SPERRY RAND

UNIVAC

1106
SYSTEM
1108
MULTI-PROCESSOR
SYSTEM

EXEC 8
INDEXED SEQUENTIAL FILE MANAGEMENT SYSTEM
(ISFMS)

PROGRAMMERS
REFERENCE

UP-7780

This manual is published by the Univac Division of Sperry Rand Corporation in loose leaf format. This format provides a rapid and complete means of keeping recipients apprised of UNIVAC® Systems developments. The information presented herein may not reflect the current status of the product. For the current status of the product, contact your local Univac Representative.

The Univac Division will issue updating packages, utilizing primarily a page-for-page or unit replacement technique. Such issuance will provide notification of hardware or software changes and refinements. The Univac Division reserves the right to make such additions, corrections, and/or deletions as, in the judgment of the Univac Division, are required by the development of its Systems.

PRINTED IN U.S.A.

CONTENTS

CONTENTS	1 to 3
1. INTRODUCTION	1-1 to 1-11
1.1. INDEXED SEQUENTIAL FILE MANAGEMENT SYSTEM (ISFMS)	1-1
1.2. USER RESPONSIBILITY	1-1
1.3. ISFMS TERMINOLOGY	1-2
1.4. FUNCTIONAL DESCRIPTION OF ISFMS	1-3
1.4.1. Data Files	1-3
1.4.2. Index Files	1-5
1.4.3. Block Organization	1-6
1.4.3.1. Data Block	1-6
1.4.3.2. Index Block	1-7
1.4.3.3. Overflow Blocks	1-9
1.4.3.4. Information Block	1-9
1.4.4. Indexing Technique	1-10
2. ISFMS COMMAND REPERTOIRE	2-1 to 2-29
2.1. GENERAL	2-1
2.2. OUTPUT FILES	2-1
2.2.1. Open Output	2-1
2.2.2. Write Random Output	2-3
2.2.3. Close Output	2-5
2.3. INPUT FILES	2-8
2.3.1. Open Input	2-8
2.3.2. Read Sequential Input	2-10
2.3.3. Read Random Input	2-12
2.3.4. Close Input	2-14
2.4. INPUT/OUTPUT FILES	2-15
2.4.1. Open Input/Output	2-15
2.4.2. Read Sequential Input/Output	2-17
2.4.3. Read Random Input/Output	2-19
2.4.4. Write Sequential Input/Output	2-21
2.4.5. Write Random Input/Output	2-23
2.4.6. Write Random Delete Input/Output	2-25
2.4.7. Close Input/Output	2-26
2.5. INFORM COMMAND	2-28

APPENDIXES

A. COBOL INTERFACE	A–1 to A–2
A.1. IDENTIFICATION DIVISION	A–1
A.2. ENVIRONMENT DIVISION	A–1
A.3. DATA DIVISION	A–1
A.4. PROCEDURE DIVISION	A–2
B. ASSEMBLER INTERFACE	B–1 to B–2
B.1. GENERAL	B–1
B.2. FUNCTION CONTROL TABLE (FCT)	B–1
B.3. EQUF STATEMENTS	B–1
B.4. RESERVE WORDS	B–2
B.5. ADDITIONAL RESERVE WORDS	B–2
C. OPTIMIZATION PROCEDURES	C–1 to C–4
C.1. GENERAL	C–1
C.2. BLOCKS	C–1
C.3. FILE	C–3
D. SAMPLE PROGRAMS	D–1 to D–4
D.1. COBOL	D–1
D.2. ASSEMBLER	D–3
E. ISFMS ERRORS	E–1 to E–2
E.1. GENERAL	E–1
E.2. FATAL ERRORS	E–1
E.3. NONFATAL ERRORS	E–1
F. SUMMARY OF COMMANDS	F–1 to F–4
F.1. COBOL COMMANDS	F–1
F.2. ASSEMBLER COMMANDS	F–3

FIGURES

1-1. ISFMS File 1-4
1-2. Data File With Associated Separate Index File 1-5
1-3. Data Block 1-6
1-4. Index Block 1-8
1-5. Multi-Level Indexing 1-8
1-6. Hierarchical Data Files 1-10
1-7. Cross-Referenced Files 1-11

TABLES

E-1. ISFMS Nonfatal Errors E-2

1. INTRODUCTION

1.1. INDEXED SEQUENTIAL FILE MANAGEMENT SYSTEM (ISFMS)

The Univac Indexed Sequential File Management System (ISFMS) is a UNIVAC 1106/1108 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 UNIVAC 1106/1108 EXEC 8 Operating System (see *UNIVAC 1108 Multi-Processor System Operating System EXEC 8 Programmers Reference, UP-4144* (current version)). ISFMS requires approximately 5000_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).

1.2. USER RESPONSIBILITY

The user's interface with ISFMS is outlined here. 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).
- File control tables (FCT's) 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 @COPIN and @COPOUT control control cards.
- 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. 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 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 records 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 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 or records.

- Item

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

- Mass Storage

Storage, other than main storage, which can be accessed on a direct or random basis (which implies that it be a FASTRAND or other magnetic drum).

- 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 words 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: data files and index files. These files are subdivided into blocks. Each block contains 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 in 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 – see 1.4.2) at any given time.

There are three types of data files: output, input, and input/output. The manner in which ISFMS manipulates these files corresponds to that described in *UNIVAC Fundamentals of COBOL-Language, 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, that is, update, 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: data, index, and 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. A schematic overview of an ISFMS-prepared file is provided in Figure 1–1.

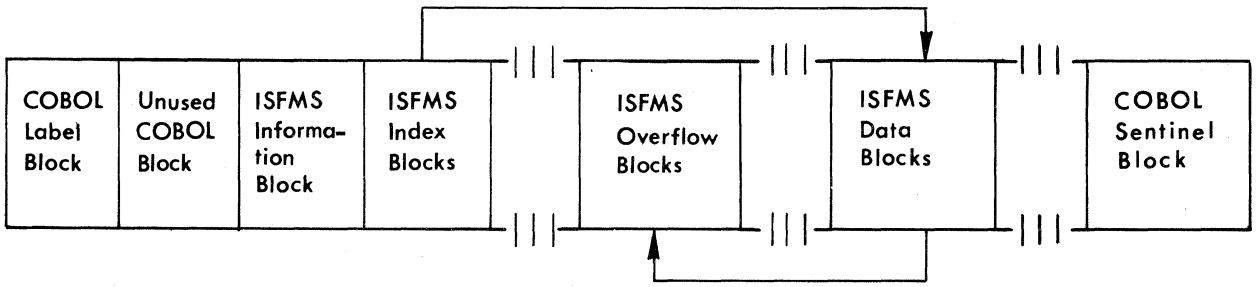


Figure 1–1. ISFMS File

The relationships 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 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 and the proper data block is then 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.

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 queueing 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.

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.

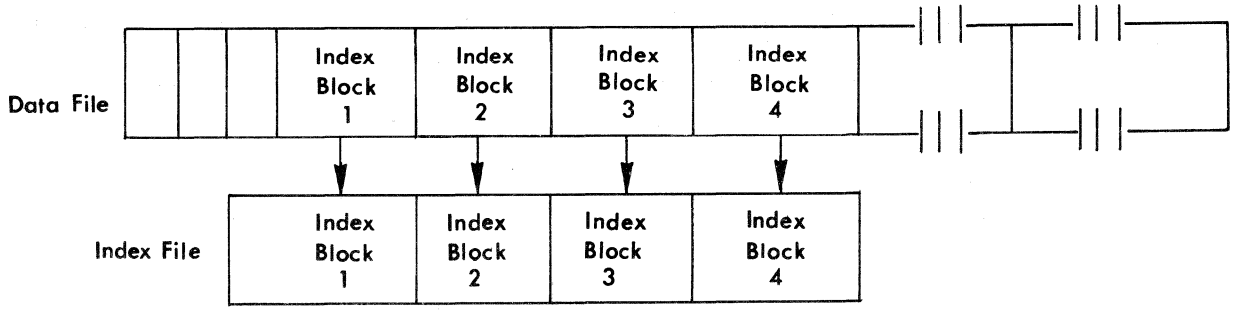


Figure 1–2. Data File With Associated Separate Index File

1.4.3. Block Organization

Three basic types of blocks are set up by ISFMS: data blocks, index blocks, and overflow blocks. The following paragraphs discuss the relationships 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.

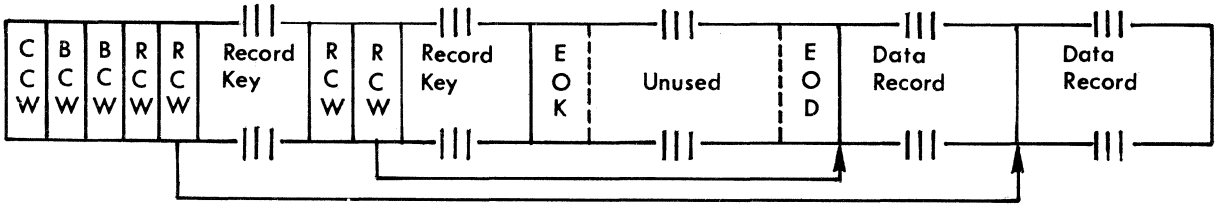


Figure 1–3. Data Block (Part 1 of 2)

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, the number of words of data records, and record keys.
- Word 2 of the BCW contains the relative block number and relative address of the first record key.

(3) RCW indicates record control words:

- Word 1 of the RCW contains the relative block number, record key length, relative address of the next record key, and a delete flag.
- Word 2 of the RCW contains the data record length and relative address of the data record, or time and date of deletion if the delete flag is set.

(4) EOK indicates end of keys sentinel.

(5) EOD indicates end of data sentinel.

Figure 1–3. Data Block (Part 2 of 2)

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.

Variable-length records are always treated as overflow records when the file is open for input/output. That is, they are always inserted into an overflow block regardless of the status of the logical data block. Note that this is only on insertion of new records on input/output. When the data file is being created (open for output), variable-length records are inserted in physically sequential order in the data blocks.

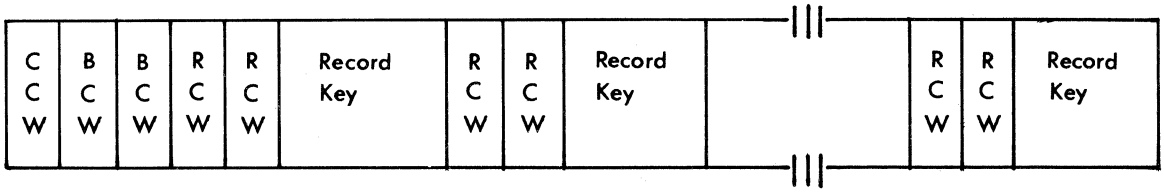
1.4.3.2. Index Block

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.

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 block type and the number of words used for record keys.
- Word 2 is not used.

(3) RCW indicates record key control words:

- Word 1 contains the data block number, record key length, and relative address of the record key.
- 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.

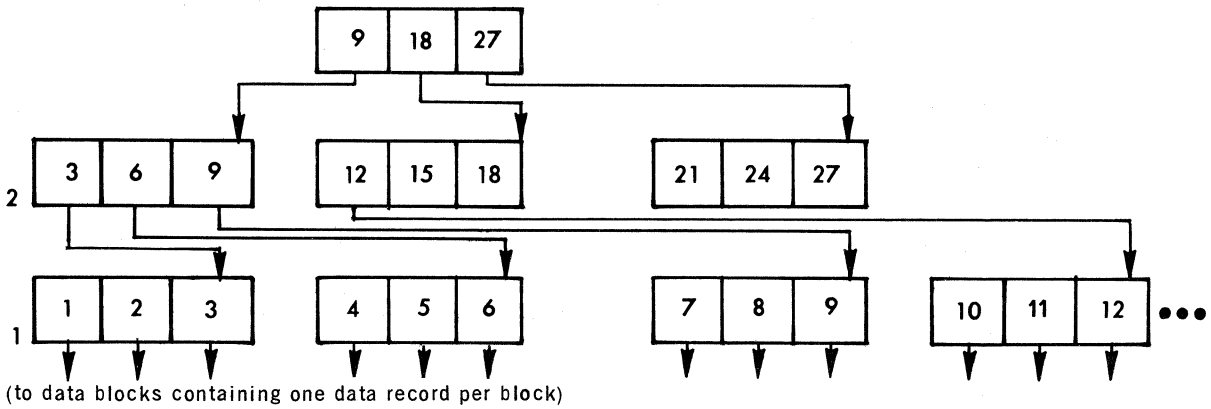


Figure 1–5. Multilevel Indexing

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.

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 excessive 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, for example, 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.

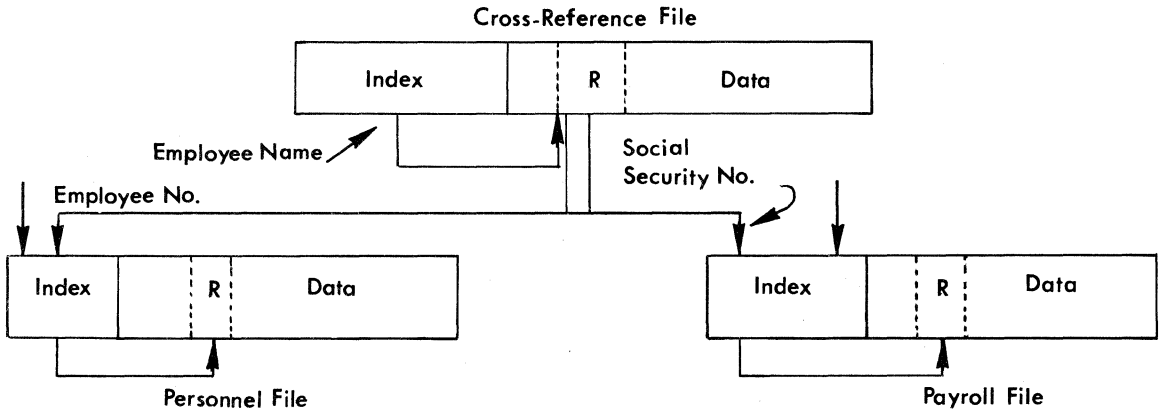


Figure 1–6. Hierarchical Data Files

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.

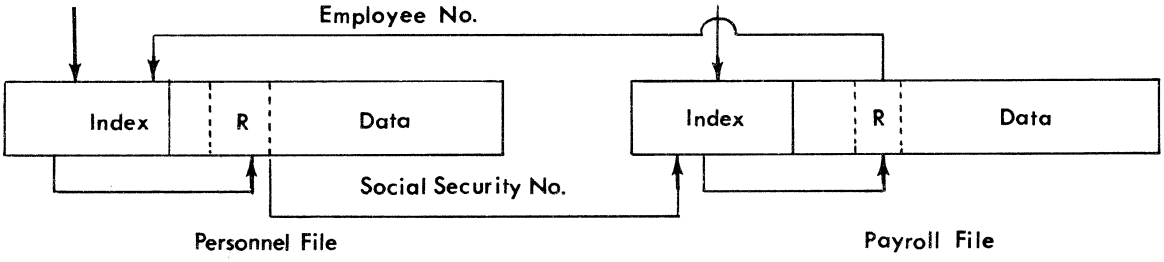


Figure 1–7. Cross-Referenced Files

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 above 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: input, output, and 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 8, 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.

2.2. 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
  Variable-length format: average (estimate) record length in characters

- Maximum record length

  Fixed-length format: 0
  Variable-length format: maximum record length in characters

- Record key length in characters
- 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 8 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.

(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) 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.

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

(7) 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.

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.

(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 as 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 8 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 moved 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 moved 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.

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) Assigns and opens the index file based on the information contained in the status-word.

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

(6) 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, the record-key-area must contain the key of the previous logical record. For example, to begin sequential processing with a data record having a record key of 100, the record-key-area must be set to the previous logical record key (assume it is 99), if known, and a Read Sequential Input command must be issued. If the key of the previous logical record is not known, 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 the sentinel block and reads the data block from FASTRAND mass storage.

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

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

(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) Increments the number of records read by 1.

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

(3) Reads this index block from magnetic drum.

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

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

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

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

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

(9) 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 8 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 moved 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 moved 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.

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) Reads the first data block from FASTRAND mass storage.

(6) 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, the record-key-area must contain the key of the previous logical record. For example, to begin sequential processing with a data record having a record key of 100, the record-key-area must be set to the previous logical record key (assume it is 99), if known, and a Read Sequential Input/Output command must be issued. If the key of the previous logical record is not known, 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 the sentinel block and reads the data block from FASTRAND mass storage.

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

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

(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) Increments the number of records read by 1.

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

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

(3) Reads this index block from magnetic drum.

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

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

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

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

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

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

(10) 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 which contains the new record.

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 lay-out 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 record is of variable length, in which case it is written in the independent overflow area (IOF), and the number of records in IOF is incremented by 1, for statistics stored in the information block. If the record is of fixed length and 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 writing of the data block on FASTRAND mass storage, when filled. 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, see *UNIVAC 1108 Multi-Processor System Operating System EXEC 8 Programmers Reference, UP-4144,* (current version)). 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 2^{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 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 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):

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

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 has to 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 has to 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 appropriate 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 PICTURE 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).

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. To provide the proper interface with ISFMS, the user must include the following in his program:

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

(2) EQUF statements, used to specify the lengths of the parameters used in calling the PROCs;

(3) reserve words, used to set up the buffer areas;

(4) additional reserve words, for the parameters used in calling the PROCs.

The above information corresponds to that automatically generated by the Data Division of COBOL.

B.2. FUNCTION CONTROL TABLE (FCT)

The FCT is 131_8 or 89_{10} words long and it provides the block buffering package (BBP) interface for the assembler-generated program. This BBP/FCT interface is explained in *UNIVAC 1108 Multi-Processor System Operating System EXEC 8 Programmers Reference Manual, UP-4144* (current version).

B.3. EQUF STATEMENTS

The EQUF statements provide the information used by the assembler procedures to generate the FCT.

```
L1(1)   EQUF   length of block in characters (see C.2, formula 12).
L1(2)   EQUF   length of status word in characters.
L1(3)   EQUF   length of file description in characters (see 2.2.1).
L2(1)   EQUF   length of block in characters (see C.2, formula 12).
L2(2)   EQUF   length of data records in characters (see Appendix C).
L2(3)   EQUF   length of record key in characters.
L3(1)   EQUF   length of block in characters (see C.2, formula 12).
L3(2)   EQUF   length of inform-area in characters (see 2.2.1).
L3(3)   EQUF   length of record key in characters.
L4(1)   EQUF   length of block in characters (see C.2, formula 12).
L4(2)   EQUF   length of record key in characters.
```

B.4. RESERVE WORDS

These large reserves are used for working storage, I/O storage, and the I/O block buffers.

```
LABEL   RES   0700†     working storage
        +     0703†,0   I/O storage
        +     0703†     I/O storage
        +     0         I/O storage
        +     0         I/O storage
LABEL   RES   0700†     I/O block buffer
LABEL   +     0         address of ACTUAL KEY
```

B.5. ADDITIONAL RESERVE WORDS

The following words are used as parameters in calling the PROCs:

- Inform-area is a reserve of nine words (see 2.2.3).
- File description is a reserve of six words (see 2.2.1).
- Status-word is a reserve of one word (six Fieldata zeros).
- Record-area and record-key-area are explained in C.2.

---

† *The numbers 0700 and 0703 are used here as an example of a quarter-track. The extra three words used for the I/O handler are control words. A half-track is 895 words and a full track is 1792 words. The three words used for the I/O handler are needed only once. The word BCWX is needed for the address of the ACTUAL KEY.*

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:

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

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:

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

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 6. 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 claculate the number of data blocks:

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

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

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:

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

from which is obtained:

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

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:

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

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:

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

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:

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

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. 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 FORM00
   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).


PROCEDURE DIVISION.
START-TEST.
   OPEN INPUT CARDS.
   OPEN OUTPUT RESULTS.
   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

```
START
            LMJ     X11, SEVENS.          . LMJ SEVENS
            L,U     R1,20.                . LOAD R1 WITH 20
            L,U     A7,1.                 . START OF RECORD KEY
            S       A7,KEY.               . START OF RECORD KEY
            L       A8,('PAT+++').        . LOAD WITH PAT+++
            DS      A7,ACTREC.            . DS INTO ACTRED
            OPO     ADO,STATUX,ADDER.     . OPEN FILE SAM
                    TE A0,('000000').     . TE A0=0
                    J EX.                 . JP TO EXIT
WRT         WRRO    ADO,ACTREC,KEY.       . WRITE TO FILE SAM
                    TE  A0,('000000').    . TE A0=0
                    J EX.                 . JP TO EXIT
            A,U     A7,1.                 . ADD 1 TO A7
            S       A7,KEY.               . STORE A7 TO KEY
            DS      A7,ACTREC.            . DS INTO ACTREC
            JGD     R1,WRT.               . JP IF R1>0
            CLO     ADO,INFO,KEY.         . CLOSE FILE SAM
            J       EX                    . JP TO EXIT
SEVENS      L,U     X10,27D.              . LOAD X10 WITH 27D
            L       A1,(-0).              . LOAD A1 WITH ALL 7'S
            S       A1,ACTREC,X10.        . STORE INTO ACTREC
            JGD     X10,$-1.
            J       0,X11.
EX          ER      EXIT$.
```

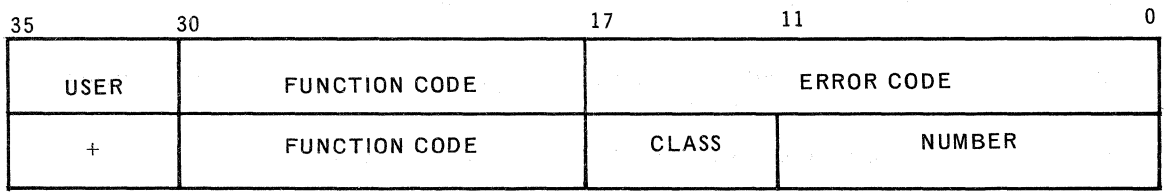
```
$(0)
L1(1)       EQUF    05172           . RECORD LENGTH
L1(2)       EQUF    06              . STATUS LENGTH
L1(3)       EQUF    044             . FILE-DESCRIPTION IN CHARACTERS
L2(1)       EQUF    05172           . RECORD LENGTH
L2(2)       EQUF    48              . DATA RECORD IN CHARACTERS
L2(3)       EQUF    06              . STATUS LENGTH
L3(1)       EQUF    05172           . RECORD LENGTH
L3(2)       EQUF    066             . LENGTH OF INFORM-AREA
L3(3)       EQUF    06              . STATUS LENGTH
L4(1)       EQUF    05172           . RECORD LENGTH
L4(2)       EQUF    06              . STATUS LENGTH


ACTREC      RES     48              . DATA RECORD LENGTH IN CHARACTEF
KEY         +       0               . RECORD KEY AREA
ADDER       +       0,000025        . FILE DESCRIPTION
            +       0,000360
            +       0,0
            +       0,000006
            +       0,000001
            +       0,000001
INFO        RES     9               . INFORM-AREA
STATUX      +       0606060606060   . STATUS WORD
BUF1        RES     0700            . WORKING AREA
            +       0703,0
            +       0703
            +       0
            +       0
BUF         RES     0700
BCWX        +       0
ADO                                 . FIRST TWO WORDS OF THIS FCT
                                    . CONTAIN THE FILE NAME. THE
                                    . REST CAN BE SEEN IN APPENDIX B
```

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 zerofilled if no error has occurred and has the following format in Fieldata:



E.2. FATAL ERRORS

The run is aborted when one of the following fatal errors occurs:

(a) An attempt is made to open an already opened file.

(b) An attempt is made to refer to an unopened file.

(c) An invalid function code is encountered.

(d) The status-word is improperly defined.

(e) An attempt is made to have more than 10 files open at the same time.

(f) The file has not been assigned.

E.3. NONFATAL ERRORS

Table E–1 lists the 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 is returned to the user program and recovery from the error is possible.

ERROR CODE	ERROR
101	ACTUAL KEY is larger than the end of file key and is not equal to next key.
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: (1) Incoming record is greater in size than that defined in the tables. (2) Check if status word is equal to one. (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: (1) Incoming record is greater than that defined in the tables. (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 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.
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.
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.

Table E–1. ISFMS Nonfatal Errors

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

(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.

F.2. ASSEMBLER COMMANDS

OPEN INPUT

OPI FCT-address, status-word.

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

READ SEQUENTIAL INPUT

RESI 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.

For Inform command, see 2.2.3.

OPEN OUTPUT

OPO FCT-address, status-word, file description. (6 words, all H9(10))

. See COBOL Open Output command.

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.

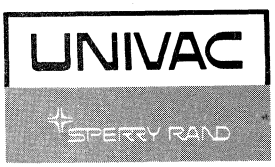
CLOSE INPUT/OUTPUT

CLIO FCT-address, inform-area.

INFORM

INFORM FCT-address, inform-area.

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


UNIVAC
SPERRY RAND