SOFTWARE MANUAL

ISAM SYSTEM

USER'S GUIDE

DWM-00100-06

REV. A02

alpha micro

SOFTWARE MANUAL

ISAM SYSTEM

USER'S GUIDE

DWM-00100-06

REV. A03

alpha micro

This printing of the manual contains Change Page Packet #1 for the "ISAM System User's Guide", (DSS-10000-22), which may be ordered separatedly from Alpha Micro.

First printing: 6 December 1977
Second printing: April 1979
Third printing: 30 April 1981
Fourth printing: 31 October 1981

ALPHA MICROSYSTEMS
17881 Sky Park North
Irvine, CA 92714

This document reflects AMOS Versions 4.6 and later

October 1981

IMPORTANT NOTICE FOR ISAM USERS

1.0 INTRODUCTION

ISAM is software package that allows you to organize and retrieve data in a file via a system of index files. You may call ISAM from within your own AlphaBASIC or assembly language programs. For more information on ISAM, see the ISAM System User's Guide, (DWM-00100-07), Revision A03, and the AlphaBASIC User's Manual, (DWM-00100-01).

The following notice is intended for experienced assembly language programmers whose assembly language programs make use of ISAM:

You should be aware of the following change to ISAM--

As of AMOS Release 4.6, ISAM supports a new feature: when using ISAM to find the next data record (by order of the symbolic keys as they appear in the index file), you may optionally ask ISAM to return the symbolic key as well as the relative record number of the data record.

The implementation of this feature has changed the way that the .SREDR ISAM call works: ISAM now looks at the contents of a register that was not formerly checked. If you wish to use the AMOS 4.6 version of ISAM, you may need to check your assembly language programs to see that they do not conflict with the new ISAM's calling convention. If you do not wish to make use of this new feature (and do not want to check all .SREDR calls in your programs), you may wish to keep your AMOS 4.5 version of ISAM when updating to AMOS 4.6.

For exact information on the change to the way you must use the .SREDR call, see the ISAM System User's Guide, (DWM-00100-07), Revision A03 or later. (NOTE: This information is also in the Change Page Packet #1 for the "ISAM System User's Guide", (DSS-10000-22).)

PREFACE

This manual is aimed at the experienced assembly language or BASIC programmer who wishes to make use of the Alpha Micro ISAM system within his or her own programs. If you are not familiar with Alpha Micro Assembly Language, please refer to the AMOS Assembly Language Programmer's Reference Manual, (DWM-00100-43). If you wish information on AlphaBASIC, refer to the AlphaBASIC User's Manual, (DWM-00100-01).

Table of Contents

CHAPTER 1 INTRODUCTION TO ISAM

1.1 THE INDEXED SEQUENTIAL ACCESS METHOD 1-1
1.2 DESCRIPTION OF INDEXED SEQUENTIAL FILES 1-3
1.2.1 The Data File 1-4
1.2.2 The Index File 1-4
1.3 ISAM ACCESS MODES 1-5
1.3.1 Counted Update Mode 1-5
1.3.2 Exclusive Open Mode 1-6
1.3.3 Hints and Restrictions 1-7
1.4 CONVERTING VERSION 4.2 ISAM FILES TO ISAM VERSIONS 4.3 OR LATER 1-8

PART I THE ISAM UTILITY PROGRAMS

CHAPTER 2 CREATING AND LOADING AN ISAM FILE WITH ISMBLD

2.1 GENERAL OPERATING INSTRUCTIONS 2-1
2.2 CREATION MODE 2-1
2.2.1 Specifying File Parameters 2-2
2.2.1.1 Size of key: 2-2
2.2.1.2 Position of key: 2-2
2.2.1.3 Size of data record: 2-2
2.2.1.4 Number of records to allocate: 2-2
2.2.1.5 Entries per index block: 2-2
2.2.1.6 Empty index blocks to allocate: 2-3
2.2.1.7 Primary Directory? 2-3
2.2.1.7.1 Secondary File 2-3
2.2.1.8 Data File Device? 2-3
2.3 FILE LOADING MODE 2-3
2.3.1 Suppressing Exclusive Open Mode (the /N Switch) 2-4
2.4 CHANGING THE DATA FILE DEVICE (THE /D SWITCH) 2-4
2.5 OPTIMIZING FILE PARAMETERS 2-4
2.5.1 Entries per Index Block 2-5
2.5.2 Empty Index Blocks to Allocate 2-5

CHAPTER 3 DUMPING AN ISAM FILE WITH ISMDMP

3.1 GENERAL OPERATING INSTRUCTIONS 3-1
3.1.1 Suppressing Exclusive Open Mode 3-1
3.2 FILE DUMP MODE 3-2
3.2.1 Sample Data File Display 3-2
3.3 INDEX FILE DUMP MODE 3-2
3.3.1 Sample Index File Display 3-3

CHAPTER 4 COMPRESSING INDEX FILES WITH ISMCOM

PART II PROGRAMMING WITH ISAM

CHAPTER 5 USING ISAM FROM THE ASSEMBLY LANGUAGE LEVEL

5.1 GENERAL CALLING SEQUENCE 5-1
5.2 INITIALIZING THE ISAM SYSTEM (.INIT) 5-2
5.2.1 Calling Sequence 5-2
5.2.2 User Supplied Allocation and Deallocation Routines 5-2
5.3 FINALIZING ISAM PROCESSING (.IFIN) 5-3
5.3.1 Calling Sequence 5-3
5.4 OPENING A FILE FOR PROCESSING (.IOPNR) 5-3
5.4.1 Calling Sequence 5-4
5.5 CLOSING THE FILE AFTER PROCESSING (.ICLOS) ... 5-4
5.5.1 Calling Sequence 5-4
5.6 LOCATING A FREE DATA RECORD (.IGTFR) 5-5
5.6.1 Calling Sequence 5-5
5.7 DELETING A DATA RECORD (.IDLFR) 5-5
5.7.1 Calling Sequence 5-5
5.8 READING A DATA RECORD BY RELATIVE RECORD NUMBER (.IRLRD) 5-6
5.8.1 Calling Sequence 5-6
5.9 WRITING A DATA RECORD BY RELATIVE RECORD NUMBER (.IRLWT) 5-6
5.9.1 Calling Sequence 5-6
5.10 FINDING A RECORD (.IREDR) 5-7
5.10.1 Calling Sequence 5-7
5.11 ADDING A SYMBOLIC KEY (.IWRTR) 5-7
5.11.1 Calling Sequence 5-7
5.12 DELETING A KEY (.IDELK) 5-8
5.12.1 Calling Sequence 5-8
5.13 FINDING THE NEXT SEQUENTIAL KEY (.SREDR) 5-8
5.13.1 Calling Sequence 5-9

CHAPTER 6 STANDARD ISAM SYMBOLS FOR ASSEMBLY LANGUAGE PROGRAMMERS

6.1 CALLING SYMBOLS 6-1
6.2 COMPLETION CODE SYMBOLS 6-1

CHAPTER 7 USING ISAM FROM WITHIN BASIC

7.1 OPENING AN INDEXED SEQUENTIAL FILE 7-1
7.2 THE ISAM STATEMENT 7-3
7.2.1 ISAM Statement Codes 7-3
7.3 READING AND WRITING DATA IN AN ISAM DATA FILE 7-4
7.4 CLOSING FILES 7-5
7.5 ERROR PROCESSING 7-5
7.6 USING THE ISAM FUNCTIONS WITHIN A BASIC PROGRAM 7-6
7.6.1 Adding Data to an Indexed Sequential File 7-6
7.6.2 Reading Data Records in Symbolic Key Order 7-8
7.6.3 Reading Data Records Randomly by Symbolic Key 7-8
7.6.4 Updating Data Records 7-9
7.6.5 Deleting a Data Record 7-9
7.7 SAMPLE ISAM PROGRAM 7-11

INDEX

CHAPTER 1

INTRODUCTION TO ISAM

The purpose of this short manual is: 1. to give you an introduction to ISAM; and, 2. to discuss how you can create and access indexed sequential files using the various programs of the ISAM system, as well as write programs in either assembly language or BASIC to locate, update, add, and delete data in those files.

ISAM is a method for organizing and retrieving data. The name of the method (Indexed Sequential Access Method) refers to the manner in which the data is organized. The information in the ISAM data file is accessed by searching a separate index file that contains a group of symbolic keys and pointers to records in the data file with which those keys are associated. By searching several levels of indices within the index file, we can locate records in a separate file much more quickly and efficiently than if we had to search the actual data file itself. Some examples of data for which symbolic keys can be specified are:

Customer information-- the name of the customer is the key (that is, an element of the data record) on which you base your search.

Payroll-- the key is an employee number.

Inventory control-- the key is a part number.

1.1 THE INDEXED SEQUENTIAL ACCESS METHOD

Finding a convenient and efficient way to access information in a file is an important problem for a programmer. Suppose, for example, that you have a phone book of five thousand names and phone numbers. If you need to find a specific person's phone number, you can start with page number one and scan every entry in the book until you find the proper name. That process is very slow and inefficient, however, because you have to deal with so many entries. A more efficient method would involve dividing the phone book into sections, and searching only those sections that might contain the data you need instead of searching the entire data base.

Organizing your data so that it is easier and quicker to search is the main idea behind ISAM. If we were to organize the phone book in somewhat the same way as ISAM would do it, we might do this:

A. First we build a file containing one logical record for each entry in the phone book; each record consists of a person's name, an address, and a phone number. We assign each entry a number (called the relative record number or the relative key) that marks its position in the file. For example, the five hundredth entry is number 499 (the first record is number 0, not 1). (The record number is called "relative" because it marks the position of the record from the front of the file; it is not an absolute disk address.) This file that contains all of our data corresponds to the ISAM data file.

B. Next we construct a file that contains information about the data file that helps us search the data file. (The file we are constructing corresponds to the ISAM index file.) When you open a phone book, you notice that the top of each page contains two words; the first and the last names that appear on that page. These two names give you an "index" into the data on that page. So, if the two words at the top of a particular page are "PENDERGRASS-PENNINGTON," you know that the names associated with all entries on that page fall somewhere in that range. Suppose, then, that this second file we are creating contains the words at the top of the phone book pages, along with the relative record numbers of the entries that fall on each page.

   Instead of searching the entire data file, we can search this much smaller "index" file. If we want to find the entry for the name PENHALL, we can search the page indices in our index file until we find two names that PENHALL falls between. Then we can search just the data file records associated with that range of names until we find PENHALL.

   When we build an index file, we say that the file contains symbolic keys. A symbolic key is an element of a logical record on which we base our search. In this example, the symbolic key we are using is the name associated with each phone book entry. We might just as easily have set up the files so that we can base our search on phone numbers or city names.

C. We have improved our original file-search procedure, but it can be improved upon still further. We now have a data file and an index file. The index file contains one level of indices (the words at the top of each page in the phone book). The next step is to provide another level of indices within the index file.

   When you look for a name in a phone book, you first find the proper page by glancing at the names at the top of each page. Then you might look at the first and last names of each column on the page to narrow your search still further. If the name for which you are searching falls between the names at the top and bottom of the

column, you begin to search each entry in that column; otherwise, you move to the next column on the page. In the same way, our index file contains a first-level index (the names at the top of the page); then it further divides the data on the page by giving indices into subgroups of entries on the page (the first and last names in each column make up the second-level index). The final level of indices (the third-level) in the index file consists of lists of names for each column in the book along with the actual record number in our original data file that contains the entry associated with that name.

D. Journeying through the levels of indices in our index file, then, we first find the page on which the name appears, then we find the column in which the name appears, then we find the actual record number of the file in which the entire entry associated with that name appears. At no time do we ever need to search the actual data file itself.

   Note that the keys in the index file are grouped alphabetically. Since we find a data record by searching the index file, the data records in the data file do not need to be arranged in any particular order. An index file may not contain duplicate keys; that is, no two data records in the data file may have the same symbolic key.

The example above discusses a data file that has one index file (called the primary index file). A data file always has one primary index file; it may also have one or more secondary index files. A secondary index file is structured in the same way as the primary index file except that it contains different symbolic keys. For example, if we want to base our search of phone book entries on phone numbers as well as names, we might construct a secondary index file that contains phone numbers.

Although we constructed the example above ourselves, the ISAM program automatically creates all data files and index files for you in response to information and file specifications that you supply.

1.2 DESCRIPTION OF INDEXED SEQUENTIAL FILES

In summary, an indexed sequential file consists of two files: 1. the data file, containing the actual data; and 2. the index file, containing pointers to symbolic keys within the data file. You specify the location of the symbolic key within each record when you build an indexed sequential file using the ISMBLD program (discussed in Chapter 2, "Creating and Loading an ISAM File with ISMBLD"). To build an indexed sequential file, you supply certain parameters to the ISMBLD program; ISMBLD then produces an empty file. To load the file with data, you may write your own program or you may use the ISMBLD program to copy the data from an ordinary sequential file into the data file (updating the index file in the process).

The ISAM program does all reading and writing of the index file; you will not have to handle these functions yourself. Your BASIC or assembly language program will add, delete, or update data in the ISAM data file based on the relative record number returned to your program by ISAM.

1.2.1 The Data File

The data in your data file may be in any data format; however, the index file orders keys in ASCII collating sequence (i.e., ascending binary order) which may affect operation of the ISAM program when data is recorded in other than ASCII form. When you build an indexed sequential file (via the ISMBLD program), you supply various items of information about your data file (e.g., the size of the data records, the location of the symbolic key within the data record, and so on); ISMBLD then builds both the data file and its primary index file.

Your programs use the ISAM functions to add and delete data records in the data file. When you add a record, ISAM inserts it into the first free space in your data file. When you delete a record, ISAM does so by recovering the space in the data file used by that record, and returning that area to the free record list so that it is available for new records. Because the Alpha Micro operating system (AMOS) requires that contiguous files (e.g., an ISAM data file) be preallocated, once the data file is full it must be reorganized before it can be used further. For this reason, be careful to allocate as many records as you will need for the file.

All ISAM data files MUST have the extension .IDA.

1.2.2 The Index File

The ISMBLD program automatically creates the index file from a description of the data file. The index file contains three levels of indices, the lowest of which contains pointers to the records in the data file. Each successive index level points to all the blocks containing the next lower level index. Index levels are provided so that the entire index need not be searched each time a symbolic key is accessed. When a symbolic key is accessed, ISAM reads the highest level index to find which lower level index contains a pointer to the approximate location of that key. ISAM then searches the block of that lower level index; that index block in turn points to a lower index block which points to the data record in which the key is stored.

In addition to the index blocks, the index file contains another block named the Directory Rock, so called because it never moves. This block contains information describing the index and data files as well as maintenance information (e.g., free record links, access counts, etc.).

Each data file must have a primary index file; in addition to this, it may have several secondary index files. A typical example of the use of this

feature would be a mailing list maintenance program, where the data is keyed on both a hashed retrieval code for unique reference and also keyed on the person's name. (For an example of this kind of program, see the sample BASIC ISAM program in Chapter 7, "Using ISAM From Within BASIC.")

Your programs use the ISAM functions to add and delete keys from the index files, and to locate data records in the corresponding data files.

The extension of an ISAM index file MUST be .IDX.

1.3 ISAM ACCESS MODES

Beginning with AMOS version 4.2, changes have been made to ISAM that greatly increase its file access speed. The increase in speed was made possible by the two new access modes, Counted Update mode and Exclusive Open mode, which allow ISAM to avoid unnecessary processing of your index files.

Counted Update mode is the normal access mode for assembly language or BASIC programs. Exclusive Open mode is the normal access mode for the ISMDMP and ISMBLD programs and is the only mode for the ISMCOM program. ISAM always processes indexed sequential files in one or the other of these modes. The next two sections discuss both of these modes.

NOTE: The paragraphs below mention the need for file interlocking. It is most important that your programs guard against the possibility of more than one user trying to update the same data file at the same time. If several users were to try to write to the same file record at the same time, severe damage to your data file could result. For information on file interlocking procedures, see the documents FLOCK - BASIC Subroutine to Coordinate Multi-user File Access and XLOCK - BASIC Subroutine for Multi-user Locks in the "BASIC Programmer's Information" section of the AMOS Software Update documentation packet.

1.3.1 Counted Update Mode

Counted Update mode allows ISAM to increase its speed by avoiding any unnecessary processing. Every time ISAM updates a file in any way, it increments a counter in the Rock portion of the index file. At the time of file access, ISAM checks this counter to see if the file has been updated since the last access. If the file has not been updated, ISAM can skip futher access initialization and take advantage of its prior knowledge about the file. These actions are completely transparent to the user and the speed gains (3 to 70 times faster access times) are free.

IMPORTANT NOTE: The Counted Update mode does NOT eliminate file interlock requirements from your programs. If anyone might possibly be updating the file, your program must continue to use file interlock programs such as XLOCK or FLOCK to prevent simultaneous updates or accesses. The preferred method for locking files is to use the FLOCK non-exclusive "open" locking

(action 0, mode 0 or 4) for reading, and to use the FLOCK exclusive "open" locking (action 0, mode 2 or 6) for updating. Use the FLOCK "close" (action 1, mode 0) to release the file for other users.

Note that it is not necessary to open and close the ISAM file with each manipulation even though the FLOCK commands are so named. It is acceptable to leave the file open during the whole interlocking and release process and is, in fact, the only way to gain the speed increase made possible by the Counted Update mode.

1.3.2 Exclusive Open Mode

When a program opens a file exclusively, ISAM renames the .IDX file to a .IDY extension. ISAM also sets a flag in the Rock that identifies the file as an exclusive file. If any other job tries to open that file, it receives a "?File not found" error; if another job tries to access the file once it is open, the job receives a "?Link structure smashed" error (IS.LSS).

As a result of the exclusive open, ISAM knows that no other program will be updating or accessing the file. It can therefore take full advantage of the single-process situation for initialization and change posting. Except for the process of opening the file and the need to properly close the file, use of ISAM is the same as in previous versions. The use of this mode results in an extremely large gain in access speed.

The only file interlock problem occurs at the moment of the ISAM OPEN call; no one may update the file while you are opening it. You MUST prevent this situation from occurring by using one of the file interlock programs, FLOCK or XLOCK, or by simply making sure that no other user is running a program that can update that file. Once your program has executed the ISAM OPEN call, your program needs no interlocks since no one else can access the file.

Invoke the Exclusive Open mode from within BASIC by using the file mode of INDEXED'EXCLUSIVE in the file OPEN statement in your program. Your assembly language program may select the Exclusive Open mode by setting bit number three of R0 for the .IOPNR call. You MUST close the file when you are completely done with manipulating that file so that ISAM can post the final updates and remove the Exclusive Open conditions from that file. If an error occurs during processing, you should close the file to remove the Exclusive Open conditions (although you can also remove them manually). A file in which an error occurred during updating is probably badly damaged.

Use of the Exclusive Open mode can result in significant gains when printing reports and other such batch-type operations. It does have the drawback that no one else can access the file for any reason while it is exclusively opened. Be warned that any attempts to circumvent the exclusive properties of such a file by clever manipulations will probably meet with disaster. If several people need to access the file at the same time, use the normal mode, Counted Update mode; if no one updates the file, you will lose very little speed in changing to that mode.

The ISMBLD, ISMDMP, and ISMCOM programs use the Exclusive Open mode. To prevent ISMBLD (when loading or cross-indexing an existing file) or ISMDMP from using Exclusive Open mode, use the /N switch. The /N switch must appear at the very end of the command line that invokes the program. For example:

```
.ISMBLD LABELS/N RET
```

ISMBLD (when creating/loading or creating/cross-indexing) and ISMCOM always use the Exclusive Open mode.

1.3.3 Hints and Restrictions

The new access modes make possible a dramatic increase in the speed of ISAM data accesses. They also may result in slightly peculiar situations of which you should be aware:

1. If the Counted Update mode counter has not changed, ISAM assumes that no updates have been made to the file since the last time an access was made, and that it may therefore make certain assumptions about file status and contents. The counter cycles on a count of 16,777,216. If by some very unlikely chance the file were to remain open for an incredibly long time and exactly 16,777,216 updates were made between accesses, ISAM would access and/or update the file using out-of-date information.

   Although not strictly impossible, it is very unlikely that this situation will occur. We estimate that you would have to leave your machine up and running for several weeks with the ISAM file open without making any accesses to that file in order to see this happen.

2. When you open a file in Exclusive Open mode, ISAM must be able to write to the disk that contains the file. This means that you must make sure that the disk is not write-protected even if all accesses to that file are going to be read operations.

3. The most visible quirk of the Exclusive Open mode is that it causes ISAM to rename the extension of the file being opened from .IDX to .IDY. If such a file is not properly closed (for whatever reason), then the name of that file will not be correct in the disk directory. You can cure this problem very easily by using the RENAME command. For example:

   ```
   .RENAME *.IDX=*.IDY RET
   ```

   A file OPEN also changes a flag in the Rock of the ISAM index file. You do not need to worry about changing the flag yourself in the event of an improperly closed file, since the situation is automatically self-correcting the next time the file is opened (in either Exclusive Open or Counted Update mode).

4. If you open a file in Counted Update mode but the file was last used in Exclusive Open mode and was never closed, the file OPEN will cause ISAM to write to the file to correct the exclusive flag. If this situation is going to occur, make sure that the disk is write-enabled.

1.4 CONVERTING VERSION 4.2 ISAM FILES TO ISAM VERSIONS 4.3 OR LATER

If you have ISAM files built under ISAM version 4.2, you will need to use the ISMFIX program to convert them over to ISAM versions 4.3 and later. (Although using ISMFIX on files that were built under ISAM versions 4.3 and later doesn't do anything useful, it doesn't harm the files either.)

Because various conversion steps may be necessary to convert your ISAM files from one ISAM version to another, it is wisest not to skip any ISAM versions. (For example, going directly from ISAM version 4.1 to ISAM version 4.5 with existing ISAM files is not a good idea and, in fact, won't work.)

For information on ISMFIX, see the ISMFIX reference sheet in the AMOS System Commands Reference Manual, (DWM-00100-49).

ISAM SYSTEM USER'S GUIDE

PART I

THE ISAM UTILITY PROGRAMS

The next few chapters discuss the ISAM utility programs. These programs: 1. create and (optionally) load an indexed sequential file; 2. display the contents of your data and index files; and, 3. allow more efficient use of your index files by compressing index block entries.

CHAPTER 2

CREATING AND LOADING AN ISAM FILE WITH ISMBLD

The ISMBLD program provides a convenient method for creating and loading indexed sequential files. It gives you the ability to create a new indexed sequential file, to add records to the data file from an ordinary sequential data file, and to create a secondary index file that cross-indexes to a primary index file.

2.1 GENERAL OPERATING INSTRUCTIONS

ISMBLD has three operating modes: 1. create a new indexed file; 2. add data to the new file or to an existing file; and, 3. change the device specification of a data file. All modes are called via the general command:

.ISMBLD filespec{/D}{/N}(RET)

If the indexed sequential file specified by filespec does not exist, ISMBLD enters the creation mode. If the file already exists, ISMBLD enters the data loading mode unless you have specified the optional /D maintenance switch. (NOTE: If the file already exists, you may specify Counted Update mode by using the /N switch. See Section 2.3.1, "Suppressing Exclusive Open Mode.")

2.2 CREATION MODE

The creation mode is the most commonly used mode. In this mode you input series of parameters that describe the desired indexed sequential file. From these parameters the ISMBLD program generates a data file/primary index file combination or a secondary index file that cross-indexes to an existing primary index file.

2.2.1 Specifying File Parameters

Before actually creating the file, ISMBLD asks you a number of questions about your data file. In response to each of the questions, you are expected to enter a valid answer. Because of the myriad ways that you can set up an indexed sequential file, very little validity checking is done on your answers. It is therefore possible to create totally useless files. Be careful. For an example of the ISMBLD dialog, turn to Chapter 7, "Using ISAM From Within BASIC."

The following sections describe the questions asked and the expected responses:

2.2.1.1 Size of key: - Enter the size of the desired key in decimal bytes. To minimize index search time, keep this size as small as possible. The maximum key size is 256. When you later access the ISAM files you are now creating, you must remember to pad with blanks or other characters keys that are smaller than this specified size. Pad numeric fields in the front of the field; pad symbolic keys at the end. One side effect of this is that both binary and floating point keys may be used.

2.2.1.2 Position of key: - This parameter specifies the location of the key within the data record. The symbolic key position is used when loading indexed sequential files from sequential files as the means of determining the symbolic key. Enter the number of the first character-position in the record which the key occupies; the first position within a record is position number one.

2.2.1.3 Size of data record: - This parameter defines the size of the records in the data file or the maximum data record size in the case of variable length records. Specify this size in bytes (decimal). The data record size must be greater than or equal to the key size plus the key position.

2.2.1.4 Number of records to allocate: - This parameter defines the number of records which the data file is to contain.

2.2.1.5 Entries per index block: - This parameter allows you to specify the number of entries contained in an index block; this value can greatly affect the efficiency of searches and inserts within the file. See Section 2.5, "Optimizing File Parameters," for more information.

2.2.1.6 Empty index blocks to allocate: - ISMBLD allocates for you the bare minimum number of index blocks you will need to contain keys for the specified number of data records. This calculation is based on the assumption that the index file tree structure will be perfectly balanced. Since this is rarely the case, you will probably need to specify an additional number of index blocks.

2.2.1.7 Primary Directory? - If you are creating a primary index and data file combination, enter Y; if you are creating a secondary index file, enter N.

2.2.1.7.1 Secondary File - If you are building a secondary index file, ISMBLD prompts you for the file specification of the primary index file:

Secondary index to file:

Enter the specification of the primary index file to which this secondary file cross-indexes. Type just a RETURN to exit ISMBLD. You may create as many secondary index files as you want that cross-index to a particular primary index file by re-invoking ISMBLD with the specification of that primary index file and specifying a new secondary index file.

If you have created a secondary index file, your dialog with ISMBLD is now over. ISMBLD returns you to AMOS command level. If you are creating a data file/primary index file combination, ISMBLD asks you for more information (see below).

2.2.1.8 Data File Device? - ISMBLD now asks you:

Data File Device?

If the data file is to be on a different device than the index file, enter the name (and number) of that device. If they are to be on the same device, enter a RETURN. For example, if the data file is to be on unit 1 of device "DSK," enter:

Data File Device? DSK1: RET

2.3 FILE LOADING MODE

After an indexed sequential file has been created, it is often desirable to load the data and index files with data from an ordinary sequential data file. To allow this, ISMBLD enters the data loading mode once it creates the indexed sequential file.

If you want to load data into an existing data file, invoke ISMBLD with the name of that file. ISMBLD then responds:

[Processing existing file]

This notifies you that you are in the file loading mode and not the creation mode.

ISMBLD now prompts you for a sequential file specification by typing:

Load from file:

You may now enter the file specification that selects the sequential data file from which you want to load. A default extension of .SEQ is assumed by ISMBLD. (If you do not want ISMBLD to load the new file for you or if you have made an error in the file specification you gave to ISMBLD, type a RETURN after the "Load from file:" prompt; no data will be added to the data file.)

2.3.1 Suppressing Exclusive Open Mode (the /N Switch)

When loading an existing file, ISMBLD normally uses Exclusive Open mode. If you wish it to use Counted Update mode instead, include the /N switch at the end of the ISMBLD command line. For example:

.ISMBLD MAIL/N RET

2.4 CHANGING THE DATA FILE DEVICE (THE /D SWITCH)

The only creation data that you can change is the data file device. The /D switch provides this field for examination and change. Simply enter the new device name or a RETURN (to leave the device unchanged). To change the device to the same device that the index file uses, enter a period (.) only. It is your reponsiblity to move the file to the specified device.

2.5 OPTIMIZING FILE PARAMETERS

This section provides some hints on how to organize an indexed sequential file for maximum efficiency.

Once your file has stabilized and you aren't changing it much, re-evaluate the original file parameters. If your evaluation so indicates, rebuild the file with different parameters.

2.5.1 Entries per Index Block

This parameter is a two-edged sword. A small value means faster in-core searches, but more disk accesses and more block splits during record additions. A large value reduces the number of disk accesses and block splits, but increases in-core search time and increases the amount of memory used for buffers. (A block split occurs if you add a key to an index block, but there is no more room in that block; ISAM automatically "splits" that block and redistributes the keys among the two new blocks.)

Since the index structure is fixed at three levels deep, the maximum number of keys that you may add to an index without the top index block splitting is n^3, where n is the number of entries per index block. When the top index block splits, the search time through the index increases due to the possibility of having to do more disk reads.

When you use a floppy disk, the in-core search time is so small compared to a disk seek/transfer that any increase/decrease will not be apparent. When you use a faster disk the trade-off becomes trickier. As a rule, keep the number of entries as large as possible, consistent with the user memory partition size. The amount of index buffer space required is:

```
5 * ((entries-per-block * (keysize + 4))+2)
```

where key size is rounded to an even number of bytes. Given this, you should be able to determine a reasonable value for the number of entries. (NOTE: (keysize + 4) * entries-per-block MUST be less than or equal to 510.) The more entries per block, the more memory you use. It is sometimes more efficient to have the top block split a few times rather than to eat up a large amount of memory.

2.5.2 Empty Index Blocks to Allocate

During creation, enough index blocks are allocated to support a balanced index file tree with sufficient nodes for the number of data records allocated. In practice, the index file tree is rarely balanced (unless you add records in a truly random number with an even distribution of key values). Because of this, you should allocate empty index blocks. Practice has shown that the number of data records divided by the number of entries in an index block gives a good number of empty blocks.

CHAPTER 3

DUMPING AN ISAM FILE WITH ISMDMP

The ISMDMP program provides a convenient method for unloading an indexed sequential file into a sequential file. It also provides a means of examining the index file structure to determine how balanced that structure is.

3.1 GENERAL OPERATING INSTRUCTIONS

ISMDMP has two operating modes: the first allows you to output the contents of an indexed sequential file to an ordinary sequential file; the second allows you to display the index file structure on a terminal to allow analysis thereof. Both are invoked via the general command form:

.ISMDMP filespec{/N}RET

where filespec specifies an indexed sequential file and the optional /N switch suppresses Exclusive Open mode. (See below.) After performing some initialization procedures, ISMDMP asks:

Output to:

Supply another file specification; this one selects the sequential output file. ISMDMP assumes a default file extension of .SEQ. If you want to enter the index file dump mode, enter TTY: as the file specification. For example:

Output to: TTY:RET

3.1.1 Suppressing Exclusive Open Mode

ISMDMP normally uses Exclusive Open mode when performing its file accesses. If you wish it to use Counted Update mode instead, use the /N switch at the end of the ISMDMP command line. For example:

```
.ISMDMP STAT/N(RET)
```

3.2 FILE DUMP MODE

In this mode, ISMDMP outputs the records of the indexed sequential file to an ordinary sequential file in ascending key order. ISMDMP does no translation of the records; it outputs the records in exactly the same form as they were input at some earlier date.

3.2.1 Sample Data File Display

We used ISMBLD to create a small ISAM data file named LABELS. Then we used the sample program in Chapter 7 ("Using ISAM From Within BASIC") to place five records in the file. We then asked ISMDMP to place the data in that file into a file named DATDMP:

```
.ISMDMP LABELS(RET)

Output to: DATDMP(RET)

5 records dumped
.
```

If we use the TYPE command to display the new file (e.g., TYPE DATDMP.SEQ), we see:

```
FILMORE SUSAN          230 STILWOOD LOWELLMA15673200
HINCHEY EDSEL          6712 VIA MALAGA TUSTINCA90245102
LAWRENCE T.E.          1023 W. SANDS PANGUITCHUT98344100
MUKLUK, H.             345 PRAIRIE DOG LN BAKERCA98766120
SAVOY JOHN             891 E. DECATUR LAS VEGASNE89023103
```

Each record contains: 1. Customer name; 2. street address; 3. city; 4. state (two letters); 5. zip code; and, 6. three-digit identifying number (called a hash number).

3.3 INDEX FILE DUMP MODE

The dump mode is intended primarily as a debugging tool, and will not find much use among general users. Therefore we provide little documentation on its use. Those of you who understand the basic structure of the index file should be able to figure out the display quite easily. Remember that you can type a Control-S to freeze the screen display and a Control-R to release the display.

3.3.1 Sample Index File Display

Let's say that we want to display the structure of the primary index file that belongs to our sample data file, LABELS:

```
.ISMDMP LABELS(RET)

Output to: TTY:(RET)
```

Now you see something like this (our comments on the information in this display are in square brackets):

```
Size of data record:    67
Size of dir entry:      30
Size of dir block:      302
Size of key:            25
Type of key:            0
Entries per dir block:  10              [keys per index block]
Record key position:    1
Blocking factor:        7
IDA freelist pointer:   000000000517    [first free record in data file]
IDA freecount:          45              [number of free data file records]
IDX freelist pointer:   000004          [first free index file block]
IDX freecount:          22              [number of free index file blocks]
Records allocated:      5               [number of data records]
Top dir blk pointer:    000001          [points to top index block]

[index file block number:]
000001:    000000000002 [points to next index level]
                                        000000000000
                                        000000000000
                                        000000000000
                                        000000000000
                                        000000000000
                                        000000000000
                                        000000000000
                                        000000000000
        000000                  177777

000002:  ----------------------------   000000000003
                                        000000000000
                                        000000000000
                                        000000000000
                                        000000000000
                                        000000000000
                                        000000000000
                                        000000000000
                                        000000000000
        000000                  1777777
```

```
000003: FILMORE SUSAN                    000000000414 [points to data record]
        HINCHEY EDSEL                    000000177777 [first record entered]
        LAWRENCE T.E.                    000000000206
        MUKLUK H.                        000000000311
        SAVOY JOHN                       000000000103
        ------------------------         177777177776 [indicates last record]
                                         000000000000
                                         000000000000
                                         000000000000
                                         000000000000
        000000                   044506  [in used blocks, this number is
                                         junk-- ignore it.]
000004:                                  000000000000
                                         000000000000
                                         000000000000
                                         000000000000
                                         000000000000
                                         000000000000
                                         000000000000
                                         000000000000
                                         000000000000
        000000                   000005  [in unused blocks, points to next
                                         free index block]
000005:                                  000000000000
                                         000000000000
                                         000000000000
                                         000000000000
                                         000000000000
                                         000000000000
                                         000000000000
                                         000000000000
                                         000000000000
        000000                   000006

                  .
                  .
                  .
                  .
              [Etc.]
```

CHAPTER 4

COMPRESSING INDEX FILES WITH ISMCOM

ISMCOM.PRG compresses the upper level of ISAM index files; this increases access speed and may recover some storage room in the index file. To use ISMCOM, enter:

```
.ISMCOM filespec<RET>
```

where filespec selects the index file you want to compress. The program now reports its intended compression factor (initially based on 95%). If you wish denser or looser compression, enter the percentage of compression you want ISMCOM to use. If that value is valid for the file (based on the number of entries per index block), the program proceeds; otherwise, it reports the actual effective value and allows you to enter a new value. The only way to get 100% compression is to enter 100. The program will not accept input of a percentage of less than 50. (In actual practice, 50% can be rounded down to, say, 47% in some cases.) Below is a sample ISMCOM dialog:

```
.ISMCOM DATA.IDX<RET>

NOBODY else may use this file while I'm processing it

I am planning to compress each block to at least 90 percent full
If that is not acceptable, enter the percentage you desire 76<RET>
It will actually work out to be 80 percent full
If that is not acceptable, enter the percentage you desire


No blocks unchanged, No blocks freed, No blocks compressed
```

Note that a compression factor of 100% will cause a block split the next time a top level index is created. The number 95% was chosen as the optimum compression factor for most files. At the end of the compression, ISMCOM prints some statistics that tell you how much compression was done and how much good it should do.

PART II

PROGRAMMING WITH ISAM

This section contains information on writing assembly language programs and BASIC programs that use the ISAM functions to access and update ISAM files. For information on writing assembly language programs on the AMOS system, refer to the AMOS Assembly Language Programmer's Reference Manual, (DWM-00100-43), and the AMOS Monitor Calls Manual, (DWM-00100-42). For information on BASIC, refer to the AlphaBASIC User's Manual, (DWM-00100-01).

CHAPTER 5

USING ISAM FROM THE ASSEMBLY LANGUAGE LEVEL

NOTE: This section assumes that you are an experienced assembly language programmer and that you are familiar with the Alpha Micro CPU instruction set and the AMOS monitor calls. For information on these topics, refer to the AMOS Assembly Language Programmer's Reference Manual, the WD16 Microcomputer Programmer's Reference Manual, (DWM-00100-04), and the AMOS Monitor Calls Manual.

The ISAM program is implemented as a FETCHable memory module which allows the assembly language programmer easy access to the features of indexed sequential files. (NOTE: FETCH is an AMOS monitor call. Refer to the AMOS Monitor Calls Manual for information on the routines within the operating system (called "monitor calls") that have been made available to your assembly language programs.) It is through the ISAM module that high level languages such as BASIC gain access to indexed sequential files. The ISAM program is fully re-entrant, and could therefore be made resident in system memory if more than one user at a time is going to be using indexed sequential files.

The ISAM program itself takes up approximately 4K bytes of memory. In addition to this space, another 1 to 4K bytes is required for each indexed sequential file that you are processing. This memory space is usually allocated by the ISAM system using the GETMEM monitor call; you may, however, allocate your own buffer areas (see Section 5.2.2).

5.1 GENERAL CALLING SEQUENCE

The various ISAM subroutines are called via a dispatch table at the start of the ISAM program. To make things easier, the file ISUSYM.MAC defines the table offsets. This file also contains symbols for the various return codes. All table offsets begin with a period (e.g., .ICLOS, the close routine). All return codes have the general form IS.xxx (e.g., IS.EOF, the end-of-file return code). ISUSYM.MAC is designed to be COPYed by your assembly language program.

To call the close routine (.ICLOS) with the base of the ISAM.PRG module contained in register R4, use the following code:

```
CALL    .ICLOS(R4)
```

All arguments are passed in registers. Each call returns with a completion code in R0. A successful return (IS.SUC) is indicated by a zero in R0; the indicators (also known as condition codes or condition flags) on return reflect success or error status. The Z-bit is set if successful (BNE branches on error).

5.2 INITIALIZING THE ISAM SYSTEM (.INIT)

Before your program can access an indexed sequential file, you must tell the ISAM system that you exist; this is done via the .INIT call. The .INIT call allocates space for the user's impure variables and does minor housekeeping chores. NOTE: Your program calls .INIT only once regardless of the number of ISAM files that are to be opened.

5.2.1 Calling Sequence

Parameters:	R2	User allocation routine address (optional)
	R3	User deallocation routine address (optional)
	R4, R5	Used to pass information to user memory allocation routines (optional).
	CALL	.INIT(Rn)
Returns:	R0	Completion code
	R5	User memory pointer
	Indicators	Z if no error

The user memory pointer that is returned in R5 is a pointer to your impure area. This pointer is needed by all other ISAM calls; if convenient, leave it in R5 since all calls look for it there.

5.2.2 User Supplied Allocation and Deallocation Routines

In many cases, the program calling the ISAM program will do its own memory management, and not want ISAM to use GETMEMs to do so. To allow you to do your own allocation, the .INIT call allows the passing of allocation and deallocation routine addresses. .INIT uses its own routines (which use GETMEMs) if you pass a zero instead of an address.

The user allocation routine is called with the desired module size in R1. The .INIT call expects the address of the assigned module to be returned in R1. You may not modify any other registers.

The user deallocation routine is called with the address of the module to be deleted in R1. Do not modify any other registers. If you pass a zero to .INIT in R3, no deallocation occurs.

The current version of ISAM allows you to move any of the modules that ISAM requests as well as the ISAM program itself. ISAM is immune to such movement as long as the user memory pointer (in R5) and the FPN (file pair number, see Section 5.6.1) associated with a given file are updated to show any movement. (The FPN is usually in R1.)

5.3 FINALIZING ISAM PROCESSING (.IFIN)

When you are through processing indexed sequential files, you must call the .IFIN routine. This call deallocates any space used by ISAM if a user deallocation routine has been provided; otherwise the modules are not deleted until the job EXITs. (EXIT is an AMOS monitor call.)

5.3.1 Calling Sequence

Parameters:		R5	User memory pointer
		CALL	.IFIN(Rn)
Returns:		R0	IS.SUC
	Indicators		Z if no error

The .IFIN routine cannot fail, therefore it always returns the successful completion code in R0.

5.4 OPENING A FILE FOR PROCESSING (.IOPNR)

You must open an indexed sequential file via this call before you can process the file in any way. Also use this call when opening a secondary index file for processing a previously opened data file. If you execute this call on a primary index file, the call also opens the associated data file; if you execute the call on a secondary index file, the call opens the index file only. Thus to process a data file with a secondary index file, you must execute two .IOPNR calls: once to open the data file and primary index file, and once to open the secondary index file.

5.4.1 Calling Sequence

Parameters:		R0	Flags: Bit <3> (decimal) ISAM will open file in Exclusive Open mode; otherwise, Counted Update mode is used. Bit <10> (decimal) Operating system will print system and device error messages before returning.
		R2	Pointer to ASCII filespec string describing the index file to be opened. If the index file is a primary index, the data file must have the same name.
		R5	User memory pointer
		CALL	.IOPNR(Rn)
Returns:		R0	Completion code
		R1	Unique File Pair Number (FPN)
	Indicators		Z if no error

The file pair number (FPN) is a pointer to the memory module that has been allocated for the storage needed by this particular indexed sequential file. The read, write, and delete routines use the FPN to tell the ISAM program which indexed sequential files to process of the ones you may have open.

If you must move the module allocated by .IOPNR, you may do so as long as you also update the FPN.

5.5 CLOSING THE FILE AFTER PROCESSING (.ICLOS)

After you have finished processing a file, you must close it. The .ICLOS call does some housekeeping and also deallocates any space used by the file if a user deallocation routine has been provided.

5.5.1 Calling Sequence

Parameters:		R1	File pair number (FPN)
		R5	User memory pointer
		CALL	.ICLOS(Rn)
Returns:		R0	Completion code
	Indicators		Z if no error

The file pair number used in R1 is that value returned by .IOPNR.

5.6 LOCATING A FREE DATA RECORD (.IGTFR)

Use this call to get the relative record number of the next available data record in the data file.

5.6.1 Calling Sequence

Parameters:	R1	File pair number (FPN) of primary index file
	R5	User memory pointer
	CALL	.IGTFR(Rn)
Returns:	R0	Completion code
	R1	Low-order relative record number of the data record
	R2	High-order relative record number of the data record
	Indicators	Z if no error

The FPN supplied in R1 must refer to the primary index file associated with the data file from which a free record is to be obtained.

5.7 DELETING A DATA RECORD (.IDLFR)

Use this call to return a data record to the free record list.

5.7.1 Calling Sequence

Parameters:	R1	File pair number (FPN) of primary index file
	R2	Low-order relative record number of the data record
	R3	High-order relative record number of the data record
	R5	User memory pointer
	CALL	.IDLFR(Rn)
Returns:	R0	Completion code
	Indicators	Z if no error

The FPN supplied in R1 must refer to a primary index file.

5.8 READING A DATA RECORD BY RELATIVE RECORD NUMBER (.IRLRD)

Use this call to read the data record pointed to by the relative record number.

5.8.1 Calling Sequence

Parameters:	R1	Low-order relative record number of the data record
	R2	High-order relative record number of the data record
	R3	File pair number (FPN)
	R4	Buffer address
	R5	User memory pointer
	CALL	.IRLRD(Rn)
Returns:	R0	Completion code
	Indicators	Z if no error

The FPN supplied in R3 must refer to the primary index file associated with the data file.

5.9 WRITING A DATA RECORD BY RELATIVE RECORD NUMBER (.IRLWT)

Use this call to write or update the data record pointed to by the relative record number.

5.9.1 Calling Sequence

Parameters:	R1	Low-order relative record number of the data record
	R2	High-order relative record number of the data record
	R3	File pair number (FPN)
	R4	Buffer address
	R5	User memory pointer
	CALL	.IRLWT(Rn)
Returns:	R0	Completion code
	Indicators	Z if no error

5.10 FINDING A RECORD (.IREDR)

Use this call to use a symbolic key to find the relative record number of a data record.

5.10.1 Calling Sequence

Parameters:	R1	File pair number (FPN) of the desired index file
	R3	Pointer to symbolic key
	R5	User memory pointer
	CALL	.IREDR(Rn)
Returns:	R0	Completion code
	R1	Low-order relative record number of data record
	R2	High-order relative record number of data record
	Indicators	Z if no error

The FPN supplied in R1 may refer to any open index file.

5.11 ADDING A SYMBOLIC KEY (.IWRTR)

Use this call to add a key entry to an index file given a user supplied data record number.

5.11.1 Calling Sequence

Parameters:	R1	File pair number (FPN) of desired index file
	R2	Pointer to symbolic key
	R3	Low-order relative record number of the data record
	R4	High-order relative record number of the data record
	R5	User memory pointer
	CALL	.IWRTR(Rn)
Returns:	R0	Completion code
	Indicators	Z if no error

The FPN supplied in R1 may refer to any open index file. The relative

record number in R3 and R4 will usually be a record number returned by the .IGTFR call.

5.12 DELETING A KEY (.IDELK)

Use this call to delete a key from an index file.

5.12.1 Calling Sequence

Parameters:	R1	File pair number (FPN) of desired index file
	R3	Pointer to symbolic key
	R5	User memory pointer
	CALL	.IDELK(Rn)
Returns:	R0	Completion code
	R1	Low-order relative record number of deleted key
	R2	High-order relative record number of deleted key
	Indicators	Z if no error

The relative record number in R1 and R2 refers to the data record associated with the deleted key within the index file referred to by the supplied FPN. The data record is not deleted in the data file; do this by using the .IDLFR call when you are sure that there are no keys left in the index file that refer to that data record.

5.13 FINDING THE NEXT SEQUENTIAL KEY (.SREDR)

When printing reports or posting data, it is often useful to be able to go through the records in the indexed sequential file in ascending key order. The .SREDR call makes this possible; it returns the relative record number of the record that immediately follows the one returned by the last call to ISAM. It thus makes it possible to start sequential processing by key anywhere in the file. To do so, use the .IREDR call to get the first key you wish to use. Then call .SREDR to get the key following the one read by .IREDR. You can get the next key by doing another .SREDR, ad infinitum. If .IREDR does not find the key specified, the following .SREDR returns the record with the key closest to (but greater than) the one not found. Thus, to read the file from the very beginning, try to do a .IREDR with a key of zero. This call will almost always fail, but the following .SREDR grabs the very first record in the file.

If the file contains a key of all zero, the initial .IREDR will succeed and that record should be processed as the first record before doing any .SREDRs. When the file is initially opened by .IOPNR, it is set up so that .SREDR gets the first key, unless the first key is all zero. Therefore it is almost always possible to open the file and read it sequentially. Since keys are expected to be ASCII, and an all-null key is not very sensible, the various utilities assume that the first key is not zero. When the end of the file is reached by the last .SREDR, the end-of-file (IS.EOF) completion code is returned.

5.13.1 Calling Sequence

Parameters:	R1	File pair number (FPN) of desired index
	R3	Contains 0 if symbolic key is not to be returned; otherwise, R3 contains a pointer to the location in memory that the returned symbolic key is to be placed in.
	R5	User memory pointer
	CALL	.SREDR(Rn)
Returns:	R0	Completion code
	R1	Low-order relative record number of data record
	R2	High-order relative record number of the data record
	Indicators	Z if no error

NOTE: If you want ISAM to behave as in previous releases (that is, if you do not want the symbolic key returned), be sure that R3 is cleared to zero before your program calls .SREDR. Remember that if R3 is nonzero, ISAM will return the symbolic key to the memory location specified by the contents of R3-- if R3 happens to contain invalid data, ISAM may damage important data in memory.

CHAPTER 6

STANDARD ISAM SYMBOLS FOR ASSEMBLY LANGUAGE PROGRAMMERS

We have provided a symbols file (ISUSYM.MAC) to make life easier for the assembly language programmer. This appendix describes the contents of that file. We have broken this information into two sections: 1. those offsets used when invoking ISAM; 2. the completion codes returned by the various ISAM functions.

6.1 CALLING SYMBOLS

The following symbols define entry offsets in the ISAM package. See Section 5.1 of this manual for more information.

.INIT	Call the initialization routine
.IFIN	Call the finalization routine
.IOPNR	Call the file open routine
.ICLOS	Call the file close routine
.IGTFR	Call the get free record routine.
.IDLFR	Call the data record deletion routine.
.IRLRD	Call the read data record via relative record number routine.
.IRLWT	Call the write data record via relative record number routine.
.IREDR	Call the read returning relative record number routine.
.IWRTR	Call the write using relative record number routine.
.IDELK	Call the key deletion routine.
.SREDR	Call the read sequential relative record number routine.

6.2 COMPLETION CODE SYMBOLS

The following symbols name the completion codes returned in R0 upon completion of an ISAM call. Always use these symbols rather than their values in ISUSYM.MAC, since those values could change in the future.

IS.SUC	The successful completion code (will always be zero).
IS.DNR	Parameters supplied do not match those in the Directory Rock.
IS.RNF	Record not found.
IS.DPK	Attempt to add duplicate key.
IS.LSS	Index file link structure is smashed.
IS.XFL	Index file is full.
IS.AFL	Data file is full.
IS.EOF	End of file encountered on sequential read.

An error code of 2 (handled by BASIC as a SYSTEM ERROR; which is why this code was chosen) means either that something in ISAM or the ISAM file structure is in error (as in a bug) or that you used an obsolete ISAM call.

CHAPTER 7

USING ISAM FROM WITHIN BASIC

The following pages are a brief summary of the BASIC ISAM commands that your BASIC programs can use to access ISAM indexed sequential files. (Remember that an indexed sequential file is made up of both an ISAM data file and at least one ISAM index file.) For more information on the BASIC ISAM functions and on BASIC itself, refer to the AlphaBASIC User's Manual, (DWM-00100-01). The following discussions assume that you are already familiar with opening and closing random files, and that you understand the BASIC READ and WRITE statements. For more information on using files, refer to the AlphaBASIC User's Manual.

You must use the ISAM utility program ISMBLD to create an indexed sequential file before your BASIC program can access it. Although no features exist within BASIC to create an indexed sequential file, your BASIC program can create and execute a command file that invokes ISMBLD with a list of file parameters. All data files must have an extension of .IDA and all index files must have the .IDX extension.

Before you run your BASIC program, make sure that ISAM.PRG has been loaded into memory if the System Operator has not arranged to have ISAM.PRG resident in system memory.

7.1 OPENING AN INDEXED SEQUENTIAL FILE

To open an ISAM indexed sequential index file, use the BASIC OPEN command. Your program must include an OPEN statement that assigns a file channel to the indexed sequential file before the program makes any other references to that file.

This statement takes the same form as the OPEN statement for ordinary random files except that you must specify either INDEXED or INDEXED'EXCLUSIVE mode rather than RANDOM mode. (Remember that your ISAM files are random data files. For information on using random files, see Chapter 15 of the AlphaBASIC User's Manual.) The OPEN statement takes this form:

```
OPEN #file-channel,filespec,mode,record-size,relative-record-number
```

1. #file-channel - specifies the file channel you want to assign to the indexed sequential file. Any numeric expression that evaluates to an integer from 0-65535 (0 is the user terminal). This is the number you reference when using the ISAM, READ, and WRITE statements.

2. Filespec - The filespec is any string expression that evaluates to a legal AMOS file specification. (It may optionally include account and device specifications.) It specifies the name of the indexed sequential file you created using ISMBLD (that is, the data file/primary index file combination you built) or specifies the name of a secondary index file created with ISMBLD. (If the OPEN statement refers to a secondary index file, you must have previously opened the corresponding data file/primary index file on another file channel.)

   Note that the primary index file always has the same name as the data file, but has a .IDX extension; the data file has a .IDA extension.

3. Mode - If you wish ISAM to access the indexed sequential file in Counted Update mode, use the INDEXED keyword as the file mode; if you want ISAM to access the file in Exclusive Open mode, use the keyword INDEXED'EXCLUSIVE. (For information on Counted Update mode and on Exclusive Open mode, see Section 1.3, "ISAM Access Modes.")

4. Record-size - An expression that specifies the logical record size for read/write operations.

5. Relative record number - A floating point variable that will hold the relative record number returned by an ISAM function. (See Section 7.2, "The ISAM Statement.")

For an example of the use of the OPEN statement, refer to the sample BASIC program at the end of this chapter. Below are several sample OPEN statements:

```
220     OPEN #1, "LABELS", INDEXED, RECSIZE, RELKEY1
230     OPEN #2, "HASH", INDEXED, RECSIZE, RELKEY1
```

The two program lines above assume that there exists a data file named LABELS.IDA and a primary index file named LABELS.IDX. Line 220 opens that indexed sequential file. Line 230 opens a secondary index file associated with LABELS.IDA. Note that RECSIZE and RELKEY1 are identical for both OPEN statements; this is because both OPEN statements refer to the SAME data file, LABELS.IDA. The RECSIZE and RELKEY1 are used by subsequent READ and WRITE commands to access the data file.

7.2 THE ISAM STATEMENT

The purpose of the ISAM statement is to allow you to use the ISAM program from within your BASIC program to: 1. find a record in the data file by symbolic key (returning the relative record number in the variable specified by the indexed sequential file OPEN statement); 2. find the next data record (by the order in which the symbolic keys occur in the index file); 3. add a symbolic key to an index file; 4. delete a symbolic key from an index file; 5. locate next free data record in data file (returning relative record number in the variable specified by the appropriate OPEN statement); 6. delete a record from a data file, and return that record to the free list; and, 7. find the next data record (by the order in which the symbolic keys occur in the index file) and return the symbolic key.

The ISAM statement follows this form:

```
ISAM #file-channel, code, symbolic-key
```

1. #file-channel - Specifies the file channel assigned by an OPEN statement to either the data file/primary index file or the secondary index file (depending on which set of symbolic keys you want to access).

2. Code - A numeric value that selects one of the functions mentioned above. May be any legal numeric expression which is resolved at runtime.

3. Symbolic-key - Specifies the symbolic key to be used in locating a data record. You must always specify a symbolic key even if a function does not require the use of one. (This simplifies syntax checking.) If you wish, you may use a dummy string variable in such cases.

7.2.1 ISAM Statement Codes

The ISAM statement can perform six different functions. You may select one of these functions by supplying the appropriate code number (see below) to the ISAM statement. An error will result if you do not supply a valid code number.

Some of the functions below require a relative record number as input; others return a relative record number to be used when your READ and WRITE statements access the data file. In either case, the ISAM functions pass the relative record number in the variable specified in the OPEN statement for the data file/primary index file. READ and WRITE statements also use that variable for locating the data file record that they are going to access. Remember that the ISAM statement does <u>not</u> directly access the data file. Instead, it gives you the information you need to access the data file yourself using the relative record number returned by ISAM.

CODE 1 - Searches the index file selected by #file-channel for the key that matches the symbolic-key. If it finds a match, it returns the relative record number of the data file record containing that key. If it does not find a match, it returns an error code 33 in ERF(X). (See Section 7.5, "Error Processing").

CODE 2 - Accesses the index file selected by #file-channel and finds the next symbolic key. Returns the relative record number of the data file record associated with that symbolic key in preparation for a READ or a WRITE to the data file. If this is the first access to the file after the OPEN statement, it finds the first symbolic key in the index file. If this function follows a previous code 1 statement, the function finds the next symbolic key after the code 1 symbolic key. If there are no more keys in the index file, the function returns an end-of-index-file error (38): make no further accesses to the data file until you make another ISAM call that returns a valid relative record number.

CODE 3 - Adds the specified symbolic key to the index file selected by #file-channel. Also adds the relative record number specified by the variable in the OPEN statement. You will usually set this relative record number just prior to the code 3 call by using a code 5 ISAM statement. (A code 5 call returns the relative record number of the next free data record.)

CODE 4 - Delete the specified symbolic key from the index file selected by #file-channel. This function returns the corresponding relative record number so that you can use a code 6 ISAM statement to delete the data record and return it to the free list. If the function cannot find the symbolic key in the index file, it returns a "?Record not found" error (33).

CODE 5 - Finds the next available data record on the free list. (The free list is a linked list that keeps track of all available records in the data file. ISMBLD initially builds the free list.) Returns the relative record number of that record so that you can use a code 3 ISAM statement to add a symbolic key/relative record number pair to the index file. If no more data records are free in the data file, the function returns a "?Data file full" error. A code 5 ISAM statement does not modify the index file; it simply locates the next free record in the data file. The function ignores the symbolic key in the ISAM statement. The #file-channel in the code 5 ISAM statement must be the file channel assigned to the primary index file.

CODE 6 - Returns to the free list the data record specified by the relative record number in the OPEN statement. Does not modify the index file. The #file-channel in the code 6 ISAM statement must be the file channel assigned to the primary index file. A code 6 call ignores the symbolic key in the ISAM statement.

CODE 7 - Accesses the index file selected by #file-channel and finds the next symbolic key. Returns the relative record number of the data file record associated with that symbolic key in preparation for a READ or a WRITE to the data file. If this is the first access to the file after the OPEN statement, it finds the first symbolic key in the index file. If this function follows a previous code 1 statement, the function finds the next symbolic key after the code 1 symbolic key. If there are no more keys in the index file, the function returns an end-of-index-file error (38): make no further accesses to the data file until you make another ISAM call that returns a valid relative record number.

NOTE: This code performs exactly the same function as code 2 above, except that it returns the symbolic key as well as the relative record number.

It is very important that the symbolic key variable that appears in your code 7 ISAM statement be the same size as or larger than the key defined in the ISAM index file. If the variable is smaller than the key, data following the symbolic key in memory will be overwritten, damaging your running program.

7.3 READING AND WRITING DATA IN AN ISAM DATA FILE

ISAM statements do not access data records, but instead return their relative record numbers. To actually read or write data records, you must use the BASIC READ and WRITE commands. When you read or write data in a specific ISAM data file, BASIC selects the record to be accessed by referring to the relative record number variable in the OPEN statement for that file.

```
READ #file-channel, variable1, variable2,... variableN

WRITE #file-channel, variable1, variable2,... variableN
```

The #file-channel in the READ or WRITE statement MUST be the file channel that appears in the OPEN statement for the primary index file you want to access. The relative record number variable in the OPEN statement must contain a valid relative record number or an error will result.

7.4 CLOSING FILES

To ensure that ISAM has rewritten all data records to the data file and that it has properly updated all links in the index file, it is VERY important that you close all index files (primary and secondary) via the normal CLOSE statement. Failing to close the file when you are through with it may destroy the linking structure of the indexed sequential file. The CLOSE statement takes the form:

```
CLOSE #file-channel
```

where #file-channel is the file channel assigned to the file you want to close For example:

```
CLOSE #2
```

where file channel #2 was assigned to an indexed sequential file by a previous OPEN statement. Remember to close both primary and secondary index files. NOTE: The order in which you close the ISAM files makes no difference; however, remember that you cannot access a secondary index file if you have already closed the primary index file/data file.

7.5 ERROR PROCESSING

Any ISAM operation can result in some kind of error. If the error is a system error (for example, the disk is not mounted), BASIC interrupts your program and aborts to the monitor. (Or, if error trapping is enabled, BASIC transfers control to your error handling routine.) For information on dealing with the usual system errors (e.g., "?File not found" or "?Disk not mounted") refer to the AlphaBASIC User's Manual, in particular the section titled "Error Trapping."

Special ISAM errors can also occur as a result of an ISAM operation. These errors do not generate an error message or result in an error trap. It is therefore very important that your program check for these errors after every ISAM statement; otherwise, you have no way of knowing whether or not the ISAM function was performed successfully. To do so, use the ERF(X) function, where X is the file channel number used by the preceding ISAM statement. (The ERF(X) function operates in much the same way as the EOF(X) function.)

If ERF(X) returns a zero, the preceding ISAM statement was successful. If ERF(X) returns a nonzero value, then an error was detected. If an error occurred, your program should correct the problem before going on to access the file. The nonzero value returned tells you which error occurred. For example:

```
! If a "Record not found" error (#33), go to routine that asks for new key.
100 IF ERF(2)=33 THEN PRINT "RECORD NOT FOUND" : GOTO PROMPT
```

The current ISAM error codes are:

32 - Illegal ISAM statement code.
33 - Record not found in index file search.
34 - Duplicate key found in index file during attempted key addition.
35 - Link structure is smashed and mustt be re-created.
36 - Index file is full.
37 - Data file is full (free list is empty).
38 - End of file during sequential key read.

REMEMBER: Always check after performing an ISAM function to see if an error occurred. If you do detect an error, your program must take corrective action before continuing on.

7.6 USING THE ISAM FUNCTIONS WITHIN A BASIC PROGRAM

Below are some examples of the ways you can combine the ISAM statements and other BASIC commands to access and use indexed sequential files. For a look at a sample ISAM prgram, turn to Section 7.7.

7.6.1 Adding Data to an Indexed Sequential File

At the time that you use ISMBLD to create an indexed sequential file, you have the option of loading data into the ISAM data and primary index file from an ordinary sequential data file. Your BASIC programs may also add data to the indexed sequential file by using code 5 and code 3 ISAM statements. For each new data record to be added:

1. Open the indexed sequential file with an OPEN statement. For example:

```
OPEN #1,"PHONES",INDEXED,RECSIZE,RELKEY
```

Remember to open any secondary index files that you might want to use via separate OPEN statements on different file channels.

2. Use a code 1 statement to see if the index entry you want to add already exists. For example:

```
ISAM #1, 1, NAME
```

Check to see if an error was returned. For example:

```
IF ERF(1) = 0 THEN PRINT "Duplicate Name." : GOTO GET'NAME
```

(If no error occurred, then the index entry already exists and you can't add it.) If you are using secondary index files, also check to see that the secondary index entries don't already exist.

3. Retrieve the next free data record (a code 5 ISAM statement). For example:

```
ISAM #1, 5, DUMMY
```

Check to make sure that an error (e.g., 37 - "?Data file is full (free list is empty)" did not occur. For example:

```
IF ERF(1) <> 0 THEN GOTO ISAM'ERROR
```

4. If no error occurred, the record number of the next free record is in the relative record number variable defined by the OPEN statement for the indexed sequential file. Now you can write the data into the record by using a WRITE statement. For example:

```
WRITE #1,INFO
```

5. Now you must add the symbolic keys for that data record to the index files, using a code 3 statement. (Those symbolic keys will then link to that data record.) Be sure to check for an ISAM error after each addition.

6. After adding all data records, close the ISAM files. For example:

```
CLOSE #1        ! Close primary index file/data file
CLOSE #2        ! Close secondary index file
```

7.6.2 Reading Data Records in Symbolic Key Order

ISAM stores symbolic keys in the index file in ASCII collating sequence. To retrieve records in the order in which their keys appear in an index file:

1. Open the indexed sequential file with an OPEN statement. (If you also wish to open one or more secondary index files that cross-index to the primary index file, use one OPEN statement for each secondary index file.)

2. Execute a code 2 ISAM statement to find the next symbolic key.

3. Check to make sure that the ISAM statement didn't return an error. For example:

```
IF ERF(1) = 38 THEN PRINT "End of file." : GOTO PROMPT
IF ERF(1) <> 0 THEN GOTO ISAM'ERROR
```

4. The proper record number is now in the relative record number defined by the OPEN statement for the file, so you can use a READ statement to read in the data. For example:

```
READ #1, INFO
```

   (Remember that the READ statement must include the file channel assigned to the primary index file even if the code 2 ISAM statement included a symbolic key contained in a secondary index file; this is because the data you want to read is in the data file.)

5. Check for an end-of-file error by using the ERF(X) function.

6. Repeat these procedures to step through the data records in the order of the symbolic keys in the index files until you reach the end of the file, or until you have accessed all the records you need. Be sure to check for an ISAM error after each access.

7. Close all files when you are done.

7.6.3 Reading Data Records Randomly by Symbolic Key

1. Open the indexed sequential file with an OPEN statement. You must include one OPEN statement for the data file/primary index file. You must also include one OPEN statement for each secondary index file you want to access.

2. Locate each data record by using a code 1 ISAM statement. The statement must contain the symbolic key associated with the record for which you are searching and the file channel associated with the index file containing the symbolic key.

3. Check for a "record not found" error; this indicates that the symbolic key was not located in the specified index file.

4. If the record was found, use a READ statement to read in the data record. (The READ statement includes the file channel associated with the data file/primary index file, even if the symbolic key used belonged to a secondary index file.)

5. Repeat steps 2 through 4 for each record you want to find.

6. Close all files.

7.6.4 Updating Data Records

1. Open the indexed sequential file with an OPEN statement.

2. Locate the data record you want to update via one of the methods above (i.e., by using a code 1 or code 2 ISAM statement).

3. Check to make sure that the record was found. (Use the ERF function.)

4. Use a WRITE statement to update the data record. (The WRITE statement includes the file channel associated with the data file/primary index file, even if the symbolic key used to find the record belonged to a secondary index file.)

5. This operation does not change the index files, so do not change the symbolic key in the record you rewrite. If you need to alter data that is part of a symbolic key, you must delete the key in the correct index file (a code 4), and then add the new key to the index file (code 3). You do not need to delete and re-create the data record during this operation unless you are entering completely new data.

6. Close all files.

7.6.5 Deleting a Data Record

Deleting a data record from an indexed sequential file entails not only deleting the record itself but also deleting all symbolic keys associated with that data record from all index files.

1. Open the primary index file and all secondary index files needed.

2. Locate the data record via one of the symbolic keys (a code 1 ISAM statement).

3. Check to make sure that the statement executed without error. For example:

```
IF ERF(2) = 33 THEN PRINT "Record not found." : GOTO PROMPT
IF ERF(2) <> 0 THEN GOTO ISAM'ERROR
```

4. Read the data record with a READ statement (whose #file-channel is the file channel number associated with the primary index file).

5. Extract each symbolic key from that data record. Use each symbolic key to delete each key from its associated index file with code 4 ISAM statements.

6. After all symbolic keys have been deleted from all index files, delete the record itself via a code 6 ISAM statement.

7. Close all files.

NOTE: A good way to check the structure of the indexed sequential file might be to store the relative record number in another variable; then compare the relative record numbers returned by each code 4 ISAM statement to check that the symbolic keys did indeed all link to the correct data record. You should also check each ISAM statement for any possible error that might otherwise go unnoticed.

7.7 SAMPLE ISAM PROGRAM

The sample program below will make clearer the use of the commands discussed above. For more information on using ISAM from within a BASIC program, consult the manual AlphaBASIC User's Manual.

Before we can begin to use ISAM, we must load it into memory if it is not already resident in system memory:

```
.LOAD SYS:ISAM.PRG<RET>
```

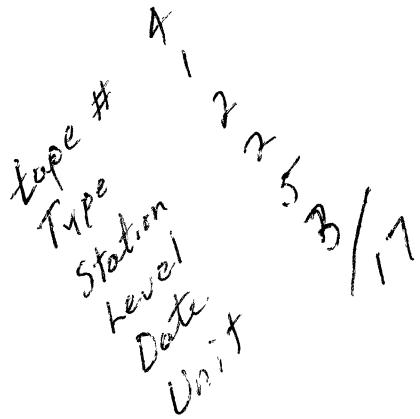
Before we run the sample program below, we first use the program ISMBLD to build the ISAM files LABELS.IDA (the data file), LABELS.IDX (the primary index file), and HASH.IDX (the secondary index file). Note that we build an empty file (i.e., we type a RETURN after the "Load from file:" prompt). We use the BASIC program below to place data into the file.

```
.ISMBLD LABELS<RET>
Size of key: 25<RET>
Position of key: 1<RET>
Size of data record: 67<RET>
Number of records to allocate: 50<RET>
Entries per index block: 10<RET>
Empty index blocks to allocate: 20<RET>
Primary Directory: Y<RET>
Data file device:<RET>

Load from file:<RET>

.ISMBLD HASH<RET>
Size of key: 10<RET>
Position of key: 58<RET>
Size of data record: 67<RET>
Number of records to allocate: 50<RET>
Entries per index block: 10<RET>
Empty index blocks to allocate: 20<RET>
Primary Directory? N<RET>

Secondary index to file: LABELS<RET>
End of primary file
No records loaded
```



Now we can run our sample program:

```
.RUN MAIL<RET>
```

SAMPLE BASIC ISAM PROGRAM

```
10 ! ISAM Sample Program.
20 !
30 ! This program is a simple example of how to handle ISAM files, both
40 ! primary and secondary.  It simulates a very simple-minded mailing
50 ! list program, with the addresses keyed by both name and user
60 ! defined hash code.
70 !
80 ! Define the Mailing List file record.
90 !
100 MAP1 LABEL
110     MAP2 NAME,S,25
120     MAP2 ADDRESS,S,25
130     MAP2 STATE,S,2
140     MAP2 ZIP,S,5
150     MAP2 HASH,S,10
160
170 ! Define record sizes.
180
190 MAP1 RECSIZE,F,6,67                 ! Size of data record.
200
210 ! Open the primary and secondary files.
220     OPEN #1, "LABELS", INDEXED, RECSIZE, RELKEY1
230     OPEN #2, "HASH", INDEXED, RECSIZE, RELKEY1
240
250 PROMPT:
260     PRINT
270     INPUT "ENTER FUNCTION &
          (1=ADD,2=DELETE,3=INQUIRE,4=PRINT,99=END): "; FUNCTION
280     ON FUNCTION GOTO ADD'RECORD,DELETE'RECORD,INQUIRE'RECORD,PRINT'LABELS
290     IF FUNCTION=99 THEN GOTO END'IT
300     GOTO PROMPT
310
320 ADD'RECORD:
330     INPUT "ENTER NAME: "; NAME
340     INPUT "ENTER ADDRESS: "; ADDRESS
350     INPUT "ENTER STATE: "; STATE
360     INPUT "ENTER ZIP: "; ZIP
370     INPUT "ENTER HASH: "; HASH
380 ! Add Trailing blanks to the keys.
390     NAME = NAME + SPACE(25-LEN(NAME))
400     HASH = HASH + SPACE(10-LEN(HASH))
410 ! Look up name to verify that it is not a duplicate. (If ERF(1)=0, then
415 ! ISAM found the key in the data file.)
420     ISAM #1, 1, NAME
430     IF ERF(1) = 0 THEN PRINT "DUPLICATE NAME" : GOTO ADD'RECORD
440 ! Verify that hash is not a duplicate.
450     ISAM #2, 1, HASH
460     IF ERF(2) = 0 THEN PRINT "DUPLICATE HASH" : GOTO ADD'RECORD
```

```
470 ! Get free data record from primary file and write record out.
480     ISAM #1, 5, NAME
490     IF ERF(1) <> 0 THEN GOTO ISAM'ERROR
500     WRITE #1, LABEL
510 ! Add key to primary index file.
520     ISAM #1, 3, NAME
530     IF ERF(1) <> 0 THEN GOTO ISAM'ERROR
540 ! Add key to secondary index file.
550     ISAM #2, 3, HASH
560     IF ERF(2) <> 0 THEN GOTO ISAM'ERROR
570     GOTO PROMPT
580
590 DELETE'RECORD:
600     INPUT "ENTER NAME: "; NAME
610     NAME = NAME + SPACE(25-LEN(NAME))
620 ! Verify that the key exists.
630     ISAM #1, 1, NAME
640     IF ERF(1) = 33 THEN PRINT "RECORD NOT FOUND" : GOTO PROMPT
650     IF ERF(1) <> 0 THEN GOTO ISAM'ERROR
660     READ #1, LABEL
670 ! Delete the key from the primary index.
680     ISAM #1, 4, NAME
690     IF ERF(1) <> 0 THEN GOTO ISAM'ERROR
700 ! Delete the key from the secondary index.
710     ISAM #2, 4, HASH
720     IF ERF(2) <> 0 THEN GOTO ISAM'ERROR
730 ! Delete the data record in data file.
740     ISAM #1, 6, NAME
750     IF ERF(1) <> 0 THEN GOTO ISAM'ERROR
760     GOTO PROMPT
770
780 INQUIRE'RECORD:
790     INPUT "BY NAME (1) OR HASH (2): "; FUNCTION
800     IF FUNCTION = 2 THEN GOTO BY'HASH
810     INPUT "NAME: "; NAME
820     NAME = NAME + SPACE(25-LEN(NAME))
830 ! Locate the record.
840     ISAM #1, 1, NAME
850     IF ERF(1) = 33 THEN PRINT "RECORD NOT FOUND" : GOTO PROMPT
860     IF ERF(1) <> 0 THEN GOTO ISAM'ERROR
870 ! Read the record.
880 READ'RECORD:
890     READ #1, LABEL
900     PRINT NAME, HASH
910     PRINT ADDRESS, STATE, ZIP
920     GOTO PROMPT
```

```
930 ! Locate record by hash code.
940 BY'HASH:
950     INPUT "HASH: "; HASH
960     HASH = HASH + SPACE(10-LEN(HASH))
970     ISAM #2, 1, HASH
980     IF ERF(2) = 33 THEN PRINT "RECORD NOT FOUND" : GOTO PROMPT
990     IF ERF(2) <> 0 THEN GOTO ISAM'ERROR
1000    GOTO READ'RECORD
1010
1020 PRINT'LABELS:
1030 ! Read null key to get to front of file.
1040    NAME = SPACE(25)
1050    ISAM #1, 1, NAME
1060 ! Loop thru file doing sequential reads until we hit the end.
1070 LOOP:
1080    ISAM #1, 2, NAME
1090    IF ERF(1) = 38 THEN GOTO PROMPT          ! We hit end-of-file.
1100    IF ERF(1) <> 0 THEN GOTO ISAM'ERROR
1110    READ #1, LABEL
1120    PRINT
1130    PRINT NAME, HASH
1140    PRINT ADDRESS, STATE, ZIP
1150    GOTO LOOP
1160
1170 END'IT:
1180 ! Be sure and close files before we exit.
1190    CLOSE #1
1200    CLOSE #2
1210    END
1220
1230 ISAM'ERROR:                          ! ERF(X) returned an ISAM error
1240    PRINT "?FATAL ISAM ERROR"         ! other than RECORD NOT FOUND.
1250    END
```

Index

Adding data records 7-6
Adding symbolic keys 5-7, 7-4
AMOS monitor calls 5-1
  COPY 5-1
  EXIT 5-3
  FETCH 5-1
  GETMEM . . . . . 5-1 to 5-2

BASIC
  Adding data records 7-6
  Closing files 7-5
  Deleting data records 7-10
  ERF(X) 7-4, 7-6
  Error processing 7-5
  ISAM codes 7-3
  ISAM error codes 7-6
  ISAM statement 7-3
  OPEN statement 7-1
  Opening an ISAM file 7-1
  READ statement 7-3, 7-5
  Reading data records 7-8
  Sample ISAM program 7-11
  Updating data records 7-9
  WRITE statement 7-3, 7-5
Block split 2-5

Closing ISAM files 1-6, 5-4, 7-5
Code 7-3
Completion codes 5-2, 6-1
  IS.EOF 5-9
  IS.SUC 5-2
Compressing index files 4-1
Compression factor 4-1
Condition codes 5-2
Condition flags 5-2
Contiguous file 1-4
Counted update mode 1-5, 5-4, 7-2
Creating ISAM files 2-1, 7-1

Data file . . . . . 1-2 to 1-4
Deleting data records 5-5, 7-4, 7-10
Deleting symbolic keys 5-8, 7-4
Directory Rock 1-4

Displaying the data file 3-2
Displaying the index file 3-2

ERF function 7-4
Error processing 7-5
Exclusive open mode . . . . . 1-5 to 1-6, 5-4, 7-2

File channel . . . . . 7-1 to 7-3
File interlocking . . . . . 1-5 to 1-6
File pair number . . . . . 5-3 to 5-4
File parameters 2-2
Filespec 7-2
Finalizing ISAM processing . . . 5-3
Finding data records 5-7, 7-4
Finding free data records 5-5, 7-4
Finding symbolic keys 7-4
Finding the next key 5-8, 7-4

Index file . . . . . 1-2 to 1-3
Index levels 1-4
INDEXED 7-2
Indexed sequential file 1-3
  Data file 1-3
  Index file 1-3
INDEXED'EXCLUSIVE 1-6
Indicators 5-2
Initializing ISAM 5-2
ISAM 1-1
ISAM access modes 1-5
ISAM calls 5-1, 6-1
  .ICLOS 5-1, 5-4
  .IDELK 5-8
  .IDLFR 5-5
  .IFIN 5-3
  .IGTFR 5-5
  .INIT 5-2
  .IOPNR 5-3
  .IREDR . . . . . 5-7 to 5-8
  .IRLRD 5-6
  .IRLWT 5-6
  .IWRTR 5-7
  .SREDR 5-8
ISAM codes 7-3
ISAM error codes 7-6
ISAM file extensions
  .IDA 1-4
  .IDX 1-5
  .IDY 1-6
ISAM statement 7-3
ISMBLD 1-3, 2-1, 7-1, 7-11
ISMCOM 4-1
ISMDMP 3-1
ISMFIX 1-8

ISUSYM.MAC 5-1

Loading ISAM files 2-1

Memory allocation routine 5-2
Memory deallocation routine . . . 5-2
Memory requirements 5-1
Mode 7-2

Opening ISAM files 5-3, 7-1

Primary index file . . . . . 1-3 to 1-4

Reading data records 5-6, 7-8
Record size 7-2
Relative key 1-2
Relative record number 1-2, 7-2
Return codes 5-1

Sample BASIC ISAM program 7-11
Sample ISMBLD dialog 7-11
Secondary index file . . . . . 1-3 to 1-4
Suppressing Exclusive Open mode . 1-6, 2-4, 3-1
Symbolic key 7-3

Table offsets 5-1

Unloading ISAM files 3-1
Updating data records 5-6, 7-9

Writing data records 5-6

'ARE PUBLICATIONS FILE REFERENCE NUMBER: ISAM System User's Gui

SOFTWARE DOCUMENTATION READER'S COMMENTS

reciate your help in evaluating our documentation efforts. Please feel free to attach additional comments. If you require a written response, check he

NOTE: This form is for comments on software documentation only. To submit reports on software problems, use Software Performance Reports (SPRs), available from Alpha Micro.

:omment on the usefulness, organization, and clarity of this manual.

_____

_____

_____

_____

_____

_____

_____

J find errors in this manual? If so, please specify the error and the number of the page on which it occurred.

_____

_____

_____

_____

_____

inds of manuals would you like to see in the future?

_____

_____

_____

_____

ndicate the type of reader that you represent (check all that apply):

- ☐ Alpha Micro Dealer or OEM
- ☐ Non-programmer, using Alpha Micro computer for:
  - ☐ Business applications
  - ☐ Education applications
  - ☐ Scientific applications
  - ☐ Other (please specify).

    _____
- ☐ Programmer
  - ☐ Assembly language
  - ☐ Higher-level language
  - ☐ Experienced programmer
  - ☐ Little programming experience
  - ☐ Student
  - ☐ Other (please specify).

    _____

_____ DATE: _____

_____ PHONE NUMBER: _____

NIZATION: _____

ESS: _____

_____ STATE: _____ ZIP OR COUNTRY: _____

APLE

STAPLE

FOLD

PLACE
STAMP
HERE

alpha
micro

17881 Sky Park North
Irvine, California
92714

ATTN: SOFTWARE DEPARTMENT

FOLD

CUT ALONG LINE