6. INDEXED RANDOM FILES

6.1 Description

Random files as described in the previous section are simple to set up and use. The limitations become apparent when records are referenced by non-serial items such as names, or the user wants to access the record via a number of different items such as name as well as account number.

In this case, an index must be created that provides a cross-reference between the item provided as the reference (the symbolic key) and the relevant record. These keys can be alphabetic such as names, alphanumeric such as encoded charge numbers, or numeric such as account numbers. Up to four keys can be used to reference the record. When a record is required, the instruction is supplied with the key and the system looks up the logical record number associated with the key. The symbolic keys are all held in a sequential file called an 'index file'. A summary of this file is created in order to reduce the search time for a record. This summary of the index file is called the 'master index', which is also a sequential file. The data file, index file and master index file constitute a 'file structure'.

6.1.1 *File structure*

The basic component of a file structure is a data file. Each file structure can contain *only one* data file. For every record in the data file, at least one item has been nominated as a key. The keys are gathered into an index file that has been sorted according to the pure binary form of the key and into ascending order. The index file is thus a *sequential file.* Each key in the index file is given the logical record number of the record it belongs to in the data file. The record key in an index file is called the 'symbolic key'.
The number of symbolic keys in the index file is obviously the same as the number of records in the data file. To search through the index could take a lot of time for a large file so the system divides the index into partitions. The highest value key in each partition is copied into a 'master index' with the lowest record number of that partition. This master index is also a sequential file. When a symbolic key is input at runtime, the master index is searched sequentially until there is a match or until the next highest value key is found. In either case the master index points to the partition that contains the symbolic key and hence the logical record number of the required record. Note that the master index is stored in memory while the index file is assigned.
The relationship between the data file, the index file and master index is shown in the following diagram.

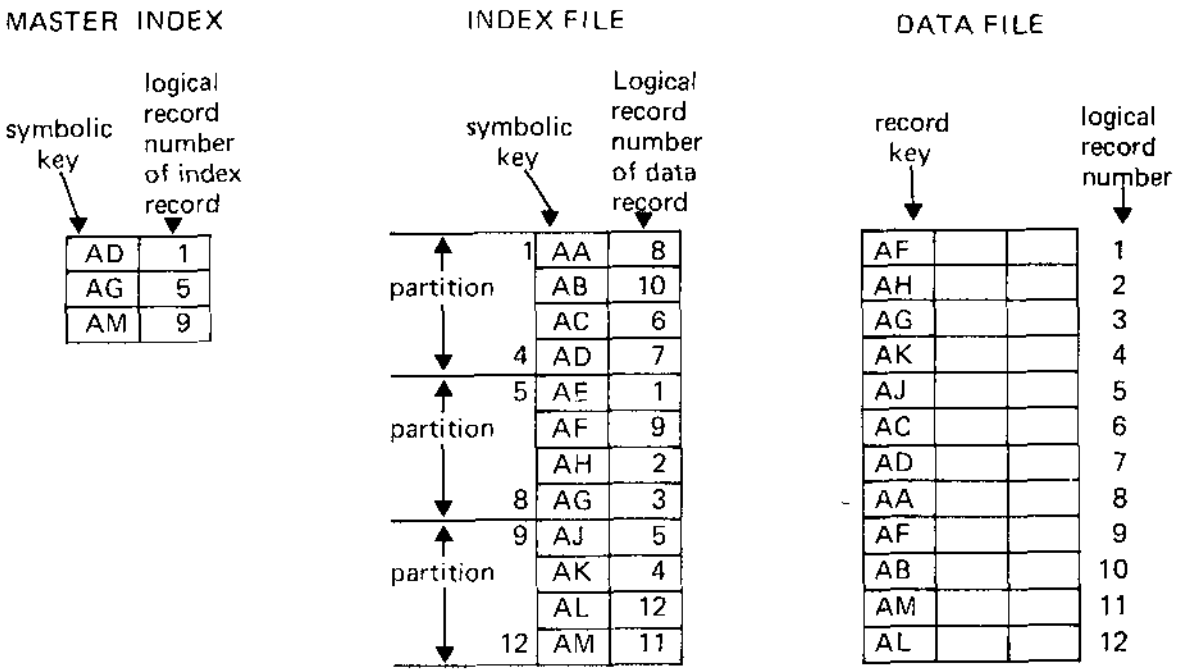

Figure 6.1 File Structure

Consider two examples of record access from a key supplied via the keyboard. First, key AG is supplied, then AL.

Example 1:

1. The operator inputs AG.
2. A sequential search is made of the master index until a match occurs, or the next highest value is found.
3. A match occurs stating that symbolic key AG is in the partition that starts with record 5.
4. A sequential search of partition 2 of the index file until a match occurs – this gives the logical record number 3 in the data file.
5. The record is accessed directly.

Example 2:

1. The operator inputs AL.
2. A sequential search is made of the master index until a match occurs or the next highest value is found.
3. AL does not exist in the master index but a higher value is encountered, i.e. AM, which gives the partition that starts with record 9.
4. A sequential search is made of partition 3 of the index file until a match occurs – this gives the logical record number 12 in the data file.
5. The record is accessed directly.

Thus the use of a master index as a summary of the index file, and the division of the index into partitions, gives considerable savings of time when searching for a record.

Any data item in the record can be used as the key and it is possible to use more than one data item for accessing the record. If a second (or third or fourth) item is required as a key then more index files must be created (and master index files), see figure 6.2 below.

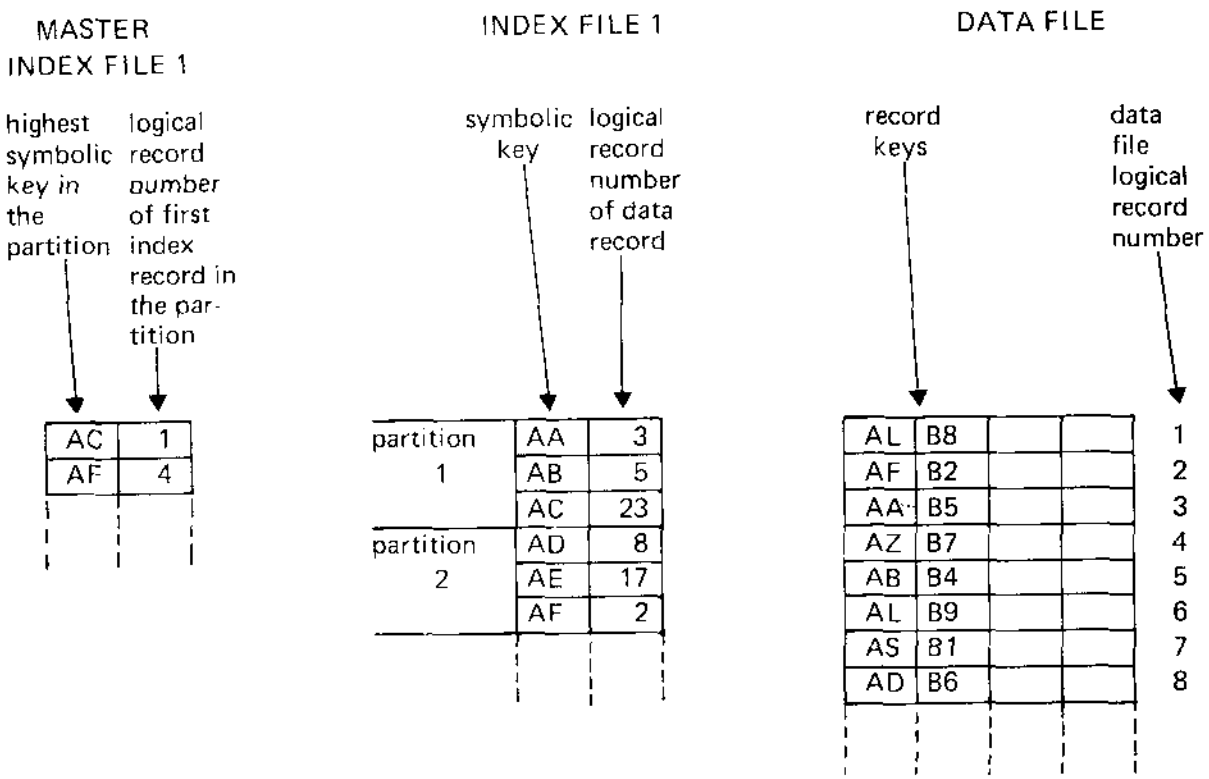

Figure 6.2 Data file with Two Sets or Index Files

6.1.2 *The Index file*

The index file is a sequential file and contains one symbolic key for each record in the data file and the logical record number of the data file record. It also contains a status indication for the record and an indication of duplicate characters in a set of symbolic keys.

Allowance must be made for the file to grow without the need for frequent reorganization. Just as a number of empty records are allowed for when creating the data file, so should empty records be present in the index file. The ratio of used to empty records is called the 'load factor' and is a parameter required for the Re-organize Index utility (RIX) which is fully described in the Utilities Reference Manual, M08. The number of empty records in the index file should be the same as that in the data file, but the empty records are put at the end of each block (i.e. sector) in the stated proportion. Consider an example data file of 100 records spaces. It is expected to grow to this size quickly but when the records are set up, only 50 data records are available. The index file is created with symbolic keys pointing to the existing records and not the empty ones. However, provision must be made in the index file for the data file to grow, so the index is created with a load factor 50%. In the data file, the empty records are positioned logically at the end of the file. The index is a sequential file and new symbolic keys must be inserted in the correct sequence so the spare index records are placed in each sector. The new key is inserted in the correct position and the following records in that sector are shifted along. Figure 6.3 summarizes this situation.

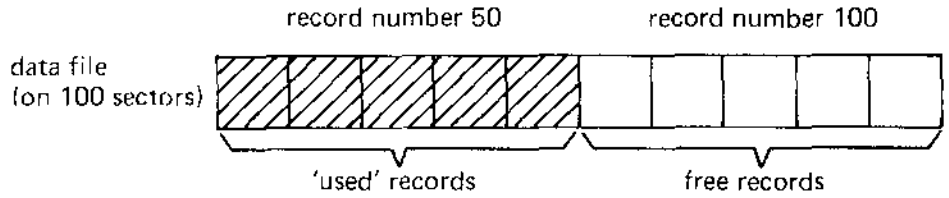

Figure 6.3a Illustration of Load factor-Data File

There is 50% utilization of record spaces after initialisation of the file, so the index is created with a load factor of 50%. A new record, logical record number 51 is placed at the end of the used area and the LRN updated.

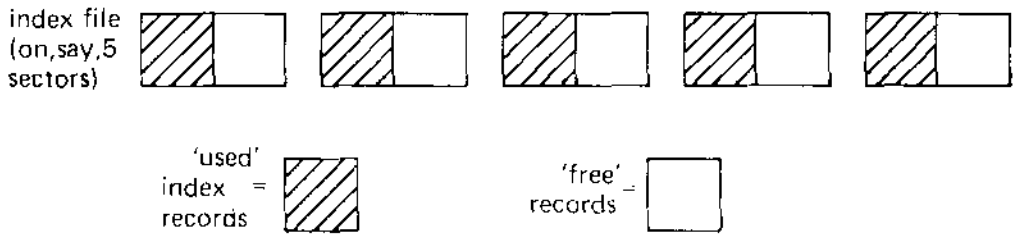

Figure 6.3b Illustration of Load factor-Index File

The index record for record 51 in the data file is placed in the correct position according to the symbolic key and the keys following in the same sector are shifted along. If the records overflow into the next sector, the records already contained in that sector are shifted along.

The format of the index record is as follows

field	length
1. Symbolic key	1-n characters
2. Dummy	2 characters
3. Duplicate key	1 character
3. Logical record number	3 characters

where:
'symbolic key' is the data item contained in the data file record that is used for identification of the record. It is left ajusted and padded with blanks if n is greater than the key used.
'Dummy' is not used.
'Duplicate key' is the binary value of the minimum number of leading characters in the key that is identical with the next symbolic key in the index file.
'Logical record number' is that of the record in the data file that is referred to.
Remember when calculating the blocking factor that one byte must be added to the record length for the status byte.

6.1.3 *The Master Index file*

The master index is used to reduce the time required to search the index file. This is created by the system during the utility Re organize Index (RIX). The size of the master index file is decided by the user and this will determine the number of partitions, and hence the number of master index file entries.
The master index file must reside on the same volume as the index and there can be a maximum of 16 master indexes in the system at the same time. When the index file is assigned to the data file by the application, the master index is completely read into memory. The size of the master index memory area has to be specified during system generation, and must be able to contain all master index files required simultaneously plus three words.

The optimum size for partitions should be related to the physical storage of the index file on the disk. That is, the master index file should be constructed to minimize disk head movements when searching for index records. The system reads a whole sector at a time even through only one record is being accessed. If the index file occupies, for example, 5 sectors then the master index file could be created with 5 records. Each master index entry would describe one sector (= 1 partition). The key value is the last record in the sector/partition (highest key value) and the lowest record number is the first record in the sector/partition. Thus only one disk head movement is required to access all the index records in the partition. Figure 6.2 above illustrates this relationship.

For very large data files this could result in a large master index. A data file with 4 000 records would have an index file with 4 000 entries. If these latter were blocked 20 per sector then the index file would occupy 200 sectors with 200 entries in the master index file. This could result in relatively long search times for records with high key values. Time could be saved by creating the master index files with enough records for 1 entry per track (number of sectors = 16). Thus the disk head only needs to move once to the relevant track and have access to any index record in that partition/track within one revolution/track of starting the search.

It is difficult to give exact values for the time required to find a particular record in a file structure because of the number of variables involved. Nevertheless, the programmer should keep these points in mind when creating a file structure and balance these to give a reasonable partition size for each circumstance. The variables to be considered are:
a. number of index file records
b. blocking factor of index file
c. load factor of index file
d. amount of memory available to hold the master index.

If 'a' is high and 'b' and 'c' low, then 1 track to a partition could be better than 1 sector to a partition. If 'a' is low then 1 sector to a partition could be better. However, with limited amounts of memory available the master index would have to be small and the partitions arranged to represent a large number of index records.

6.2 Creating the files

The files can only be created on a TOSS formatted disk, i.e. one that has been created by the utility Create Volume (CRV).
The creation of the files must be performed on 8 stages:

1. The data file is set up by using the utility Create File (CRF)
2. The actual records must be written to the file.
3. The index file and master index file must be created using the utility CRF.
4. Before the index files can be built, the utilities used in this process must have two intermediate files. These are created at this stage and for the purpose of this explanation are called IFILE1 and IFILE2.
5. The utility Build Index File (BIX) is next. It takes the data file as input file and builds a file of records that contain the key and the logical record numbers of the data records. This output file is the intermediate file IFILE1.
6. The records on IFILE1 must be sorted according to the pure binary form of the key and into ascending order. Output is to intermediate file IFILE2.
7. The sorted file IFILE2 is used as input to the utility Re-organise Index file (RIX). It takes the sorted records from IFILE2 and writes them in index record format onto the index file created at stage 3. The master index file is generated by RIX at the same time using the file created at stage 3.
8. The data file, index file and master index file are now available for use by the application. IFILE1 and IFILE2 can be deleted.

This process is summarized in figure 4.4.

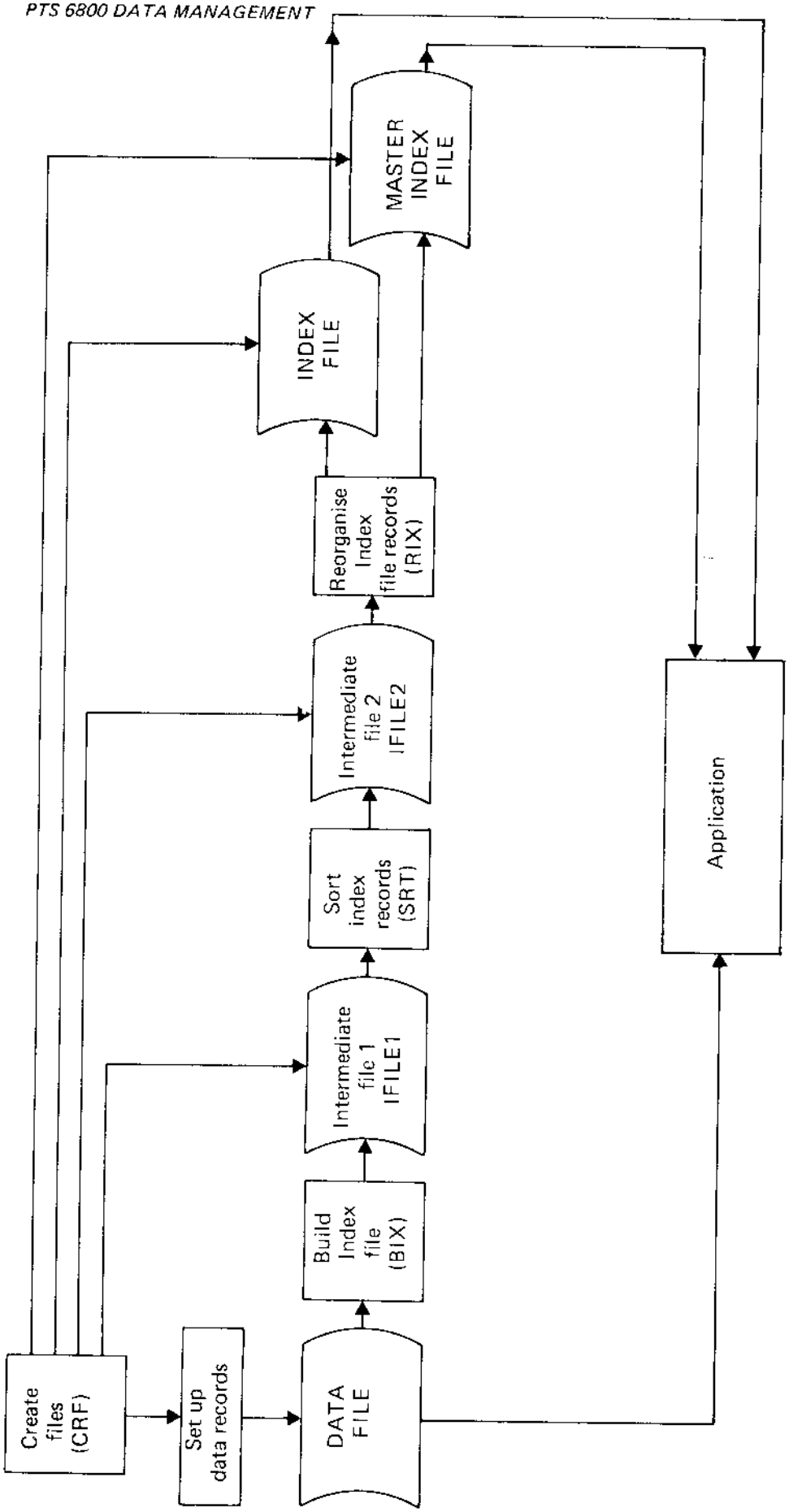

Figure 6.4 Setting up a file structure

Stage 1

The data file must be created using the utility CRF.

1. Call CRF utility under the TOSS utilities Monitor, or as a subroutine from the application, or via the DOS utility TOSSUT.
2. CRF requests a number of parameters. Most parameters can be given as required, but for an indexed random file, two parameters are obligatory. To 'File organization' give 'S', and to 'Number of index files' give 'n' where n is the number of index files required (1—4).
3. CRF searches the volume(s) for free extents large enough to hold the stated file size.
4. The file is created with the required number of records, all of which contain space characters.

Each record is set to 'FREE' status. The LRN is set to zero for this file.

Stage 2

The actual records must be written to the data file using sequential operations otherwise the LRN will not be updated and will still be set to zero the first time that the file is used. The operations for this stage depend upon the source of the records. If the records already exist, for example a bank branch putting its account records onto the PTS system then the records could be copied into the 'empty' file on the disk. If it is a new system it would save time during the next stage if the records could be written into the file in the ascending sequence of the key data item.

Stage 3

The index file and master index files must be created using the utility CRF. The number of index files must be given as '0' in both cases.
The record lengths must be:
for index file = key length + 6
for master index file = key length + 3.

Stage 4

Create the intermediate files using the utility CRF. Any file names can be used, but the example shown in figure 4.4 uses IFILE1 and IFILE2. The number of index files must be given as '0' in both cases.
The record lengths must be:
for intermediate file 1 = key length + 6
for intermediate file 2 = key length + 6.

Stage 5

Next, the utility BIX must be performed as follows:

1. Call BIX utility under the TOSS utilities Monitor.
2. BIX requests a number of parameters via the console typewriter. Most parameters are input as required. The data item that is to be used as the index is specified by two parameters. These are 'Key Address in Record' — give the address of the first character of the key relative the start of the record, in decimal; 'Key Length' — give the length of the required key, in decimal.
3. BIX then scans the data file and copies the required keys to IFILE1. The index records are written sequentially to the file without any regard to the value of the key. For this reason the next stage (sorting) must be performed.

Stage 6

IFILE1 at this point contains the required number of symbolic keys but they are unsorted. The following sequence of operations must be performed.

1. Call utility SORT under the TOSS utilities Monitor, or as a subroutine from the application.
2. SORT requires a number of parameters but this process is a standard sort and requires no special parameters.
3. The sorted records are output to IFILE2.

Stage 7

IFILE2 consists of a set of symbolic keys (with logical record number referring to the data file) that are sorted into ascending order. The index records must now be formatted onto the index file and a master index file built. The master index is structured and formatted by the same utility, Reorganize Index File, and requires no parameters from the user. This master index is stored on the same volume as the index and is assigned at the same time. The sequence of operations for this last stage is as follows:

1. Call RIX utility under the TOSS utilities Monitor, or as a subroutine.
2. RIX requests a number of parameters via the console typewriter. Most parameters are input as required but the reader should note that a value is required for 'Load Factor' which was discussed in section 6.1.2, The Index File. This parameter is the percentage of 'used' records to be written to each sector and should reflect the percentage of 'used' records in the data ifle. The RIX utility will use this factor to construct the partitions and to build the master index file.
3. Index records will be read out from the input file and written in the required format for an index record with the required number of free records at the end of each sector. Records are written to the master index file sequentially during the run. RIX performs a check on the record sequence and an error is set if a key sequence error is detected.
4. At completion, the index is properly structured and formatted and RIX has constructed the relavant master index file.

Stage 8

The file structure is now available for use as an indexed sequential file, see the instructions in the following paragraphs. It is possible to use this data file as a sequential file (for copying or generating reports) or as an ordinary random file (but only if a record key can be input which is related to the record's logical record number). However, no process should be allowed to take place on the data file that is going to disturb the order, or position, of records without also updating the index file.

Note
It is also possible to start with an 'empty' data file, LRN = '0', and hence empty index and master index files. Indexed-insert can be performed until performance considerations require index reorganization with RIX. This would probably be the case for a new application where data is generated and collected as the application goes live (a new branch office, for example, taking on new accounts).

6.3 Instructions

A definitive description of these instructions is contained in the relevant language reference manuals, either. Assembler Programmer's Reference Manual, M06, or CREDIT Programmer's Reference Manual, M04. Before these instructions can be used, both the data file and the index file must be assigned otherwise an error will be returned.
These instructions are, briefly:

ASSIGN THE FILES

An indexed random file must be assigned in two steps.
- the data file is assigned to a file code as accessible by all tasks or only accessible by one task, that is with TC = 0 for common files, TC = 1 for this task only.
- the index file is assigned using the index file assign instruction. The associated master index file is assigned implicitly and read into memory. If more than one index/master index is to be used, they must all be assigned separately.

On this data file records may be retrieved, deleted, stored, inserted, or retrieved sequential from a certain point by using the commands indexed random read, indexed delete, indexed rewrite, indexed insert, or indexed read next. The record to be fetched is indicated by means of a symbolic key.
It is allowed to assign an index-file as a data file, in this case the user is able to process this file as an ordinary data file and can create his own index records. It is not allowed to assign an index file as both a data file and index file at the same time.
An indexed rewrite, indexed delete or indexed insert may be performed when all index-files corresponding to the data file are assigned.

Assigning a file code to a data file or an index file on flexible disk results in the door of the corresponding flexible disk drive being locked.

INDEXED RANDOM READ

The task must supply a symbolic key with this instruction. Data Management searches the master index first until a matching or first highest key is found. This will point to the relevant partition in the index file and cause a search in that partition until a match occurs. This will give a logical record number and the record will then be accessed directly. The record can then be put under exclusive access if required. The CRNs for both the index file and data file will be updated.

INDEXED REWRITE

The data record must first be read and exclusive access set (if it has been specified during system generation before this instruction is used). The record is replaced by the new record, except the symbolic key which remains unchanged, and exclusive acces released. The CRN for the data file will be updated to this record number and set to zero for the index file. Note that in CREDIT this instruction is effected through the DSC1 statement.

INDEXED DELETE

This instruction will delete the data record and the entries in the index file. The task must supply the logical record number of the data record which must be under exclusive access (if this has been specified during system generation). The data record is read and the index entry is deleted only after a successful read. The data record is set to 'free' and exclusive access is released. The CRN is not affected.

INDEXED INSERT

This instruction allows the task to insert an new data record into the data file and create a new index record. The data record is added to the end of the file and the LRN is updated in memory. The symbolic key is inserted in the correct place in the index file according to its pure binary value and the records following the new index record in same disk sector shifted along into the free area. If an index record already exists with the same symbolic key, the new is inserted in front of the old. Exclusive access is not set and the CRN in the data file and index file will be set to the values of the new record and its index.

INDEXED READ NEXT

This performs the same function as indexed random read except that no symbolic key is supplied. The data record that is read is the one that is referred to by the *index record* following the *current index record number*. This implies that the 'read next' is the next highest symbolic key and the data record it points to. If the CRN is zero then the first key in the index file is used. The CRNs in the data file and index file will be set to the new values.
This instruction can only be used after -

a. indexed random read
b. indexed insert
c. or another indexed read next.

CLOSING THE FILES

A close file is issued for first the data file, then each of the index files to indicate that the file is no longer required by the task. By closing a file the LRN in the volume table of the corresponding file is updated and saved on the volume. The previously assigned file code is no longer valid for this file.

When an index file is closed the master index file will be closed implicity.

Note
If a large number of changes has occured during the time the file structure was in use, it is advisable to run utility PACK to reorganize the index and master index files. This will ensure that the master index file is a true picture of the structure of the index file.

A data file with 'Short Record' is reorganized with 'Copy File with pack' utility program (CF PA) after which the indexed must be rebuilt.