# UNIVAC 1100 SERIES

# INDEX SEQUENTIAL FILE MANAGEMENT SYSTEM ( ISFMS )

PROGRAMMER REFERENCE

SPERRY UNIVAC
COMPUTER SYSTEMS

This document contains the latest information available at the time of publication. However, the Univac Division reserves the right to modify or revise its contents. To ensure that you have the most recent information, contact your local Univac Representative.

UNIVAC is a registered trademark of the Sperry Rand Corporation.

Other trademarks of the Sperry Rand Corporation in this publication are:

FASTRAND



PRINTED IN U.S.A.

# PAGE STATUS SUMMARY

## ISSUE: UP–7780 Rev. 1

| Section | Page Number | Update Level |
|---|---|---|
| Cover/Disclaimer | | Orig. |
| PSS | 1 | Orig. |
| Contents | 1 thru 4 | Orig. |
| 1 | 1 thru 12 | Orig. |
| 2 | 1 thru 31 | Orig. |
| Appendix A | 1, 2 | Orig. |
| Appendix B | 1, 2, 3 | Orig. |
| Appendix C | 1 thru 4 | Orig. |
| Appendix D | 1 thru 10 | Orig. |
| Appendix E | 1 thru 4 | Orig. |
| Appendix F | 1 thru 5 | Orig. |
| Appendix G | 1 | Orig. |
| Appendix H | 1, 2 | Orig. |
| Appendix I | 1 thru 7 | Orig. |
| Index | 1, 2, 3 | Orig. |
| UCS | | |

| Section | Page Number | Update Level |
|---|---|---|

| Section | Page Number | Update Level |
|---|---|---|

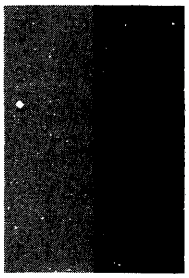
# CONTENTS

## FIGURES



## TABLES

UNIVAC
SYSTEMS
PUBLICATIONS



UNIVAC 1100 Series

# REVISION

**Index Sequential File Management System (ISFMS) Programmer Reference UP 7780 Rev. 1**

**This UNIVAC 1100 Series Library Memo announces the release and availability of "UNIVAC 1100 Series Index Sequential File Management System (ISFMS) Programmer Reference," UP-7780 Rev. 1. This is a Standard Library Item (SLI).**

This revision was developed to correspond to level 8 of the software package. The field support bulletins accompanying the release of new levels of this software will contain any needed documentation change information.

This revision includes minor corrections and provides:

- additional information on the assembler interface (Appendix B),
- additional error code information (Appendix E),
- additional command information (Appendix F),
- information on main storage index feature (Appendix H),
- information on American National Standard COBOL (Fieldata) (Appendix I), and
- an index.

Destruction Notice: This revision supersedes and replaces "UNIVAC 1106 System/1108 Multi-Processor System EXEC 8 Indexed Sequential File Management System (ISFMS) Programmer Reference," UP-7780 released on Library Memo 13 dated April 29, 1970. Also superseded and replaced is Updating Package A for UP-7780 released on P.I.E. Bulletin 1, UP-7837.1 dated September 22, 1970. Please destroy all copies of UP-7780, its Updating Package A, and the associated release memos.

Additional copies may be ordered by your Univac Manager via the Technical and Advertising Materials Requisition, UDI-578. Send to:

*Customer Information Distribution Center (CIDC)*
Univac Division Sperry Rand Corporation
P.O. Box 448
King of Prussia, Pa. 19408

*H. MASTERS*
Group Manager
Systems Publications

| LISTS | ATTACHMENTS | THIS SHEET IS: |
|---|---|---|
| 217,630 and 692 Library Memo only | UP-7780 Rev. 1 (covers and 91 pages) plus Library Memo to Univac Library Mailing Lists 37, 38, 62, 63, and 64. | Library Memo for UP-7780 Rev. 1 |
| | | DATE: May 1973 |

SPERRY RAND

# 1. INTRODUCTION

## 1.1. INDEXED SEQUENTIAL FILE MANAGEMENT SYSTEM (ISFMS)

The Univac Indexed Sequential File Management System (ISFMS) is a UNIVAC 1100 Series service routine which enables the user:

- to establish an ordered (indexed sequential) file on random storage so that each record within the file may be located directly by the user;
- to locate records, within an established file, either sequentially or randomly;
- to modify or delete old records and store new records within an established file.

ISFMS is collected and executed as part of user programs operating under the EXEC. (See *UNIVAC 1100 Series Operating System Programmer Reference, UP-4144* (current version). ISFMS requires approximately $6000_8$ words of main storage, exclusive of buffer requirements, which are user-selected. Many of the indexed sequential concepts employed by ISFMS are similar to those contained in the UNIVAC manual *Direct Access Storage Device Concepts, UE-604* (current version).

The ISFMS service routine is available through the Assembly Language described in the *UNIVAC 1100 Series American National Standard COBOL (Fieldata) Programmer Reference, UP-7845* (current version) and the COBOL that is described in *UNIVAC Fundamentals of COBOL – Series Programmer Reference, UP-7503* (current version). Unless otherwise stated, the word COBOL and non American National Standard COBOL will always specifically refer to the COBOL as it is described in the *UNIVAC Fundamentals of COBOL – Series Programmer Reference, UP-7503* (current version). Appendix I of this document describes the American National Standard COBOL (Fieldata) interface with ISFMS and therefore the word COBOL in that appendix does not refer to UP-7503.

## 1.2. USER RESPONSIBILITY

The user's interface with ISFMS is outlined below. This interface provides for the user complete flexibility in the specification, design, and usage of his files. Each user program must provide the following:

- File assignment control cards (@ASG) for each file created and maintained by ISFMS. These control cards register the files with the operating system and assign areas on the storage device for the file.
- COBOL Environment and Data Division entries appropriate to the file, when a COBOL interface routine is used (see Appendix A and Appendix I).

- File control tables (FCTs) and buffers appropriate to the file, when an assembler interface routine is used (see Appendix B).

- Error recovery in the event ISFMS encounters a nonfatal error. ISFMS returns an error status code after each unsuccessful request. The user must interpret this code and take whatever action he deems necessary. In the event of a fatal error, ISFMS automatically terminates the run.

- Loading and saving of the file through the use of @COPY control card.

- Parameters necessary to service requests. These include such things as file name, record key, function code, and so forth.

## 1.3. ISFMS TERMINOLOGY

The terminology given here reflects its current usage within ISFMS only. Whenever possible, however, ISFMS terminology does conform to the meanings commonly attached to these words in file management/data management literature.

- Block

  A grouping of records into a size more suitable for mass storage I/O transfers in order to minimize mass storage accesses and shorten the range index.

  – Data Block

    A grouping or collection of data records and their corresponding record keys. Data blocks contain the information or data the user has supplied to ISFMS for storage.

  – Index Block

    A grouping or collection of record keys and associated data block pointers. Only the highest record key per data block is entered into the index block, thereby creating a range index.

  – Overflow Block

    A collection of data records (and their corresponding record keys), which are added after the file has been created and for which no space is available in the proper data block.

- Control Word

  Any ISFMS supplied words. These words are attached to blocks and record to indicate starting positions, lengths, words available, and so forth.

- File

  A physical area on a storage device assigned, using an @ASG control card, to a file. ISFMS operates only on files contained on mass storage devices. A file contains a set of records grouped into blocks.

- Data File

  A file which contains data records and their corresponding record keys combined into data blocks. It also contains index blocks and overflow blocks. A data file is a complete ISFMS file.

- Index File

  A file which contains only index records (in blocks). This file has been built from the data file for the purpose of high speed index access.

- Input File

  A data file which has been selected for input processing. Only read commands may be issued to an input file. The file must have been created previously.

- Input/Output File

  A data file which has been selected for input/output processing, that is, updating. Read, write, insert and delete commands may be issued to an input/output file. The file must have been previously created.

- Output File

  A data file which has been selected for output processing. The file did not exist before and is being created. Only write commands are allowed.

■ Indexed Sequential

A method of file organization in which records are stored in sequential order and are accessed by means of a range index, within which each entry points to a block of records.

■ Item

A unit of data within a record (see *UNIVAC Fundamentals of COBOL-Language Programmer Reference, UP-7503.1* (current version).

■ Mass Storage

Storage, other than main storage, which can be accessed on a direct or random basis. This implies that it be a FASTRAND drum, magnetic drum, disc or unitized channel storage.

■ Range Index

An index in which each entry points to a block of data records.

■ Record

The basic unit of data being accessed through ISFMS. A record is a group of related words or items, the contents of which are determined by the user.

- Record Key

  The user-supplied word(s) by which a data record in a file is identified, sequenced, and controlled.

## 1.4. FUNCTIONAL DESCRIPTION OF ISFMS

ISFMS utilizes two types of files:

(1) Data Files

(2) Index Files

These files are subdivided into blocks. Each block contains record keys, or record keys and data records related to the other blocks by means of the indexed sequential technique. While this is provided automatically by ISFMS, the user must be aware that ISFMS is a routine which must be collected with his program and that he must pass to ISFMS certain parameters which aid ISFMS in establishing the needed files and blocks, and the relationships between their contents. These parameters are passed to ISFMS through either a COBOL or an assembler interface program.

### 1.4.1. Data Files

Any file that has been made known to ISFMS (the file has been opened) can be referenced by the user. The user can reference up to 10 completely independent data files (each data file may or may not have a separate index file associated with it) at any given time (see 1.4.2).

There are three types of data files:

(1) output

(2) input

(3) input/output

The manner in which ISFMS manipulates these files corresponds to that described in *UNIVAC Fundamentals of COBOL-Language Programmer Reference, UP-7503.1* (current version). The user must be familiar with COBOL because ISFMS, when referenced in the COBOL or assembly language user programs, utilizes some of the COBOL-provided input/output routines to access a data file.

To initially create a data file, the file must be declared to be an output file, and only output commands can be used to create the file. Once created, the file may be declared an input file, and input commands are used to read the file.

Finally, the file may be declared to be an input/output file and input/output commands can be used to read, write, modify, or delete records in the file. A data file may be closed during a run and then reopened during the same run as a different type of data file. It is also possible to have up to 10 different data files open simultaneously during a run.

A data file is defined as a collection of blocks, each of which is a collection of records. There are three basic types of blocks:

(1) data

(2) index

(3) overflow

These blocks are described in detail in 1.4.3. In addition, there are COBOL-provided label and sentinel blocks and an ISFMS-provided information block. The latter contains a description of the positions of the data, index, and overflow blocks. All blocks generated by ISFMS are the same length, i.e., ¼, ½ or track length. A schematic overview of an ISFMS-prepared file is provided in Figure 1–1.

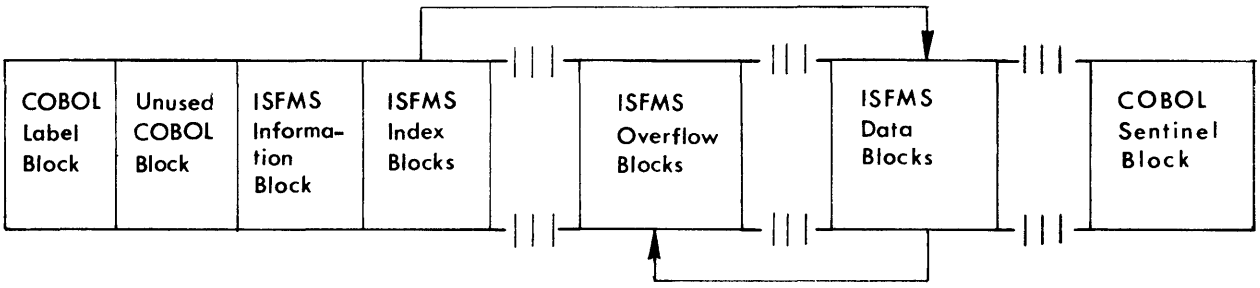



*Figure 1–1. ISFMS File*

The relationship between the blocks are provided by ISFMS. Based upon block numbers retained within the information block, ISFMS can determine the location and number of index blocks. The index blocks, in turn, point to the data blocks. When necessary, the data blocks point to the overflow blocks. The overflow blocks are used to contain records which will not fit in their respective data blocks. This is discussed in detail in 1.4.3. The ISFMS information block is read during the open and close commands. Other than requiring consideration when allocating file size, information blocks are not of concern to the user of ISFMS.

Through the use of the @ASG and @CAT control cards, a data file may be made simultaneously available to more than one user. ISFMS, however, is separately collected with each user and no copy is cognizant of any other copy accessing that data file. To prevent situations occurring where one user is referencing data records which another user is updating, data files being accessed simultaneously by more than one user should be open for input only. Updating runs, opening the file for input/output, should be made only when no other user is accessing the file.

### 1.4.2. Index Files

A separate index file may or may not be generated for a data file. If not, the index blocks maintained within the data file are referenced as necessary when a command is issued to the data file. If an index file is generated, it is considered to be a run temporary file, that is, created by the open command, and released by the close command. If used, the index file is internally assigned by ISFMS by means of the @ASG control card. This is done based upon parameters supplied by the user in his open data file command (see Section 2).

The index file is a copy of the index blocks contained within the data file. Assuming that an index file is specified by the open command (input or input/output only), ISFMS copies the index blocks from the data file to the index file. This is a straight block-for-block transfer to the index file, which is always considered empty when the open command is issued. When a reference to the data file is made (read or write), the index file is first consulted if necessary and then the proper data block is read. The index blocks within the data file are never consulted. Upon receiving a close command, the index file becomes extraneous and is deassigned. The next open command will recreate the index file from the data file. Index blocks are never modified when opened for input or input/output.

The procedure is somewhat different if the data file does not exist, that is, opening the file for output. At this point, there are no index blocks to transfer. The number of index blocks required is calculated and the index file is initialized. As the data blocks are filled and placed in the data file through the write command, the index blocks are filled and placed in the index file. Upon receiving the close command, the index blocks are copied from the index file to the data file where they are found when the file is opened for input or input/output. The index file becomes extraneous and is deassigned after the index blocks have been copied.

There are distinct advantages to having separate index and data files:

- The index file, which is usually small relative to the data file, may be placed on high speed drum or Fastband; the data file, on slower but larger FASTRAND devices.
- The index file may be placed on different mass storage devices to minimize queuing and head positioning on any one channel/device.

Obviously, more mass storage is required when separate index and data files are maintained; however, this is only for the duration of the run and is always optional. The user may elect to have ISFMS reference the index blocks within the data file and no separate index file is established.

The user may also elect to use the main storage index feature. The purpose of the main storage index feature is to minimize the mass storage accesses when processing an ISFMS file. The advantage of separate index and data files become less distinct when the main storage index feature is utilized. Appendix A describes the main storage index feature in detail.

Figure 1–2 shows a data file with its associated index file. The unlabeled areas within the data file correspond to those in Figure 1–2.

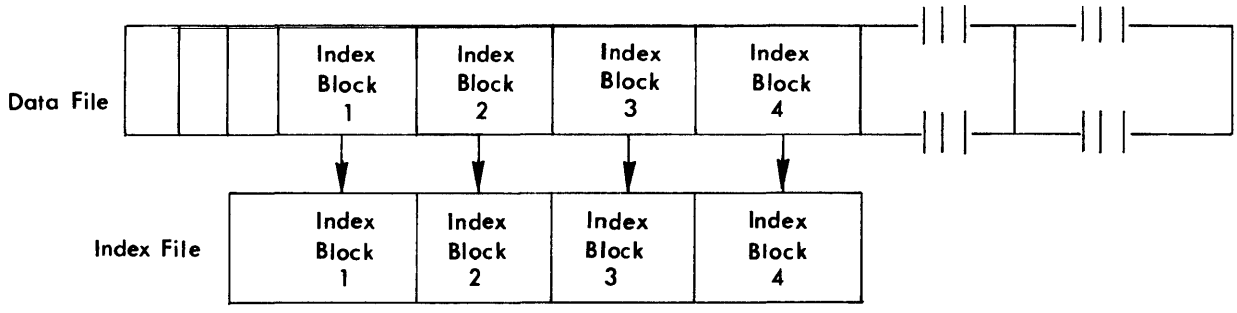



*Figure 1–2. Data File With Associated Separate Index File*

### 1.4.3. Block Organization

Three basic types of blocks are set up by ISFMS:

(1) data blocks

(2) index blocks

(3) overflow blocks

The following paragraphs discuss the relationship existing between blocks and also examine relationships existing within blocks.

A fourth type of block, the information block, is also examined.

#### 1.4.3.1. Data Block

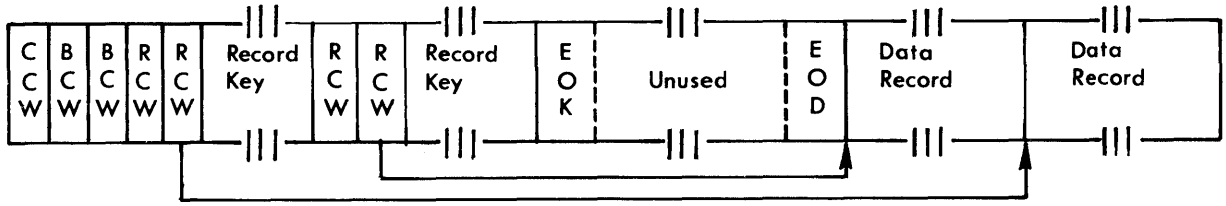
A data block consists of:

- data records
- record keys
- ISFMS control words

The data records and their associated record keys are copies of the user-supplied records and keys. The ISFMS-supplied control words locate the start of the records, record length, and number of words available within the block.

Assume that a file has been opened as an output file. This indicates that the file is to be created (no data existed in it before) and that the records are being submitted sequentially. As the data records are submitted, they are placed in the data block in inverse order from the rear of the data block. The corresponding record keys are inserted (along with ISFMS-supplied control words) in ascending order from the start of the block. When the unused area in the center of the block diminishes to the point at which not enough space is available for another data record and record key, or when the limit (to allow for later additions) specified in the file description of the open command is reached, ISFMS writes that block to mass storage, reads the current index block, and inserts the last record key and data block number into it, that is, the highest record key of that block.

This process is illustrated in Figure 1–3. The first two words are ISFMS control words pertaining to the block. Following these two words are the two ISFMS-supplied control words pertaining to the first data record, and then the record key. This sequence, two control words plus record key, is repeated for each data record. An end of key sentinel follows. The area between this sentinel and the end of record sentinel is available for additional records. Immediately following the end of record sentinel is the last data record received, the next to the last, and so forth. Data records are not separated by control words or sentinels.




NOTES:

(1) CCW indicates a COBOL control word which contains the ACTUAL KEY of that block.

(2) BCW indicates block control words:

- Word 1 of the BCW contains the block type (i.e., 0003) in T1, the number of words of data records in T2, and the number of words of record keys in T3.
- Word 2 of the BCW contains the relative block number in H1 and the relative address of the first record key in H2.

(3) RCW indicates record control words:

- Word 1 of the RCW contains the relative block number in H1, the record key length in S4, the relative address of the next record key in T3, and the leftmost bit is used as a delete flag. The record is marked as deleted if the delete flag is set to 1.
- Word 2 of the RCW contains the data record length in T2 and the relative address of the data record in T3. If the delete flag is set in word 1, then word 2 will contain the time and date of deletion.

(4) EOK indicates end of keys sentinel.

(5) EOD indicates end of data sentinel.

*Figure 1–3. Data Block*

If, at a later date, the file is reopened for input/output, additional records may be inserted into the unused area within the block. Assuming that this is the correct block and sufficient room exists, the data record is inserted before the other data records and the end of data sentinel is moved forward. The record keys are searched for the proper position; when found, all higher record keys, record control words, and the end of keys sentinel are moved backward the necessary number of words to allow insertion of the new record key and record control words.

Making the same assumptions as previously except that there is insufficient room within the data block, a somewhat different procedure is followed. Upon finding the logical insert point and determining that insufficient space exists for the new record, the record control word (RCW) of the record logically preceding the insert point is altered to point to the current overflow block (assuming the existence of records 4 and 6 and the insertion of record 5, the RCW of 4 is changed to point to record 5). The new record (record 5) is placed in the overflow block, as is the RCW of the next logical record (record 6). The same procedure is followed when inserting a new record logically between two existing overflow records. Thus, every record in the data file is implicitly linked together and a sequential search is never required, even if the record is in an overflow block.

### 1.4.3.2. Index Block

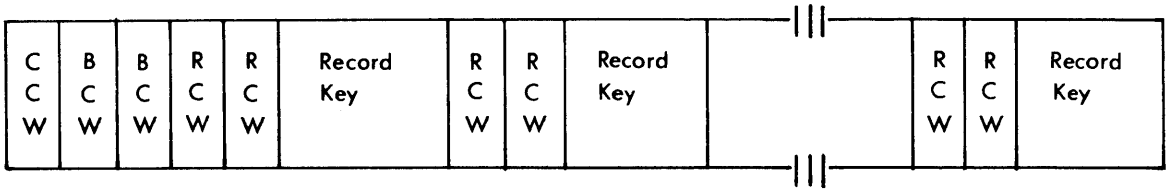
An index block contains:

- the record keys of the highest ascending, that is, last entered, data record within each data block;
- ISFMS-supplied control information.

Entries are made to index blocks only when the file is being created (output file). Each entry is made when the data block is full and must be written on mass storage. The record key of the last data record entered in the data block is retained, the current index block is read, the record key is placed following the record key associated with the previous data block, and the index block is returned to mass storage.

If the user supplies a main storage index buffer the number of mass storage accesses required for each data block written on mass storage is reduced from approximately three per data block to approximately one per data block. The main storage index will maintain the lowest level of index block in the main storage buffer until the time it becomes necessary to write the index block on mass storage.

Figure 1–4 illustrates the contents of an index block. Note that each record key entry points back to its own data block, which may contain many data records; hence, it is called a range index.




NOTES:

(1) CCW indicates a COBOL control word which contains the ACTUAL KEY of that block.

(2) BCW indicates block control words:

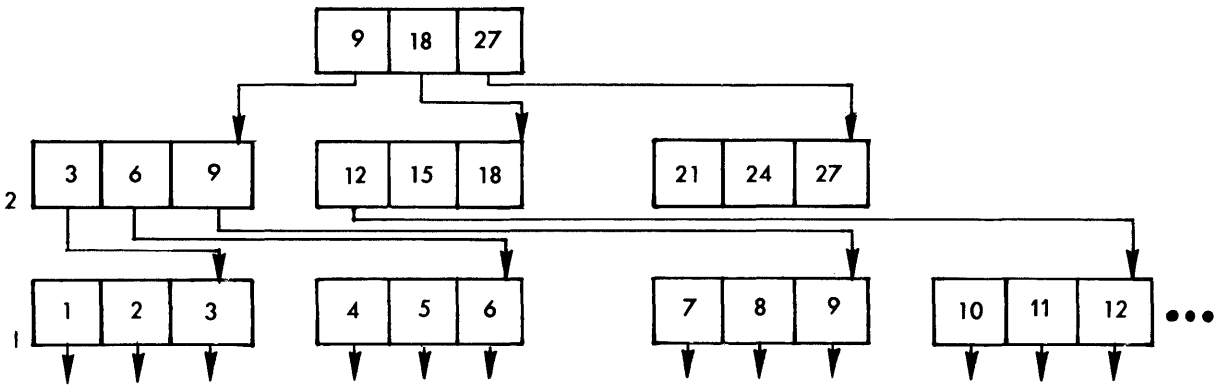
- Word 1 contains the block type (i.e., 0002) in T1 and the number of words used for record keys in T3.
- Word 2 is not used.

(3) RCW indicates record key control words:

- Word 1 contains the data block number in H1, the record key length in S4, and the relative address of the record key in T3.
- Word 2 is not used.

*Figure 1–4. Index Block*

When the number of record keys (the highest of each data block) becomes large enough to fill the initial index block, ISFMS automatically generates a new level of index blocks. The block at level 2 is then used to index into the level 1 blocks. When the initial block of level 2 is full, a third level is started and the remaining possible level 2 blocks are filled out, also pointing into level 1 (see Figure 1–5). Control words are implied by arrows.




(to data blocks containing one data record per block)

*Figure 1–5. Multilevel Indexing*

ISFMS is capable of handling up to seven such levels. To minimize access time, however, use as few levels as possible. This may be accomplished by using a small record key and a large block size. The latter implies more data records per block which reduces the absolute number of index entries (in the form of record keys) as well as providing for additional index entries per index block. Each additional level of index blocks implies one additional mass storage access, in addition to the access for the data block. Thus, a 2-level index requires three mass storage accesses to randomly obtain a data record, a 3-level index requires four accesses, and so forth. One additional access is required if the desired record is in an overflow block. ISFMS, however, always attempts to minimize the number of accesses by first examining the block presently in main storage and only accesses mass storage when necessary.

If the user supplies a main storage index buffer the number of mass storage accesses to access a data block is decreased by one. Thus, a 1-level index requires one mass storage access to randomly obtain a data record, a 2-level index requires two accesses, a 3-level index requires three accesses, and so forth.

### 1.4.3.3. Overflow Blocks

An overflow block is similar to a data block. It consists of:

- data records
- record keys
- ISFMS-supplied pointers and parameters

The major difference between a data block and an overflow block is that within an overflow block no attempt is made to maintain the record keys in ascending sequence; instead, they are stored on a first arrived, first stored basis. Overflow records, however, while being randomly stored, are not randomly accessed. When a record is not found in its data block, a link directly into the proper overflow block is picked up and followed. If more than one overflow record is placed logically between two physically sequential records in the data block, all are linked together. Thus, any overflow record may be located in a minimum number of accesses.

The execessive use of overflow blocks is the best indication of the need to reform the data file, that is, use this file as input to create a new file. Overflow block usage may be monitored by means of the Inform command described in 2.5.

#### 1.4.3.4. Information Block

The information block, generated and maintained by ISFMS, provides various summary information on:

- the starting and ending positions of the various types of blocks;
- information taken from the open command as to lengths and number of records;
- the name and qualifier of the internally assigned index file;
- statistics collected for printout by the Inform command.

Other than allocating space (one block) within the file, the user need not make special allowances for the information block. He may not reference it directly.

### 1.4.4. Indexing Technique

ISFMS uses the indexed sequential method. The term indexed implies that a separate index is built through which the data records are accessed. The term sequential implies that the record keys within the index are arranged in ascending order and that the data records are logically sequenced within the data block. This form of indexed sequential uses a range index; that is, an index entry is made only for that data record having the highest record key within its block. This imposes an ordering of data records from block to block, where each block must contain data records which fall within its low-high record key range, and a particular block cannot overlap the range of any other data block. The exception to this rule is the possible use of overflow blocks.

The major advantages of indexed sequential are its speed and simplicity. Each data record within the file is equally accessible with a maximum access equal to the number of index levels plus the read of the data block plus the read of an overflow block, if necessary. A limitation is that it is impossible to build multiple indexes automatically into the same data records, and file updating can become costly if overflow blocks are used excessively.

It is possible to build hierarchical data relationships and multiple indexes; however, it becomes the user's responsibility to establish and maintain such structures. If a hierarchy is to exist, there must be some logical relationship existing between different types of data records. For example, a payroll record and a personnel record have a logical relationship in that both refer to the same employee. The payroll records may be placed within the first data file, using Social Security number as the record key (ISFMS will automatically generate an index based upon Social Security number and the user may always reference this file accordingly). The personnel records may then be placed within a second data file using employee number as the record key on which the file is to be indexed. Again, the user may reference this file independently by using employee number. The user may now develop a cross-reference between these two files by placing their record keys within a data record and storing this into a third file. This data record has its own record key, for example, employee name, upon which ISFMS will also automatically generate an index. This example is illustrated in Figure 1–6. R indicates a data record within the data portion of the file.

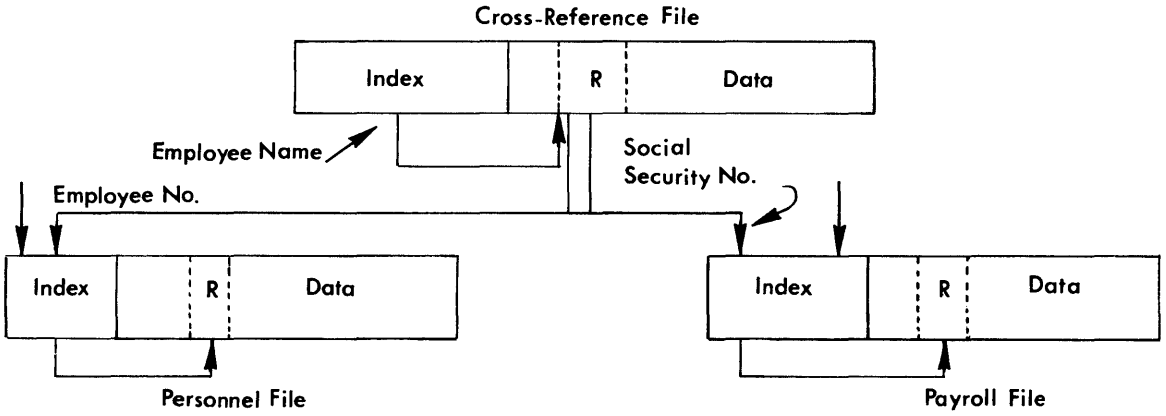



*Figure 1–6. Hierarchical Data Files*

If the cross-reference file index is searched on the record key SMITH, ISFMS produces the data record of SMITH. Within the data record are two items, each containing a record key. The first is SMITH's Social Security number; the second is SMITH's employee number. The user may choose either record key (or both) and use it to find the payroll and/or personnel data record. Thus, the user could choose the employee number of SMITH,#13450, and use this as the record key for a read request of the personnel file. The index of the personnel file would be searched for record key #13450 and SMITH's personnel record would be presented. SMITH's Social Security number could be used in a similar manner. Note that if the proper record key had been known in advance, the cross-reference file search could have been avoided, and the personnel and/or payroll files entered directly.

A similar example may be developed in which one of the items of the personnel or payroll data record (or perhaps both) contains the record key of the other data record. This eliminates the need for a third file. For example, if the payroll file data records contain the employee number and the personnel file data records contain the Social Security number, complete cross-referencing is possible. Figure 1–7 illustrates this possibility.

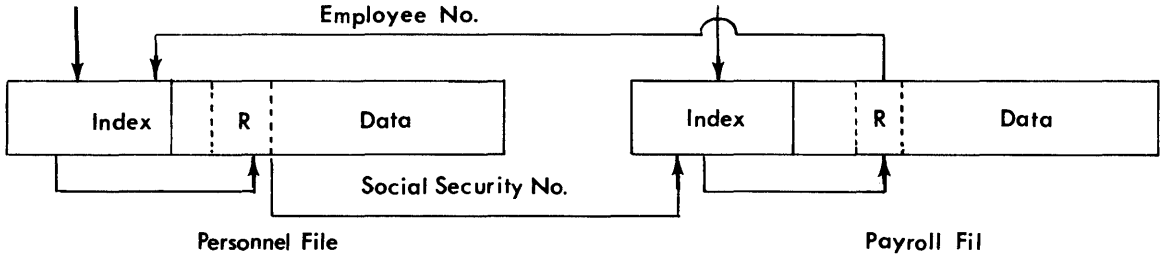



*Figure 1–7. Cross-Referenced Files*

The user can enter the personnel file index, searching for employee number #13450, and ISFMS will present the associated data record. The user may extract the item containing SMITH's Social Security number and enter the payroll file index with this as the record key of the payroll data record. SMITH's payroll record is also presented.

The previous examples are purely hypothetical; the user could just as well have indexed and searched both files on employee name, that is, SMITH. The user could have combined the payroll and personnel records into a joint record called employee record. The area of file structure and definition must be determined based upon the user's requirements and operating environment.

# 2. ISFMS COMMAND REPERTOIRE

## 2.1. GENERAL

There are three types of files, each distinguished by its use:

(1) input

(2) output

(3) input/output.

For each of these files, ISFMS provides a basic set of open, close, read, and write commands. There is also a special command, Inform.

These commands, if used through the assembly language, are used in the context of the file control table described in Appendix B. If used through COBOL, they are used in the context of the Environment Division. In the first case, they are called using PROCs; in the second, they are called by the COBOL command ENTER.

There are no functions provided for transferring of data from media; therefore, the normal utility routine of EXEC that is, @COPY, is used for this purpose. Nor are there functions for reorganization of the file; therefore, the following procedure is recommended:

(1) Read the file sequentially from FASTRAND mass storage, using ISFMS, and write it on a tape.

(2) Read the tape and, using ISFMS, write the file on FASTRAND mass storage.

Small files may be reorganized directly on FASTRAND mass storage; read the file sequentially and then write the new file. Naturally, the necessary changes to improve efficiency must have been made to the Environment Division of the new file.

### 2.1.1. Main Storage Index

ISFMS has been expanded so that the user may specify that he would like to maintain an index block in main storage. (See Appendix H for specifics).

## 2.2. OUTPUT FILES

### 2.2.1. Open Output

**Application:**

The Open Output command is used to open an output file.

**Format 1 (COBOL):**

ENTER ISFMS SUBROUTINE REFERENCING
20 *file-name status-word file-description.*

where:

20 is the function code for the Open Output command.

File-name is the name of the file, SELECTed in the INPUT-OUTPUT SECTION.

Status-word is a word which has a special meaning during the entire time a program is processing a file. After each function is performed, the status-word will contain information about the success or failure of the function (see E.2.).

File-description is a fixed-length area (six words) and contains the following information:

- Number of records (estimate)
- Record length

  Fixed-length format: length in characters (must be a multiple of six)
  Variable-length format: average (estimate) record length in characters (must be a multiple of six)

- Maximum record length

  Fixed-length format: 0
  Variable-length format: maximum record length in characters (must be a multiple of six)

- Record key length in characters (must be a multiple of six)
- Number of additional records per block to be inserted in the data area
- Independent overflow area – IOF (number of overflow records for the whole file)

**Format 2 (Assembler):**

OPO FCT-*address, status-word, file-description.*

where:

OPO is the call to a PROC which generates the function code (20) for the Open Output command. The PROC also generates the address and character length of each parameter.

FCT-address is the address of the user-built file control table for the file. For a detailed layout of the file control table, see Appendix B.

Status-word and file-description are as described in format 1.

**Prerequisite Functions:**

There are no prerequisite commands for the Open Output command. Open Output is the first command that must be issued to ISFMS and failure to do so results in an error message and termination of the run. Nevertheless, there is one prerequisite condition, that the file must be assigned to the run using an EXEC control card as follows:

@ASG,CP file-name, F2/number/TRK or POS/number

where file-name must be the file-name SELECTed and ASSIGNed in the INPUT-OUTPUT SECTION (see A.1) if COBOL is used, or the name which occupies words 1 and 2 of the file control table (see Appendix B) if the assembler is used. The options CP stand for a catalogued public file; the user may replace them with a T for a temporary file. The number of tracks or positions is discussed in Appendix C.

**Entry Conditions:**

If COBOL is used, file-name must have been SELECTed and ASSIGNed in the INPUT-OUTPUT SECTION; status-word must be defined as 01 level in the WORKING-STORAGE SECTION with PICTURE x(6); file-description must be defined as 01 level in the WORKING-STORAGE SECTION with 02 level-numbers for the number of records, record length, maximum record length, record key length, addition records per data block, and independent overflow area (IOF), all with PICTURE H9(10) and VALUE as indicated in Appendix C.

If the assembler is used, the file control table must be set up by the user, together with the necessary buffer areas (see Appendix B). The six words of the file description must be given, as indicated in Appendix C, plus one word (Fieldata zeros) for the status-word.

**Exit Conditions:**

Upon return to the user program, the status-word contains either of the following:

- Fieldata zeros, if no error has occurred; or
- the error code (see E.2).

**Error Conditions:**

The run is aborted if errors (a), (c), (d), (e), or (f) occur (see E.2). For nonfatal errors, the codes 101, 102, 210, 213, or 240 through 247 may appear in the status-word. For an explanation of the error codes, see E.3.

**Operation:**

The user supplies the Open Output command with the status-word, the file-name, or the address of the file control table, depending on the format chosen, and the file description.

The command:

(1) Checks the parameters.

(2) Registers the file and internally assigns an index file on drum if there is enough drum available.

(3) Sets up an internal file control table with entries for the internally assigned index file.

(4) Opens the files.

(5) Initializes the index blocks and the data blocks. If the user has supplied a mass storage index buffer, that area will also be initialized.

(6) Returns control to the user program with the status in the status-word.

### 2.2.2 Write Random Output

**Application:**

The Write Random Output command is used to write a record and its record key into an output file.

**Format 1 (COBOL):**

ENTER ISFMS SUBROUTINE REFERENCING
24 *file-name record-area record-key-area.*

where:

24 is the function code for Write Random Output command.

File-name is the name of the file, SELECTed in the INPUT-OUTPUT SECTION.

Record-area is the user program area in which the record to be written is located.

Record-key-area is the user program area in which the key of the record to be written is located.

**Format 2 (Assembler):**

WRRO FCT-*address, record-area, record-key-area.*

where:

WRRO is the call to a PROC which generates the function code (24) for Write Random Output. The PROC also generates the address and character length of each parameter.

FCT-address is the address of the user-built file control table for the file. For a detailed layout of the file control table, see Appendix B.

Record-area and record-key-area are as described in format 1.

**Prerequisite Function:**

The file must have been opened for output.

**Entry Conditions:**

If COBOL is used, file-name must have been SELECTed and ASSIGNed in the INPUT-OUTPUT SECTION, and record-area must be defined as 01 level in the WORKING-STORAGE SECTION with PICTURE X (number of characters in the record).

Record-key-area must also be defined as 01 level in the same section with PICTURE X (number of characters in the record key). Note that if the key consists of numeric characters, 9 may be used in place of X in the PICTURE clause. If the number of characters in the record or record key is not a multiple of six (word boundaries), FILLER must be used appropriately.

If the assembler is used, the address of the file control table is given and the appropriate areas reserved (number of words) for the record and the record key. In both cases, the key of the records to be written must be ascending.

**Exit Conditions:**

Upon return to the user program, the status-word contains one of the following:

- Fieldata zeros, if no error has occurred; or
- the error code (see E.3).

The record-area and record-key-area still contain the record and record key just written.

**Error Conditions:**

The run is aborted if error (b) or (c) occurs (see E.2). For nonfatal errors, the codes 101, 102, 210, 211, 230, 240, 248, or 362 appear in the status-word. For an explanation of the error codes, see E.3.

**Operation:**

The user supplies the Write Random Output command with the file-name or the address of the file control table, depending on the format chosen, and the areas which contain the record to be written and its associated record key.

The command:

(1) Checks the parameters.

(2) Transfers the record and the record key into the data block.

(3) Increments the number of records written in the file by one (for the statistics stored in the information block).

(4) Writes the data block, when filled, on FASTRAND mass storage and initializes a new data block.

(5) Updates the index blocks.

(a) Without main storage index:

Reads the index block from magnetic drum or FASTRAND mass storage and updates it after checking for the seventh and maximum index level which, if occurring, causes an error code to be stored in the status word.

Writes back on magnetic drum or FASTRAND mass storage the index block and initializes a new index block when the previous one is filled.

(b) With main storage index: (See Appendix H.)

Updates the main storage index block after checking for the seventh and maximum index level which, if occurring, causes an error code to be stored in the status word.

The main storage index buffer will be written to mass storage only when a new key is to be inserted and there is insufficient space in the current buffer. Writing the main storage index buffer on magnetic drum or FASTRAND mass storage will cause the buffer to be re-initialized and updated.

(6) Returns control to the user program with the status in the status-word.

### 2.2.3. Close Output

**Application:**

The Close Output command is used to close an output file. It also adds a sentinel record to the file so that the user can process this file sequentially without encountering a fatal COBOL end of file condition. The ISFMS one word sentinel record and its associated key are words of all sevens. If the last record supplied by the user has a key of all sevens, ISFMS will not add its sentinel record to the file.

**Format 1 (COBOL):**

Enter ISFMS SUBROUTINE REFERENCING
27 *file-name inform-area record-key-area.*

where:

27 is the function code for Close Output command.

File-name is the name of the file, SELECTed in the INPUT-OUTPUT SECTION.

Inform-area is the area into which information will be placed as to what has been done to the file. This area is required on all Close commands be they on input, output, or input/output files or the special command, Inform. The inform-area is of fixed length and must be nine words. It contains the following information pertaining to the status of the file as a whole; all information is relative to the time the file was initially opened for output:

- Number of blocks
- Number of index blocks
- Number of overflow blocks (IOF)
- Number of records
- Number of records in independent overflow area (IOF)
- Number of records deleted
- Number of records read
- Number of records read from IOF
- Number of records written

Record-key-area must be defined, but need not be initialized; it enables ISFMS to write the sentinel record.

**Format 2 (Assembler):**

CLO FCT-*address, inform-area, record-key-area.*

where:

CLO is the call to a PROC which generates the function code (27) for the Close Output command. The PROC also generates the address and character length of each parameter.

Inform-area and record-key-area are as described in format 1.

**Prerequisite Functions:**

The file must have been opened for output.

**Entry Conditions:**

If COBOL is used, file-name must have been SELECTed and ASSIGNed in the INPUT-OUTPUT SECTION, and inform-area must be defined as 01 level in the WORKING-STORAGE SECTION with 02 level-numbers for the nine words of information described above (each with PICTURE H9 (10)). Record-key-area must be defined as 01 level in the same section with PICTURE X (number of characters in the record-key) and FILLER with the appropriate PICTURE to make the number of characters a multiple of six (word boundaries).

If the assembler is used, the address of the file control table is given and the appropriate areas reserved (number of words) for the inform-area and the record-key-area.

**Exit Conditions:**

Upon return to the user program, the status-word contains either of the following:

- Fieldata zeros, if no error has occurred; or
- the error code (see E.3).

The inform-area contains statistics, as described under format 1.

**Error Conditions:**

The run is aborted if either error (b) or (c) occurs (see E.2). For nonfatal errors, the codes 101, 102, 230, 240, or 248 may appear in the status-word. For an explanation of the error codes, see E.3.

**Operation:**

The user supplies the Close Output command with the file-name or the address of the file control table, depending on the format chosen, the area which will contain the statistics, and the area for the record key.

The command:

(1) Checks the parameters.

(2) Writes a sentinel record if necessary.

(3) Writes the data block on FASTRAND mass storage.

(4) Transfers the index blocks from magnetic drum to FASTRAND mass storage.

(5) Initializes all IOF blocks.

(6) Creates and writes the information block.

(7) Closes the magnetic drum.

(8) Closes FASTRAND mass storage.

(9) Returns control to the user program with the status in the status-word.

## 2.3. INPUT FILES

### 2.3.1. Open Input

**Application:**

The Open Input command is used to open input files.

**Format 1 (COBOL):**

ENTER ISFMS SUBROUTINE REFERENCING
10 *file-name status-word.*

where:

10 is the function code for Open Input command.

File-name is the name of the file which had been created as an output file and which has been SELECTed in the INPUT-OUTPUT SECTION.

Status-word is the word which contains information about the success or failure of the function (see E.3).

**Format 2 (Assembler):**

OPI FCT-*address, status-word.*

where:

OPI is the call to a PROC which generates the function code (10) for the Open Input command. The PROC also generates the address and the character length of each parameter.

FCT-address is the address of the user-built file control table for the file. For a detailed layout of the file control table, see Appendix B.

Status word is described in format 1.

**Prerequisite Functions:**

There is no prerequisite command for the Open Input command. Open Input is the first command that must be issued to ISFMS and failure to do so results in an error message and termination of the run. Nevertheless, there are the prerequisite conditions that the file has been created as an output file; it may or may not be catalogued, and it must have been assigned to the run using an EXEC control card as follows:

@ASG,A file-name, F2/number/TRK or POS/number

The A option represents an already catalogued file (it could be replaced by T – for temporary file). File-name is the same name as the one on the @ASG control card used to create the output file, and which will appear in the INPUT-OUTPUT SECTION, to which the SELECTed file-name is ASSIGNed (see A.1) if COBOL is used, or the name which occupies words 1 and 2 of the file control table (see Appendix B) if the assembler is used.

Status-word must contain a right-justified Fieldata value with the following meaning:

0 – The user is asking for the index blocks to be copied from FASTRAND mass storage to magnetic drum (FH-432); but, if there is not enough space, they are left on FASTRAND mass storage. ISFMS automatically determines the number of tracks required and internally submits an @ASG.

1 – The user is asking for the index blocks to be copied from FASTRAND mass storage to magnetic drum (FH-432); however, if there is not enough space, control is returned to the user's program and status-word contains an error code.

2 – The user is asking for the index blocks to be left on FASTRAND mass storage.

If the proper value (Fieldata 0, 1 or 2) has not been placed in the status word by the user, ISFMS will force the default value of 0 to the status word.

Finally, the number of tracks, or positions, specifies the length of the file.

**Entry Conditions:**

If COBOL is used, file-name must have been SELECTed and ASSIGNed in the INPUT-OUTPUT SECTION and status-word must be defined as 01 level in the WORKING-STORAGE SECTION with PICTURE X(6).

If the assembler is used, the file control table must be set up by the user, together with the necessary buffer areas (see Appendix B), plus one word for the status-word, as explained in format 1.

**Exit Conditions:**

Upon return to the user program, the status-word contains either of the following:

- Fieldata zeros, if no error has occurred; or
- the error code (see E.3).

The first record is not read by this command.

**Error Conditions:**

The run is aborted if the errors (a), (c), (d), (e), or (f) occur (see E.2). For nonfatal errors, the codes 101 or 102 may appear in the status word. For an explanation of the error codes, see E.3.

**Operation:**

The user supplies the Open Input command with the status-word and the file-name or the address of the file control table, depending on the format chosen.

The command:

(1) Checks the parameters.

(2) Opens the file.

(3) Reads the information block from FASTRAND mass storage and transfers the information to an internally set up file control table.

(4) May assign and open the index file, based on the information contained in the status-word.

(5) Will move the highest level index block to main storage, if the user has supplied a main storage index buffer.

(6) Reads the first data block from FASTRAND mass storage.

(7) Returns control to the user program with the status in the status-word.

### 2.3.2. Read Sequential Input

**Application:**

The Read Sequential Input command is used to read the file sequentially, that is, according to the ordering of the record keys.

**Format 1 (COBOL):**

ENTER ISFMS SUBROUTINE REFERENCING
11 *file-name record-area record-key-area.*

where:

11 is the function code for Read Sequential Input command.

File-name is the name of the file, SELECTed in the INPUT-OUTPUT SECTION.

Record-area is the user program area in which the record is stored by the Read Sequential Input command.

Record-key-area is the user program area in which the key of the record just read is stored by the Read Sequential Input command.

**Format 2 (Assembler):**

RDSI FCT-*address, record-area, record-key-area.*

where:

RDSI is the call to a PROC which generates the function code (11) for the Read Sequential Input command. The PROC also generates the address and character length of each parameter.

FCT-address is the address of the user-built file control table for the file. For a detailed layout of the file control table, see Appendix B.

Record-area and record-key-area are as described in format 1.

**Prerequisite Functions:**

The file must have been opened for input.

**Entry Conditions:**

If COBOL is used, file-name must have been SELECTed and ASSIGNed in the INPUT-OUTPUT SECTION, and record-area must be defined as 01 level in the WORKING-STORAGE SECTION with PICTURE X (number of characters in the record). Record-key-area must also be defined as 01 level in the same section with PICTURE X (number of characters in the record key). If the number of characters in the record or in the record key is not a multiple of six (word boundaries), FILLER must be used appropriately.

If the assembler is used, the address of the file control table must be given and the appropriate areas reserved (number of words) for the record and the record key.

Regardless of the format used, a Read Sequential Input command following the Open Input command automatically releases the first record of the file to the user program. The user does not have to initialize the key in order to obtain the first record. Before issuing a Read Sequential Input command at any other point in the user program, a Read Random Input command must be given to initiate input at the desired point. For example, to begin sequential processing with a data record having a record key of 100, then record 100 must be read with a Read Random Input command and sequential processing may proceed from then on. On a Read Sequential Input command, the record key of the newly read record is placed in the record-key-area, which automatically prepares the way for the user to issue the Read Sequential Input command to read the next record.

**Exit Conditions:**

Upon return to the user program, the status-word contains either of the following:

- Fieldata zeros, if no error has occurred; or
- the error code (see E.3).

The record-area contains the record read and record-key-area contains its key.

**Error Conditions:**

The run is aborted if either error (b) or (c) occurs (see E.2). For nonfatal errors, the codes 101, 230, 248, 350, or 360 may appear in the status-word. For an explanation of the error codes, see E.3.

**Operation:**

The user supplies the Read Sequential Input command with the file-name or the address of the file control table, depending on the format chosen and the areas which contain the record and its key after the command is executed.

The command:

(1) Checks the parameters.

(2) Checks for more data records in the data block; if not found, it reads the next data block from FASTRAND mass storage.

(3) Checks for the sentinel record.

(4) Checks whether the record exists or has been deleted.

(5) Checks if the record is an overflow record and, if it is, the number of records read from the IOF area is incremented by one (for statistics stored in the information block).

(6) Transfers the record to the user-specified area.

(7) Increments the number of records read by 1.

(8) Returns control to the user program with the status in the status-word.

### 2.3.3. Read Random Input

**Application:**

The Read Random Input command presents the requested record, as specified by the record key, to the user program.

**Format 1 (COBOL):**

ENTER ISFMS SUBROUTINE REFERENCING
12 *file-name record-area record-key-area.*

where:

12 is the function code for the Read Random Input command.

File-name is the name of the file SELECTed in the INPUT-OUTPUT SECTION.

Record-area is the user program area in which the record is stored by the Read Random Input command.

Record-key-area is the user program area in which the user has placed the key of the record to be read.

**Format 2 (Assembler):**

RDRI FCT-*address, record-area, record-key-area.*

where:

RDRI is the call to a PROC which generates the function code (12) for the Read Random Input command. The PROC also generates the address and character length of each parameter.

FCT-address is the address of the user-built file control table for the file. For a detailed layout of the file control table, see Appendix B.

Record-area and record-key-area are as described in format 1.

**Prerequisite Functions:**

The file must have been opened for input.

**Entry Conditions:**

If COBOL is used, file-name must have been SELECTed and ASSIGNed in the INPUT-OUTPUT SECTION, and record-area must be defined as 01 level in the WORKING-STORAGE SECTION with PICTURE X (number of characters in the record). Record-key-area must also be defined as 01 level in the same section with PICTURE X (number of characters in the record key). If the number of characters in the record or in the record key is not multiple of six (word boundaries), FILLER must be used appropriately.

If the assembler is used, the address of the file control table must be given and the appropriate areas reserved (number of words) for the record and the record key.

Before issuing a Read Random Input command, the record-key-area must contain the proper key.

A Read Random Input command may follow a Read Sequential Input command and vice versa. A Read Sequential Input command following a Read Random Input command reads the next logical record that follows the last previous record, that is, the one obtained by a Read Random Input command.

**Exit Conditions:**

Upon return to the user program, the status word contains either of the following:

- Fieldata zeros, if no error has occurred; or
- the error code (see E.3).

The record-area contains the record read and the record-key-area still contains the given record key.

**Error Conditions:**

The run is aborted if either error (b) or (c) occurs (see E.2). For the nonfatal errors, the codes 101, 102, 230, 240, 248, or 361 may appear in the status-word. For an explanation of the error codes, see E.3.

**Operation:**

The user supplies the Read Random Input command with the file-name or the address of the file control table, depending on the format chosen, the area which will contain the record after the function is performed, and the area which contains the specified key.

The command:

(1) Checks the parameters.

(2) Checks the record key to determine if the data record should reside in the data block which is already in main storage; if yes go to (7), otherwise continue with (3).

(3) Finds the index block in which the given record key exists (range index).

(4) Reads this index block from magnetic drum.

(5) Matches the record key with a key in the index block, which then points to a data block.

(6) Reads this data block from FASTRAND mass storage.

(7) Searches for the record in this data block and, if not found, continues the search in the overflow block.

(8) Transfers the record, if found, into the user-specified record-area.

(9) Increments the number of records read by 1.

(10) Returns control to the user program with the status in the status-word.

### 2.3.4. Close Input

**Application:**

The Close Input command is used to close an input file. In the case of sequential processing, it is not necessary to have processed the file to the end.

**Format 1 (COBOL):**

ENTER ISFMS SUBROUTINE REFERENCING
17 *file-name inform-area.*

where:

17 is the function code for the Close Input command.

File-name is the name of the file, SELECTed in the INPUT-OUTPUT SECTION.

Inform-area is the area which will contain information on what has been done to the file.

For a description of this area, see 2.2.3.

**Format 2 (Assembler):**

CLI FCT-*address, inform-area.*

where:

CLI is the call to a PROC which will generate the function code (17) for the Close Input command. The PROC also generates the address and character length of each parameter.

FCT-address is the address of the user-built file control table for the file.

Inform-area is the area which will contain information on what has been done to the file (see 2.2.3).

**Prerequisite Functions:**

The file must have been opened for input.

**Entry Conditions:**

If COBOL is used, file-name must have been SELECTed and ASSIGNed in the INPUT-OUTPUT SECTION, and inform-area must be defined as 01 level in the WORKING-STORAGE SECTION, with 02 level-numbers for the nine words of information described in 2.2.3, each with PICTURE H9(10).

If the assembler is used, the address of the file control table is given and the appropriate nine-word area reserved for the inform-area.

**Exit Conditions:**

Upon return to the user program, the status-word contains either of the following:

- Fieldata zeros, if no error has occurred; or
- the error code (see E.3).

The inform-area contains statistics as described in 2.2.3.

**Error Conditions:**

The run is aborted if either error (b) or (c) occurs (see E.2), or the nonfatal error codes 230, 248 may appear in the status-word. For an explanation of the error codes, see E.3.

**Operation:**

The user supplies the Close Input command with the file-name or the address of the file control table, depending on the format chosen, and the area which will contain the statistics.

The command:

(1) Checks the parameters.

(2) Transfers the statistics to the user-specified inform-area.

(3) Closes and frees the magnetic drum, if it was used for the index file.

(4) Closes FASTRAND mass storage file.

(5) Returns control to the user program with the status in the status-word.

## 2.4. INPUT/OUTPUT FILES

### 2.4.1. Open Input/Output

**Application:**

The Open Input/Output command opens an input/output file.

**Format 1 (COBOL):**

```
ENTER ISFMS SUBROUTINE REFERENCING
30 file-name status-word.
```

where:

30 is the function code for the Open Input/Output command. File-name is the name of the file which has been created as output before and which has been SELECTed in the INPUT-OUTPUT SECTION.

Status-word is the word which contains information about the success or failure of the function (see E.3).

**Format 2 (Assembler):**

OPIO FCT-*address, status-word.*

where:

OPIO is the call to a PROC which generates the function code (30) for the Open Input/Output command. The PROC also generates the address and the character length of each parameter.

FCT-address is the address of the user-built file control table for the file. For a detailed layout of the file control table see Appendix B. Status-word is as described in format 1.

**Prerequisite Functions:**

There are no prerequisite command for the Open Input/Output command. Open Input/Output is the first command that must be issued to ISFMS and failure to do so results in an error message and termination of the run. Nevertheless, there are the prerequisite conditions that the file has been created as output and may or may not be catalogued and that it will be assigned to the run using an EXEC control card as follows:

@ASG,A file-name, F2/number/TRK or POS/number

The A option represents an already catalogued file (it could be replaced by T – for temporary file). File-name is the same name as the one on the @ASG control card used to create the output file, and which will appear in the INPUT-OUTPUT SECTION, to which the SELECTed file-name is ASSIGNed (see A.1) if COBOL is used, or the name which occupies words 1 and 2 of the file control table (see Appendix B) if the Assembler is used.

Status-word must contain a right-justified Fieldata value, with the following meaning:

0 – The user is asking for the index blocks to be copied from FASTRAND mass storage to magnetic drum (FH-432); but if there is not enough space, they are left on FASTRAND mass storage. ISFMS automatically determines the number of tracks required and internally submits an @ASG.

1 – The user is asking for the index blocks to be copied from FASTRAND mass storage to magnetic drum (FH-432); however, if there is not enough space, control is returned to the user's program and status-word contains an error code.

2 – The user is asking for the index blocks to be left on FASTRAND mass storage.

If the proper value (Fieldata 0, 1 or 2) has not been placed in the status word by the user, ISFMS will force the default value of 0 to the status word.

Finally, the number of tracks, or positions, specifies the length of the file.

**Entry Conditions:**

If COBOL is used, file-name must have been SELECTed and ASSIGNed in the INPUT-OUTPUT SECTION and status-word must be defined as 01 level in the WORKING-STORAGE SECTION with PICTURE X (6).

If the assembler is used, the file control table must be set up by the user, together with the necessary buffer areas (see Appendix B), plus one word for the status-word, as explained in format 1.

**Exit Conditions:**

Upon return to the user program, the status-word contains either of the following:

- Fieldata zeros, if no error has occurred; or
- the error code (see E.3).

The first record is not read by this command.

**Error Conditions:**

The run is aborted if any of the errors (a), (c), (d), (e), or (f) occurs (see E.2). For the nonfatal errors, the codes 101 or 102 may appear in the status-word. For an explanation of the error codes, see E.3.

**Operation:**

The user supplies the Open Input/Output command with the status-word and the file-name or the address of the file control table, depending on the format chosen.

The command:

(1) Checks the parameters.

(2) Opens the file.

(3) Reads the information block from FASTRAND mass storage and transfers the information to an internally set up file control table.

(4) May assign and open the index file, based on the information contained in the status-word.

(5) Will move the highest level index block to main storage, if the user has supplied a main storage index buffer.

(6) Reads the first data block from FASTRAND mass storage.

(7) Returns control to the user program with the status in the status-word.

### 2.4.2. Read Sequential Input/Output

**Application:**

The Read Sequential Input/Output command is used to read the file sequentially, that is, according to the ordering of the record keys.

**Format 1 (COBOL):**

ENTER ISFMS SUBROUTINE REFERENCING
31 *file-name record-area record-key-area.*

where:

31 is the function code for Read Sequential Input/Output command.

File-name is the name of the file, SELECTed in the INPUT-OUTPUT SECTION.

Record-area is the user program area in which the record is stored by the Read Sequential Input/Output command.

Record-key-area is the user program area in which the key of the record just read is stored by the Read Sequential Input/Output command.

**Format 2 (Assembler):**

RDSIO FCT-*address, record-area, record-key-area.*

where:

RDSIO is the call to a PROC which generates the function code (31) for the Read Sequential Input/Output command. The PROC also generates the address and character length of each parameter.

FCT-address is the address of the user-built file control table for the file.

Record-area and record-key-area are as described in Format 1.

**Prerequisite Functions:**

The file must have been opened for input/output.

**Entry Conditions:**

If COBOL is used, file-name must have been SELECTed and ASSIGNed in the INPUT-OUTPUT SECTION, and record-area must be defined as 01 level in the WORKING-STORAGE SECTION with PICTURE X (number of characters in the record). Record-key-area must also be defined as 01 level in the same section with PICTURE X (number of characters in the record key). If the number of characters in the record or in the record key is not a multiple of six (word boundaries), FILLER must be used appropriately.

If the assembler is used, the address of the file control table must be given and the appropriate areas reserved (number of words) for the record and the record key.

Regardless of the format used, a Read Sequential Input/Output command following the Open Input/Output command automatically releases the first record of the file to the user program. The user does not have to initialize the key in order to obtain the first record. Before issuing a Read Sequential Input/Output command at any other point in the user program, a Read Random Input/Output command must be given to initiate input at the desired point. For example, to begin sequential processing with a data record having a record key of 100, then record 100 must be read with a Read Random Input/Output command and sequential processing may proceed from then on. On a Read Sequential Input/Output command, the record key of the newly read record is placed in the record-key-area, which automatically prepares the way for the user to issue the Read Sequential Input/Output command to read the next record.

**Exit Conditions:**

Upon return to the user program, the status-word contains either of the following:

- Fieldata zeros, if no error has occurred; or
- the error code (see E.3).

The record-area contains the record read and record-key-area contains its key.

**Error Conditions:**

The run is aborted if either error (b) or (c) occurs (see E.2). For the nonfatal errors, the codes 101, 230, 240, 248, 350, or 360 may appear in the status-word. For an explanation of the error codes, see E.3.

**Operation:**

The user supplies the Read Sequential Input/Output command with the file-name or the address of the file control table, depending on the format chosen, and the areas which will contain the record and its corresponding key after the command is executed.

The command:

(1) Checks the parameters.

(2) Checks for more data records in the data block; if not found, it reads the next data block from FASTRAND mass storage.

(3) Checks for the sentinel record.

(4) Checks whether the record exists or has been deleted, in which case it continues to the next record.

(5) Checks if the record is an overflow record and, if it is, the number of records read from the IOF area is incremented by 1 (for statistics stored in the information block).

(6) Transfers the record to the user-specified area.

(7) Increments the number of records read by 1.

(8) Updates an entry in the internal file control table for the last function with an R for read.

(9) Returns control to the user program with the status in the status-word.

### 2.4.3. Read Random Input/Output

**Application:**

The Read Random Input/Output command presents the requested record, as specified by the record key, to the user program.

**Format 1 (COBOL):**

ENTER ISFMS SUBROUTINE REFERENCING
32 *file-name record-area record-key-area.*

where:

32 is the function code for the Read Random Input/Output command.

File-name is the name of the file SELECTed in the INPUT-OUTPUT SECTION.

Record-area is the user program area in which the record is stored by the Read Random Input/Output command.

Record-key area is the user program area in which the key of the record to be read is located.

**Format 2 (Assembler):**

RDRIO FCT-*address, record-area, record-key-area.*

where:

RDRIO is the call to a PROC which generates the function code (32) for the Read Random Input/Output command. The PROC also generates the address and character length of each parameter.

FCT-address is the address of the user-built file control table.

Record-area and record-key-area are as described in format 1. For a detailed layout of the file control table, see Appendix B.

**Prerequisite Functions:**

The file must have been opened for input/output.

**Entry Conditions:**

If COBOL is used, file-name must have been SELECTed and ASSIGNed in the INPUT-OUTPUT SECTION, and record-area must be defined as 01 level in the WORKING-STORAGE SECTION with PICTURE X (number of characters in the record). Record-key-area must also be defined as 01 level in the same section with PICTURE X (number of characters in the record key). If the number of characters in the record or in the record key is not a multiple of six (word boundaries), FILLER must be used appropriately.

If the assembler is used, the address of the file control table must be given and the appropriate areas reserved (number of words) for the record and the record key.

Before issuing a Read Random Input/Output command, the record-key-area must contain the proper key.

A Read Random Input/Output command may follow a Read Sequential Input/Output command and vice versa. A Read Sequential Input/Output command following a Read Random Input/Output command reads the next logical record that follows the last previous record, that is, the one obtained by a Read Random Input/Output command.

**Exit Conditions:**

Upon return to the user program, the status-word contains either of the following:

- Fieldata zeros, if no error has occurred; or
- the error code (see E.3).

The record-area contains the record read and the record-key-area still contains the given record-key.

**Error Conditions:**

The run is aborted if either error (b) or (c) occurs (see E.2) or the nonfatal error codes 101, 102, 230, 240, 248, or 361 may appear in the status word. For an explanation of the error codes, see E.3.

**Operation:**

The user supplies the Read Random Input/Output command with the file-name or the address of the file control table, depending on the format chosen, the area which will contain the record after the function is performed, and the area which contains the specified key.

The command:

(1) Checks the parameters.

(2) Checks the record key to determine if the data record should reside in the data block which is already in main storage; if yes go to (7), otherwise continue with (3).

(3) Finds the index block in which the given record key exists (range index).

(4) Reads this index block from magnetic drum.

(5) Matches the record key with a key in the index block, which then points to a data block.

(6) Reads this data block from FASTRAND mass storage.

(7) Searches for the record in this data block and, if not found, continues the search in the overflow block.

(8) Transfers the record, if found, into the user-specified record-area.

(9) Increments the number of records read by 1.

(10) Updates an entry in the internal file control table for the last function with an R for read.

(11) Returns control to the user program with the status in the status-word.

### 2.4.4. Write Sequential Input/Output

**Application:**

The Write Sequential Input/Output command rewrites the record which was accessed with the last Read Input/Output command, either Random or Sequential.

**Format 1 (COBOL):**

ENTER ISFMS SUBROUTINE REFERENCING
33 *file-name record-area record-key-area.*

where:

33 is the function code for Write Sequential Input/Output command.

File-name is the name of the file, SELECTed in the INPUT-OUTPUT SECTION.

Record-area is the user program area in which the record to be written is located.

Record-key-area is the user program area in which the key of the record to be written is located.

**Format 2 (Assembler):**

WRSIO FCT-*address, record-area, record-key-area.*

where:

WRSIO is the call to a PROC which generates the function code (33) for Write Sequential Input/Output command. The PROC also generates the address and character length of each parameter.

FCT-address is the address of the user-built file control table.

Record-area and record-key-area are as described in format 1.

**Prerequisite Functions:**

The file must have been opened for input/output and the previous command must be either a Read Random Input/Output or a Read Sequential Input/Output.

**Entry Conditions:**

If COBOL is used, file-name must have been SELECTed and ASSIGNed in the INPUT-OUTPUT SECTION, and record-area must be defined as 01 level in the WORKING-STORAGE SECTION with PICTURE X (number of characters in the record). Record-key-area must also be defined as 01 level in the same section with PICTURE X (number of characters in the record key). If the key consists of numeric characters, 9 may be used in place of X in the PICTURE. If the number of characters in the record or in the record key is not a multiple of six (word boundaries), FILLER must be used appropriately.

If the assembler is used, the address of the file control table must be given and the appropriate areas reserved (number of words) for the record and record key.

**Exit Conditions:**

Upon return to the user program, the status-word contains either of the following:

- Fieldata zeros, if no error has occurred; or
- the error code (see E.3).

The record-area and record-key-area still contain the record and record key just written. This key has remained unchanged since the last Read Input/Output command.

**Error Conditions:**

The run is aborted if either error (b) or (c) occurs (see E.2). For the nonfatal errors, the codes 101, 230, 240, 248, or 363 may appear in the status-word. For an explanation of the error codes, see E.3.

**Operation:**

The user supplies the Write Sequential Input/Output command with the file-name or the address of the file control table, depending on the format chosen, and the areas which contain the record to be written and its record key.

The command:

(1) Checks the parameters.

(2) Checks if the last command was Read Input/Output.

(3) Checks for a match of the key of the record previously read with the key of the record to be written.

(4) Checks for a match of the record length with the length of the record previously read.

(5) Transfers the record into the proper area as determined by the key.

(6) Rewrites the current data block on FASTRAND mass storage.

(7) Updates the entry in the internal file control table for the last function with a W for write.

(8) Returns control to the user program with the status in the status-word.

### 2.4.5. Write Random Input/Output

**Application:**

The Write Random Input/Output command adds a new record to the file. If the file already contains a record with the same key, the existing record is not overwritten.

**Format 1 (COBOL):**

ENTER ISFMS SUBROUTINE REFERENCING
34 *file-name record-area record-key-area.*

where:

34 is the function code for the Write Random Input/Output command.

File-name is the name of the file SELECTed in the INPUT-OUTPUT SECTION.

Record-area is the user program area in which the record to be written is located.

Record-key-area is the user program area which contains the key of the record to be written.

**Format 2 (Assembler):**

WRRIO FCT-*address, record-area, record-key-area.*

where:

WRRIO is the call to a PROC which generates the function code (34) for the Write Random Input/Output command. The PROC also generates the address and character length of each parameter.

FCT-address is the address of the user-built file control table. For a detailed layout of the file control table, see Appendix B.

Record-area and record-key-area are as described in format 1.

**Prerequisite Functions:**

The file must have been opened for input/output.

**Entry Conditions:**

If COBOL is used, file-name must have been SELECTed and ASSIGNed in the INPUT-OUTPUT SECTION, and record-area must be defined as 01 level in the WORKING-STORAGE SECTION with PICTURE X (number of characters in the record).

Record-key-area must also be defined as 01 level in the same section with PICTURE X (number of characters in the record key). If the number of characters in the record or in the record key is not a multiple of six (word boundaries), FILLER must be used appropriately.

If the assembler is used, the address of the file control table must be given and the appropriate areas reserved (number of words) for the record and the record key.

**Exit Conditions:**

Upon return to the user program, the status-word contains either of the following:

- Fieldata zeros, if no error has occurred; or
- the error code (see E.3).

The record-area and record-key-area still contain the record and record key just written.

**Error Conditions:**

The run is aborted if either error (b) or (c) occurs (see E.2), or the nonfatal error codes 101, 102, 212, 230, 240 248, or 364 may appear in the status-word. For an explanation of the error codes, see E.3.

**Operation:**

The user supplies the Write Random Input/Output command with the file-name or the address of the file control table, depending on the format chosen, plus the areas which contain the new record and its key.

The command:

(1) Reads this record randomly and checks if it already exists in the file. If so, error status 364 (see Table E-1) is returned.

(2) Checks if the read random in (1) required entry into an overflow block, the data block is known to be full and the record is automatically placed in an overflow block. If the read random in (1) did not require entry into an overflow block, an attempt is made to place the record in its proper data block. The function determines whether this block is already in main storage, in which case it determines if there is space available in this data block for a subsequent transferring of the record into it and if filled, the writing of the data block on FASTRAND mass storage. If these conditions are not met, the procedure for the IOF is repeated.

(3) Updates the entry in the internal file control table for the last function with a W for write.

(4) Increments the number of records written by 1.

(5) Returns control to the user program with the status in the status-word.

### 2.4.6. Write Random Delete Input/Output

**Application:**

The Write Random Delete Input/Output command replaces a part of the record key control words with the date of deletion (date and time of day). The record key and record itself are not altered.

**Format 1 (COBOL):**

ENTER ISFMS SUBROUTINE REFERENCING
36 *file-name record-key-area.*

where:

36 is the function code for the Write Random Delete Input/Output command.

File-name is the name of the file SELECTed in the INPUT-OUTPUT SECTION.

Record-key-area is the user program area which contains the key of the record to be deleted.

**Format 2 (Assembler):**

WRRD FCT-*address, record-key-area.*

where:

WRRD is the call to a PROC which generates the function code (36) for the Write Random Delete Input/Output command. The PROC also generates the address and character length of each parameter.

FCT-address is the address of the user-built file control table. For a detailed layout of the file control table, see Appendix B.

Record-key-area is the user program area which contains the key of the record to be deleted.

**Prerequisite Functions:**

The file must have been opened for input/output.

**Entry Conditions:**

If COBOL is used, file-name must have been SELECTed and ASSIGNed in the INPUT-OUTPUT SECTION, and record-key-area must be defined as 01 level in the WORKING-STORAGE SECTION with PICTURE X (number of characters in the record). If this number of characters is not a multiple of six (word boundaries), FILLER must be used appropriately.

If the assembler is used, the address of the file control table is given and the appropriate area reserved (number of words) for the record key. In both cases, the record-key-area contains the key of the record to be deleted.

**Exit Conditions:**

Upon return to the user program, the status-word contains either of the following:

- Fieldata zeros, if no error has occurred; or
- the error code (see E.3).

Key control word 1 has bit 35 set to 1; key control word 2 contains the date and time of the data deletion.

**Error Conditions:**

The run is aborted if either error (b) or (c) occurs (see E.2), or the nonfatal error codes 101, 102, 230, 248, or 361 may appear in the status-word. For explanation of the error codes, see E.3.

**Operation:**

The user supplies the Write Random Delete Input/Output command with the file-name or the address of the file control table, depending on the format chosen and the area which contains the key of the record to be deleted.

The command:

(1) Reads randomly the record to be deleted.

(2) Sets the key control words to "deleted".

(3) Writes back the data block on FASTRAND mass storage.

(4) Updates the entry for the last function in the internal file control table with W for write.

(5) Increments the number of records deleted by 1.

(6) Returns control to the user program with the status in the status-word.

### 2.4.7. Close Input/Output

**Application:**

The Close Input/Output command is used to close an input/output file. In the case of sequential processing, it is not necessary to have processed the file to the end.

**Format 1 (COBOL):**

```
ENTER ISFMS SUBROUTINE REFERENCING
37 file-name inform-area.
```

where:

37 is the function code for the Close Input/Output command.

File-name is the name of the file SELECTed in the INPUT-OUTPUT SECTION.

Inform-area is the area which contains information on what has been done to the file. For a description of this area, see 2.2.3.

**Format 2 (Assembler):**

CLIO FCT-*address, inform-area.*

where:

CLIO is the call to a PROC which generates the function code (37) for the Close Input/Output command. The PROC also generates the address and character length of each parameter.

FCT-address is the address of the user-built file control table.

Inform-area is the area which contains information on what has been done to the file (see 2.2.2).

**Prerequisite Functions:**

The file must have been opened for input/output.

**Entry Conditions:**

If COBOL is used, file-name must have been SELECTed and ASSIGNed in the INPUT-OUTPUT SECTION, and inform-area must be defined as 01 level in the WORKING-STORAGE SECTION, with 02 level-numbers for the nine words of information described in 2.2.3, each with PICTURE H9(10).

If the assembler is used, the address of the file control table is given and the appropriate nine-word area reserved for the inform-area.

**Exit Conditions:**

Upon return to the user program, the status-word contains either of the following:

- Fieldata zeros, if no error has occurred; or
- the error code (see E.3).

The inform-area contains statistics, as described in 2.2.3.

**Error Conditions:**

The run is aborted if either error (b) or (c) occurs (see E.2), or the nonfatal error codes 101, 230, or 248 may appear in the status-word. For an explanation of the error codes, see E.3.

**Operation:**

The user supplies the Close Input/Output command with the file-name or the address of the file control table, depending on the format chosen and the area which will contain the statistics.

The command:

(1) Checks the parameters.

(2) Reads the information block from FASTRAND mass storage.

(3) Updates the number of records in the file.

(4) Transfers the statistics to the user-specified inform-area.

(5) Writes the information block back on FASTRAND mass storage.

(6) Closes FASTRAND mass storage.

(7) Closes the index-file, if it had been opened.

(8) Returns control to the user program with the status in the status-word.

## 2.5. INFORM COMMAND

**Application:**

The inform command provides the user with the ability to obtain information about the status of the file during processing.

**Format 1 (COBOL):**

ENTER ISFMS SUBROUTINE REFERENCING
77 *file-name inform-area.*

where:

77 is the function code for the Inform command.

File-name is the name of the file SELECTed and the INPUT-OUTPUT SECTION.

Inform-area is the area which contains information on what has been done to the file. For a description of this area, see 2.2.3.

**Format 2 (Assembler):**

INFORM FCT-*address, inform-area.*

where:

INFORM is the call to a PROC which generates the function code (77) for the Inform command. The PROC also generates the address and character length of each parameter.

FCT-address is the address of the user-built file control table.

Inform-area is the area which contains information on what has been done to the file (see 2.2.3).

**Prerequisite Functions:**

The file must be in an open state.

**Entry Conditions:**

If COBOL is used, file-name must have been SELECTed and ASSIGNed in the INPUT-OUTPUT SECTION, and inform-area must be defined as 01 level in the WORKING-STORAGE SECTION, with 02 level-numbers for the nine words of information described in 2.2.3, each with PICTURE H9(10).

If the assembler is used, the address of the file control table is given and the appropriate nine-word area reserved for the inform-area.

**Exit Conditions:**

Upon return to the user program, the status-word contains either of the following:

- Fieldata zeros, if no error has occurred; or
- the error code (see E.3).

The inform-area contains statistics as described in 2.2.3.

**Error Conditions:**

The run is aborted if either error (b) or (c) occurs (see E.2), or the nonfatal error code 230 may appear in the status-word. For an explanation of this error code, see E.3.

**Operation:**

The user supplies the Inform command with the file-name or the address of the file control table, depending on the format chosen and the area which contains the statistics.

The command:

(1) Checks the parameters.

(2) Transfers the statistics which are stored in an internally built file control table into the user-specified inform-area.

(3) Returns control to the user program with the status in the status-word.

# APPENDIX A. COBOL INTERFACE (NON AMERICAN NATIONAL STANDARD COBOL)

## A.1. IDENTIFICATION DIVISION

This contains the standard information of every COBOL program.

## A.2. ENVIRONMENT DIVISION

### CONFIGURATION SECTION

This section contains the standard information of every COBOL program.

### INPUT-OUTPUT SECTION

In FILE-CONTROL, the user, besides assigning the card reader, printer, and so forth, has to SELECT his file-name and ASSIGN it to the external MASS-STORAGE file-name, that is, the file name which appears on the @ASG card. The user must specify that the ACCESS MODE is RANDOM and give a data-name to the ACTUAL KEY. This data-name is the one with which the blocks are written on MASS STORAGE, and must be defined with PICTURE H9(10) in the WORKING-STORAGE SECTION of the Data Division. The ACTUAL KEY should not be confused with the real record key.

## A.3. DATA DIVISION

### FILE SECTION

The FD for the file-name specifies:

BLOCK CONTAINS 1 RECORD 1 CONTROL WORD

LABEL RECORD IS FORM00

DATA RECORD is a data-name for the buffer area in which the ISFMS blocks are read to or written from. The buffer size is either 1/4, 1/2, or 1 track.

One software control word is subtracted from each of these lengths. This makes the lengths respectively 2682, 5370, and 10746 characters long, and it is given in the PICTURE for the buffer data-name.

**WORKING-STORAGE SECTION**

The ACTUAL KEY is described in level-number 77 as having PICTURE H9(10). Next on the 01 level are the areas, describing the record-area and the record-key-area. FILLER is used, when necessary, to make the lengths of the above areas a multiple of six (word boundaries). Following this, also on the 01 level, are the file-description, the status-word, and the inform-area. The file-description contains, on 02 levels, the following:

(1) number of records, with PICTURE H9(10) and the approriate VALUE;

(2) record-length, with PICTURE H9(10) and VALUE the length in characters (a multiple of six);

(3) maximum record-length, with PICTURE H9(10) and VALUE 0 for fixed length or length in characters (a multiple of six);

(4) record-key-length, with PICTURE H9(10) and VALUE the length in characters (a multiple of six);

(5) number of additional records, with PICTURE H9(10) and VALUE evaluated as in C.2, formula (9);

(6) independent overflow area, with PICTURE H9(10) and VALUE the number of records that are to be written in updating runs, as in C.2, formula (6).

The status-word contains, on the 02 level, the following:

(1) function code, with PICTURE X(3);

(2) error code, with PICUTRE X(3) which contains, on the 03 level, the following:

  (a) error-class, with PICTURE X; and

  (b) error number, with PICTURE X(2).

For a description of error codes (class and number), see Appendix E.

The inform-area contains, on the 02 level, the information passed with every Close command as well as the special command Inform. This area will be nine words (see 2.2.3) each with PICTURE H9(10).

The main storage index feature is optional and if used, it requires an area described on the 01 level. This area must be equal to (or greater than) the buffer area reserved for the DATA RECORD as it is described in the FILE SECTION.

## A.4. PROCEDURE DIVISION

ISFMS is used through COBOL by the ENTER verb in the following way:

ENTER ISFMS SUBROUTINE REFERENCING

Parameter-1 Parameter-2 Parameter-3 [Parameter-4]

where:

Parameters are as explained in Section 2. A sample program is given in Appendix D.

# APPENDIX B. ASSEMBLER INTERFACE

## B.1. GENERAL

This appendix describes the assembler/ISFMS interface. This interface is available through a set of assembler PROCs (see Section 2 for the individual commands or Appendix F for the exact calling sequence). The symbolics of these PROCs are maintained in file 2 of the ISFMS release tape. The assembler program must be collected with an ISFMS relocatable element assembled for American National Standard COBOL (Fieldata) interface (also available in file 2). Additionally, the American National Standard COBOL (Fieldata) file handler (CFH) must be available at collection time. To provide proper interface with ISFMS, the user must include the following in his assembler program:

(1) A file control table (FCT) for the file;

(2) EQUF statements specifying the lengths of parameters used in the PROC calls;

(3) Reserves for the buffer areas and the parameters used in the PROC calls.

The above information corresponds to the information automatically generated by the data division of COBOL.

## B.2. FILE CONTROL TABLE (FCT)

A FCT is required by the CFH. The following PROC generates and initializes the CFH FCT.

Label FCT 'file-name',buffer-name,buffer-length,actual-key-name,main-storage-index-area,mass-storage-flag.

where:

Label is the name given to the FCT and becomes the first subfield of all commands issued to ISFMS for file-name.

FCT is the name of the PROC.

File-name is a 12-character literal which is identical (left justified, space filled) to the file-name specified in the @ASG control statement.

Buffer-name is the label attached to the user-specified reserve area for the ISFMS I/O buffer. This reserve must be one of the following, depending upon desired buffer length:

```
Label RES  448.   1/4 track
Label RES  896.   1/2 track
Label RES 1792.     1 track
```

Buffer-length is the number of words specified in the above RES.

Actual-key-name is the label assigned to the location designated to contain the address of the ACTUAL-KEY. Thus, actual-key-name must contain the label of the location designated to contain the ACTUAL-KEY. This initialization is a user responsibility.

For example, if ACTUAL is the label to be placed in the FCT calling sequence and ACTKEY is location of the ACTUAL-KEY, then:

```
ACTUAL + ACTKEY.
ACTKEY + 0.
```

The initialization of ACTKEY is an ISFMS responsibility.

The last two fields in the previous proc are optional and if values are supplied they must be as follows:

(1) The first additional field, if supplied, must contain the label of the user reserved area where ISFMS can process index blocks. If the user supplies a reserved area for processing index blocks, it is likewise his responsibility to reserve area equal to (or greater than) the user supplied buffer.

(2) The second additional field, if supplied, should contain a "1". A "1" in this field informs ISFMS and CFH that this ISFMS file should start at a logical increment of 112 words. This option provides more efficient processing for ISFMS files which reside on disc.

*NOTE:*

*Most users will probably want their disc packs prepped at 112 words.*

## B.3. EQUF STATEMENTS

The following EQUF statements provide required information on parameter lengths; all lengths must be defined as follows:

```
L1(1)   EQUF   length.  Block length in characters
  .                     length = 2688 if 1/4 track
  .                     length = 5376 if 1/2 track
  .                     length = 10752 if 1 track
L1(2)   EQUF   6.       Status word length (fixed)
L1(3)   EQUF   36.      File description length (fixed)
L2(1)   EQUF   length.  See L1(1)
L2(2)   EQUF   length.  Maximum record length in characters
L2(3)   EQUF   length.  Record-key length in characters
L3(1)   EQUF   length.  See L1(1)
```

```
L3(2)   EQUF  54.      Inform length (fixed)
L3(3)   EQUF  length.  Record-key length in characters
L4(1)   EQUF  length.  See L1(1)
L4(2)   EQUF  length.  Record-key length in characters
```

## B.4. RESERVE WORDS

Two of the necessary reserve areas have already been discussed; they are for the I/O buffer and the ACTUAL-KEY location and contents. In addition, the user must reserve sufficient space for the maximum record size and for the symbolic record-key. The labels placed on these reserve areas correspond to those used in the individual commands.

For example:

```
RECORD-AREA       RES   value.   maximum size of record
RECORD-KEY-AREA   RES   value.   maximum size of key
```

Finally, the user must establish a status word location, a file description, and an inform area. These have the following format:

For the status word:

```
Label + value,
```

where value equals $60_8$, $61_8$, or $62_8$, depending upon where the index is to be positioned (see Sections 2.2.1 and 2.2.3, OPEN commands).

For the file description (see the respective OPEN commands – Sections 2.2.1 and 2.2.3 – for details):

```
Label   +value.   Maximum records
        +value.   Record length in characters
        +value.   Maximum length in characters if records area variable in length
        +value.   Record-key length in characters
        +value.   Overflow records in block
        +value.   Overflow records in overflow
```

For the inform-area:

```
Label     RES 9.   inform results.
```

The contents of these nine words are initialized upon a CLOSE or INFORM command (see Section 2.2.3).

# APPENDIX C. OPTIMIZATION PROCEDURES

## C.1. GENERAL

The following paragraphs give formulas used in building up a file.

## C.2. BLOCKS

To calculate the number of words:

Conversion of characters to words:

$$\text{number of words} = \frac{\text{number of characters}}{6} \quad (1)$$

*NOTE:*

*If the remainder is not zero, the quotient must be rounded to the next integer.*

To calculate the number of records per block:

The number of records per block can be found by taking into consideration the record length and the record-key length:

$$\text{number of records} = \frac{\text{block length} - 2}{\text{record-key length} + \text{record length} + 2} \quad (2)$$

The block length is a quarter-track (447 words) or a half-track (895 words) or a full-track (1791 words); the words in parentheses are one word less than the actual length. This one word is a software-control-word.

The record-key must have fixed-length format. Its length is specified in characters and must be a multiple of six. The minimum key length is six characters. The maximum is 378 characters.

The record may have fixed-length format or variable-length format. Its length is specified in characters and must be a multiple of six. The minimum record length is six characters. The maximum, which is for one record per block, varies depending on the key length (these two together must equal the block length) from 10,344 characters, when the key length 378 characters, to 10,716 characters when the key length is six characters.

To calculate the number of data blocks:

The number of data blocks is given by the following formula:

$$\text{number of data blocks} = \frac{\text{number of records in file}}{(\text{number of records per block}) - (\text{number of overflow records per block})} \quad (3)$$

where number of records per block is found from (2), and number of overflow records per block is found from (8).

To calculate the number of record keys per index block:

The number of record keys per index block is determined by using the following formula:

$$\text{number of record keys per index block} = \frac{\text{block length in words} - 2}{\text{key length in words} + 2} \quad (4)$$

where the block length (in words) is given following (2).

To calculate the number of index blocks:

The number of index blocks is determined by using the following formula:

$$\text{number of index blocks} = \frac{\text{number of data blocks}}{(\text{number of record keys per index block}) - 1} + 3 \quad (5)$$

where number of data blocks is given by (3), and number of record keys per index block by (4).

Overflow records in the independent overflow area (IOF):

The processing speed of an indexed sequential file can be greatly increased by writing all overflow records inside the data blocks. This, however, may prove impractical. Often numerous additional records must be written in the same data block. If there is not enough space available, some may be placed in the data block, if possible, and the others in the independent overflow area. In general, the number of IOF records is the same as the number of new records which can be written in updating runs, that is:

$$\text{IOF blocks} = \frac{\text{IOF records}}{\text{number of records per block}} \quad (6)$$

where the number of records per block is found from (2).

If the IOF area is almost filled, the file should be reorganized.

Additional records inside the data blocks:

It is advantageous, when updating, to have the new records logically as close as possible to the already existing ones. The user should, at the time a file is created, reserve a certain number of spare records inside each data block. This reserved area permits the user to write new records in the data block during the updating run.

A uniform formula for determining the number of additional records inside a data block cannot be given. It depends on how the file is built and how the run records are distributed. However, the proportion of the number of data records to the number of records to be added in an updating run is the same proportion used for the number of data records per block to the number of additional records per block, that is:

$$\frac{\text{number of data records}}{\text{number of update records}} = \frac{\text{number of data records/block}}{\text{number of additional records/block}} \qquad (7)$$

from which is obtained:

$$\text{additional records per data block} = \frac{\text{update records x data records per block}}{\text{data records}} \qquad (8)$$

where update records are known by the user, the data records per block are known from (2), and data records are the records in the file. This will minimize the IOF area needed.

Additional Blocks:

Every ISFMS file contains the following additional blocks:

Label block: 1

Block number 0: 1 (In random files, block number 0 is not used by COBOL.) (9)

Information block for ISFMS: 1

Sentinel block: 1

## C.3. FILE

To calculate the size of a file:

The size of a file is determined by the number of data blocks, index blocks, independent overflow blocks, one label block, one unused block, one sentinel block, and one information block which is used by ISFMS.

The total number of blocks in the file is given by the formula:

$$\text{number of blocks in the file} = \text{data blocks} + \text{index blocks} + \text{IOF blocks} + 4 \text{ additional blocks} \quad (10)$$

where data blocks are given by (3), index blocks by (5), IOF blocks by (6), and additional blocks are as explained in (9).

To convert to tracks:

The following formula is used to convert the size of the file to tracks:

$$\text{number of tracks} = \text{number of blocks} \times \text{block size in tracks} \quad (11)$$

where number of blocks is given by (10) and block size in tracks is quarter-track, half-track, or full-track as explained following (2).

To convert to positions:

To convert the size of the file in positions, use the following formula:

$$\text{number of positions} = \frac{\text{number of tracks}}{64} \quad (12)$$

where number of tracks is given by (11).

*NOTE:*

*In all these calculations, the result must be rounded to the next highest integer.*

# APPENDIX D. SAMPLE PROGRAMS

## D.1. NON AMERICAN NATIONAL STANDARD COBOL

```
IDENTIFICATION DIVISION.
PROGRAM-ID.  ISFMS-TEST-0.
AUTHOR. J P JONES.
INSTALLATION. SPSP.
DATE-COMPILED.
REMARKS.  TEST PROGRAM FOR THE ISFMS.
    EXAMPLE 1.  OUTPUT FILE.  CREATE AN IS-FILE.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE COMPUTER.  UNIVAC-1108.
OBJECT COMPUTER.  UNIVAC-1108.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT NEW-FILE,
    ASSIGN TO MASS-STORAGE ASGRDFLNM,
    ACCESS MODE IS RANDOM,
    ACTUAL KEY IS KLEIDI.
    SELECT CARDS,
    ASSIGN TO CARD-READ-EIGHTY.
    SELECT RESULTS,
    ASSIGN TO PRINTER.

DATA DIVISION.
FILE SECTION.
FD NEW-FILE
   LABEL RECORD IS FORMOO
   BLOCK CONTAINS 1 RECORD 1 CONTROL WORD
   DATA RECORD IS MASTER-RECORD.
01 MASTER-RECORD                PIC X(2682).
FD CARDS
   LABEL RECORDS OMITTED
   DATA RECORD IS DATA-CARDS.
01 DATA CARDS.
   02 NUMBER                    PIC 9(2).
   02 FILLER                    PIC X(4).
   02 CHARS                     PIC X(6).
   02 FILLER                    PIC X(68).
```

```
FD RESULTS
   LABEL RECORDS OMITTED
   DATA RECORD IS PRINTOUT.
01 PRINTOUT                          PIC X(132).

WORKING-STORAGE SECTION.
77 KLEIDI                            PIC H9(10).
77 NOTHING                           PIC X VALUE IS SPACE.
01 ACTUAL-RECORD.
   02 ARL                            PIC A(500).
   02 FILLER                         PIC X(4).
01 RECORD-KEY.
   02 RKL                            PIC 9(56).
   02 FILLER                         PIC X(4).
01 FILE DESCRIPTION.
   02 NUMBER-OF-RECORDS              PIC H9(10) VALUE IS 30.
   02 RECORD-LENGTH                  PIC H9(10) VALUE IS 504.
   02 RECORD-LENGTH-MAX              PIC H9(10) VALUE IS 0.
   02 RECORD-KEY-LENGTH              PIC H9(10) VALUE IS 60.
   02 OVERFLOW-AREA                  PIC H9(10) VALUE is 1.
   02 I-O-F                          PIC H9(10) VALUE IS 5.
01 STATOUS.
   02 F-CODE                         PIC X(3).
   02 ERROR-CODE                     PIC 9(3).
      03 ERROR-CLASS                 PIC 9.
      03 ERROR-NUMBER                PIC 9(2).
01 INFORM.
   02 NUMBER-OF-BLOCKS               PIC H9(10).
   02 NUMBER-OF-INDX-BLOCKS          PIC H9(10).
   02 NUMBER-OF-OVERFL-BLOCKS        PIC H9(10).
   02 NUMBER-OF-RECORDS              PIC H9(10).
   02 NUMBER-OF-RECORDS-IN-IOF       PIC H9(10).
   02 NUMBER-OF-RECORDS-DELETD       PIC H9(10).
   02 NUMBER-OF-RECORDS-READ         PIC H9(10).
   02 NMBR-OF-REC-RED-FROM-IOF       PIC H9(10).
   02 NUMBER-OF-RECORDS-WRITEN       PIC H9(10).
01 INFORM-FL-DATA.
   02 NUMBER-OF-BLOCKS               PIC 9(6).
   02 NUMBER-OF-INDX-BLOCKS          PIC 9(6).
   02 NUMBER-OF-OVERFL-BLOCKS        PIC 9(6).
   02 NUMBER-OF-RECORDS              PIC 9(6).
   02 NUMBER-OF-RECORDS-IN-IOF       PIC 9(6).
   02 NUMBER-OF-RECORDS-DELETD       PIC 9(6).
   02 NUMBER-OF-RECORDS-READ         PIC 9(6).
   02 NMBR-OF-REC-RED-FROM-IOF       PIC 9(6).
   02 NUMBER-OF-RECORDS-WRITEN       PIC 9(6).
01 MAIN-STORAGE-INDEX-BUFF.          PIC X(2682).
```

```
PROCEDURE DIVISION.
START-TEST.
   OPEN INPUT CARDS.
   OPEN OUTPUT RESULTS.
   ENTER ISFMS SUBROUTINE REFERENCING
   05 NEW-FILE STATOUS MAIN-STORAGE-INDEX-BUFF.
   ENTER ISFMS SUBROUTINE REFERENCING
   20 NEW-FILE STATOUS FILE-DESCRIPTION.
   IF STATOUS NOT EQUAL TO '000000' GO TO END-RUN.
CARD-READ.
   READ CARDS AT END GO TO FILE-READY.
   WRITE PRINT-OUT FROM DATA-CARDS.
   MOVE CHARS TO ACTUAL-RECORD.
   MOVE NUMBER TO RKL.
   ENTER ISFMS SUBROUTINE REFERENCING
   24 NEW-FILE ACTUAL-RECORD RECORD-KEY.
   IF STATOUS NOT EQUAL TO '000000' GO TO END-RUN.
   ENTER ISFMS SUBROUTINE REFERENCING
   77 NEW-FILE INFORM.
   IF STATOUS NOT EQUAL TO '000000' GO TO END-RUN.
   MOVE CORR INFORM TO INFORM-FL-DATA.
   WRITE PRINTOUT FROM INFORM-FL-DATA.
   GO TO CARD READ.
FILE-READY.
   WRITE PRINT-OUT FROM NOTHING AFTER ADVANCING 10 LINES.
   ENTER ISFMS SUBROUTINE REFERENCING
   27 NEW-FILE INFORM RECORD-KEY.
   MOVE CORR INFORM TO INFORM-FL-DATA.
   WRITE PRINT-OUT FROM NOTHING AFTER ADVANCING 2 LINES.
   WRITE PRINT-OUT FROM INFORM-FL-DATA.
END-RUN.
   WRITE PRINT-OUT FROM STATOUS AFTER ADVANCING 2 LINES.
   CLOSE CARDS, RESULTS.
   STOP RUN.
```

## D.2. ASSEMBLER

The following sample assembler program creates 40 27-word records while open for OUTPUT. It then closes and reopens for INPUT-OUTPUT, reads records, modifies and rewrites records, deletes records, and inserts new records. Then it closes and reopens for INPUT, sequentially reading and printing the records. Finally it reads randomly, does an inform, closes, and terminates.

```
@ASM,LI     .TEST,.TEST
             AXR$
$(0)
CVTFD*       PROC     2
             L        A6,CVTFD(1,1).       ROUTINE WILL TRANSFER
             L        X1,(-1,1).           BINARY TO FD AN PUT THESE
             SZ       CVTFD(2,1).          TWO WORDS IN THE RECORD..
             SZ       A5
             L,U      R1,5
```

```
DIVIDE      DI,U     A5,10
            DSL      A6,6
            DSL      A5,36
            JGD      R1,DIVIDE
            S        A7,CVTFD(2,1),*X1
            JNZ      A5,DIVIDE-1
            DL       A5,CVTFD(2,1)
            DA       A5,(060606060606060606060606060)
            DS       A5,CVTFD(2,1)
            END

$(1),START
            L,U      X10,26.              LOAD X10 WITH 26
            L        A1,(0757575757575).  LOAD A1 WITH 75'S
SER         S        A1,ACTREC,X10.       STR A1 INTO ACTREC+X10
            JGD      X10,SER.             JP SER IF 0
            DL       A0,('XXX    OPEN ').  DBL LOAD A0+1
            DS       A0,ACTREC.           DBL STR THIS IN ACTREC
            DL       A0,('OUTPUT   XXX').  DBL LOAD A0+1
            DS       A0,ACTREC+2.         DBLSTR A0 + A1 INTO ACTREC+2
            LMJ      X11,PRINT1.          GO PRINT IT
            L,U      A11,1.               STR ALL WITH 1
            SZ       A10.                 MAKE IT 0
            S        A11,KEY.             STR A11 INTO KEY
            DS       A10,ACTREC.          DBLSTR A10+A11 INTO ACTREC
            DL       A12,('      RECORD').  DBL LOAD A12+13
            DS       A12,ACTREC+2.        DBLSTR A12+13+
            OPO      ADO,STATUX,ADDER.    OPEN OUTPUT
            TE       A0,('000000').       TEST TO SEE IF A0=00
            ER       ERR$.                NO ITS NOT
            L,U      R1,40.               LOAD R1 WITH 40
WRT         WRRO     ADO,ACTREC,KEY.      WRITE RANDOM
            TE       A0,('000000').       TEST TO SEE IF A0=00
            ER       ERR$.                NO ITS NOT
            A,U      A11,1.               ADD TO A11 EACH TIME THRU
            S        A11,KEY.             STR A11 INTO KEY
            S        A11,ACTREC+1.        STR A11 INTO ACTREC+1
            JGD      R1,WRT.              JP TO WRT,IF R1#0
            CLO      ADO,INFO,KEY.        CLOSE OUTPUT
            TE       A0,('000000').       TEST TO SEE IF A0=00
            ER       ERR$.                NO ITS NOT
            DL       A0,('XXX OPEN INP').  DBL LOAD A0+A1
            DS       A0,ACTREC.           DBL STR THIS IN ACTREC
            DL       A0,('UT/OUTPUTXXX').   DBL LOAD A0 + A1
            DS       A0,ACTREC+2.         DBLSTR A0 + A1 INTO ACTREC+2
            LMJ      X11,PRINT1.          GO PRINT IT
            L        A0,('000000').
            S        A0,STATUX.           SET STATUS WORD
            OPIO     ADO,STATUX.          OPEN I/O
            TE       A0,('000000').       TEST TO SEE IF A0=00
            ER       ERR$.                NO ITS NOT
            L,U      A11,1.               STR A11 WITH 1
            SZ       A10.                 MAKE IT 0
```

```
S       A11,KEY.              STR A11 INTO KEY
DS      A10,ACTREC.           DBLSTR A10+A11 INTO ACTREC
RDSIO   ADO,ACTREC,KEY.       READ SEQ I/O
TE      A0,('000000').        SEE IF IT IS NEXT ONE
ER      ERR$.                 NO ITS NOT
L,U     A1,11.                ENTER A1 WITH 11
S       A1,KEY.               STR IT IN KEY
DS      A0,ACTREC.            DBL STR A0+1 INTO ACTREC
RDRIO   ADO,ACTREC,KEY.       READ RANDOM I/O
TE      A0,('000000').        TEST TO SEE IF A0=00
ER      ERR$.                 NO ITS NOT
DL      A12,('      MODIFY'). DBL LOAD A12+A13
DS      A12,ACTREC+2.         DBL STR A12+13 INTO ACTREC+2
WRSIO   ADO,ACTREC,KEY.       WRITE SEQ I/O
TE      A0,('000000').        TEST TO SEE IF A0=00
ER      ERR$.                 NO ITS NOT
L,U     A1,11.                ENTER A1 WITH 11
S       A1,KEY.               STR IT IN KEY
DS      A0,ACTREC.            DBL STR A0+1 INTO ACTREC
LMJ     X11,PRINT.            GO PRINT IT
L,U     A1,23.                ENTER A1 WITH 23
S       A1,KEY.               STR IT IN KEY
DS      A0,ACTREC.            DBL STR A0+1 INTO ACTREC
RDRIO   ADO,ACTREC,KEY.       READ RANDOM I/O
TE      A0,('000000').        TEST TO SEE IF A0=00
ER      ERR$.                 NO ITS NOT
DL      A12,('      MODIFY'). DBL LOAD A12+A13
DS      A12,ACTREC+2.         DBL STR A12+13 INTO ACTREC+2
WRSIO   ADO,ACTREC,KEY.       WRITE SEQ I/O
TE      A0,('000000').        IS TEST EQUAL TO A0
ER      ERR$.                 NO ITS NOT
L,U     A1,23.                ENTER A1 WITH 23
S       A1,KEY.               STR IT IN KEY
SZ      A0.                   MAKE IT 0
DS      A0,ACTREC.            DBL STR A0+1 INTO ACTREC
DL      A12,('      MODIFY'). DBL LOAD A12+A13
DS      A12,ACTREC+2.         DBL STR A12+13 INTO ACTREC+2
LMJ     X11,PRINT.            GO PRINT IT
L,U     A1,55.                LOAD A1 WITH 55
S       A1,KEY.               STR IT IN KEY
SZ      A0.                   MAKE IT 0
DS      A0,ACTREC.            DBL STR A0+1 INTO ACTREC
DL      A12,('      INSERT'). DBL LOAD A12+13
DS      A12,ACTREC+2.         DBL STR A12+13 INTO ACTREC+2
WRRIO   ADO,ACTREC,KEY.       WRITE RANDOM I/O
TE      A0,('000000').        TEST TO SEE IF A0=00
ER      ERR$.                 NO ITS NOT
L,U     A1,55.                LOAD A1 WITH 55
S       A1,KEY.               STR IT IN KEY
SZ      A0.                   MAKE IT 0
DS      A0,ACTREC.            DBL STR A0+1 INTO ACTREC
DL      A12,('      INSERT'). DBL LOAD A12+13
```

```
                DS      A12,ACTREC+2.       DBL STR A12+13 INTO ACTREC+2
                LMJ     X11,PRINT.          GO PRINT IT
                L,U     A1,56.              LOAD A1 WITH 56
                S       A1,KEY.             STR IT IN KEY
                SZ      A0.                 MAKE IT 0
                DS      A0,ACTREC.          DBL STR A0+1 INTO ACTREC
                WRRIO   ADO,ACTREC,KEY.     WRITE RANDOM I/O
                TE      A0,('000000').      TEST TO SEE IF A0=00
                ER      ERR$.               NO ITS NOT
                L,U     A1,56.              LOAD A1 WITH 56
                S       A1,KEY.             STR IT IN KEY
                DS      A0,ACTREC.          DBL STR A0+1 INTO ACTREC
                DL      A12,('      INSERT'). DBL LOAD A12+13
                DS      A12,ACTREC+2.       DBL STR A12+13 INTO ACTREC+2
                LMJ     X11,PRINT.          GO PRINT IT
                L,U     A1,39.              LOAD A1 WITH 39
                S       A1,KEY.             STR IT IN KEY
                DS      A0,ACTREC.          DBL STR A0+1 INTO ACTREC
                DL      A12,('      DELETE'). DBL LOAD A12+13
                DS      A12,ACTREC+2.       DBL STR A12+13 INTO ACTREC+2
                WRRD    ADO,KEY.            WRITE RANDOM DELETE
                TE      A0,('000000').      TEST TO SEE IF A0=00
                ER      ERR$.               NO ITS NOT
                L,U     A1,39.              LOAD A1 WITH 39
                S       A1,KEY.             STR IT IN KEY
                DS      A0,ACTREC.          DBL STR A0+1 INTO ACTREC
                DL      A12,('      DELETE'). DBL LOAD A12+13
                DS      A12,ACTREC+2.       DBL STR A12+13 INTO ACTREC+2
                LMJ     X11,PRINT.          GO PRINT IT
                CLIO    ADO,INFO.           CLOSE I/O
                TE      A0,('000000').      TEST TO SEE IF A0=00
                ER      ERR$.               NO ITS NOT
                DL      A0,('XXX    OPEN '). DBL LOAD A0+1
                DS      A0,ACTREC.          DBL STR THIS IN ACTREC
                DL      A),('INPUT    XXX'). DBL LOAD A0+1
                DS      A0,ACTREC+2.        DBLSTR A0 + A1 INTO ACTREC+2
                LMJ     X11,PRINT1.         GO PRINT IT
                L       A0,('000000').
                S       A0,STATUX.          SET STATUS WORD
                OPI     ADO,STATUX.         OPEN INPUT
                TE      A0,('000000').      TEST TO SEE IF A0=00
                ER      ERR$.               NO ITS NOT
SERR            RDSI    ADO,ACTREC,KEY.     READ SEQ
                TE      A0,('000000').      SEE IF IT IS NEXT ONE
                J       EOF.                GO TO END OF FILE
                LMJ     X11,PRINT.          GO PRINT IT
                J       SERR.               JP BACK TO CHECK NEXT
EOF             TNE     A0,('+11350').      TEST FOR EOF
                J       RDNIO.              GO TO READ NEXT I/O
                ER      ERR$.               NO ITS NOT
PRINT           CVTFD   KEY ACTREC.  CVRT BIN TO FD THEN STR IN ACTREC.
```

```
PRINT1    L       A0,(0000204,ACTREC). SET PACKET FOR PRINT$
          ER      PRINT$.
          J       0,X11.                JP BACK TO 0+X11
RDNIO     L,U     A0,30.                LOAD A0 WITH 30
          S       A0,KEY.               STR IT IN KEY
          RDRI    AD0,ACTREC,KEY.       READ RANDOM
          TE      A0,('000000').        TEST TO SEE IF A0=00
          ER      ERR$.                 NO ITS NOT
          LMJ     X11,PRINT.            GO PRINT IT
          IFORM   AD0,INFO.             INFORM US TO WHAT HAPPENED
          TE      A0,('000000').        TEST TO SEE IF A0=00
          ER      ERR$.                 NO ITS NOT
          CLI     AD0,INFO.             CLOSE INPUT
          TE      A0,('000000').        TEST TO SEE IF A0=00
          ER      ERR$.                 NO ITS NOT
          ER      EXIT$.
L1(1)     EQUF    2688.      RECORD LENGTH IN CHARACTERS (1/4 TRACK)
L1(2)     EQUF    6.         STATUS LENGTH IN CHARACTERS (1 WORD)
L1(3)     EQUF    36.        FILE DESCRIPTION IN CHARACTERS (6 WORDS)
L2(1)     EQUF    2688.      RECORD LENGTH IN CHARACTERS (1/4 TRACK)
L2(2)     EQUF    162.       DATA RECORD IN CHARACTERS (27x6)
L2(3)     EQUF    6.         RECORD KEY LENGTH IN CHARACTERS (1 WORD)
L3(1)     EQUF    2688.      RECORD LENGTH IN CHARACTERS (1/4 TRACK)
L3(2)     EQUF    54.        LENGTH OF INFORM IN CHARACTERS (9 WORDS)
L3(3)     EQUF    6.         RECORD KEY LENGTH IN CHARACTERS (1 WORD)
L4(1)     EQUF    2688.      RECORD LENGTH IN CHARACTERS (1/4 TRACK)
L4(2)     EQUF    6.         RECORD KEY LENGTH IN CHARACTERS (1 WORD)
ADDER     +       42.        NUMBER OF RECORDS MAXIMUM
          +       162.       DATA LENGTH IN CHARACTERS (27X6)
          +       0.         FIXED (0) LENGTH OR MAX. LENGTH
          +       6.         REC.-KEY-LENGTH IN CHAR. (1X6)
          +       1.         NBR. OVERFLOW REC. RESERVED DATA AREA
          +       1.         OVERFLOW REC. FOR THE WHOLE AREA
INFO      +       0.         NUMBER OF BLOCKS
          +       0.         NUMBER OF INDEX BLOCKS
          +       0.         NUMBER OF OVERFLOW BLOCKS (IOF)
          +       0.         NUMBER OF RECORDS
          +       0.         NBR. REC. INDEPENDENT OVERFLOW (IOF)
          +       0.         NUMBER OF RECORDS DELETED
          +       0.         NUMBER OF RECORDS READ
          +       0.         NUMBER OF REC. READ FROM IOF
          +       0.         NUMBER OF RECORDS WRITTEN
ACTUAL    +       ACTKEY.    ACTUAL KEY FOR THE RANDOM FILES
ACTKEY    +       0.         SECOND WORD FOR ACTUAL KEY
KEY       +       0.         ONE WORD RESERVED FOR KEY
BUFFAD    RES     448.       1/4 OF A TRACK AS STATED IN FCT CODE
BUFF2     RES     448.       MAIN STORAGE INDEX BUFFER-SIZE OF I/O BUFFER
ACTREC    RES     27.        SIZE OF THE RECORD
STATUX    +       0606060606060.     ONE WORD RESERVED FOR STATUS
$(30)
AD0       FCT     'SAM         ',BUFFAD,448,ACTUAL,BUFF2,1.
.         CONTAINS A LABEL (AD0) WHICH NAMES THE FILE CONTROL TABLE.
```

```
.          THE PROC NAME, FCT, WHICH GENERATES THE FILE CONTROL TABLE,
.          FILE NAME, BUFFER ADDRESS, BUFFER SIZE, ACTUAL KEY FOR THE
.          RANDOM FILES, ADDRESS OF MAIN STORAGE INDEX BUFFER IF ONE
.          EXISTS, A 1 IN THE LAST FIELD INFORMS ISFMS AND CFH THAT ISFMS
.          FILE SHOULD START AT A LOGICAL INCREMENT OF 112 WORDS.
.          NOTE - LAST TWO FIELDS ARE OPTIONAL AND CAN BE OMITTED.
           END START
```

## D.3. AMERICAN NATIONAL STANDARD COBOL (FIELDATA)

The following sample American National Standard COBOL (Fieldata) program creates 20 four-word records while open for OUTPUT. It closes and reopens for INPUT/OUTPUT, reads records, modifieds and rewrites records, deletes records, and inserts new records. It then closes and reopens for INPUT, sequentially reading and printing the records, after which it closes, and terminates.

```
IDENTIFICATION DIVISION.
PROGRAM-ID.  INDEXED-SEQUENTIAL.
REMARKS.  PROGRAM WILL CREATE FILE, THEN REVISE IT WITH UPDATES,
          DELETIONS, AND INSERTIONS.  THE FILE WILL THEN BE LISTED.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.  SELECT IS-FILE     ASSIGN TO MASS-STORAGE-112  FF
    ORGANIZATION IS INDEXED WITH INDEX-BUFFER
    ACCESS MODE IS RANDOM
    FILE-DESCRIPTION IS F-DES
    SYMBOLIC KEY IS KEY-IS.
    SELECT PRINT-FILE             ASSIGN TO PRINTER.

DATA-DIVISION.
FILE-SECTION.
FD  IS-FILE  LABEL RECORDS ARE STANDARD DATA RECORD IS IS-REC.
01  IS-REC                                PICTURE X(5376).
FD  PRINT-FILE
    LABEL RECORDS ARE OMITTED     DATA RECORD IS PRINT-LINE.
01  PRINT-LINE                            PICTURE X(30).
WORKING-STORAGE SECTION.
    77  KEY-IS                            PICTURE 9(6).
01  F-DES.
    05  NUMBER-RECORDS      VALUE 20      PICTURE 9(10)   COMPUTATIONAL.
    05  RECORD-LENGTH       VALUE 24      PICTURE 9(10)   COMPUTATIONAL.
    05  FILLER              VALUE 0       PICTURE 9(10)   COMPUTATIONAL.
    05  RECORD-KEY-SIZE     VALUE 6       PICTURE 9(10)   COMPUTATIONAL.
    05  RESERVE-OVERFLOW    VALUE 4       PICTURE 9(10)   COMPUTATIONAL.
    05  RESERVE-IOF         VALUE 20      PICTURE 9(10)   COMPUTATIONAL.
    05  STATUS-WD       VALUE SPACES      PICTURE X(6).
01  REC-1.
    05  FILLER VALUE 'LOGICAL RECORD'     PICTURE X(18).
    05  REC-NUMBER                        PICTURE 9(6).
PROCEDURE DIVISION.
START.  OPEN OUTPUT IS-FILE, PRINT-FILE.  MOVE ZERO TO KEY-IS.
```

```
LOOP-1.  ADD 10 TO KEY-IS.
    IF KEY-IS > 200 GO TO CLOSE-1.
    MOVE KEY-IS TO REC-NUMBER.
    WRITE IS-REC FROM REC-1 INVALID KEY STOP RUN.
    GO TO LOOP-1.
CLOSE-1.  CLOSE IS-FILE.
    MOVE  '000000' TO STATUS-WD.
    OPEN I-O IS-FILE.
    MOVE 210 TO KEY-IS.
LOOP-2.
    SUBTRACT 10 FROM KEY-IS.
    READ IS-FILE INTO REC-1 INVALID KEY STOP RUN.
    ADD 1000 TO REC-NUMBER.
    WRITE IS-REC FROM REC-1.       NOTE  ** RECORD IS UPDATED**.
    SUBTRACT 10 FROM KEY-IS.
    READ IS-FILE INTO REC-1 INVALID KEY STOP RUN.
    WRITE IS-REC FROM REC-1 FOR DELETION INVALID KEY STOP RUN.
    NOTE **RECORD IS DELETED**.
    SUBTRACT 5 FROM KEY-IS.
    READ IS-FILE INTO REC-1 INVALID KEY NEXT SENTENCE.
    MOVE KEY-IS TO REC-NUMBER.
    WRITE IS-REC FROM REC-1 INVALID KEY STOP RUN.
    NOTE **RECORD IS INSERTED**.
    SUBTRACT 5 FROM KEY-IS.  IF KEY-IS > 100 GO TO LOOP-2.
    MOVE ZERO TO KEY-IS.  CLOSE IS-FILE.

    MOVE '000002' TO STATUS-WD.
    OPEN INPUT IS-FILE.
LOOP-3.
    READ IS-FILE INTO REC-1 AT END
        CLOSE IS-FILE, PRINT-FILE, STOP RUN.
    WRITE PRINT-LINE FROM REC-1.
    GO TO LOOP-3.
```

**Results of executing sample program:**

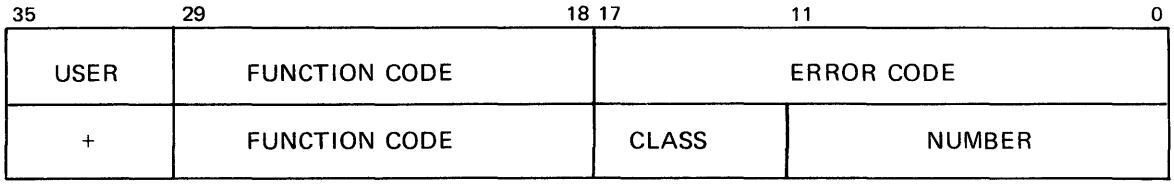
```
LOGICAL RECORD     000010
LOGICAL RECORD     000020
LOGICAL RECORD     000030
LOGICAL RECORD     000040
LOGICAL RECORD     000050
LOGICAL RECORD     000060
LOGICAL RECORD     000070
LOGICAL RECORD     000080
LOGICAL RECORD     000090
LOGICAL RECORD     000095
LOGICAL RECORD     001110
LOGICAL RECORD     000120
LOGICAL RECORD     000125
LOGICAL RECORD     001140
LOGICAL RECORD     000150
LOGICAL RECORD     000155
```

```
LOGICAL RECORD      001170
LOGICAL RECORD      000180
LOGICAL RECORD      000185
LOGICAL RECORD      001200
```

# APPENDIX E. ISFMS ERRORS

## E.1. GENERAL

The status word (defined in WORKING-STORAGE SECTION in COBOL, or a reserved word in the assembler) is Fieldata zero filled if no error has occurred and has the following format in Fieldata:




## E.2. FATAL ERRORS

When ISFMS encounters a fatal error condition, it will print an error message on primary output and then cause the run to be aborted. The ISFMS generated error messages and the fatal error conditions which caused them are as follows:

(a) "FILE ALREADY OPENED. FILE-NAME..."

cause: Attempting to open a file which is currently open.

(b) "FILE NOT OPENED. FILE-NAME..."

cause: Attempting to refer to a file which is not open.

(c) "BAD ISFMS FUNCTION-CODE. CODE IS "

cause: Attempting a function which is invalid.

(d) "INCORRECT ISFMS STATUS-WORD. FILE-NAME..."

cause: The status-word is improperly defined.

(e) "10 ISFMS FILES ALREADY OPEN. FILE-NAME..."

cause: Attempting to have more than 10 ISFMS files open at the same time.

(f) "FILE NOT ASSIGNED. FILE-NAME..."

cause: Attempting to open a file which was not assigned; i.e., the required @ASG card was missing.

The FILE-NAME... as shown in the above messages will be the name of the ISFMS file as it was supplied by the worker program.

## E.3. NONFATAL ERRORS

Table E–1 lists the ISFMS nonfatal errors. If the error code, which indicates the type of error, begins with a 1 or a 2, control is returned to the user program and recovery from the error is usually impossible. If the error code begins with a 3, control is returned to the user program and recovery from the error is possible.

## E.4. ERROR CODE 101 UNDER NON AMERICAN NATIONAL STANDARD COBOL

ISFMS, when run under non-American National Standard COBOL, interfaces with the non-American National Standard COBOL ITEM HANDLER to create output files. This routine attempts to prevent the destruction of germane data by testing for a meaningful ACTUAL KEY in a mass storage area prior to writing into that area (see *UNIVAC 1106/1108 COBOL EXEC II and EXEC 8 Supplementary Reference, UP-7626* (current version)). The ACTUAL KEY written by the non-American National Standard COBOL ITEM HANDLER contains the first two characters of the file qualifier as specified on the @ASG card or as implied by the project ID specification on the @RUN card. When the mass storage area containing data from the creation of an ISFMS output file is released and a subsequent attempt is made to recreate that output file, the same physical area of mass storage may be selected to receive output a second time. If this occurs, error number 101 occurs during the writing and/or closing of that file.

If the first two characters of the qualifier used for the output file are changed between successive runs which create and recreate any given output file, this problem cannot occur.

*NOTE:*

*The above problem cannot arise when ISFMS is employed from American National Standard COBOL or through an assembler interface.*

*Table E–1. ISFMS Nonfatal Errors (Part 1 of 2)*

| ERROR CODE | ERROR |
|---|---|
| 101 | One of two conditions can cause this error:<br>(1) ACTUAL KEY is larger than the end-of-file key and is not equal to next key (most probable cause is insufficient maximum size specification on @ASG card).<br>(2) An attempt was made to create an output file under non American National Standard COBOL such that the physical area of mass storage referenced corresponded to an area which contained data from a previous attempt. See Section E.4. |
| 102 | A program contingency occurred that concerns either the user's manipulation of the file or a physical malfunction has occurred on the FH-432 drum. |
| 210 | Insufficient space is available on FASTRAND mass storage to initialize a new index block. |
| 211 | While updating the index, an attempt was made to write an index block at a level higher than index level 7. |
| 212 | An attempt was made to write an IOF record and no more IOF blocks are available. |
| 213 | An attempt was made to assign an index file to magnetic drum or FASTRAND mass storage and the request cannot be honored because no space is available. |
| 230 | One of the following may be wrong with parameter 3 of the ISFMS instruction:<br>(1) Incoming record is greater in size than that defined in the tables.<br>(2) Check if status word is equal to one.<br>(3) Check if inform area contains anything other than nine words. |
| 240 | One of the following may be wrong with parameter 4 of the ISFMS instruction:<br>(1) Incoming record key is not equal to that defined in the tables.<br>(2) Check if file description contains anything other than six words. |
| 241 | Block size is not quarter-track, half-track, or full track. |
| 243 | Record length (RL), record key length (RKL), or maximum record length (RLM) is incorrect. |
| 244 | Number of additional records per data block is equal to or greater than the number of records per block or is set negative. |
| 245 | Number of independent overflow records has not been specified. |
| 246 | Number of records in the file has not been specified. |
| 247 | File needs more FASTRAND mass storage tracks than have been assigned. |
| 248 | When the file mode and the function to be performed were checked, a mismatch was found. Function codes for input files should start with 1, the mode for that file being 1; function codes for output files should start with 2, the mode for the file being 2; function codes for input/output files should start with 3, the mode for that file being 3. The inform function, starting with 7, has no mode. The main storage index function, starting with 0, also has no mode. |
| 350 | Sentinel record has been read. |
| 360 | When an attempt to read the next record is made, it is found that the sentinel record has already been read. |

*Table E–1. ISFMS Nonfatal Errors (Part 2 of 2)*

| ERROR CODE | ERROR |
|---|---|
| 361 | Unsuccessful search for a record in the data block. |
| 362 | During the creation of an output file, the key of the record presented to ISFMS is not ascending. |
| 363 | The record could not be rewritten because the previous command was not a Read Random or a Read Sequential, or because the old record and the modified record do not match. |
| 364 | An attempt is made to write a record that already exists into an input/output file. |
| 365 | Main storage index area supplied is too small and therefore the main storage index feature is unavailable. |

# APPENDIX F. SUMMARY OF COMMANDS

## F.1. COBOL COMMANDS

OPEN INPUT

ENTER ISFMS SUBROUTINE REFERENCING

10 *file-name status-word.*

The status-word has a meaning during the whole processing of the file (see Appendix E).

READ SEQUENTIAL INPUT

ENTER ISFMS SUBROUTINE REFERENCING

11 *file-name record-area record-key-area.*

READ RANDOM INPUT

ENTER ISFMS SUBROUTINE REFERENCING

12 *file-name record-area record-key-area.*

CLOSE INPUT

ENTER ISFMS SUBROUTINE REFERENCING

17 *file-name inform-area.*

For Inform command, see 2.2.3.

OPEN OUTPUT

ENTER ISFMS SUBROUTINE REFERENCING

20 *file-name status-word file-description.* (6 words, all H9(10))

(1) Number of records (estimate)

(2) Record length:

Fixed: Length in characters

Variable: Average (estimate) record length in characters

(3) Maximum record length:

Fixed: 0

Variable: Maximum record length in characters

(4) Record-key-length in characters

(5) Number of additional records to be inserted in data area

(6) Overflow records for the whole area

WRITE RANDOM OUTPUT

ENTER ISFMS SUBROUTINE REFERENCING

24 *file-name record-area record-key-area.*

For Inform command, see 2.2.3.

CLOSE OUTPUT

ENTER ISFMS SUBROUTINE REFERENCING

27 *file-name inform-area record-key-area.*

OPEN INPUT/OUTPUT

ENTER ISFMS SUBROUTINE REFERENCING

30 *file-name status-word.*

READ SEQUENTIAL INPUT/OUTPUT

ENTER ISFMS SUBROUTINE REFERENCING

31 *file-name record-area record-key-area.*

READ RANDOM INPUT/OUTPUT

ENTER ISFMS SUBROUTINE REFERENCING

32 *file-name record-area record-key-area.*

WRITE SEQUENTIAL INPUT/OUTPUT

ENTER ISFMS SUBROUTINE REFERENCING

33 *file-name record-area record-key-area.*

WRITE RANDOM INPUT OUTPUT

ENTER ISFMS SUBROUTINE REFERENCING

34 *file-name record-area record-key-area.*

WRITE RANDOM DELETE INPUT/OUTPUT

ENTER ISFMS SUBROUTINE REFERENCING

36 *file-name record-key-area.*

CLOSE INPUT/OUTPUT

ENTER ISFMS SUBROUTINE REFERENCING

37 *file-name inform-area.*

For Inform command, see 2.2.3.

INFORM

ENTER ISFMS SUBROUTINE REFERENCING

77 *file-name inform-area.*

This function may be performed only when the file is open. For Inform command, see 2.2.3.

USE MAIN STORAGE INDEX

ENTER ISFMS SUBROUTINE REFERENCING

05 *file-name status-word main-storage-index-area.*

This function is described in Appendix H.

## F.2. ASSEMBLER COMMANDS

OPEN INPUT

OPI FCT-*address, status-word.*

READ SEQUENTIAL INPUT

RDSI FCT-*address, record-area, record-key-area.*

READ RANDOM INPUT
RDRI FCT-*address, record-area, record-key-area.*

CLOSE INPUT
CLI FCT-*address, inform-area.*

OPEN OUTPUT
OPO FCT-*address, status-word, file description.*

WRITE RANDOM OUTPUT
WRRO FCT-*address, record-area, record-key-area.*

CLOSE OUTPUT
CLO FCT-*address, inform-area, record-key-area.*

OPEN INPUT/OUTPUT
OPIO FCT-*address, status-word.*

READ SEQUENTIAL INPUT/OUTPUT
RDSIO FCT-*address, record-area, record-key-area.*

READ RANDOM INPUT/OUTPUT
RDRIO FCT-*address, record-area, record-key-area.*

WRITE SEQUENTIAL INPUT/OUTPUT
WRSIO FCT-*address, record-area, record-key-area.*

WRITE RANDOM INPUT/OUTPUT
WRRIO FCT-*address, record-area, record-key-area.*

WRITE RANDOM DELETE INPUT/OUTPUT
WRRD FCT-*address, record-key-area.*

CLOSE INPUT/OUTPUT
CLIO FCT-*address, inform-area.*

INFORM
INFORM FCT-*address, inform-area.*

This function may be performed only when the file is open.

## F.3. AMERICAN NATIONAL STANDARD COBOL (FIELDATA) COMMANDS

The following is a list of the American National Standard COBOL (Fieldata) commands and the corresponding ISFMS function codes as used in a non-American National Standard COBOL program.

+10 OPEN INPUT *file-name-1*
+11 READ *file-name-1* INTO *record-name-2* AT END (input file)
+12 READ *file-name-1* INTO *record-name-2* INVALID KEY (input file)
+17 CLOSE *file-name-1* (which was an input file)
+20 OPEN OUTPUT *file-name-1*
+24 WRITE *record-name-1* FROM *record-name-2* INVALID KEY
+27 CLOSE *file-name-1* (which was an output file)
+30 OPEN INPUT-OUTPUT *file-name-1*
+31 READ *file-name-1* INTO *record-name-2* AT END (I-O file)
+32 READ *file-name-1* INTO *record-name-2* INVALID KEY (I-O file)
+33 WRITE *record-name-1* FROM *record-name-2* (I-O file)
+34 WRITE *record-name-1* FROM *record-name-2* INVALID KEY (I-O file)
+36 WRITE *record-name-1* FROM *record-name-2* FOR DELETION INVALID KEY
+37 CLOSE *file-name-1* (which was an I-O file)

# APPENDIX G. ISFMS RELEASE TAPE

## G.1. GENERAL

ISFMS is now available to operate under both non-American National Standard COBOL and American National Standard COBOL (Fieldata). File 1 of the ISFMS tape contains a relocatable element named ISFMS. This relocatable element must be used whenever non-American National Standard COBOL is used because it contains an interface to the COBOL I/O package L$CBIO.

Similarly, File 2 of the ISFMS tape contains a relocatable element named ISFMS. This element must be used whenever American National Standard COBOL (Fieldata) is used because the element contains an interface to the American National Standard COBOL (Fieldata) file handler (CFH). The American National Standard COBOL (Fieldata) relocatable library (which contains CFH and an interface element, ISFMSC) must be available. Because ISFMS itself is also included in the American National Standard COBOL (Fieldata) relocatable library, American National Standard COBOL (Fieldata) may be used without reference to the ISFMS release tape.

File 2 of the ISFMS tape also contains the assembler PROC symbolics. These are assembler PROCs which generate a CFH file control table and a COBOL-like calling sequence for ISFMS commands. To use these PROCs, the American National Standard COBOL (Fieldata) ISFMS relocatable element (in this file) and the American National Standard COBOL (Fieldata) relocatable library must be available at collection time. These PROCs are not used with L$CBIO.

File 3 of the ISFMS tape contains a symbolic element of ISFMS, named ISFMS. There is only one symbolic element of ISFMS. This element contains an assembler EQU directive which forces assembly to occur with either an L$CBIO or a CFH interface. The symbolic is initially prepared with

```
CFH EQU 0
```

which forces assembly for a non-American National Standard COBOL L$CBIO interface. If an assembly of an American National Standard COBOL (Fieldata) CFH interface element is necessary, the above instruction must be replaced by:

```
CFH EQU 1
```

# APPENDIX H. MAIN STORAGE INDEX FEATURE

## H.1. GENERAL

The ISFMS has been expanded, allowing the user to maintain an index block in main storage. The input and I/O files will maintain the highest level index block in main storage and therefore he can access records randomly with one less mass storage read instruction. The output file will maintain the lowest level index block in main storage as it is being filled. Therefore, the mass storage requests will be reduced approximately 60% when the file is initially created.

## H.2. AMERICAN NATIONAL STANDARD COBOL (FIELDATA)

The American National Standard COBOL (Fieldata) user can receive the main storage index feature by modifying the SELECT statement in the FILE-CONTROL paragraph. The modification is to change "ORGANIZATION IS INDEXED" to read "ORGANIZATION IS INDEXED WITH INDEX-BUFFER". This change causes the American National Standard COBOL (Fieldata) compiler to generate a buffer of the appropriate size (¼, ½, or 1 track) and store the address in the FCT. ISFMS will interrogate the COBOL FCT for the main storage index buffer information and proceed accordingly.

## H.3. NON AMERICAN NATIONAL STANDARD COBOL

The non American National Standard COBOL user can receive the main storage index feature by using a new command and supplying ISFMS with a reserve area of the appropriate size (¼, ½, or 1 track) in working storage. The command is USE MAIN STORAGE INDEX and it must be executed prior to opening an ISFMS file. The function of this command is to inform ISFMS that a main storage index area and its location is being provided by the user. Once the command has been issued, the user receives the main storage index feature, regardless of the number of times the file is opened and closed during the execution of the task.

The format of the USE MAIN STORAGE INDEX command is as follows:

ENTER ISFMS SUBROUTINE REFERENCING
05 *file-name status-word main-storage-index-area.*

Where:

05 is the function code for the USE MAIN STORAGE INDEX command.

File-name is the name of the file which had been, or is being created as an output file and which has been SELECTed in the INPUT-OUTPUT SECTION.

Status-word is the word which contains information concerning the success or failure of the function. A status code of "0" signifies it was successful. A status code of "365" signifies the user supplied area was too small and therefore the user will not receive the main storage index feature.

Main-storage-index-area is the user program area assigned by the user for ISFMS to process the file's index blocks.

## H.4. ASM

The ASM user can receive the main storage index feature by the use of an additional field on the ISFMS proc, "FCT", and supplying ISFMS with an appropriate size buffer area. The format of the ISFMS proc "FCT" as illustrated in Appendix D has been modified as follows:

```
ADO   FCT   'SAM        ',BUFFAD,448,ACTUAL,BUFF2,1.
```

The last two fields in the previous proc are optional and if values are supplied they must be as follows:

(1) The first additional field, if supplied, must contain the label of the user reserved area where ISFMS can process index blocks. If the user supplies a reserved area for processing index blocks, it is likewise his responsibility to reserve area equal to (or greater than) the user supplied buffer.

(2) The second additional field, if supplied, should contain a "1". A "1" in this field informs ISFMS and CFH that this ISFMS file should start at a logical increment of 112 words. This option provides more efficient processing for ISFMS files which reside on disc.

*NOTE:*

*Most users will probably want their disc packs prepped at 112 words.*

# APPENDIX I. AMERICAN NATIONAL STANDARD COBOL (FIELDATA) INTERFACE

## I.1. GENERAL

This appendix describes the American National Standard COBOL (Fieldata) ISFMS interface. The American National Standard COBOL (Fieldata) program links up with ISFMS by means of the interface element ISFMSC. The American National Standard COBOL (Fieldata) compiler will determine if it should reference the interface element ISFMSC while it is interrogating the FILE-CONTROL paragraph.

The information contained in this appendix is basically a condensed version of the *UNIVAC 1100 Series American National Standard COBOL (Fieldata) Programmer Reference, UP-7845* (current version) Section 11. This appendix is designed to provide the American National Standard COBOL (Fieldata) user a working knowledge of the minor variations in the previously described non American National Standard COBOL ISFMS interface.

## I.2. IDENTIFICATION DIVISION

This contains the standard information of every American National Standard COBOL (Fieldata) program.

## I.3. ENVIRONMENT DIVISION

Within the FILE-CONTROL paragraph, the SELECT statement for an indexed sequential file should contain:

SELECT *file-name*
ASSIGN TO MASS-STORAGE [–112] *file-system-name*
ACCESS MODE IS RANDOM
SYMBOLIC KEY IS *data-name-1*
ORGANIZATION IS INDEXED [WITH INDEX-BUFFER]
FILE-DESCRIPTION IS *data-name-2.*

The file-system-name is the name appearing on the @ASG control statement. The name of the record key, which appears in WORKING-STORAGE, is given as data-name-1. Descriptive information about the file is presented in a WORKING-STORAGE record named data-name-2.

### –112 Option

Use of the –112 option causes the first COBOL data block (i.e., the ISFMS information block) to be written at mass storage location 112. This feature was added to American National Standard COBOL (Fieldata) since mass storage files can reside on disc and the packs may be prepped at 28, 56 or 112 words. If a disc pack is prepped at 112 words, this option causes the program's execution to be more efficient since all the ISFMS generated blocks will start on a pack's prepped record boundaries.

### INDEX-BUFFER Option

Use of the INDEX-BUFFER option will cause the file table for this file to be generated large enough so that it can contain the highest level index block and hence save I/O references when using the file.

## I.4. DATA DIVISION

### FILE SECTION

Each indexed sequential file is described in an FD entry as follows:

FD *file-name*

LABEL RECORDS ARE STANDARD
DATA RECORD IS *data-name-1.*

01 *data-name-1* PICTURE X (block-size).

The block-size is either 1/4, 1/2, or 1 track. Thus, block size must be 2688, 5376, or 10752. Block-size depends on the record size and also the degree of discrimination of indexing desired. Smaller blocks produce a fine indexing, but they may also lead to additional index levels with a concomitant price in processing time. If the block size is not specified correctly 1/2 track blocks will be generated.

### WORKING-STORAGE SECTION

The record(s) and the record key (as named in the SELECT statement SYMBOLIC KEY clause) are given here. The record size plus record key length may not exceed block-size minus 24. The record key must begin on a word boundary. Also given is the file description record (named in the FILE-DESCRIPTION clause), which contains 6 items each with PICTURE 9(10) and USAGE COMPUTATIONAL, and 2 items with PICTURE X(3). Records are to be assigned level 01, with contained items at level 02 or a higher number. An optional record which may also be given is the information area record (see I.5 for CLOSE verb).

The File Description record contains the following:

01 *data-name-1.*

02 *data-name-2* PICTURE 9(10) COMPUTATIONAL
VALUE *number-of-records-maximum* .

02 *data-name-3* PICTURE 9(10) COMPUTATIONAL
VALUE *record-length* (must be a multiple of six characters).

02 *data-name-4* PICTURE 9(10) COMPUTATIONAL
VALUE *maximum-or-zero* (must be a multiple of six characters).

02 *data-name-5* PICTURE 9(10) COMPUTATIONAL
VALUE *record-key-length* (must be a multiple of six characters).

02 *data-name-6* PICTURE 9(10) COMPUTATIONAL
VALUE *overflow-area-size.*

02 *data-name-7* PICTURE 9(10) COMPUTATIONAL
VALUE *overflow-area-size.*

02 *data-name-8.*
*03 data-name-9* PICTURE X(3).
*03 data-name-10* PICTURE X(3).

If only records of a specific size are to be written into a file, record-length gives the length of those records. If variable-length records (records of different lengths) are to be written, record-length is the average (estimate) record length. For fixed-length records, maximum-or-zero is zero; for variable-length records, maximum-or-zero is the maximum record length.

The overflow-area-size should be computed as:

(anticipated number of update records) ((block size)/(record size +
record key length + 12))/(number of records)

The independent-overflow-area-size is a number of records sufficient to accommodate update records which do not fit in the data block overflow area. Reorganization takes place as the number of such records becomes significant, and before the independent overflow area (IOF) capacity is exceeded. In deciding on an independent overflow area size, the user should consider the possibility that insertion update records are concentrated in a small range of his file rather than being uniformly distributed over the file. In such a case, the overflow area in the normal data block is quickly filled up, and independent overflow is used for all remaining insertions in that area.

The last item in the File Description record is used for error analysis. Following an INVALID KEY or a USE AFTER ERROR PROCEDURE exit from a READ or WRITE verb, data-name-10 may be interrogated for more specific information as to the problem (see I.6 error codes). The value in data-name-9 corresponds to the operation which produced the error.

The final record type, which may be included in WORKING-STORAGE, is the Information Area record. When used, it is named in the USING clause of a CLOSE verb. The record contains nine elementary items each with PICTURE 9(10) and USAGE COMPUTATIONAL.

The contents of these nine words are:

Number of blocks
Number of index blocks
Number of overflow blocks (IOF)
Number of records
Number of records in Independent Overflow area (IOF)
Number of records deleted
Number of records read
Number of records read from IOF
Number of records written

### COMMON-STORAGE

If an indexed sequential file is to be referenced from more than one sequential, independently compiled program, the data necessary in WORKING-STORAGE should be placed in COMMON-STORAGE. This will allow the same key area to be used for all ISFMS references.

## I.5. PROCEDURE DIVISION

The input/output verbs generally have the same significance as with non-indexed-sequential files. Specific differences are the requirement for INTO on READ, FROM on WRITE, FOR DELETION option to the WRITE verb, and the USING option on the CLOSE verb.

### OPEN

**Format:**

OPEN { INPUT / OUTPUT / I-O } *file-name-1 [, file-name-2]... .*

**Description:**

The same statement may be used to open both indexed sequential and other files. Indexed sequential files are always on mass storage, and must not have REWIND or REVERSED options.

### READ

**Format 1:**

READ *file-name* RECORD INTO *record-name* AT END *imperative-statement.*

**Format 2:**

READ *file-name* RECORD INTO *record-name* INVALID KEY *imperative-statement.*

**Description:**

The INTO record-name is required. Record-name names a data area in WORKING-STORAGE, the size of which must be as stated in the File Description record. Record-name may be qualified but not subscripted.

AT END is used for a sequential read. The record key (named data-name, in the SYMBOLIC KEY clause of the SELECT statement) does not need to be initialized if sequential reading is to begin at the first record of file-name. If sequential reading is to begin at some other record, a random read is given to initiate input at the desired point. As each record is conveyed to record-name, the record key of that record is moved to data-name.

INVALID KEY is used for random input. The record key (data-name) must be initialized to the value of the key of the desired record. If the record cannot be found, the INVALID KEY imperative statement is executed.

### WRITE

**Format:**

WRITE *record-name-1* FROM *record-name-2* [FOR DELETION] [INVALID KEY *imperative-statement*] .

**Description:**

Record-name-1 is the name of the record as given in the FILE SECTION. Record-name-2 names the record area in WORKING-STORAGE; it may be qualified, but not subscripted.

INVALID KEY is used for a random write. The value of the record key must be initialized prior to the write. The INVALID KEY imperative statement is executed if the key matches that of a record already in the file, except for a write FOR DELETION; on a write FOR DELETION, the INVALID KEY imperative statement (which must be supplied) is executed if the record key does not match that of a record in the file. The random write is used for initial file creation in a file opened for output. The record keys for successive writes must be in ascending sequence on a file creation. The random write is also used to insert new records in a file opened for input-output. A random write FOR DELETION applies to input-output files, and is used to delete a record. Random writes on input-output files are preceded by a read operation for the same record key.

A sequential write does not use the INVALID KEY clause. It is used on an input-output file only, and is employed to rewrite a record which has first been read in and updated. The read which precedes a sequential write may be either sequential or random.

### CLOSE

**Format:**

CLOSE *file-name* [USING *record-name*] .

**Description:**

The USING option is a special extension for closing indexed sequential files. If used, record-name names a record in WORKING-STORAGE which communicates certain information about the contents of the file.

## I.6. ERROR CONDITIONS

AT END, INVALID KEY, and USE AFTER ERROR PROCEDURE exits are associated with error codes which are found in the last item for the File Description record.

These error codes, which may be analyzed by the program, are as follows.

For INVALID KEY or AT END:

350 – Normal AT END exit following end of sequential file reading.
361 – Unsuccessful search for a record in the data block.
362 – During the creation of an output file, the key of the record presented to ISFMS is not ascending.
363 – The record cannot be rewritten because the previous command was not a Read Random or a Read Sequential, or because the old record and the modified record do not match.
364 – An attempt is made to write a record that already exists into an input/output file.

For USE AFTER ERROR PROCEDURE:

102 – A program contingency occurred that concerns either the user's manipulation of the file or a physical malfunction has occurred on the FH-432 drum.
230 – Incoming record is greater in size than that defined in File Description record.
241 – Block size is not quarter-track, half-track, or full track.
243 – Record length, record key length, or maximum record length is incorrect.

The value in data-name-9 of the file description record contains a value corresponding to the operation which produced the error code.

These values are as follows:

+10 OPEN INPUT *file-name-1*
+11 READ *file-name-1* INTO *record-name-2* AT END (input file)
+12 READ *file-name-1* INTO *record-name-2* INVALID KEY (input file)
+17 CLOSE *file-name-1* (which was an input file)
+20 OPEN OUTPUT *file-name-1*
+24 WRITE *record-name-1* FROM *record-name-2* INVALID KEY
+27 CLOSE *file-name-1* (which was an output file)
+30 OPEN INPUT-OUTPUT *file-name-1*
+31 READ *file-name-1* INTO *record-name-2* AT END (I-O file)
+32 READ *file-name-1* INTO *record-name-2* INVALID KEY (I-O file)
+33 WRITE *record-name-1* FROM *record-name-2* (I-O file)
+34 WRITE *record-name-1* FROM *record-name-2* INVALID KEY (I-O file)
+36 WRITE *record-name-1* FROM *record-name-2* FOR DELETION INVALID KEY
+37 CLOSE *file-name-1* (which was an I-O file)

## I.7. ABORT CONDITIONS

Several error conditions may arise for which there is no possible recovery. These cause the program to abort, with a message produced on the printer to identify the problem. These fatal errors are of two types:

(1) these identified with an ISFMS message, and

(2) those identified by a COBOL-produced message.

The ISFMS messages are already enumerated in Appendix E. The fatal errors detected by ISFMSC (the American National Standard COBOL (Fieldata) ISFMS interface element) causes the following message to be produced:

ERROR CODE code-value FILENAME file-system-name error-producing-operation.

Where:

The value error-producing-operation may be:

OPEN FILE { INPUT / OUTPUT / I-O }

SEQUENTIAL READ

RANDOM READ

SEQUENTIAL WRITE

RANDOM WRITE

CLOSE FILE

WRITE FOR DELETION

The error identifying code-value may be:

101 – An ISFMS error concerned with the actual key occurred.
102 – A program contingency occurred that concerns the user's manipulation of the file, or a physical malfunction has occurred on the FH-432 drum.
210 – Insufficient space is available on FASTRAND mass storage to initialize a new index block.
211 – ISFMS can handle only 7 index levels. While updating the index, an attempt was made to write an index block at an index level higher than 7.
212 – An attempt was made to write an IOF record and no more IOF blocks are available.
244 – Number of overflow records per data block is equal to or greater than the number of records per block.
245 – Number of IOF records has not been specified.
246 – Number of records in the file has not been specified.
247 – File needs more FASTRAND tracks than have been assigned.
248 – File opened inconsistent with operation, for example, an attempt to write on an input file.
360 – Attempt to read beyond end of file; AT END exit was taken on previous sequential read.

# INDEX

Comments concerning this manual may be made in the space provided below. Please fill in the requested information.

System: _____

Manual Title: _____

UP No: _____ Revision No: _____ Update: _____

Name of User: _____

Address of User: _____

Comments:

CUT

FOLD

FIRST CLASS
PERMIT NO. 21
BLUE BELL, PA.

**BUSINESS REPLY MAIL** NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

POSTAGE WILL BE PAID BY

UNIVAC

P.O. BOX 500

BLUE BELL, PA. 19422

ATTN: SYSTEMS PUBLICATIONS DEPT.

FOLD

CUT