related fields (must be the same as for the creation run)

  Extended file organization (NEW)

- CALL OPENM (AKFIT, 'I-O')

  This statement opens the file for input/output processing; the PD field is set to IO and the ON field is set to OLD.

- CALL DLTE (AKFIT)

  This statement deletes a record from the file. The primary key for the record to be deleted is in the variable KEY beginning in character position 8.

- CALL GET (AKFIT)

  This statement reads a record randomly and returns it to the working storage area AKWSA. The primary key for the record to be retrieved is in the variable KEY beginning in character position 8.

- CALL REPLC (AKFIT)

  This statement replaces an existing record with the record in the working storage area (AKWSA).

- CALL PUT (AKFIT)

  This statement writes the record in the working storage area AKWSA to the file.

- CALL GETN (AKFIT)

  This statement reads the next record in sequence and returns it to the working storage area AKWSA.

- IF (IFETCH (AKFIT, 'FP') .NE. 0"20") GO TO 80

  This statement checks the file position (FP) field in the FIT for a valid read. The FP field is set to 20_8 when a record is read from the file.

- CALL CLOSEM (AKFIT)

  This statement initiates close processing, which includes updating the FSTT and writing any data blocks in the buffer to the file.

Printed output from program AKUPDAT is shown in part D of figure 6-5. The first part of the listing shows the input records and the record read by the random read request. The last part of the listing shows the contents of the updated actual key file. This can be compared with the original file contents shown in part B of the figure. The updated file reflects the following transaction processing:

- Record number 5 was deleted and record number 27 was inserted because the animals in cage 5 were moved to cage 27.
- Record number 45 was added for the new animals in cage 45.
- Record number 62 was replaced; the updated record contains another trailer item for the additional animal in cage 62.

POSITIONING AN ACTUAL KEY FILE

Normally, file positioning is performed for subsequent sequential access to the file in primary key order. Because the primary keys in an actual key file indicate the storage locations of the records, sequential access is meaningless in most applications. An actual key file can be positioned in the following ways:

- An open request positions the file at the beginning of the first record in the first block.
- A skip request positions the file forward or backward to the beginning of a record, as indicated by the current position and the skip count.
- A rewind request positions the file at the beginning of the first record in the file.
- A random read request positions the file at the end of the record read.

When a series of sequential read requests are being issued, the sequential file position is not affected by intervening write, replace, or delete requests.

SKIPPING RECORDS

You can position an actual key file by skipping records forward or backward by executing a CALL SKIP statement. The format of this statement is shown in appendix B.

The following statement specifies that the file is to be positioned backward three records from the current position:

```
CALL SKIP (AKFIT, -3)
```

No FIT fields are set by the skip request. Skipping stops when beginning-of-information or end-of-information is reached before the specified number of records have been skipped.

In either direction, the skip request always positions the file to the beginning of a record. Empty record positions are ignored and are not included in the skip count. An overflow record is counted when the position corresponding to its key value is encountered, not when the actual record is encountered. Only small skips should be made because AAM reads and counts each intervening record, thus increasing execution time.

REWINDING THE FILE

Execution of a CALL REWND statement positions the actual key file to the beginning of the record in the first occupied record slot in the first block. The format of this statement is shown in appendix B.

When the following statement is executed, the file is rewound to beginning-to-information:

```
CALL REWND (AKFIT)
```

The file must be opened before the rewind request can be issued. Rewinding the file is more efficient than extensive backward skipping of records. The name of the array containing the FIT is the only parameter in the rewind request.

A. NOS Operating System

```
Job statement
USER statement
CHARGE statement
FTN5.
ATTACH(NEWZOO/UN=userno)
ATTACH(AKFILE/UN=userno,M=W)
LGO.
CRMEP(LO,RU)
```

NOS/BE Operating System

```
Job statement
ACCOUNT statement
FTN5.
ATTACH (NEWZOO,ID=AAM)
ATTACH(AKFILE,ID=AAMUG)
LGO.
CRMEP(LO,RU)
```

B. Input File: AKFILE

```
 1MONKEY    BANABA MIX6 JOE   BIMBA TOMMY SUSIE SALLY JIMBO
 5POLAR BEARFISH      2 SAM   PEARL
12BABOON    BANANA MIX5 JANEY BOBBY BUCKO MANDY CARRIE
18GOAT      MASH      4 BILLY BETTY BUTTS BECKY
24SHEEP     MASH      1 CURLY
32RACCOON   MEAT MIX  3 RINGO KINKY KELLY
41PENGUIN   FISH MIX  2 PENNY MR MAC
62BROWNBEAR MEAT MIX  2 ROCKY TESSA
```

Input File: NEWZOO

```
05        -1
27         1
27POLAR BEARFISH      2 SAM   PEARL
45         1
45LION      RAW MEAT  1 LEO
62         0
62BROWN BEARMEAT MIX  3 ROCKY TESSA BRUNO
```

Figure 6-5. Processing Existing Actual Key File (Sheet 1 of 3)

C. Source Program

```
      PROGRAM AKUPDAT
C
C  ****************************************************************
C  * THIS PROGRAM ILLUSTRATES UPDATING AN EXISTING AK FILE       *
C  * (AKFILE).  UPDATE INFORMATION EXISTS ON THE INPUT FILE      *
C  * NEWZOO.  TRANSACTION CODES (TRANS) ARE AS FOLLOWS:          *
C  *                -1 MEANS DELETE A RECORD                     *
C  *                 1 MEANS ADD A RECORD                        *
C  *                 0 MEANS REPLACE A RECORD                    *
C  ****************************************************************
C
      IMPLICIT INTEGER (A-Z)
      DIMENSION AKFIT (35), AKWSA (8)
      CALL FILEAK (AKFIT, 'LFN', 'AKFILE', 'ORG', 'NEW',
     +            'WSA', AKWSA,
     +            'RT', 'T', 'HL', 32, 'TL', 6, 'CP', 30, 'CL', 1,
     +            'MNR', 32, 'MRL', 80,
     +            'KA', KEY, 'KL', 2, 'EMK', 'YES', 'KP', 8,
     +            'EFC', 3, 'DFC', 3)
      OPEN (2, FILE = 'NEWZOO')
      CALL OPENM (AKFIT, 'I-O')
      IF (IFETCH (AKFIT, 'ES') .NE. 0) GO TO 90
      PRINT *, '  TRANS'
   10 READ (2, '(2I10)', END = 60)  KEY, TRANS
      PRINT *,   KEY, TRANS
C
C  DELETE RECORD FROM EXISTING FILE
C
      IF (TRANS .LT. 0) THEN
      CALL DLTE (AKFIT)
      IF (IFETCH (AKFIT, 'ES') .NE. 0) GO TO 90
      GO TO 10
      END IF
C
C  READ EXISTING RECORD AND REPLACE
C
      IF (TRANS .EQ. 0) THEN
      CALL GET (AKFIT)
      IF (IFETCH (AKFIT, 'ES') .NE. 0) GO TO 90
      PRINT 110, AKWSA
      READ (2, '(I10, 7A10)', END = 60) AKWSA
      PRINT 130, AKWSA
      CALL REPLC (AKFIT)
      IF (IFETCH (AKFIT, 'ES') .NE. 0) GO TO 90
      GO TO 10
      END IF
C
C  ADD NEW RECORD TO THE FILE
C
      IF (TRANS .GT. 0) THEN
      READ (2, '(I10, 7A10)', END = 60) AKWSA
      PRINT 130, AKWSA
      CALL PUT (AKFIT)
      IF (IFETCH (AKFIT, 'ES') .NE. 0) GO TO 90
      GO TO 10
      END IF
   60 CALL REWND (AKFIT)
```

Figure 6-5. Processing Existing Actual Key File (Sheet 2 of 3)

Source Program (Contd)

```
C
C  READ UPDATED FILE SEQUENTIALLY
C
      PRINT *, 'NEW FILE'
  70  DO 75 I = 1, 8
  75  AKWSA (I) = 10H
      CALL GETN (AKFIT)
      IF (IFETCH (AKFIT, 'ES') .NE. 0) GO TO 90
      IF (IFETCH (AKFIT, 'FP') .NE. O"20") GO TO 80
      PRINT 130, AKWSA
      GO TO 70
  80  CALL CLOSEM (AKFIT)
      STOP
  90  PRINT 902, IFETCH (AKFIT, 'ES')
      CALL CLOSEM (AKFIT)
      STOP 'CRM ERROR RETURNED'
 110  FORMAT (' REPLACE RECORD - ', I10, 7A10)
 130  FORMAT (1X, I10, 7A10)
 902  FORMAT ('ES = ', O3)
      END
```

D. Sample Output

```
   TRANS
 5 -1 <------------------------------------------------------------ Delete Record 5
27  1 <------------------------------------------------------------ Add Record 27
          27POLAR BEARFISH      2 SAM   PEARL
45  1 <------------------------------------------------------------ Add Record 45
          45LION      RAW MEAT  1 LEO
62  0 <------------------------------------------------------------ Replace Record 62
 REPLACE RECORD -          62BROWNBEAR MEAT MIX  2 ROCKY TESSA
          62BROWN BEARMEAT MIX  3 ROCKY TESSA BRUNO
NEW FILE
          1MONKEY     BANABA MIX6 JOE   BIMBA TOMMY SUSIE SALLY JIMBO
         12BABOON     BANANA MIX5 JANEY BOBBY BUCKO MANDY CARRIE
         18GOAT       MASH      4 BILLY BETTY BUTTS BECKY
         24SHEEP      MASH      1 CURLY
         27POLAR BEARFISH       2 SAM   PEARL
         32RACCOON    MEAT MIX  3 RINGO KINKY KELLY
         41PENGUIN    FISH MIX  2 PENNY MR MAC
         45LION       RAW MEAT  1 LEO
         62BROWN BEARMEAT MIX   3 ROCKY TESSA BRUNO
       .040 CP SECONDS EXECUTION TIME.
/
```

Figure 6-5. Processing Existing Actual Key File (Sheet 3 of 3)

DEBUGGING TOOLS 7

AAM performs various checks to ensure proper file processing and maintains information in a number of FIT fields related to error processing. You can set several of these FIT fields. AAM sets other fields that you can examine.

During program execution, AAM generates error messages and notes. Error messages are generated for fatal and trivial errors. Notes output by AAM are informative and statistical messages. You control the disposition of error messages and notes through two FIT fields. Unless otherwise directed, AAM only outputs fatal error messages to the dayfile. You can examine the error file through the CRMEP control statement.

The contents of FIT fields can be captured at various points during processing and recorded on the error file. This capability is available through the CALL FITDMP statement, which can appear anywhere within the source program.

FIT FIELDS UNDER USER CONTROL

The most common FIT fields that you can set and use for error processing are DFC, EFC, and ERL.

These fields are summarized in table 7-1 and described in the following paragraphs.

DAYFILE CONTROL, DFC

The DFC field controls the listing of error messages on the dayfile. This field can be set by the CALL FILExx statement, the CALL STOREF statement, or the FILE control statement.

Fatal error messages are always written on the dayfile. The messages written on the dayfile depend on the setting of the DFC field as follows:

0	Fatal messages only (default)
1	All error messages to the dayfile
2	Notes to the dayfile
3	Error messages and notes to the dayfile

ERROR FILE CONTROL, EFC

The EFC field controls the listing of error messages on the error file. This field can be set by the CALL FILExx statement, the CALL STOREF statement, or the FILE control statement.

The error file is a special file created with the logical file name ZZZZZEG. The messages written on the error file depend on the setting of the EFC field as follows:

0	No messages to the error file (default)
1	Error messages to the error file
2	Notes to the error file
3	Error messages and notes to the error file

TRIVIAL ERROR LIMIT, ERL

Trivial error conditions can interfere with a particular operation but do not deny further file access. Trivial errors should not be ignored on the assumption they are unimportant. Trivial errors might reveal that the correct file is not being processed. Trivial errors might also reveal that an entire job completed normally, but the file was not created because the file was never opened.

TABLE 7-1. ERROR PROCESSING FIT FIELDS UNDER USER CONTROL

FIT Field	Definition	Set By	Values
DFC	Dayfile control	FILExx STOREF FILE control statement	0 fatal messages only (default) 1 error messages 2 notes 3 error messages and notes
EFC	Error file control	FILExx STOREF FILE control statement	0 no entries (default) 1 error messages 2 notes 3 error messages and notes
ERL	Trivial error limit	FILExx STOREF FILE control statement	0 no limit (default) n error limit where n = 1 thru 511

The ERL field places a limit on the number of trivial errors allowed. When the limit is reached, a fatal error occurs. For example, if ERL is set to 10, nine trivial errors can occur before the job aborts.

The ERL field can be set by the CALL FILExx statement, the CALL STOREF statement, or the FILE control statement. The default is no limit on trivial error messages.

FIT FIELDS UNDER SYSTEM CONTROL

FIT fields that are set by AAM and that you can interrogate are:

Trivial error count	ECT
Error status	ES
Fatal/nonfatal flag	FNF

These fields are summarized in table 7-2 and described in the following paragraphs.

TABLE 7-2. ERROR PROCESSING FIT FIELDS UNDER SYSTEM CONTROL

FIT Field	Definition	IFETCH Return Value
ECT	Trivial error count	0 thru 511
ES	Error status	Error code (001 thru nnn)
FNF	Fatal/nonfatal flag	0 nonfatal 1 fatal

TRIVIAL ERROR COUNT, ECT

The ECT field holds the trivial error count. When you set the trivial error limit (ERL) field to a value greater than zero, the ECT field is incremented by AAM whenever a trivial error occurs. As long as the value of the ECT field is less than the value of the ERL field, the trivial error causes control to pass to the error exit, if specified, or to the in-line code. When the error count is the same as the trivial error limit (ECT=ERL), a fatal error occurs.

ERROR STATUS, ES

The ES field holds a three-digit octal error code. When a fatal or trivial error occurs, AAM sets the ES field to the appropriate code.

FATAL/NONFATAL FLAG, FNF

The FNF field holds a value that indicates whether an error is fatal or nonfatal (trivial). AAM sets the FNF field to 1 to indicate a fatal error and to 0 to indicate a nonfatal error. When FNF is set to 1, the file cannot be processed further; if an attempt is made to process the file, the job is aborted.

PROCESSING THE ERROR FILE

The error file is a local mass storage file that disappears at job termination. To read the information stored on the error file, you must call the post error processor by the CRMEP control statement.

The error file buffer is always flushed when the job terminates abnormally. At the normal completion of a job step, however, the buffer is flushed only if all files are closed. Any messages in the buffer are lost if the buffer is not flushed.

Parameters in the CRMEP control statement specify the output file to be used and select the error file information to be listed on the output file. If no parameters are specified, all fatal and data manager error messages are listed on the system file OUTPUT. Data manager error messages are transmitted to the error status (ES) field by the CYBER Database Control System (CDCS) component of the DMS-170 data management system. These messages comprise the 600 category and can appear when the CDCS interface applies.

Parameters are specified in two ways: the mnemonic alone or the mnemonic followed by an equal sign and one or more options. Multiple options are separated by a slash. Table 7-3 lists the various parameters for the CRMEP control statement and the possible settings for each parameter. Figure 7-1 shows some examples of the CRMEP control statement.

CRMEP(LO,SF=ISFILE)

All messages for the file ISFILE are to be listed on the file OUTPUT.

CRMEP(LO=-D,ON,L=ERRFILE)

All messages except data manager messages and those with error codes 142 and 143 are written to the output file ERRFILE.

CRMEP(LO=N,SN=1000)

Only notes with the numbers 1000, 1001, and 1002 which are the numbers for the FIT dumps are to be listed.

Figure 7-1. CRMEP Control Statement Examples

In addition to error messages, various notes and statistics are written to the error file. Notes are output by AAM to the error file at various times during program execution. Statistics, which are output when the file is closed, include accumulated totals for each type of file processing request. Figure 7-2 shows the error file notes and statistics printed after execution of the program shown in figure 3-10.

DUMPING THE FIT

You can dump the contents of the FIT to the error file as a note by executing a CALL FITDMP statement. The format of this statement is as follows:

```
CALL FITDMP (fit,id)
```

TABLE 7-3. CRMEP CONTROL STATEMENT PARAMETERS

Mnemonic	Omitted	Mnemonic Only	Mnemonic and Option
LO	Fatal and data manager error messages are listed.	All messages in the error file are listed.	Select (N) or omit (-N) notes. Select (F) or omit (-F) fatal messages. Select (D) or omit (-D) data manager messages. Select (T) or omit (-T) trivial messages.
SF	Select messages for all files.	Select messages for all files.	Select messages only for the specified files.
OF	Omit messages for no files.	Omit messages for no files.	Omit messages only for the specified files.
SN	Select all message numbers.	Select hardware and parity error messages.	Select only messages with the specified numbers.
ON	Omit no message numbers.	Omit only error numbers 142 and 143.	Omit messages with the specified numbers.
L	Output file is OUTPUT.	Output file is LIST.	Output file is the specified file.
RU	Error file remains at EOI after processing.	Error file is returned/unloaded after processing.	Not applicable.

```
                 CRMEP,LO,RU.
        RM NOTE  1001 ON LFN ISFILE   FILE OPENED
        RM NOTE  1011 ON LFN ISFILE   GETN REACHED EOI
        RM NOTE  1002 ON LFN ISFILE   FILE CLOSED
        RM NOTE  1003 ON LFN ISFILE   NUMBER OF INDEX LEVELS          0
        RM NOTE  1004 ON LFN ISFILE   ***NUMBER OF GETS THIS OPEN               2
        RM NOTE  1005 ON LFN ISFILE   ***NUMBER OF PUTS THIS OPEN               1
        RM NOTE  1006 ON LFN ISFILE   ***NUMBER OF REPLACES THIS OPEN           2
        RM NOTE  1007 ON LFN ISFILE   ***NUMBER OF DELETES THIS OPEN            1
        RM NOTE  1033 ON LFN ISFILE   ***NUMBER OF GET NEXTS THIS OPEN         10
        RM NOTE  1010 ON LFN ISFILE   ***TOTAL DIAKAREA***              256 WORDS
```

Figure 7-2. Error File Notes and Statistics

The first parameter in the FIT dump request is the name of the array that contains the FIT. When more than one FIT is being dumped to the error file, a 10-character identifier can be associated with each FIT. The identifier is specified as the second parameter. If the error file control (EFC) field is set to 0, it is forced to 2; if the field is set to 1, it is forced to 3.

CALL FITDMP (ISFIT)

When this statement is executed, the FIT in the array ISFIT is dumped to the error file as note 1000.

CALL FITDMP (ISFIT, M)

When this statement is executed, the FIT in the array ISFIT is dumped to the error file as note 1000 and identified by the name indicated in the variable M. You can easily differentiate two FIT dumps for the same file in different parts of the program by using this convention.

Program EXEXAMP, shown again in figure 7-3, includes a CALL FITDMP statement to dump the contents of the FIT to the error file and a CRMEP control statement to print the contents of the error file. Figure 7-4 shows the first three entries on the error file. Note 1001 indicates that the file was successfully opened. The second entry specifies that an error (0167) occurred. The third entry is note 1000, which is the FIT dump produced at the time the error occurred. The content of each FIT field reflects the value of the field at the time subroutine ERROR was entered. Applicable fields and their octal values are as follows:

CL=1	Trailer count field length of one character
CP=36	Count field begins in character position 30
DFC=3	Errors and statistics/notes to the dayfile
EFC=3	Error messages and notes to the error file
EMK=1	Primary key is embedded in the record
ES=167	Error code 167 - record length outside min-max range
FO=6	Actual key file organization
HL=40	Record fixed length portion of 32 characters
KA=345	Address of the primary key
KL=2	Key length of two characters
KP=10	Key begins in eighth character position of word at location KA
LFN=AKFILE	Logical file name
MNR=40	Minimum record length of 32 characters (logical equivalent of HL for T type records)
MRL=120	Maximum record length of 80 characters
OC=1	File opened
ON=0	Not a file creation run (old file)
ORG=1	Extended AAM file organization
PD=2	Processing direction of output
RKP=10	Key begins in eighth character position of RKW
RKW=0	Key begins in first word of the record
RT=5	T type records
TL=6	Trailer length of six characters
WSA=345	Address of working storage area

FILE LIMIT

The file limit option establishes an upper limit on the number of records in the file. This does not affect the number of file transactions; it applies only to the maximum number of records in the file at any time.

The file limit (FLM) field in the FIT is set to place a maximum on the number of records allowed in the file. When the limit specified by the FLM field is reached, an attempt to write more records to the file produces a trivial error and the write is ignored.

If the FLM field is set to zero, which is also the default value, an unlimited number of records can be written to the file. The installation, however, can limit the amount of mass storage a file or job can occupy.

A. NOS Operating System

```
Job statement
USER statement
CHARGE statement
FTN5.
ATTACH(AKFILE/UN=userno,M=W)
LGO.
CRMEP,LO.
```

NOS/BE Operating System

```
Job statement
ACCOUNT Statement
FTN5.
ATTACH(AKFILE,ID=AAMUG)
LGO.
CRMEP,LO.
```

B. Source Program

```
      PROGRAM EXEXAMP
C
C  *************************************************************
C  * THIS PROGRAM ILLUSTRATES USING THE CALL FITDMP STATEMENT *
C  * TO DISPLAY THE CONTENTS OF THE FIT WHEN AN ERROR OCCURS  *
C  *************************************************************
C
      IMPLICIT INTEGER (A-Z)
      DIMENSION AKFIT (35), AKWSA (8)
      CALL FILEAK (AKFIT, 'LFN', 'AKFILE', 'ORG', 'NEW',
     +            'WSA', AKWSA, 'EMK', 'YES',
     +            'RT', 'T', 'HL', 32, 'TL', 6, 'CP', 30, 'CL', 1,
     +            'KA', AKWSA (1), 'KP', 8,
     +            'EFC', 3, 'DFC', 3)
      CALL OPENM (AKFIT, 'OUTPUT')
      IF (IFETCH (AKFIT, 'ES') .NE. 0) GO TO 20
   10 READ (*, '(I10, 7A10)', END = 30) AKWSA
      CALL PUT (AKFIT)
C
C  TEST FOR CRM ERROR AND DUMP THE FIT
C
   20 IF (IFETCH (AKFIT, 'ES') .NE. 0) THEN
        PRINT 902, IFETCH (AKFIT, 'ES'), AKWSA (1)
        CALL FITDMP (AKFIT)
        END IF
      GO TO 10
   30 CALL CLOSEM (AKFIT)
      STOP
  902 FORMAT ('ERROR CODE IS  ', O3, ' FOR PRIMARY KEY  ', I10)
      END
```

C. Sample Output

```
/
?         10tiger     meat mix  9 a     b     c     d     e     f     g     h
ERROR CODE IS  167 FOR PRIMARY KEY         10
?         18goat      mash      4 billy betty bucky becky
ERROR CODE IS  446 FOR PRIMARY KEY         18
?         99elephant  peanuts
ERROR CODE IS  442 FOR PRIMARY KEY         99
?
      .031 CP SECONDS EXECUTION TIME.
/
```

Figure 7-3. Dumping the FIT

```
         CRMEP,LO,RU.
RM NOTE  1001 ON LFN AKFILE   FILE OPENED
RM ERROR 0167 ON LFN AKFILE   EXIT ADDRESS 014075 RECORD LENGTH OUTSIDE
 MIN-MAX RANGE--REQUEST IGNORED
RM NOTE  1000 ON LFN AKFILE   FIT DUMP                  (FIT AT 000302)
                    0 ASCII                       00 LAC
                    0 BAL                   00000000 LBL
                    0 BBH                          0 LCR
                    0 BCK             01130611140500 LFN
                    0 BFF                     000000 LGX
               000000 BFS                         01 LL
           0000000000 BN                          00 LNG
                    0 BT                          01 LOP
               000302 BZF                         01 LOP5
                    0 B8F                   00000036 LP
               000000 CDT                          2 LT
                    0 CF                          00 LVL
                   01 CL                      000000 LX
                    0 CM                    00006154 MBL
                    1 CMPLT             000000000000 MFN
                    0 CNF                        000 MKL
             00000036 CP                    00000000 MNB
               000000 CPA                   00000040 MNR
                    0 C1                    00000120 MRL
                   00 DC                          00 MUL
               000000 DCA                          0 NDX
               000000 DCT                         00 NL
                    3 DFC                          0 NOFCP
                    0 DFLG                         1 OC
                    0 DKI                          0 OF
                  000 DP                           0 ON
                 0412 DVT                          1 ORG
               000000 DX                           0 OVF
                  000 ECT                         00 PC
                    3 EFC                          2 PD
                    1 EMK                          0 PEF
                    0 EO                      000000 PKA
           0000002101 EOIWA                        0 PM
                  000 ERL                   00000000 PNO
                  167 ES                          00 POS
               000000 EX                    00000000 PTL
                    1 EXD                       0000 RB
                    0 FF                  0000000000 RC
             00000120 FL                           0 RDR
           7777777777 FLM                          0 REL
                    0 FNF                         10 RKP
                    6 FO                        0000 RKW
                  020 FP                    00000000 RL
                    0 FPB                         01 RMK
                   36 FTS                         05 RT
               032266 FWB                          0 SB
                    0 FWI                          0 SBF
                    0 HB                           1 SDS
             00000040 HL                          00 SES
             00000000 HMB                          0 SOL
               000000 HRL                          0 SPR
             00000000 IBL                   00000006 TL
                  000 IP                          00 TRC
                  167 IRS                          0 ULP
               000345 KA                           0 VF
                  002 KL                          00 VNO
                    0 KNE                 0000000000 WA
                   10 KP                           0 WPN
                 0000 KR                    00000345 WSA
                    3 KT                      000000 XBS
               000000 LA              00000000000000 XN
```

Figure 7-4. Error File Output

STANDARD CHARACTER SETS A

Control Data operating systems offer the following variations of a basic character set:

- CDC 64-character set
- CDC 63-character set
- ASCII 64-character set
- ASCII 63-character set

Table A-1 shows these character sets. The set in use at a particular installation is specified when the operating system is installed or deadstarted.

Depending on another installation option, the system assumes an input deck has been punched either in 026 or in 029 mode (regardless of the character set in use).

Under NOS/BE, the alternate mode can be specified by a 26 or 29 punched in columns 79 and 80 of the job statement or any 7/8/9 card. The specified mode remains in effect throughout the job unless it is reset by specification of the alternate mode on a subsequent 7/8/9 card.

Under NOS, the alternate mode can be specified by a 26 or 29 punched in columns 79 and 80 of any 6/7/9 card, as described for a 7/8/9 card. In addition, 026 mode can be specified by a card with 5/7/9 multipunched in column 1; 029 mode can be specified by a card with 5/7/9 multipunched in column 1 and a 9 punched in column 2.

Graphic character representation appearing at a terminal or printer depends on the installation character set and the terminal type. Characters shown in the CDC Graphic column of table A-1 are applicable to BCD terminals; ASCII graphic characters are applicable to ASCII-CRT and ASCII-TTY terminals.

Several graphics are not common for all codes. Where these differences in graphics appear, assignment of collation positions and translation between codes must be made. Tables A-2 and A-3 show the CDC and ASCII character set collating sequences.

TABLE A-1. STANDARD CHARACTER SETS

Display Code (octal)	CDC Graphic	CDC Hollerith Punch (026)	CDC External BCD Code	ASCII Graphic Subset	ASCII Punch (029)	ASCII Code (octal)
00†	: (colon)††	8-2	00	: (colon)††	8-2	072
01	A	12-1	61	A	12-1	101
02	B	12-2	62	B	12-2	102
03	C	12-3	63	C	12-3	103
04	D	12-4	64	D	12-4	104
05	E	12-5	65	E	12-5	105
06	F	12-6	66	F	12-6	106
07	G	12-7	67	G	12-7	107
10	H	12-8	70	H	12-8	110
11	I	12-9	71	I	12-9	111
12	J	11-1	41	J	11-1	112
13	K	11-2	42	K	11-2	113
14	L	11-3	43	L	11-3	114
15	M	11-4	44	M	11-4	115
16	N	11-5	45	N	11-5	116
17	O	11-6	46	O	11-6	117
20	P	11-7	47	P	11-7	120
21	Q	11-8	50	Q	11-8	121
22	R	11-9	51	R	11-9	122
23	S	0-2	22	S	0-2	123
24	T	0-3	23	T	0-3	124
25	U	0-4	24	U	0-4	125
26	V	0-5	25	V	0-5	126
27	W	0-6	26	W	0-6	127
30	X	0-7	27	X	0-7	130
31	Y	0-8	30	Y	0-8	131
32	Z	0-9	31	Z	0-9	132
33	0	0	12	0	0	060
34	1	1	01	1	1	061
35	2	2	02	2	2	062
36	3	3	03	3	3	063
37	4	4	04	4	4	064
40	5	5	05	5	5	065
41	6	6	06	6	6	066
42	7	7	07	7	7	067
43	8	8	10	8	8	070
44	9	9	11	9	9	071
45	+	12	60	+	12-8-6	053
46	-	11	40	-	11	055
47	*	11-8-4	54	*	11-8-4	052
50	/	0-1	21	/	0-1	057
51	(	0-8-4	34	(	12-8-5	050
52	)	12-8-4	74	)	11-8-5	051
53	$	11-8-3	53	$	11-8-3	044
54	=	8-3	13	=	8-6	075
55	blank	no punch	20	blank	no punch	040
56	, (comma)	0-8-3	33	, (comma)	0-8-3	054
57	. (period)	12-8-3	73	. (period)	12-8-3	056
60	≡	0-8-6	36	#	8-3	043
61	[	8-7	17	[	12-8-2	133
62	]	0-8-2	32	]	11-8-2	135
63	% ††	8-6	16	% ††	0-8-4	045
64	≠	8-4	14	" (quote)	8-7	042
65	↱	0-8-5	35	_ (underline)	0-8-5	137
66	∨	11-0	52	!	12-8-7	041
67	∧	0-8-7	37	&	12	046
70	↑	11-8-5	55	' (apostrophe)	8-5	047
71	↓	11-8-6	56	?	0-8-7	077
72	<	12-8-2†††	72	<	-8-4 or 12-0†††	074
73	>	11-8-7	57	>	0-8-6	076
74	≤	8-5	15	@	8-4	100
75	≥	12-8-5	75	\	0-8-2	134
76	¬	12-8-6	76	⁻ (circumflex)	11-8-7	136
77	; (semicolon)	12-8-7	77	; (semicolon)	11-8-6	073

†Twelve zero bits at the end of a 60-bit word in a zero byte record are an end of record mark rather than two colons.

††In installations using a 63-graphic set, display code 00 has no associated graphic or card code; display code 63 is the colon (8-2 punch). The % graphic and related card codes do not exist and translations yield a blank (55_8).

TABLE A-2. CDC CHARACTER SET COLLATING SEQUENCE

Collating Sequence Decimal/Octal		CDC Graphic	Display Code	External BCD	Collating Sequence Decimal/Octal		CDC Graphic	Display Code	External BCD
00	00	blank	55	20	32	40	H	10	70
01	01	≤	74	15	33	41	I	11	71
02	02	%	63 †	16 †	34	42	∨	66	52
03	03	[	61	17	35	43	J	12	41
04	04	→	65	35	36	44	K	13	42
05	05	≡	60	36	37	45	L	14	43
06	06	∧	67	37	38	46	M	15	44
07	07	↑	70	55	39	47	N	16	45
08	10	↓	71	56	40	50	O	17	46
09	11	>	73	57	41	51	P	20	47
10	12	≥	75	75	42	52	Q	21	50
11	13	¬	76	76	43	53	R	22	51
12	14	.	57	73	44	54	]	62	32
13	15	)	52	74	45	55	S	23	22
14	16	;	77	77	46	56	T	24	23
15	17	+	45	60	47	57	U	25	24
16	20	$	53	53	48	60	V	26	25
17	21	*	47	54	49	61	W	27	26
18	22	−	46	40	50	62	X	30	27
19	23	/	50	21	51	63	Y	31	30
20	24	,	56	33	52	64	Z	32	31
21	25	(	51	34	53	65	:	00 †	none†
22	26	=	54	13	54	66	0	33	12
23	27	≠	64	14	55	67	1	34	01
24	30	<	72	72	56	70	2	35	02
25	31	A	01	61	57	71	3	36	03
26	32	B	02	62	58	72	4	37	04
27	33	C	03	63	59	73	5	40	05
28	34	D	04	64	60	74	6	41	06
29	35	E	05	65	61	75	7	42	07
30	36	F	06	66	62	76	8	43	10
31	37	G	07	67	63	77	9	44	11

†In installations using the 63-graphic set, the % graphic does not exist. The : graphic is display code 63, External BCD 16.

TABLE A-3. ASCII CHARACTER SET COLLATING SEQUENCE

Collating Sequence Decimal/Octal		ASCII Graphic Subset	Display Code	ASCII Code	Collating Sequence Decimal/Octal		ASCII Graphic Subset	Display Code	ASCII Code
00	00	blank	55	20	32	40	@	74	40
01	01	!	66	21	33	41	A	01	41
02	02	"	64	22	34	42	B	02	42
03	03	#	60	23	35	43	C	03	43
04	04	$	53	24	36	44	D	04	44
05	05	%	63†	25	37	45	E	05	45
06	06	&	67	26	38	46	F	06	46
07	07	'	70	27	39	47	G	07	47
08	10	(	51	28	40	50	H	10	48
09	11	)	52	29	41	51	I	11	49
10	12	*	47	2A	42	52	J	12	4A
11	13	+	45	2B	43	53	K	13	4B
12	14	,	56	2C	44	54	L	14	4C
13	15	-	46	2D	45	55	M	15	4D
14	16	.	57	2E	46	56	N	16	4E
15	17	/	50	2F	47	57	O	17	4F
16	20	0	33	30	48	60	P	20	50
17	21	1	34	31	49	61	Q	21	51
18	22	2	35	32	50	62	R	22	52
19	23	3	36	33	51	63	S	23	53
20	24	4	37	34	52	64	T	24	54
21	25	5	40	35	53	65	U	25	55
22	26	6	41	36	54	66	V	26	56
23	27	7	42	37	55	67	W	27	57
24	30	8	43	38	56	70	X	30	58
25	31	9	44	39	57	71	Y	31	59
26	32	:	00†	3A	58	72	Z	32	5A
27	33	;	77	3B	59	73	[	61	5B
28	34	<	72	3C	60	74	\	75	5C
29	35	=	54	3D	61	75	]	62	5D
30	36	>	73	3E	62	76	^	76	5E
31	37	?	71	3F	63	77	_	65	5F

†In installations using a 63-graphic set, the % graphic does not exist. The : graphic is display code 63.

SUMMARY OF FORTRAN CALL STATEMENTS B

This appendix includes the general formats of FORTRAN calls to AAM for indexed sequential, actual key, and direct access file organizations. Refer to the AAM Reference manual for a detailed description of the SEEKF macro. The following conventions are used:

- Words in uppercase must appear exactly as they are shown.
- Words in lowercase are generic terms that represent the words or symbols supplied by the programmer. In most instances, the terms are the same as actual names of FIT fields. These fields can be constants or integer variables.
- Subroutines FILExx and STOREF require specifications of field and value. The word field denotes the name of a FIT field. It must be a character string in apostrophe or left-justified format. The word value denotes the value to be placed in the field; it must be a character string in apostrophe or left-justified format for symbolic options, an integer representation for numeric options, or a program name or variable name (for example, owncode exits and working storage area).
- Function IFETCH requires a field specification. The word field denotes the name of a FIT field; it must be a character string in apostrophe or left-justified format. The word variable denotes an integer variable in which the value of the FIT field will be returned.
- Except for CALL FILExx, the order of parameters is fixed so that all parameters positioned to the left of a desired option must be specified. A parameter list can be truncated at any point after the fit; middle parameters cannot be defaulted. If a parameter is not applicable to a particular file organization and its position is needed in a statement, a zero must be specified as indicated in the formats. If a parameter is applicable to the file organization but not applicable to the record type, a zero must be specified to mark a needed position. Zeros should never be used for applicable fields meaning a parameter is not intended. A zero or the address of the constant zero will be used as the parameter.

INDEXED SEQUENTIAL FILE ORGANIZATION

CALL CLOSEM (fit,cf)

CALL DLTE (fit,ka,kp,0,ex)

CALL FILEIS (fit,field,value,. . . ,field,value)

CALL FITDMP (fit,id)

CALL GET (fit,wsa,ka,kp,mkl,0,ex)

CALL GETN (fit,wsa,ka,ex)

CALL GETNR (fit,wsa,ka,ex)

IFETCH (fit,field)

CALL IFETCH (fit,field,variable)

CALL OPENM (fit,pd,of)

CALL PUT (fit,wsa,rl,ka,kp,0,ex)

CALL REPLC (fit,wsa,rl,ka,kp,0,ex)

CALL REWND (fit)

CALL RMKDEF (lfn,rkw,rkp,kl,0,kt,ks,kg,kc,nl,ie,ch)

CALL SEEKF (fit,ka,kp,mkl,ex)

CALL SKIP (fit,±count)

CALL STARTM (fit,ka,kp,mkl,ex)

CALL STOREF (fit,field,value)

ACTUAL KEY FILE ORGANIZATION

CALL CLOSEM (fit,cf)

CALL DLTE (fit,ka,kp,0,ex)

CALL FILEAK (fit,field,value,. . . ,field,value)

CALL FITDMP (fit,id)

CALL GET (fit,wsa,ka,kp,0,0,ex)

CALL GETN (fit,wsa,ka,ex)

CALL GETNR (fit,wsa,ka,ex)

IFETCH (fit,field)

CALL IFETCH (fit,field,variable)

CALL OPENM (fit,pd)

CALL PUT (fit,wsa,rl,ka,kp,0,ex)

CALL REPLC (fit,wsa,rl,ka,kp,0,ex)

CALL REWND (fit)

CALL RMKDEF (lfn,rkw,rkp,kl,0,kt,ks,kg,kc,nl,ie,ch)

CALL SEEKF (fit,ka,kp,0,ex)

CALL SKIP (fit,±count)

CALL STOREF (fit,field,value)

DIRECT ACCESS FILE ORGANIZATION

CALL CLOSEM (fit,cf)

CALL DLTE (fit,ka,kp,0,ex)

CALL FILEDA (fit,field,value, . . . ,field,value)

CALL FITDMP (fit,id)

CALL GET (fit,wsa,ka,kp,0,0,ex)

CALL GETN (fit,wsa,ka,ex)

CALL GETNR (fit,wsa,ka,ex)

IFETCH (fit,field)

CALL IFETCH (fit,field,variable)

CALL OPENM (fit,pd)

CALL PUT (fit,wsa,rl,ka,kp,0,ex)

CALL REPLC (fit,wsa,rl,ka,kp,0,ex)

CALL REWND (fit)

CALL RMKDEF (lfn,rkw,rkp,kl,0,kt,ks,kg,kc,nl,ie,ch)

CALL SEEKF (fit,ka,kp,0,ex)

CALL STOREF (fit,field,value)

GLOSSARY C

Actual Key (AK) File -
A mass storage file in which each record is identified by a record number that is converted by AAM to the storage location (block and record slot) of the record. Access is random or sequential.

Advanced Access Methods (AAM) -
A file manager that processes indexed sequential, direct access, and actual key file organizations and supports the Multiple-Index Processor.

Alternate Key -
A key other than the primary key by which an indexed sequential, direct access, or actual key file can be accessed. Contrast with Primary Key.

Basic Access Methods (BAM) -
A file manager that processes sequential and word addressable file organizations.

Beginning-of-Information (BOI) -
The start of the first user record in a file.

Block -
A logical or physical grouping of records to make more efficient use of hardware. All files are blocked. See also Data Block, Home Block, Index Block, and Overflow Block.

Block Checksum -
A number used to check that the contents of a data block have not been altered accidentally; a means of ensuring data integrity. Block checksums can be requested for files by setting the BCK field in the file information table.

Character -
A letter, digit, punctuation mark, or mathematical symbol forming part of one or more of the standard character sets. Also, a unit of measure used to specify block length, record length, and so forth.

Close -
A set of terminating operations performed on a file when input and output operations are complete. All files processed by AAM must be closed.

Compression -
The process of condensing a record to reduce the amount of storage space required. You can either supply a compression routine or use a system-supplied routine. See Decompression.

Creation Run -
All processing of a file, from open to close, the first time the file is written or made into an AAM file. Files must be created in a separate creation run during which only write operations on the file being created are allowed.

CRMEP Control Statement -
A control statement that processes the AAM error file.

CYBER Record Manager (CRM) -
A generic term relating to the common products BAM and AAM.

Data Block -
A block in which user records are stored in an indexed sequential or actual key file. You can define data block structure or AAM defaults that are used. Contrast with Index Block for indexed sequential files.

Decompression -
The process of expanding a compressed record to restore it to its original size. You can either supply a decompression routine or use a system-supplied routine. See Compression.

Default -
A value assumed in the absence of a user-specified value declaration for the parameter involved. Values for many defaults are defined by the installation.

Direct Access (DA) File -
A file containing records stored randomly in home blocks according to the hashed value of the primary key in each record. Files must be mass storage resident. All allocation for home blocks occurs when the file is opened on its creation run. Access is random or sequential.

Directives -
The instructions that supplement processing defined by a control statement or by a program call for execution of a utility function or member of a product set. Directives do not appear in the control statement record; they are usually in a separate record of the file INPUT or a file referenced in a control statement call. Directives are required for execution of FORM, the CREATE utility, and the key analysis utility among others.

Embedded Key -
A primary key that is contained within the record.

End-of-Information (EOI) -
The end of the last user record in a file.

Error File -
A special file created with the logical file name ZZZZZEG to hold AAM error messages; the file is processed by the CRMEP control statement.

Extended AAM File -
An indexed sequential, direct access, or actual key file identified by ORG=NEW parameter.

Field -
A portion of a word or record; a subdivision of information within a record; also, a generic entry in a file information table identified by a mnemonic.

Field Length -
The area in central memory allocated to a particular job; the only part of central memory that a job can directly access. Contrasts with mass storage space allocated for a job and on which user files reside.

File -
A logically related set of information; the largest collection of information that can be addressed by a file name. It starts at beginning-of-information and ends at end-of-information. Every file in use by a job must have a logical file name.

FILE Control Statement -
A control statement that supplies file information table values after a source language program is compiled or assembled but before the program is executed. In applications such as those with a control statement call to the FORM utility, a FILE control statement must be used. Basic file characteristics such as organization, record type, and description can be specified in the FILE control statement.

File Information Table (FIT) -
A table through which a user program communicates with AAM. For direct processing through AAM, a user must initiate establishment of this table. All file processing executes on the basis of information in this table. You can either set FIT fields directly or use parameters in a file access call that sets the fields indirectly. Some product set members set the fields automatically for you.

File Statistics Table (FSTT) -
A table generated and maintained by AAM to collect statistics about each file. The FSTT is a permanent part of a file and contains information such as organization type, size of blocks, number of current accesses, and so forth.

Hashing -
The method of using primary keys to search for relative home block addresses of records in a file with direct access storage structure.

Home Block -
A block in a file with direct access storage structure whose relative address is computed by hashing keys. A home block contains synonym records whose keys hash to that relative address. If all the synonym records cannot be accommodated in the home block, an overflow block can be created by the system. When creating a direct access file you must define the number of home blocks by setting the HMB field in the file information table.

Index -
A series of keys and pointers to records associated with the keys.

Index Block -
For an indexed sequential file, a block with ordered keys and pointers to the data blocks and other index blocks, forming a directory of the records within a file.

Index File -
A file of indexes for alternate keys defined for a data file. An index file contains a keylist for each alternate key value. An index is created and maintained by AAM, but you are responsible for making the index part of a job.

Indexed Sequential (IS) File -
A file organization in which AAM maintains files in sorted order by use of a user-defined primary key, which need not be within the record. Keys can be integer, collated symbolic, or uncollated symbolic; access is random or sequential. Files contain index blocks and data blocks.

Integer Key -
A 60-bit signed binary key used with indexed sequential files. See Symbolic Key.

Key -
A group of contiguous characters or numbers that you define to identify a record in an AAM file.

Key Analysis Utility -
A utility program that provides information about hypothetical record distribution for a file with direct access organization. The utility reads the key of each record in the file and determines the home block where the record would reside.

Key of Reference -
Either the alternate key field or the primary key field currently being used to access a record in an indexed sequential, direct access, or actual key file.

Keylist -
A list of one or more primary key values associated with a specified value of an alternate key.

Logical File Name -
The name given to a file being used by a job. The name must be unique for the job and must consist of one to seven letters or digits, the first of which must be a letter.

Macro -
A single instruction that when assembled into machine code generates several machine code instructions.

Major Key -
The leading characters of a symbolic key in an indexed sequential file.

Mass Storage -
A disk pack that can be accessed randomly. ECS is not considered mass storage.

MIPGEN -
The utility that adds or deletes an index from the index file.

Multiple-Index File -
An indexed sequential, direct access, or actual key file for which additional keys, called alternate keys, are defined.

Multiple-Index Processor (MIP) -
A processor that allows AAM files to be accessed by alternate keys.

Open -
A set of preparatory operations performed on a file before input and output can take place; required for all AAM files.

Overflow Block -
A block added to the file by AAM for use when a home block in a direct access file is full.

Owncode -
A routine written by the user to process certain conditions. Control passes automatically to your owncode routines defined in the FIT by:

DX	End-of-data condition
EX	Error condition
CPA	Compression routine
DCA	Decompression routine

Padding -
The free space reserved in a file at creation time to accommodate additional records; specified as a percentage figure.

Permanent file -
A file on a mass storage permanent file device that can be retained for longer than a single job. It is protected against accidental destruction by the system and can be protected against unauthorized access.

Physical Record Unit (PRU) -
The smallest unit of information that can be transferred between a peripheral storage device and central memory. The PRU size is permanently fixed for all mass storage devices.

Primary Key -
A field in a record whose value uniquely identifies a record and determines the location of the record in a file. One primary key field exists for a given file. A file must be updated by primary key values. Contrast with Alternate Key.

PRU Device -
A mass storage device in which information has a physical structure governed by physical record units (PRUs).

Random Access -
Access method by which any record in a file can be accessed at any time in any order; applies only to mass storage files. Contrast with Sequential Access.

Record -
The largest collection of information passed between AAM and your program in a single read or write operation. You define the structure and characteristics of records within a file by declaring a record format. The beginning and ending points of a record are implicit in each format.

Rewind -
To position a file at beginning-of-information.

Sequential Access -
A method in which only the record located at the current file position can be accessed. Contrast with Random Access.

Sparse Key -
An alternate key that is used infrequently. Only those alternate key values of interest are included in the index file.

Symbolic Key -
An alphanumeric key used with indexed sequential files; 1 to 255 characters. See Integer Key.

Synonym Records -
Direct access file records whose primary keys hash to the same home block.

Working Storage Area -
An area within the user's field length intended for receipt of data from a file or transmission of data to a file.

SUMMARY OF FIT FIELDS D

This appendix summarizes the FIT fields that are applicable to the file organizations supported by AAM. The FIT fields applicable to indexed sequential files are listed in table D-1. The FIT fields applicable to actual key files are listed in table D-2. The FIT fields applicable to direct access files are listed in table D-3.

TABLE D-1. SUMMARY OF FIT FIELDS APPLICABLE TO INDEXED SEQUENTIAL FILES

FIT Field Mnemonic	Meaning	Allowable Values	Release Default If Any	Change After Creation	Notes	FILE Statement	FILExx Call	STOREF Call	IFETCH Function
BCK	Block checksums	YES, NO	NO	No	IFETCH return: 0 = NO 1 = YES	X	X	X	X
BFS	Buffer length in words	0 thru 2^{17}-1	Provided by AAM	Yes	AAM always ensures minimum needed	X	X	X	X
BZF	Input/output status word location								X
CL	Trailer count field length in characters	1 thru 6	0	No	RT=T only	X	X	X	X
CP	Trailer count field starting position in characters	0 thru 81870	0	No	RT=T only	X	X	X	X
CPA	Compression routine address	Routine number or name	0	Yes		X	X	X	X
C1	Binary length field	YES, NO	NO	No	RT=D/T only	X	X	X	X
DCA	Decompression routine address	Routine name	0	Yes	Required if CPA is routine name		X	X	X
DCT	Display code to collating sequence table	Table location	Standard CDC	No	KT=S only		X	X	X
DFC	Dayfile control	0,1,2,3	0	Yes		X	X	X	X
DP	Data block padding percentage	0 thru 99	0	No		X	X	X	X
DX	End-of-data exit	Routine name	0	Yes			X	X	X
ECT	Trivial error count								X
EFC	Error file control	0,1,2,3	0	Yes		X	X	X	X
EMK	Embedded primary key	YES, NO	NO	No		X	X	X	X
ERL	Trivial error limit	0 thru 511	0	Yes	0 allows indefinite number of errors	X	X	X	X
ES	Error status								X

TABLE D-1. SUMMARY OF FIT FIELDS APPLICABLE TO INDEXED SEQUENTIAL FILES (Contd)

FIT Field Mnemonic	Meaning	Allowable Values	Release Default If Any	Change After Creation	Notes	FILE Statement	FILExx Call	STOREF Call	IFETCH Function
EX	Error exit	Routine name	0	Yes			X	X	X
FL	Record length in characters	1 thru 81870	0	No	RT=F/Z only	X	X	X	X
FLM	File limit - maximum records	0 thru $2^{29}-1$	0	Yes	0 allows unlimited number of records	X	X	X	X
FNF	Fatal/nonfatal error				IFETCH return: 0 = nonfatal 1 = fatal				X
FO	File organization	IS	Required	No	IFETCH return: 3 = IS	X	X	X	X
FP	File position				IFETCH return: 0 = Mid record 1 = BOI 10_8 = EOK 20_8 = EOR 100_8 = EOI				X
FWB	Buffer location	Program location	Provided BY AAM	Yes			X	X	X
FWI	Flush buffer immediately	YES, NO	NO	Yes	IFETCH return: 0 = NO 1 = YES	X	X	X	X
HL	Fixed header length in characters	1 thru 81870	0	No	RT=T only	X	X	X	X
IP	Index block padding percentage	0 thru 99	0	No		X	X	X	X
KA	Key location	Program location		Yes	Required if EMK=NO		X	X	X
KL	Key length in characters	10 (KT=I) or 1 thru 255 (KT= S/U)	Required	No		X	X	X	X
KNE	Primary key not equal to alternate key	0, 1							X
KP	Key starting position within KA	0 thru 9	0	Yes	KT=S/U only	X	X	X	X
KT	Key type	I, S, U	S	No	IFETCH return: 1 = S 2 = I 3 = U	X	X	X	X
LFN	Logical file name	1 to 7 letters or digits	Required	No	Must start with a letter	X	X	X	X
LL	Length field length in characters	1 thru 6	0	No	RT=D only	X	X	X	X

TABLE D-1. SUMMARY OF FIT FIELDS APPLICABLE TO INDEXED SEQUENTIAL FILES (Contd)

FIT Field Mnemonic	Meaning	Allowable Values	Release Default If Any	Change After Creation	Notes	FILE Statement	FILExx Call	STOREF Call	IFETCH Function
LP	Length field starting position in characters	0 thru 81870	0	No	RT=D only	X	X	X	X
MBL	Data block and index block length in characters	0 thru 81870		No	AAM can calculate	X	X	X	X
MKL	Major key length in characters	0 thru KL	0	Yes	KT=S/U only; set to 0 at end of every operation.			X	X
MNR	Minimum record length in characters	0 thru MRL	0	No		X	X	X	X
MRL	Maximum record length in characters	1 thru 10(2^{13}-5)	Required	No		X	X	X	X
NDX	Index flag	0, 1	0	Yes	1 = index file operation	X	X	X	X
NL	Number of index levels	1 thru 15	1	No		X	X	X	X
NOFCP	No FILE control statement processing	YES, NO	NO	Yes			X	X	X
OC	Open/close flag				IFETCH return: 0 = never opened 1 = opened 2 = closed				X
OF	Open flag	R, E	R	Yes	Set at file open	X	X	X	X
ON	Old or new file	OLD, NEW	OLD	Yes	Can be set by OPENM	X	X	X	X
ORG	Old/new file organization	OLD, NEW	OLD	No	Must be set to NEW	X	X	X	X
PD	Processing direction	INPUT, OUTPUT, IO	INPUT	Yes	OPENM uses I-O; IFETCH return: 0 = INPUT 1 = INPUT 2 = OUTPUT 3 = IO	X	X	X	X
PKA	Primary key address	Address or 0					X	X	X
RB	Number of records per block	1 thru 4095	2	No	MBL overrides RB	X	X	X	X
RC	Record count								X
REL	Relation, key value to key position	EQ, GE, GT	EQ	Yes			X	X	X
RKP	Relative key position in RKW	0 thru 9	0	No	Required if EMK=YES	X	X	X	X
RKW	Relative key word in record	0 thru MRL	0	No	Required if EMK=YES	X	X	X	X

TABLE D-1. SUMMARY OF FIT FIELDS APPLICABLE TO INDEXED SEQUENTIAL FILES (Contd)

FIT Field Mnemonic	Meaning	Allowable Values	Release Default If Any	Change After Creation	Notes	FILE Statement	FILExx Call	STOREF Call	IFETCH Function
RL	Current record length	MNR thru MRL	0	Yes	Set by PUT			X	X
RMK	Record mark character	Any character	62_8	No	RT=R only; 62_8 = character	X	X	X	X
RT	Record type	F, R, Z, D, T, U	U	No		X	X		X
SB	Sign overpunch length field	YES, NO	NO	No	RT=D/T only	X	X	X	X
TL	Trailer length in characters	0 thru 2^{17}-1	0	No	RT=T only	X	X	X	X
WSA	Working storage area	Program location	Required	Yes	Set by file processing statements		X	X	X
XBS	Index file block size in characters	0 thru 81870	Data block size	No		X	X	X	X
XN	Index file name	1 to 7 letters or digits	0	No	Must start with letter	X	X	X	X

TABLE D-2. SUMMARY OF FIT FIELDS APPLICABLE TO ACTUAL KEY FILES

FIT Field Mnemonic	Meaning	Allowable Values	Release Default If Any	Change After Creation	Notes	FILE Statement	FILExx Call	STOREF Call	IFETCH Function
BCK	Block checksums	YES, NO	NO	No	IFETCH return: 0 = NO 1 = YES	X	X	X	X
BFS	Buffer length in words	0 thru 2^{17}-1	Provided by AAM	Yes	AAM always ensures minimum needed	X	X	X	X
BZF	Input/output status word location								X
CL	Trailer count field length in characters	1 thru 6	0	No	RT=T only	X	X	X	X
CP	Trailer count field starting position in characters	0 thru 81870	0	No	RT=T only	X	X	X	X
CPA	Compression routine address	Routine number or name	0	Yes			X	X	X
C1	Binary length field	YES, NO	NO	No	RT=D/T only	X	X	X	X
DCA	Decompression routine address	Routine name	0	Yes	Required if CPA is routine name		X	X	X

TABLE D-2. SUMMARY OF FIT FIELDS APPLICABLE TO ACTUAL KEY FILES (Contd)

FIT Field Mnemonic	Meaning	Allowable Values	Release Default If Any	Change After Creation	Notes	FILE State-ment	FILExx Call	STOREF Call	IFETCH Func-tion
DFC	Dayfile control	0,1,2,3	0	Yes		X	X	X	X
DP	Data block padding percentage	0 thru 99	0	No		X	X	X	X
DX	End-of-data exit	Routine name	0	Yes			X	X	X
ECT	Trivial error count								X
EFC	Error file control	0,1,2,3	0	Yes		X	X	X	X
EMK	Embedded primary key	YES, NO	NO	No			X	X	X
ERL	Trivial error limit	0 thru 511	0	Yes	0 allows indefinite number of errors	X	X	X	X
ES	Error status								X
EX	Error exit	Routine name	0	Yes			X	X	X
FL	Record length in characters	1 thru 81870	0	No	RT=F/Z only	X	X	X	X
FLM	File limit - maximum records	0 thru $2^{29}-1$	0	Yes	0 allows unlimited number of records	X	X	X	X
FNF	Fatal/nonfatal error				IFETCH return: 0 = nonfatal 1 = fatal				X
FO	File organization	AK	Required	No	IFETCH return: 6 = AK	X	X	X	X
FP	File position				IFETCH return: 0 = Mid record 1 = BOI 10_8 = EOK 20_8 = EOR 100_8 = EOI				X
FWB	Buffer location	Program location	Provided by AAM	Yes			X	X	X
FWI	Flush buffer immediately	YES, NO	NO	Yes	IFETCH return: 0 = NO 1 = YES	X	X	X	X
HL	Fixed header length in characters	1 thru 81870	0	No	RT=T only	X	X	X	X
KA	Key location	Program location		Yes	Required if EMK=NO		X	X	X
KL	Key length in characters	1 thru 8	Required	No		X	X	X	X
KNE	Primary key not equal to alternate key	0, 1							X

TABLE D-2. SUMMARY OF FIT FIELDS APPLICABLE TO ACTUAL KEY FILES (Contd)

FIT Field Mnemonic	Meaning	Allowable Values	Release Default If Any	Change After Creation	Notes	FILE Statement	FILExx Call	STOREF Call	IFETCH Function
KP	Key starting position within KA	0 thru 9	0	Yes		X	X	X	X
LFN	Logical file name	1 to 7 letters or digits	Required	No	Must start with a letter	X	X	X	X
LL	Length field length in characters	1 thru 6	0	No	RT=D only	X	X	X	X
LP	Length field starting position in characters	0 thru 81870	0	No	RT=D only	X	X	X	X
MBL	Data block length in characters	1 thru 81870		No	AAM can calculate	X	X	X	X
MNR	Minimum record length in characters	1 thru MRL	Required	No		X	X	X	X
MRL	Maximum record length in characters	1 thru 81870	Required	No		X	X	X	X
NDX	Index flag	0, 1	0	Yes	1 = index file operation	X	X	X	X
NOFCP	No FILE control statement processing	YES, NO	NO	Yes			X	X	X
OC	Open/close flag				IFETCH return: 0 = never opened 1 = opened 2 = closed				X
ON	Old or new file	OLD, NEW	OLD	Yes	Can be set by OPENM	X	X	X	X
ORG	Old/new file organization	OLD, NEW	OLD	No	Must be set to NEW	X	X	X	X
PD	Processing direction	INPUT, OUTPUT, IO	INPUT	Yes	OPENM uses I-O; IFETCH return: 0 = INPUT 1 = INPUT 2 = OUTPUT 3 = IO	X	X	X	X
PKA	Primary key address	Address or 0					X	X	X
RB	Number of records per block	1 thru 4095	8	No		X	X	X	X
RC	Record count								X
RKP	Relative key position in RKW	0 thru 9	0	No	Required if EMK=YES	X	X	X	X
RKW	Relative key word in record	0 thru MRL	0	No	Required if EMK=YES	X	X	X	X
RL	Current record length	MNR thru MRL	0	Yes	Set by PUT			X	X

TABLE D-2. SUMMARY OF FIT FIELDS APPLICABLE TO ACTUAL KEY FILES (Contd)

FIT Field Mnemonic	Meaning	Allowable Values	Release Default If Any	Change After Creation	Notes	FILE State-ment	FILExx Call	STOREF Call	IFETCH Func-tion
RMK	Record mark character	Any char-acter	62_8	No	RT=R only; 62_8 = character	X	X	X	X
RT	Record type	F, D, Z, D, T, U	U	No		X	X		X
SB	Sign overpunch length field	YES, NO	NO	No	RT=D/T only	X	X	X	X
TL	Trailer length	0 thru 2^{17}-1	0	No	RT=T only	X	X	X	X
WSA	Working storage area	Program location	Required	Yes	Set by file processing statements		X	X	X
XBS	Index file block size	0 thru 81870	Data block size	No		X	X	X	X
XN	Index file name	1 to 7 letters or digits	0	No	Must start with letter	X	X	X	X

TABLE D-3. SUMMARY OF FIT FIELDS APPLICABLE TO DIRECT ACCESS FILES

FIT Field Mnemonic	Meaning	Allowable Values	Release Default If Any	Change After Creation	Notes	FILE State-ment	FILExx Call	STOREF Call	IFETCH Func-tion
BCK	Block checksums	YES, NO	NO	No	IFETCH return: 0 = NO 1 = YES	X	X	X	X
BFS	Buffer length in words	0 thru 2^{17}-1	Provided by AAM	Yes	AAM always ensures mini-mum needed	X	X	X	X
BZF	Input/output status word location								X
CL	Trailer count field length in characters	1 thru 6	0	No	RT=T only	X	X	X	X
CP	Trailer count field starting position in characters	0 thru 81870	0	No	RT=T only	X	X	X	X
CPA	Compression routine address	Routine number or name	0	Yes			X	X	X
C1	Binary length field	YES, NO	NO	No	RT=D/T only	X	X	X	X
DCA	Decompression routine address	Routine name	0	Yes	Required if CPA is routine name		X	X	X
DFC	Dayfile control	0,1,2,3	0	Yes		X	X	X	X

TABLE D-3. SUMMARY OF FIT FIELDS APPLICABLE TO DIRECT ACCESS FILES (Contd)

FIT Field Mnemonic	Meaning	Allowable Values	Release Default If Any	Change After Creation	Notes	FILE Statement	FILExx Call	STOREF Call	IFETCH Function
DX	End-of-data exit	Routine name	0	Yes			X	X	X
ECT	Trivial error count								X
EFC	Error file	0,1,2,3	0	Yes		X	X	X	X
EMK	Embedded primary key	YES, NO	YES	No			X	X	X
ERL	Trivial error limit	0 thru 511	0	Yes	0 allows indefinite number of errors	X	X	X	X
ES	Error status								X
EX	Error exit	Routine name	0	Yes			X	X	X
FL	Record length in characters	1 thru 81870	0	No	RT=F/Z only	X	X	X	X
FLM	File limit - maximum records	0 thru $2^{29}-1$	0	Yes	0 allows unlimited number of records	X	X	X	X
FNF	Fatal/nonfatal error				IFETCH return: 0 = nonfatal 1 = fatal				X
FO	File organization	DA	Required	No	IFETCH return: 5 = DA	X	X	X	X
FP	File position				IFETCH return: 0 = Mid Record 1 = BOI 10_8 = EOK 20_8 = EOR 100_8 = EOI				X
FWB	Buffer location	Program location	Provided by AAM	Yes			X	X	X
FWI	Flush buffer immediately	YES, NO	NO	Yes	IFETCH return: 0 = NO 1 = YES	X	X	X	X
HL	Fixed header length in characters	1 thru $10(2^{13}-5)$	0	No	RT=T only	X	X	X	X
HMB	Number of home blocks	1 thru $2^{24}-1$	Required	No			X	X	X
HRL	Hashing routine location	Routine name	0	No	0 uses system hashing routine		X	X	X
KA	Key location	Program location		Yes	Required if EMK=NO		X	X	X
KL	Key length in characters	1 thru MRL	0	No		X	X	X	X

TABLE D-3. SUMMARY OF FIT FIELDS APPLICABLE TO DIRECT ACCESS FILES (Contd)

FIT Field Mnemonic	Meaning	Allowable Values	Release Default If Any	Change After Creation	Notes	FILE Statement	FILExx Call	STOREF Call	IFETCH Function
KNE	Primary key not equal to alternate key	0, 1							X
KP	Key starting position within KA	0 thru 9	0	Yes		X	X	X	X
LFN	Logical file name	1 to 7 letters or digits	Required	No	Must start with a letter	X	X	X	X
LL	Length field length in characters	1 thru 6	0	No	RT=D only	X	X	X	X
LP	Length field starting position in characters	0 thru 81870	0	No	RT=D only	X	X	X	X
MBL	Data block length in characters	0 thru 81870		No	AAM can calculate	X	X	X	X
MNR	Minimum record length in characters	1 thru MRL	Required	No		X	X	X	X
MRL	Maximum record length in characters	1 thru 81870	Required	No		X	X	X	X
NDX	Index flag	0, 1	0	Yes	1 = index file operation	X	X	X	X
NOFCP	No FILE control statement processing	YES, NO	NO	Yes			X	X	X
OC	Open/close flag				IFETCH return: 0 = never opened 1 = opened 2 = closed				X
ON	Old or new file	OLD, NEW	OLD	Yes	Can be set by OPENM	X	X	X	X
ORG	Old/new file organization	OLD, NEW	OLD	No	Must be set to NEW	X	X	X	X
PD	Processing direction	INPUT, OUTPUT, IO	INPUT	Yes	OPENM uses I-O; IFETCH return: 0 = INPUT 1 = INPUT 2 = OUTPUT 3 = IO	X	X	X	X
RB	Number of records per block	1 thru 4095	2	No	MBL overrides RB	X	X	X	X
RC	Record count								X
RKP	Relative key position in RKW	0 thru 9	0	No	Required if EMK=YES	X	X	X	X
RKW	Relative key word in record	0 thru MRL	0	No	Required if EMK=YES	X	X	X	X
RL	Current record length	MNR thru MRL	0	Yes	Set by PUT			X	X

TABLE D-3. SUMMARY OF FIT FIELDS APPLICABLE TO DIRECT ACCESS FILES (Contd)

FIT Field Mnemonic	Meaning	Allowable Values	Release Default If Any	Change After Creation	Notes	FILE State-ment	FILExx Call	STOREF Call	IFETCH Func-tion
RMK	Record mark character	Any char-acter	62_8	No	RT=R only; 62_8 = character	X	X	X	X
RT	Record type	F, R, Z, D, T, U	U	No		X	X		X
SB	Sign overpunch length field	YES, NO	NO	No	RT=D/T only	X	X	X	X
TL	Trailer length	0 thru 2^{17}-1	0	No	RT=T only	X	X	X	X
WSA	Working storage area	Program location	Required	Yes	Set by file processing statements		X	X	X
XBS	Index file block size	0 thru 81870	Data block size	No		X	X	X	X
XN	Index file name	1 to 7 letters or digits	0	No	Must start with letter	X	X	X	X

INDEX

COMMENT SHEET

MANUAL TITLE: CYBER Record Manager Advanced Access Methods Version 2 User's Guide

PUBLICATION NO.: 60499400

REVISION: C

This form is not intended to be used as an order blank. Control Data Corporation welcomes your evaluation of this manual. Please indicate any errors, suggested additions or deletions, or general comments on the back (please include page number references).

_____ Please reply _____ No reply necessary

FOLD FOLD

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 8241 MINNEAPOLIS, MN.

POSTAGE WILL BE PAID BY ADDRESSEE

CONTROL DATA

Publications and Graphics Division
Mail Stop: SVL104
P.O. Box 3492
Sunnyvale, California 94088-3492

FOLD FOLD

CUT ALONG LINE

NO POSTAGE STAMP NECESSARY IF MAILED IN U.S.A.
FOLD ON DOTTED LINES AND TAPE

NAME:

COMPANY:

STREET ADDRESS:

CITY/STATE/ZIP:

TAPE TAPE