AN ABSTRACT OF THE THESIS OF

MEESAJJEE CHULIT (Name of student) for the MASTER OF SCIENCE (Degree)

in Electrical and Electronics Engineering (Major) presented on Dec 12, 1972 (Date)

Title: THE EVALUATION OF DATA FILES ACCESS METHODS FOR AN ON-LINE INFORMATION SYSTEM

Abstract approved: Redacted for Privacy

Louis N. Stone

It is necessary to evaluate and compare the characteristics of various methods of accessing data-files in order to utilize economically both the hardware and the software (Space and Time) supported by the digital on-line system. The purpose of this paper is to describe and evaluate the structure and use of four conventional methods of file organization: Sequential File, Indexed Sequential File, Partitioned File, and Direct File.

Special attention is given to the Direct File, which possesses the fastest accessing time. Five selected Hash Coding Techniques, each associated with three methods of handling redundant keys, are simulated and examined with the use of a selected data model of 1024 random United States names, and the resulting "average number of search per record retrieval" are compared with their

corresponding theoretical values. As Hash 1 has offered the best results, it has been used to evaluate the organization of the Direct File, and to compare this organization with that of the other files.

The CDC-3300 system hardware parameter, control cycle time, the internal core storage, and the auxiliary storage parameters are introduced. From these values and the average number of searches per record retrieval, an expression of logical record file size, or loading factor is developed. The file size, or loading factor varies for different methods of file structure and accessing, (based upon the selected testing program). The system characteristics consisting of the average throughput per record retrieval, achievable-throughput-rate capability and user operating cost per call (unit cost) are evaluated and compared. The file system uses the full name of the record and a fixed length numerical key.

Two common internal searches, Linear search and Binary search, are evaluated and compared as the preliminary work of this investigation, as shown in Appendix B.

The Evaluation of Data File Access Methods For an On-Line Information System

by

Chulit Meesajjee

A THESIS

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Master of Science

June 1973

APPROVED:

Redacted for Privacy

Professor of Electrical and Electronics Engineering
in charge of major

Redacted for Privacy

Head of Department of Electrical and Electronics Engineering

Redacted for Privacy

Dean of Graduate School

Date thesis is presented Dec. 12, 1972

Typed by Ilene Anderton for Meesajjee Chulit

ACKNOWLEDGEMENTS

The writer desires to express his most sincere appreciation and thanks to Professor Louis N. Stone for his encouragement and advice through the course of this study and for his help in the preparation and writing of this thesis.

Thanks go to the staff of the Operating System and to the Control Data Corporation representatives at Oregon State University Computer Center for supplying the source of information for this work.

Thanks are also due to Mrs. Blanche B. Stroup for reading and checking the format of this thesis.

The writer is grateful to the Royal Thai Air Force Committee and to his parents for their approval and support of this study.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF APPENDIX TABLES

Appendix Table — Page

Appendix Table		Page
B. 33	Data results of computation of CPU busy time and average throughput time per record retrieval of direct disk file organization with random probing using full name of a record in accessing a random record from file.	334
B. 34	Data results of computation of storage space required as the function of file size of direct file organization with random probing using both record's full name and fixed-length numerical code in accessing.	335
B. 35	Result of computation of customer operating cost (unit cost) as the function of file loading factor of direct disk file with random probing using full name of record in accessing.	336
B. 36	Results of computation disk average access time per record retrieval as a function of loading factor of direct disk file using Hash 1 with direct chain probing.	337
B. 37	Data results of computation of CPU busy time and average throughput time per record retrieval of direct disk file organization with direct chain probing using full name of a record in accessing a random record from disk file.	338
B. 38	Data results of computation of storage space required as the function of file size of direct file organization with direct chaining using both record's full name and fixed-length numerical code for accessing a record from the file.	339
B. 39	Result of computation of customer operating cost (unit cost) as the function of file loading factor of direct disk file with direct chain probing using full name of record in accessing.	340

LIST OF APPENDIX FIGURES

THE EVALUATION OF DATA FILE ACCESS METHODS FOR AN ON-LINE INFORMATION SYSTEM

I. INTRODUCTION

It is apparent that time and space play a main role in computing system efficiency, so that the concepts of utilizing mass storage economically, and the reduction of processing time are the engineering goals for digital computers used in modern information processing systems. To achieve these ends the development of a computing system should be accomplished by more effective utilization of the hardware or the software, or both.

Basically the computer system software development (the problem of data organization within the constraints of information retrieval, and the organization of files) is usually much more convenient than computer system hardware development. This is especially true in the case of existing computing system hardware. In a large capacity storage computing system, the concept of virtual memory and paging is frequently utilized for economical implementation. In this type of computing system the storage is separated into two distinct levels. The first level consists of expensive, fast-access core storage which comprises the main core memory. The second level is the auxiliary memory (disk, drum, or magnetic tape) which is much less costly than the main core memory. To make the system

economical and practical, the auxiliary memory is used for the majority or bulk storage, with inexpensive but longer access time. The efficiency of the operation of the system therefore depends upon two major aspects:

1. The organization of information within the two storage levels.
2. The manner in which information is transferred between the two levels.

By consideration of these two factors the efficiency of a computing system may be improved (or optimized).

On the aspect of information organization, file organization techniques have been given attention periodically over the years, both from persons interested in information retrieval and from those interested in file-oriented computer applications in business, engineering, science, and government works.

For these reasons, it is essential to evaluate and compare the characteristics of various file organization methods and their use.

II. METHODS OF DATA FILE ORGANIZATION FOR ON-LINE SYSTEM

The development of a larger and faster computer with efficient 1/0 terminals has made it possible to store and retrieve selected information for the supporting storage of the system (disk memory). The development of information structure and indexing techniques permits selected data to be arranged in mass storage devices in a manner amenable to interaction with the users. For the achievement of this goal, the processing information (data) has to be prearranged as follows:

Data Division

In Electronic Digital Information System a data division is classified according to:

Machine oriented data division. This data is partitioned into a size particularly suitable for manipulation by the machine:

REEL (tape) or CYLINDER, TRACK (disk)

BLOCK > WORD > BIT

See illustration in Figure 2.1.

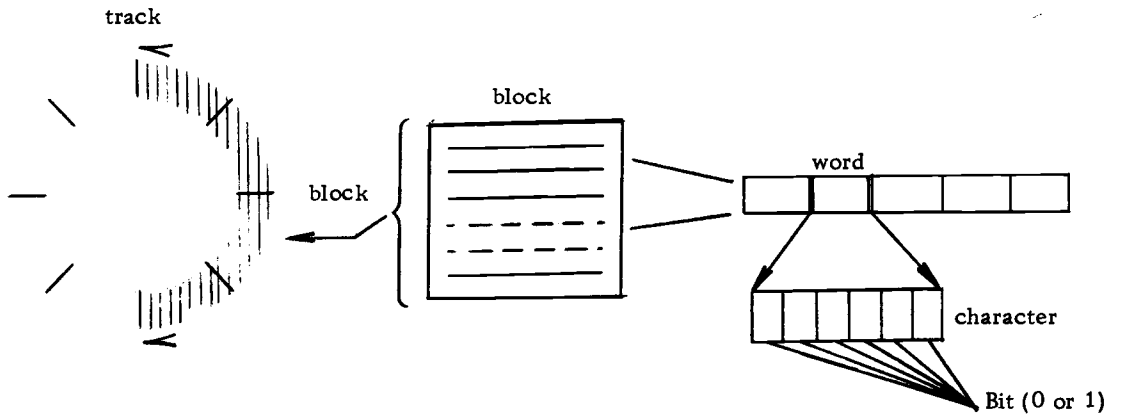


Figure 2.1. Machine-oriented data division hierarchy: Track, block.

2. Application oriented data division: This breaks the data into a size convenient for human manipulation associated with the application under consideration:

FILE > logical record or RECORD > FIELD > CHARACTER. See Figure 2.2.

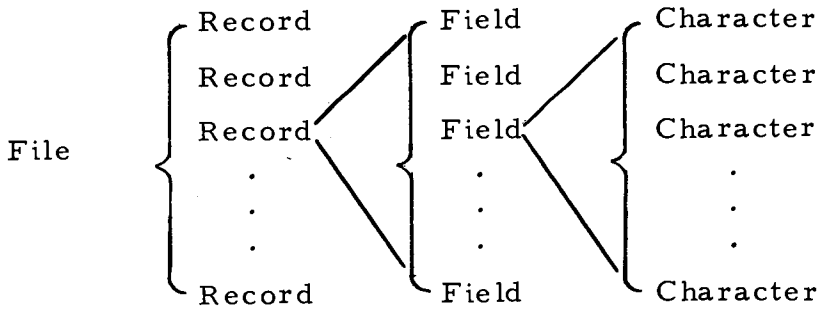


Figure 2.2. Application-oriented data division hierachy: File, record, field and character.

It is necessary to emphasize the interrelation between machine-oriented and application-oriented data divisions.

Disk-module-File: The three relationships which may exist between a Disk-module and a File are illustrated in Figure 2.3.

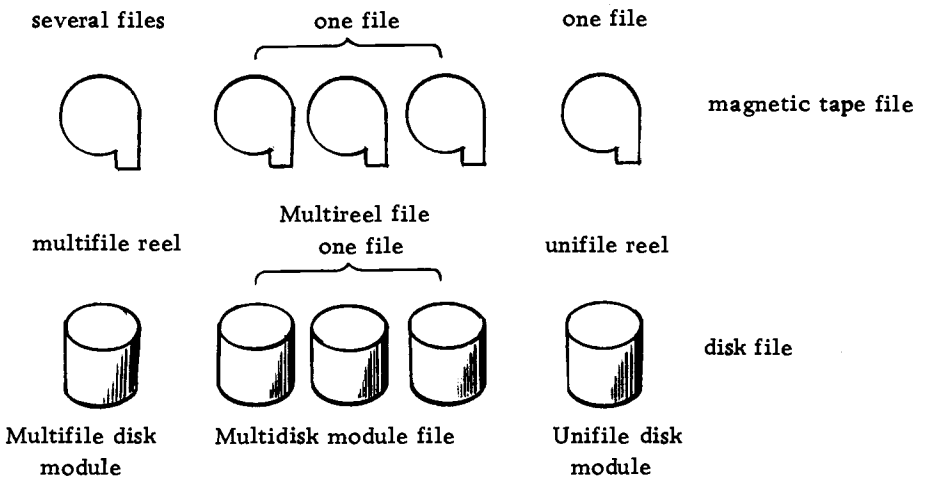


Figure 2.3. Reel-file, Disk module interaction.

1) A Multifile-Disk Module is a disk-module containing several files. It is the same as a multifile reel when files are supported by magnetic tape.
2) A Multi-Disk Module File is a large single file supported by several disk-modules, It is the same as a multireel-file.
3) A Unifile-Disk Module is a disk-module holding exactly one file. It is also the same as a unifile reel.

Block-Record: The three relationships which may exist between a Block and a Record are illustrated in Figure 2.4.

1) A Multirecord Block. When records are very small compared to block-size, it is economical to keep several records in a block.

2) A Multiblock Record. A record which contains a very large amount of imformation may require several blocks.

3) A Unirecord Block: A single record fitting exactly into a single block is called a "unirecord block" or a "uniblock".

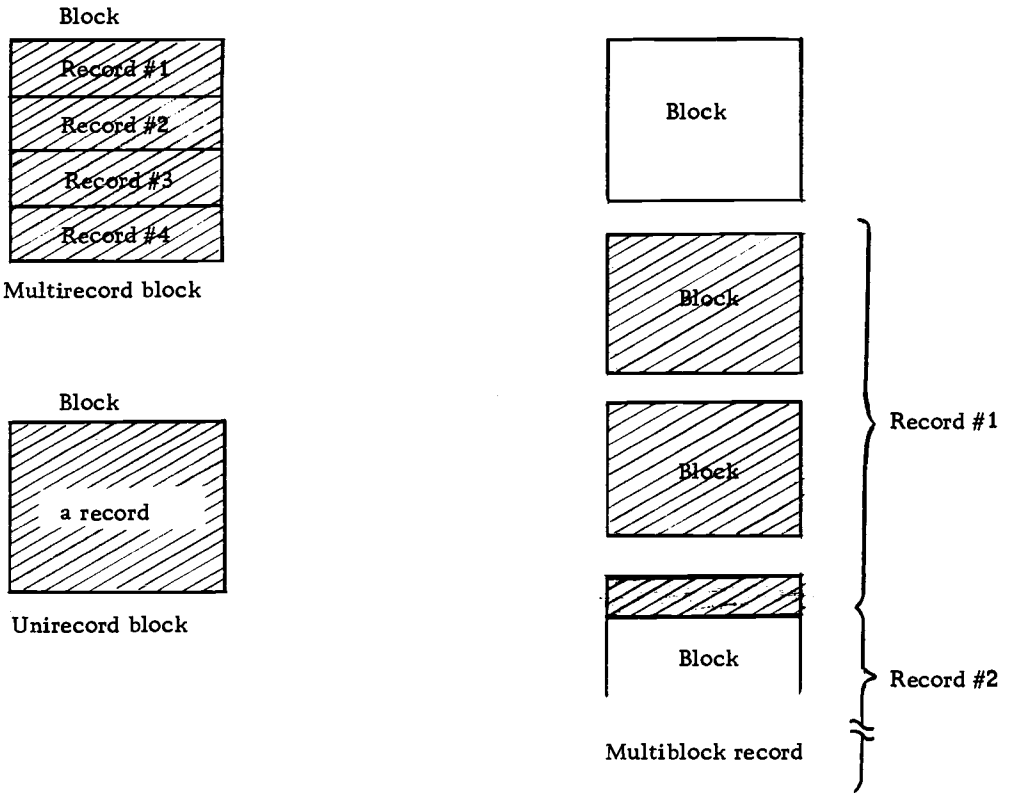


Figure 2.4. Block-record interaction.

Word-Field: The three relationships which may exist between a word (computer word) and a FIELD. See the illustration in Figure 2.5.

1) A Multifield Word. Several small fields are fit into one word.

2) A Multiword Field. A field occupies several words.

3) A Unifield Word. A word fits exactly into one field.

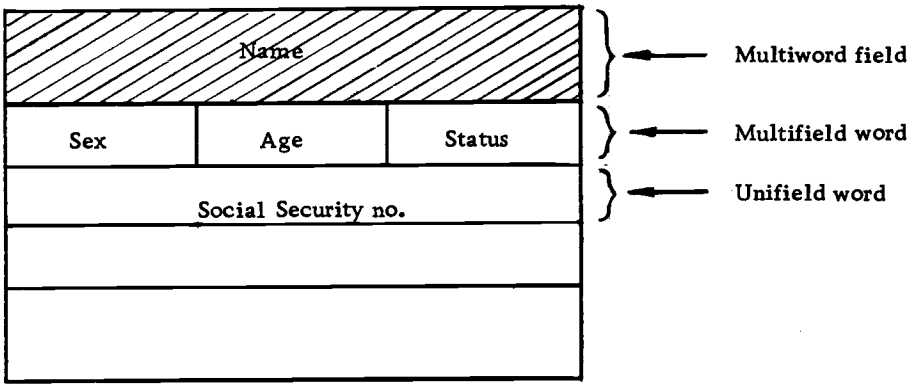


Figure 2.5 Word-field interaction.

Methods of File Organization

There are four conventional methods of file organization for direct access devices, to be used with disks for on-line computing systems. These are described briefly in this chapter and discussed fully in Chapter III through VI.

Sequential Organization

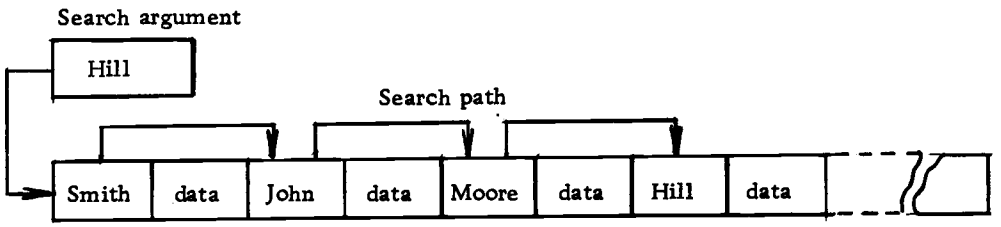
In a sequential file, the data records are organized in sequence

according to their successive physical locations in the file. There are two types of organization of the records in Sequential File.

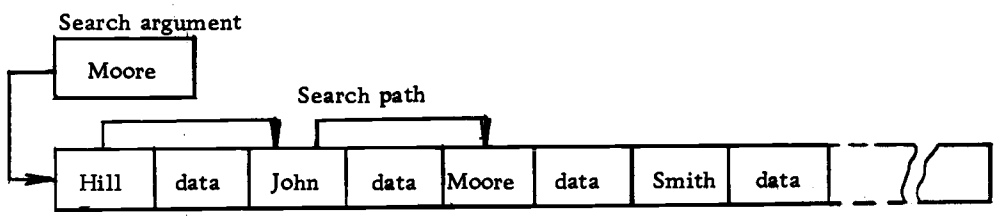
(1) In the first case the logical records are kept sequentially in the file, but not in a sequence according to their key; this type of file is called an "unsorted file". For example, the records of the magnetic tape transaction file are not in any key sequence. (2) In the second case the logical records are kept in a key sequence in the file; this type of file is called "strictly sequential file" or "sorted file". An example is the magnetic tape master file. The records in the file are kept in a sequence according to their key. For the conversion of an unsorted file into a strickly sequential file, a sorting operation is used. Usually the sorting operation is performed off-line. The records of the strictly sequential file are organized in a sequence according to their key so that the file can use either linear search or binary search to access an individual record from the file. Binary search provides fast accessing of a record from the strickly sequential file as the file size is increased, while unsorted file can make use of only linear search, and performs slowly in accessing an individual record from the file as the file size is increased.

Within a strictly sequential file, the records are usually read or updated in the same order in which they appear, and cannot be deleted or added to unless the entire file is rewritten. This type of

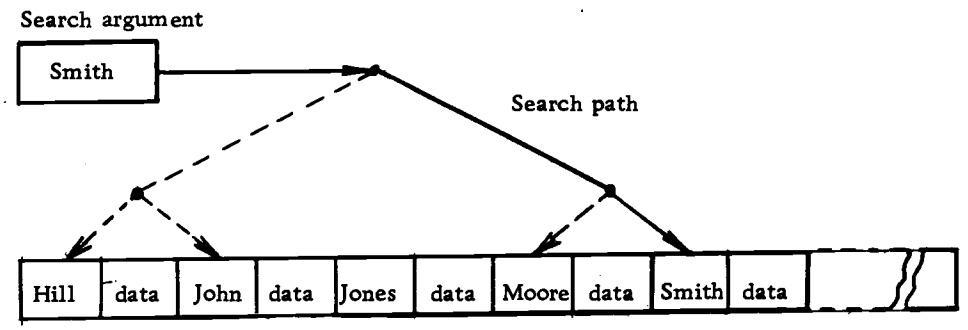
file organization is commonly used when most records are processed each time the file is used, such as for a payroll. For unsorted file, the records are read or updated sequentially according to their successive physical locations, and cannot be deleted without repacking, but provides the simple addition to the first empty space in the file. See the illustration of unsorted file and strictly sequential file in Figure 2.6.



(a) Accessing a record from unsorted file by linear search



(b) Accessing a record from strictly sequential file by linear search.



(a) Accessing a record from strictly sequential file by binary search

Figure 2.6. Accessing a record from sequential file.

The details of sequential file organization are covered in Chapter III and sample calculation Example Ia and Ib in Appendix B.

Indexed Sequential Organization

An indexed sequential organized file is a strictly sequential file associated with a reference index as its directory. The addition of an index to the file system provides rapid access to individual records in both sequential and random processing. For example, an indexed sequential file is supported by Direct Access Storage Device (DASD), Disk, and it may have a cylinder index stored on cylinder zero, which contains the address of the highest key of the record on any cylinder. This index will point to the correct cylinder. When the access arm reaches the desired cylinder, it will search a track index to determine on which track in that cylinder the desired record is located (see Figure 2.2). Separating the areas of the file permits the user to add a new record or delete the desired record to the file without rewriting the entire file, as is necessary in the sequential file. Although the added records may not be physically in key sequence, the indexes automatically organize them in key sequence. In this type of organization, the programming system has control over the location of each record. The user needs to only supply the key name of the desired record. The operating system takes care of all searching of the indexes, link field and either

presents the specified record to the user or notifies that it cannot be found. See more details in Chapter III and Example 2, Appendix B.

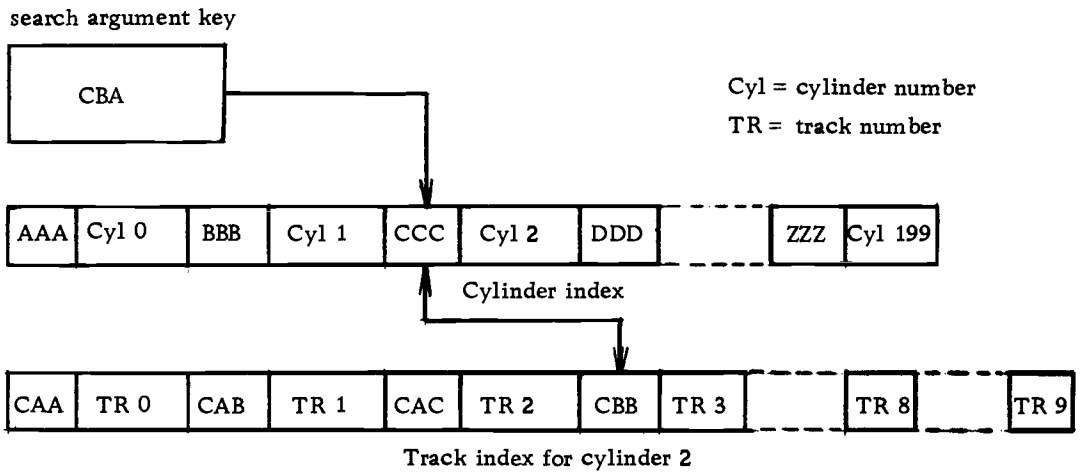


Figure 2.7.. Indexed sequential record accessing.

Partitioned Organization

In partitioned organization, the file is divided into several parts or "members", within each of these members, the data records are organized in strictly sequential file fashion. A "directory" is used with this type of file organization and is of the same nature as the index used with the Indexed Sequential File. This is done to provide fast accessing of the random record from the file. The directory contains the names and addresses of the members within the file, which are arranged in alphabetical sequence. Members may be added or deleted as required. In accessing an

individual record from the file, the user has to supply the name of the file he wishes to process (name of the desired member) and the directory is searched for "equal". The address of the corresponding name is used as the pointer, and the access arm can be positioned directly to this location (in case a disk memory is used). In some cases this partioned organization file may be used with a multilevel directory. Partitioned organization is used mainly for the storage of sequential data, such as sub-routines, table, data items for warehouse. See details in Chapter IV and Example 3, Appendix II.

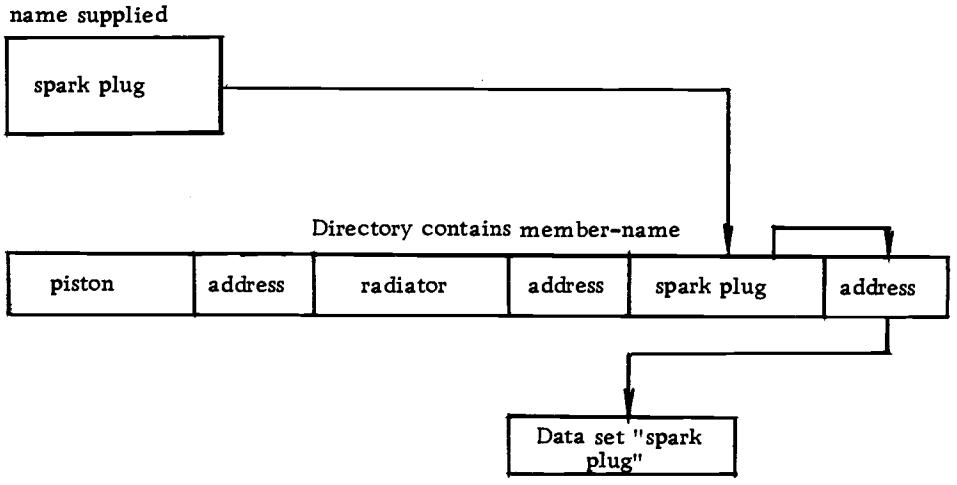


Figure 2.8. Accessing partitioned records.

Direct (Random) Organization

The Direct (Random) Organization file permits the user to add a new record, to delete a desired record, or to access the individual

record directly from the file, by using a single step, directory decoding technique (randomizing or Hash coding technique). The record-name is converted into a unique address, to which the device, may position itself and find the desired record. The programmer establishes the relationship between the record-name and its address on the supporting devices. In this organization method, the records stored in the storage devices are not in any key sequence. They may be distributed all over the supporting area depending on the nature of "mapping function" (Hash function). The major advantage of this type of organization file is that it provides very fast accessing of the individual record. It is possible to locate any record in the file with one seek and one read, in case it is disk file, which satisfies the features of the on-line information system. The disadvantage is that it is not economical to implement this method of organization when the file loading factor is too small. Although the programming system provides the routines to read or write a file of this type, the user still has great responsibility for the programming required to locate the desired records, because the user himself establishes the relationship between the record-name and its address on the storage device. See details in Chapter VI and Example 4, Appendix B.

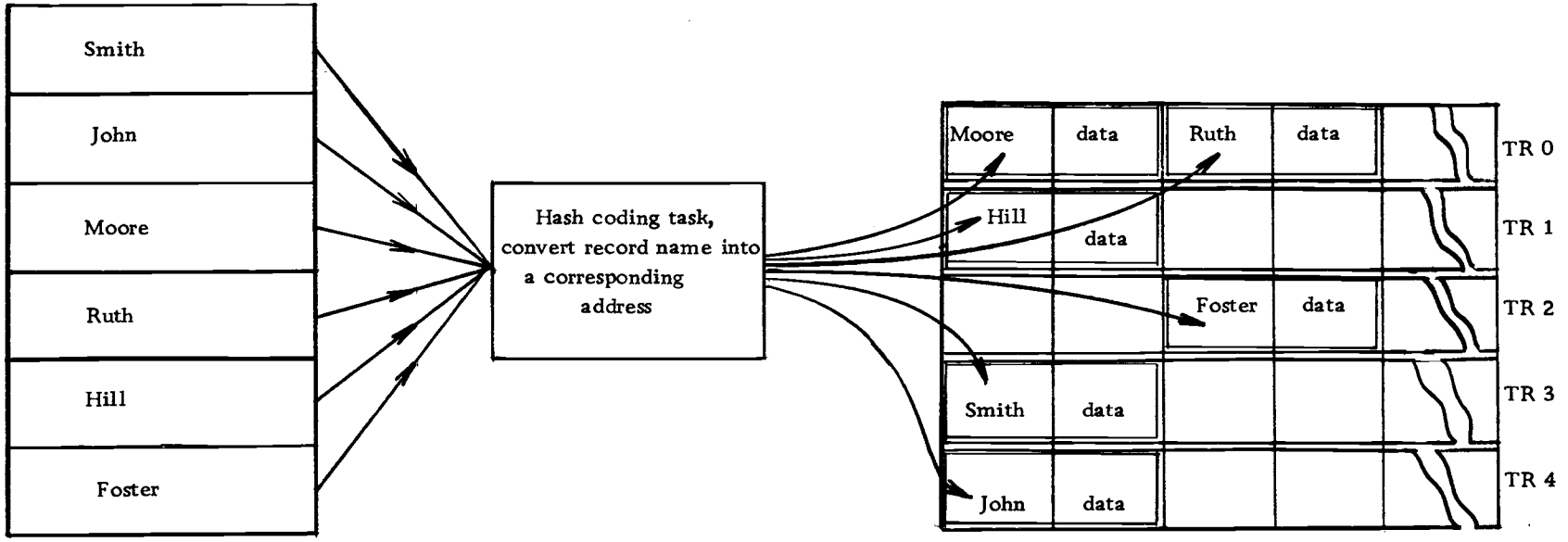


Figure 2.9. Creation and accessing records with direct organization file.

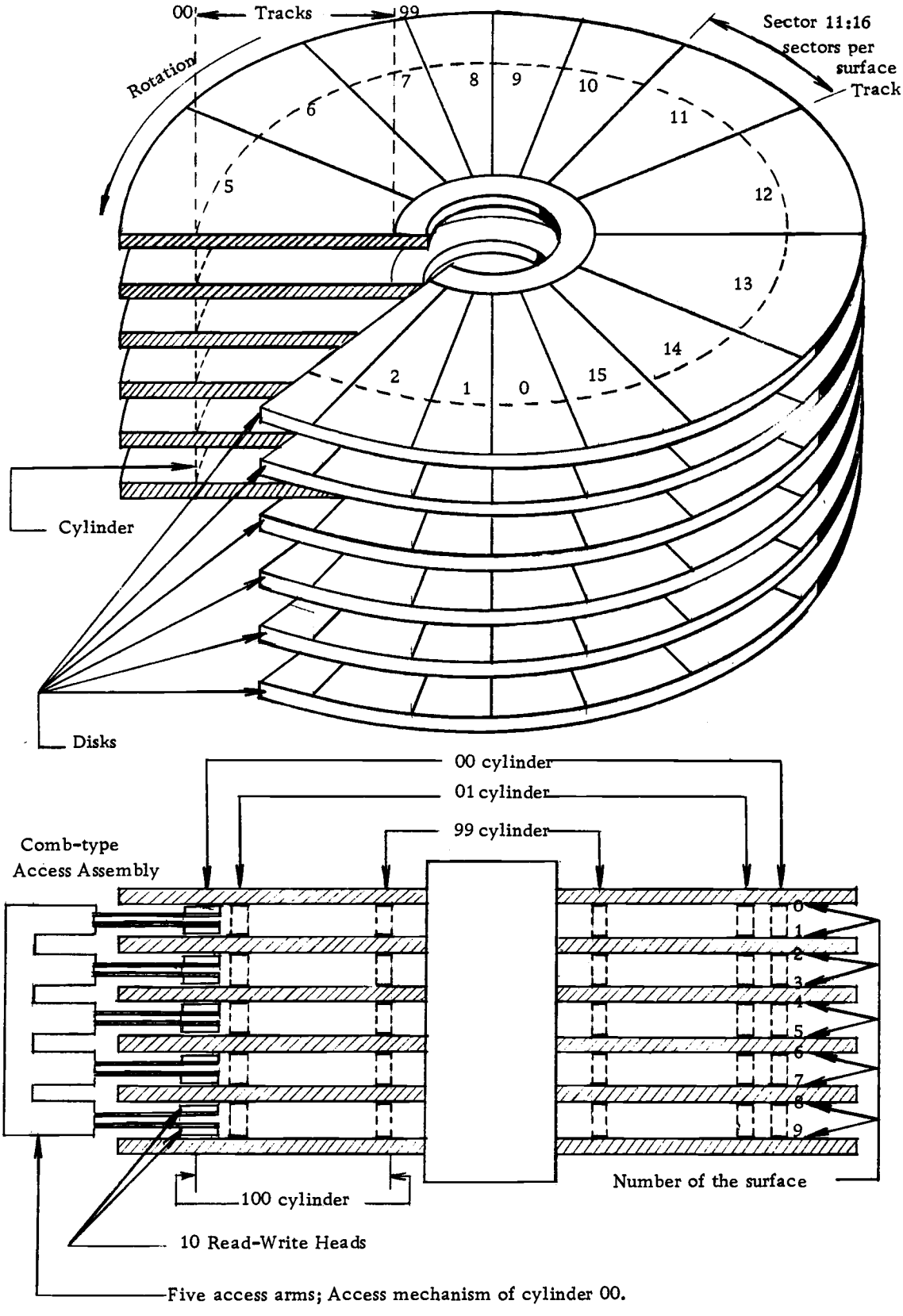


Figure 2. 10. Disk pack, Read-write heads and Cylinder Concept of data recording.

III. SEQUENTIAL FILE STRUCTURE AND USE

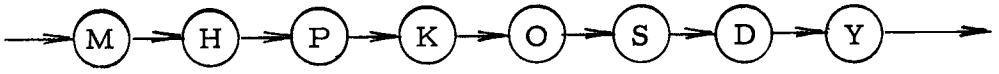
Sequential File Structure

The two conventional types of sequential file organization can be illustrated as follows.

(1) The records in the "unsorted file" are organized solely on the basis of the successive physical locations in the file.

(2) The records in the "sorted file" or "strictly sequential file" are organized in the key sequence either numerically, alphabetically or both.

A "node" can represent the item or record in the file, as shown in Figure 3. 1.



(a) Diagram of the unsorted file.

(b) Diagram of the strictly sequential file (alphabetically).

Figure 3. 1. Illustration of Sequential File Structure.

Sequential File Maintenance

Maintenance of sequential files of the system is done in the

same logically straightforward manner that the clerks use to maintain their sequential files. The system permits addition, deletion and alteration algorithms as follows:

The Algorithm of Unsorted File Maintenance

A new record can be added to the unsorted file.

Step 1: A search is made for the first empty space which may be located between the records in the file in case the file is "loose list", or at the end of the file in case the file is a "dense list" (9).

Step 2: When the first empty space is found the new record is added to the file.

Step 3: If an empty space is not found in the file, an overflow occurs, and an extension area must be introduced.

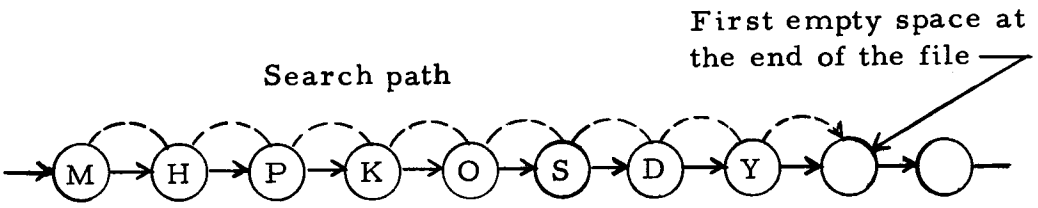
A record in the file can be deleted, or its data can be updated.

Step 1. A search is made for the desired record in the file.

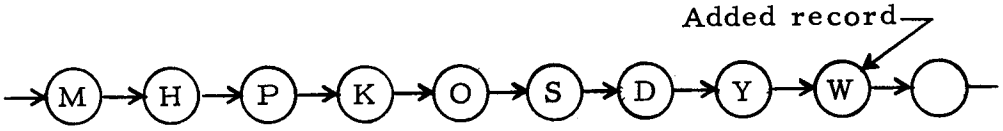
Step 2. If the desired record is found, then desired record is deleted or its data is updated.

Step 3. If the search is not satisfied until the end of file mark is found, the system has to notify the user that the desired record is not there.

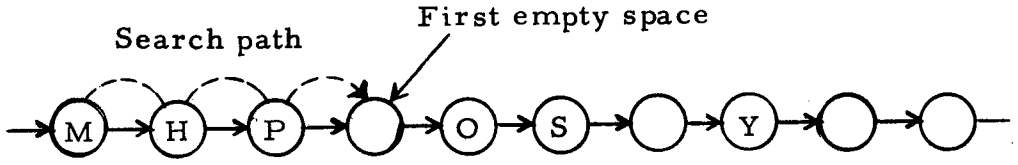
The record in the file whose key name is "S" in Figure 3.1 may need to be deleted or updated.



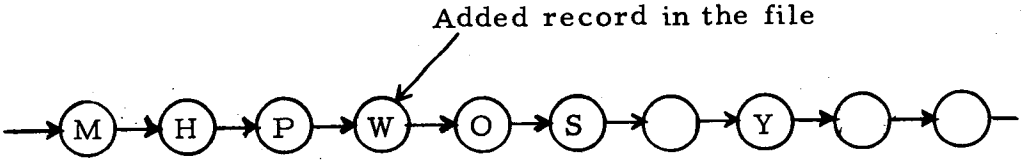
(a) Search is made for the first empty space, and a record whose key name is "W" is added into the file in Figure 3.1 (a).



(b) Graphical representation after record with key name "W" is added to the file.

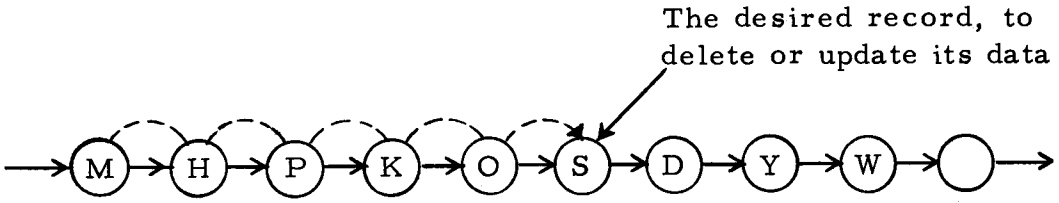


(c) Search for first empty space to add a record whose key name is "W" to the file.

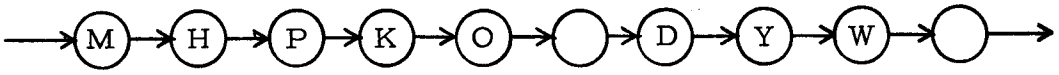


(d) Graphical representation after record with key name "W" is added to the file.

Figure 3.2. Addition of a new record to an unsorted file.



(a) Graphical representation of unsorted file, after desired record is found.



(b) Graphical representation of unsorted file after desired record is deleted.

Figure 3.3. Deletion of the record key name "S" from the file.

After deletion of the record with key name "S" is completed, the file structure is changed from density list to loose list.

The Algorithm of Strictly Sequential File Maintenance

A new record can be added to the strictly sequential file.

Step 1. A search is made for the proper location for the new added record in the file, based on the key sequence possessed by the file.

Step 2. If the proper location in the file is found the rest of the records can be shifted up one position to a higher address in the file to provide the proper location for adding a new record to the file. In some cases a part of the rest of the records including file mark in

the file have to move into the new extension area of the file, and the new extension area is now considered as a part of the body of the main file.

Step 3. If the search is not satisfied until the end of the file mark is found, the system has to notify the user that an empty space is not available.

A record is deleted from the file or its data are updated.

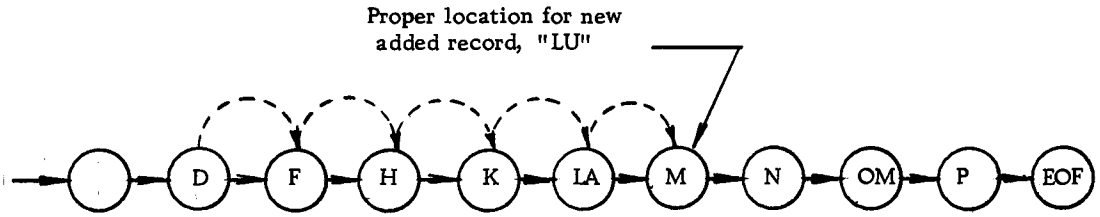
Step 1. A search is made for the desired record in the file.

Step 2. If the search is satisfied, the desired record will be added or its data updated depending on which is required.

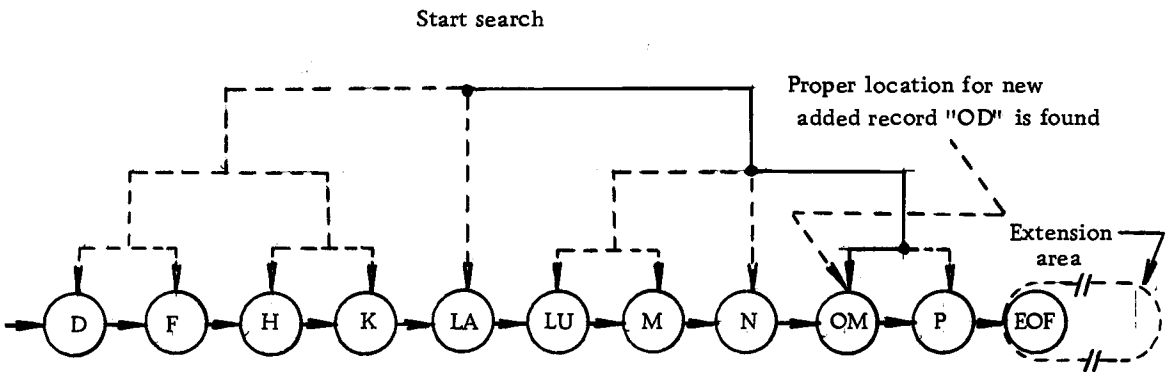
In case the user needs to delete the desired record, all information of that record is erased; the space is blank and all of the records which possess a higher key than that of the deleted record have to shift down successively one position, including the file mark, repacking.

In case the data of the record needs to be updated, then the information is to be read out.

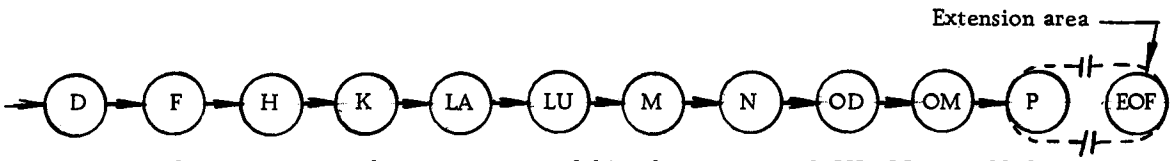
Step 3. If the search is not satisfied, and the desired record cannot be found in the file, the program will notify the user "no such record in the file".



(a) graphical representation of strictly sequential file after the proper position for the new added record is found

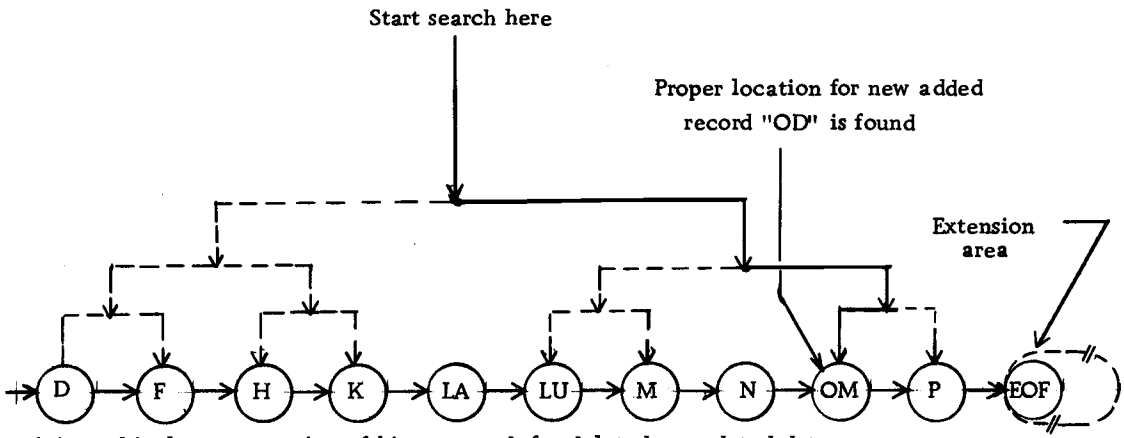


(b) graphical representation of strictly sequential file after binary search is applied and the proper position of new added record is found

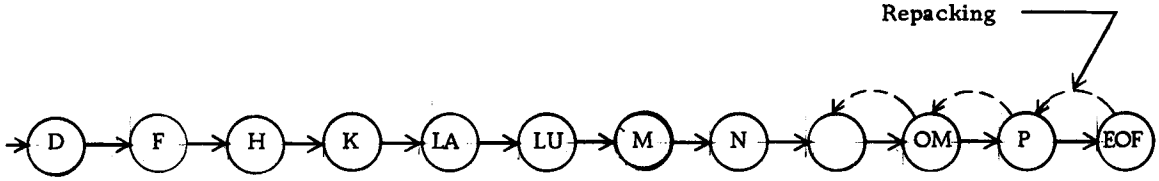


(c) graphical representation of strictly sequential file after new records LU, OD are added.

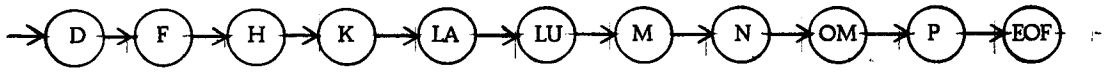
Figure 3.4. Addition of new records to strictly sequential file.



(a) graphical representation of binary search for deleted or updated data



(b) graphical representation of strictly sequential file after record OD has been deleted



(c) graphical representation of strictly sequential file after the file is repacked.

Figure 3.5. Deletion and repacking of strictly sequential file.

Sequential Disk File Use for On-Line System

The characteristics of a disk memory and its cost satisfy the requirements of the on-line processing system. The analysis of disk memory has been done by Dr. David Lepvitz (18). Consequently the disk memory system is used as an on-line supporting storage device.

In a sequential file supported by Direct Access Storage Devices (DASD) disk, records are written one after the other, track by track, cylinder by cylinder at successively higher addresses. The records are usually in key sequence.

Description of Records in File

Actually records may be fixed or variable length, blocked or unblocked or undefined. The records may be formatted with or without keys. In case the file is processed (Processing file) sequentially, as is normally the case with this method of organization, there is no need for formatting with keys. If for some reason there is an appreciable amount of random processing, records should be formatted with the key so that they can be located more quickly.

In the example in this thesis, commonly formatted records, fixed-length records, blocked, are considered (see Figure 9.2, page

Space Storage Requirements

The amount of space disk storage required must be enough to hold all the records in the file. Although it is permissible to have the file extended, actually the space requirement is directly proportionate to the number of records in the file.

Methods of Using Sequential Disk File

The sequential file provides the user with two options in processing.

1. Sequential Processing. The time required is one search per cylinder and one read per record (or block of records).
2. Random Processing. If a sequential file is processed randomly, it is, at best, very inefficient. In case it is used infrequently, the time required to locate the records may not matter. There are several ways to program random processing with significant differences in the time required.
    a) One possible way is to read the records sequentially until the desired one is located, but it is the slowest method. On the average, half of the file would have to be read. A sequential search takes less time if the records are formatted with the key. The search is

done only on search Key High or Equal to the speed of one per track revolution. When the search condition is satisfied, the corresponding record is read.

b) Another method of processing a sequential file in a random fashion is first to perform a binary search of the file in order to determine in which small section of the file the desired record is located. Then only that small section need be searched in full. In Figure 3.6 an eight cylinder file formatted with key is illustrated.

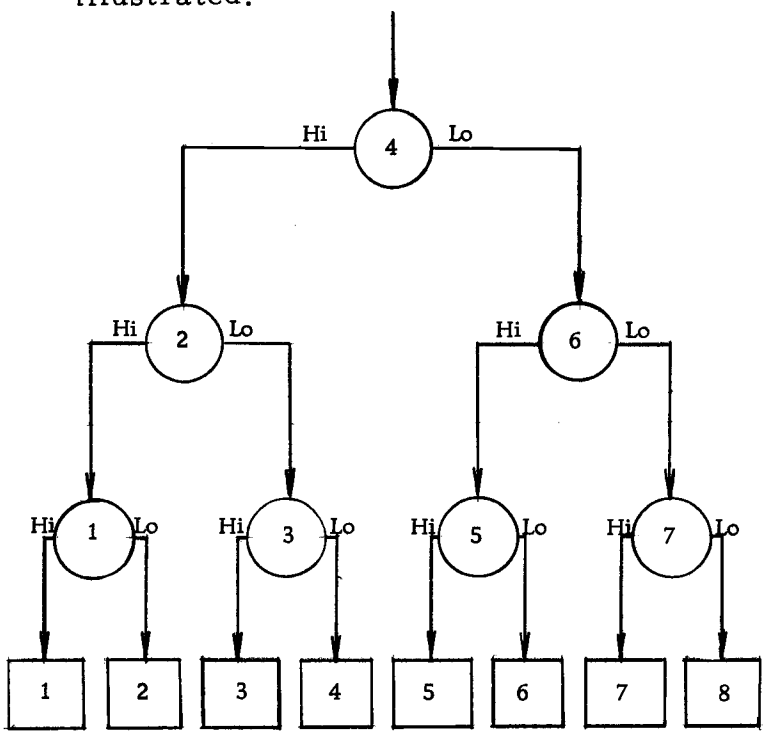


Figure 3.6. A binary search of an eight-cylinder file, multi-cylinder file.

The last record in cylinder 4 is read and compared with the search argument; then the last record in cylinder 2 or 6 is read and a comparison is made again. Then, depending on the result of that comparison, the last record in either cylinder 2 or 6 is read and a comparison performed again. Then, depending on the result of the comparison, the last record in either cylinder 1, 3, 5, or 7 is read and compared with the search argument. This last comparison indicates in which one of the eight cylinders the desired record is to be found. Only that cylinder need then be searched in full.

Sequential Disk File Maintenance

The maintenance of sequential file supported by DASD, disk is straightforward, as shown previously in the graphic illustration. Additions and deletions require a complete rewrite of a sequential file. This is desirable from a timing standpoint only in case addition and deletions can be combined with another job that also requires reading and updating all the records.

Uses of Sequential File for On-Line System

Sequential file organization is used on direct access storage devices primarily for tables and intermediate storage rather than for master files. It can be used as a master file if there is a high percentage of activity, and if virtually all processing is sequential.

Evaluation of Accessing Characteristics of Sequential Disk File

For some purposes in the comparative study of file organization, the characteristics of sequential file accessing have to be measured.

The strategy of evaluation presents two problems:

1. The systems disc file cannot be independently controlled.
2. Due to the system characteristics, OS-3 runs under the influence of this user queue, with resource allocation and I/O dependent upon the list structures connected to a particular "program status area" (PSA). At any given moment, the system is processing a single user whose PSA is indicated by a pointer. Alteration of the contents of this pointer occurs at the end of discrete time intervals (16).

According to the problems mentioned the system cannot permit measurement of the exact total time used in executing the program.

To solve problem 1. Actually the Test File is created in the disk memory, then it is read into core memory and the simulation of file processing is performed. But in the simulation of the sequential file, the test file is created directly in core memory as the initial stage, and the file simulation is then performed.

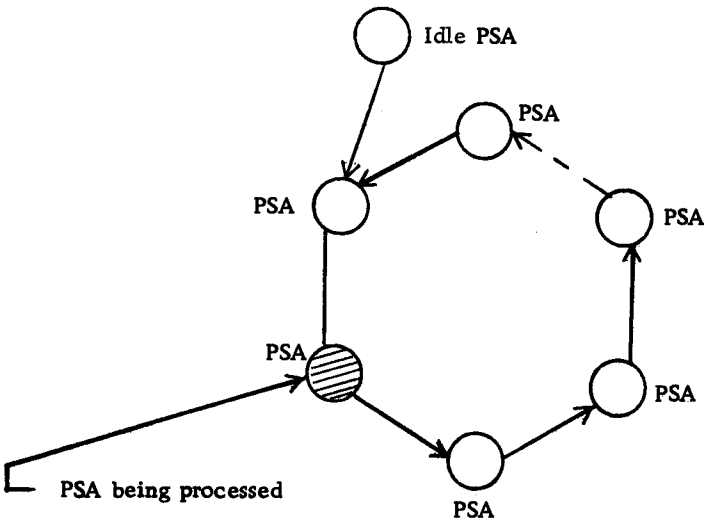


Figure 3.7. Graphical representation of User queue for CDC-3300.

To solve problem 2. "Compass," symbolic language is used for simulation. The total executing time for each program is obtained by direct computation, that is by adding the relative instruction time together according to the logical path in the "compass" program. This method of obtaining the executing time is also recommended because the result is more accurate than by directly measuring executing time.

The Purpose of the Evaluation

The purpose of this evaluation is to measure the following Sequential File parameters, and to ascertain the characteristics

of the system.

1. To measure the internal searching time per record retrieval with both Linear search and Binary search.
2. To compute the access time when the record is located on the disk.
3. To measure the space storage requirement.
4. To find the characteristics of average througput time/ record retrieval, achievable throughput rate capability.
5. To find the COST/EFFECTIVENESS characteristics, (customer operating cost per call).
6. To compare the characteristics with the other methods of file organization technique.

The philosophy of accessing characteristics of a Sequential File simulation can be illustrated as in Figure 3.8.

1. Each logical record in the file is considered to be fixed, blocked, and associated with a key. See detail in Figure 9.2, page 202.
2. The file is stored on D 854, disk memory, one W/R head per disk surface by use of cylinder concept, see Figure 2.10, page 15. The records are stored in contiguous areas, track by track, cylinder by cylinder, not in key sequence for the unsorted file, and in alphabetical sequence for the strictly Sequential file.

Simulating Block Diagram Model

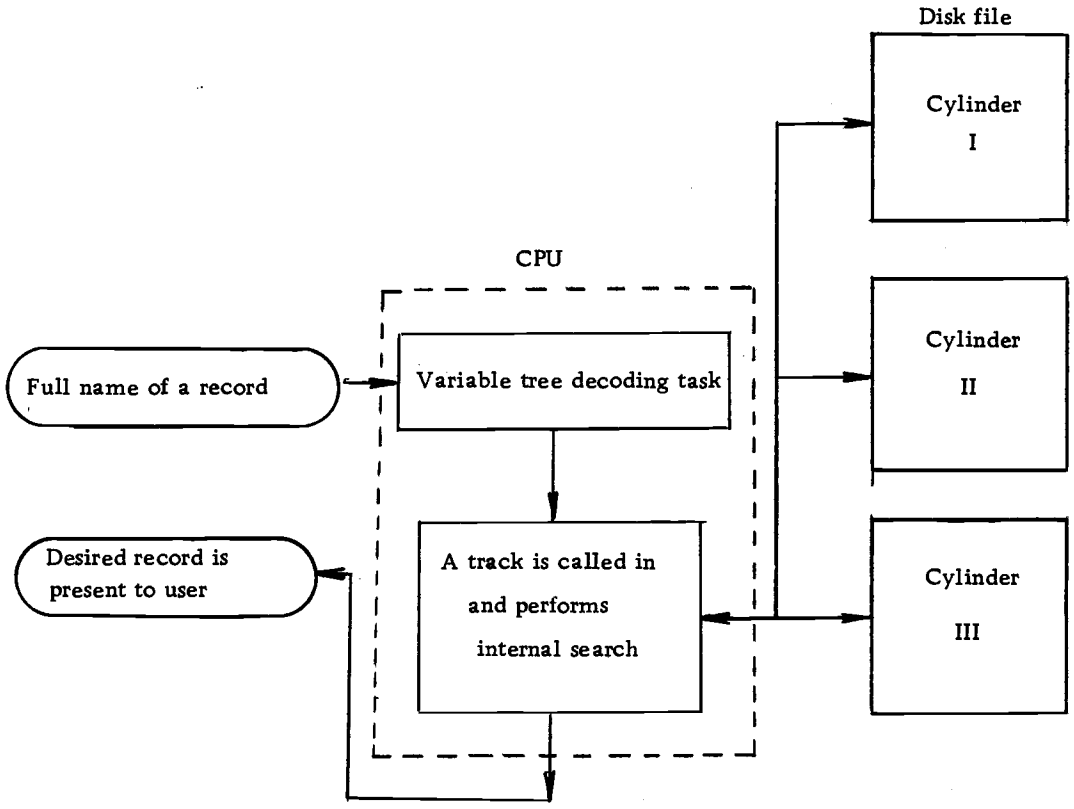


Figure 3.8. Block diagram showing the simulating of accessing a record from Disk Sequential File.

3. For accessing the desired record from the file, the full name of the desired record is applied to the system. The Tree with variable length key directory decoding program converts the full name of the desired record into one word, a uniquely fixed length key. The operating system can use the fixed length key as search argument.
4. Since for each track of Sequential File there are 64 logical records, it is more efficient to select linear internal search. See the analysis of internal linear and binary search on pages 193 - 197. So that a Sequential track is called in and linear internal search is performed for both unsorted and strictly Sequential File.
5. If the desired record is found, the operating system will present it to the user.
6. If the desired record is not found, the operating system will notify the user that it is not there. See the details in Example 1, Appendix B.

Results of Sequential Disk File Accessing Characteristics

The results of this simulation are shown as follows:

1. Figure 8. 1. The comparison of searching parameter (number of look-up per record retrieval vs. file size,

between Linear search and Binary search).

2. Figure 8.2 shows the result of the comparison of searching parameter (searching time in msec. per record retrieval vs. file size) between Linear search and Binary search.

3. Figure 8.4 shows the result of the comparison of core space requirement number of computer words vs. file size) between Linear search and Binary search.

4. Figure 7.1 shows the result of computation of retrieval time per record retrieval (throughput time) when the main records are located in disk memory.

IV. INDEXED SEQUENTIAL FILE STRUCTURE AND USE

Indexed Sequential File Structure

The indexed sequential file organization is in use by several computer manufacturers. It is an automatic file management and access method. The indexed sequential file has the same basic structure as the partitioned file, which uses the "directory" system in logical hierarchical relation with the "main file". However in the indexed sequential file the directory and the record in the main file are in physical relation according to position. In general this file is designed to use with DASD, especially disk. Hence, the basic structure of the indexed sequential file supported by disk is as follows: (see the Figure 4. 1). In case the processing of the file is based on alphabetic key, the file system has to be attached to a directory table for the variable length full name of a record to be converted into a unique fixed-length key name. Extra time is needed for decoding by searching the directory table for each random access record.

On-Line Indexed Sequential File Supported by Disk Memory

An indexed sequential file has three major components: Index Area, where the cylinder index and track index are located. Prime

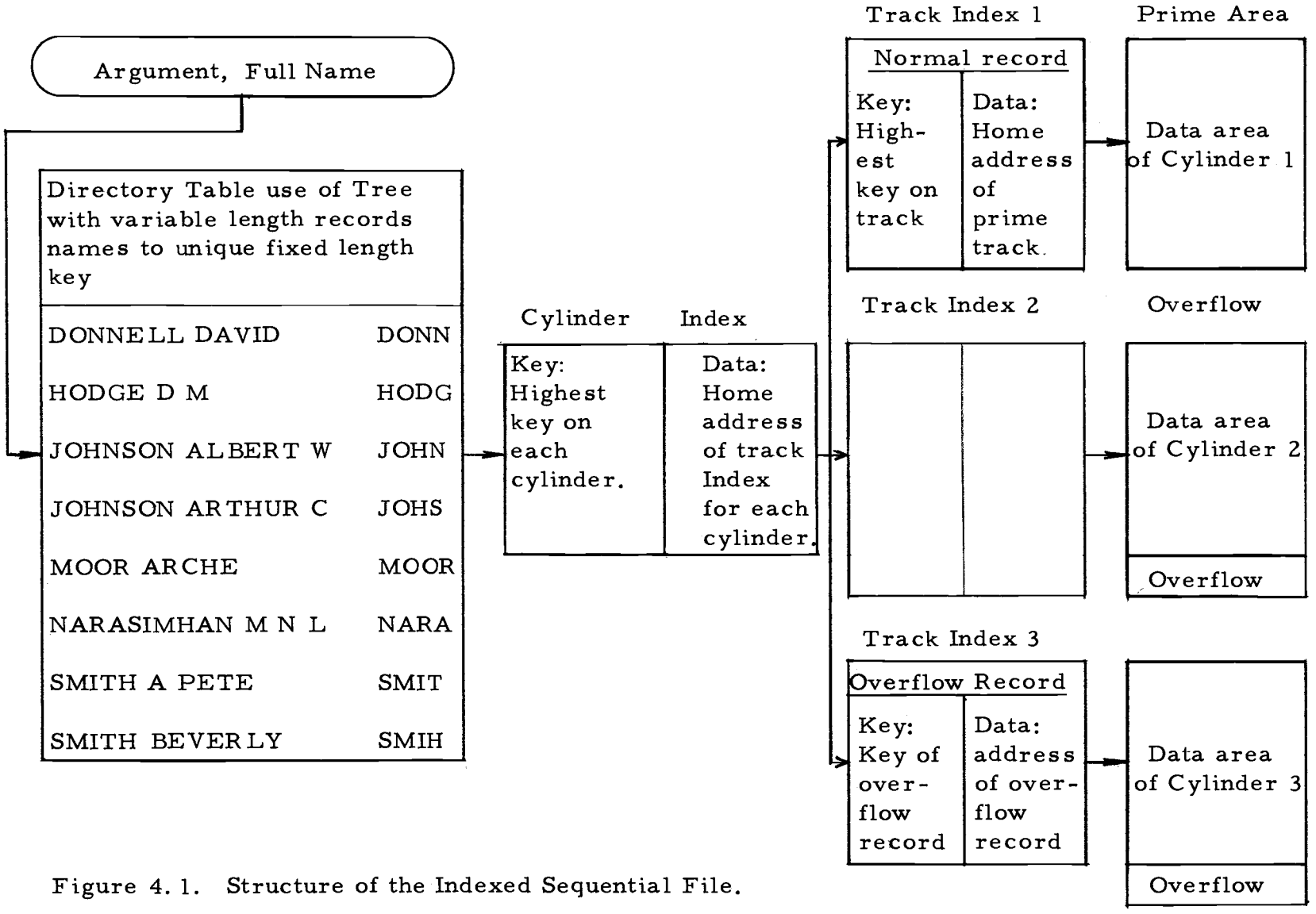


Figure 4.1. Structure of the Indexed Sequential File.

Area, where the normal records of the file system are located, and Overflow Area, where the overflow records of the main file are located.

The Cylinder Index is the "master directory" of this file system. There is only one cylinder index. It is the higher level index and is always present. Somewhere in the indexed sequential file system, its entry contains a pointer to a track index. Sometimes the cylinder index may be on a different type of disk than the rest of the file. In case it is assigned to the same disk module, the cylinder index should be put at the beginning of the file area (see Figure 4.2).

The Track Index is equivalent to a "subdirectory". This is the lowest level of index and is always present. Actually there is one track index for each cylinder in the main file area (prime area). Its entries point to data records and are always written on the first track of the cylinder that they index. Each track index may contain a special record called "Cylinder Overflow Control Record", COCR. The operating system uses this record when an overflow exists in the file system. The rest of each track index consists of alternating "normal" and "overflow" entries. There are a pair of entries for each data track in the cylinder.

The normal entry contains the home address of the prime track, and the key of the highest record on the track.

The overflow entry contains the highest key record on the track in the key area. The data area contains the home address of the overflow area which indicates "end of the chain." The entry is changed when a record is added to the file and overflow exists.

The last entry of the track index is a "dummy entry" indicating the end of the track index. The rest of track index may contain prime data records if there is room for them. In this case, the first pair of entries in the track index referes to this track.

Each track index entry, Normal, Overflow, Dummy, has the same format. It is an unblocked, fixed-length record consisting of a Count Area, a Key Area, and a Data Area. When a Key Area is as specified by the user, in the case of a normal or overflow entry, this area contains the key of the data record which is the entry point. However, the area of the dummy entry has all 1 bits, which is the highest in the collecting sequence. The Data Area is depended on the user's specifing for example that it is 10 bytes long, contains the full address of the track or record to which the index points and other information such as the level of index and type of entry (based on the IBM system).[1/] The Data Area of the dummy entry is null

---

[1/] The IBM system (16) has suggested that using of the Master Index when the cylinder index occupies more than four tracks. Master index is the highest level of index and is optional; it is used when the cylinder index is too time-consuming. It can be stored permanently in core memory.

(all 0-bits). For illustration, see Figure 4.4, page 43.

The Prime Area is the area in which data records are written when the file is first created or subsequently reorganized. Additions to the file may also be written in this area. The prime area may spand multiple cylinders, modules may consist of several non-contiguous areas. The record in the prime area has to be written in key sequence. Data records in the prime area must be formatted with key. They may be blocked or unblocked. In case a record is blocked, each logical record has to contain its key, and the key area contains the key of the highest record in the block.

The Overflow Area may be of one of two types, a cylinder overflow area or an independent overflow area. Either one or both can be used for an indexed sequential file.

a) The Cylinder Overflow Area is the most popular means of handling overflow records. A certain number of whole tracks, as specified by the use, are reserved in each cylinder for overflows form the prime tracks in that cylinder. When a cylinder overflow area is specified, record 0, the track descriptor record in the track index is used as a Cylinder Overflow Control Record (COCR). See illustration in Figure 4.2.

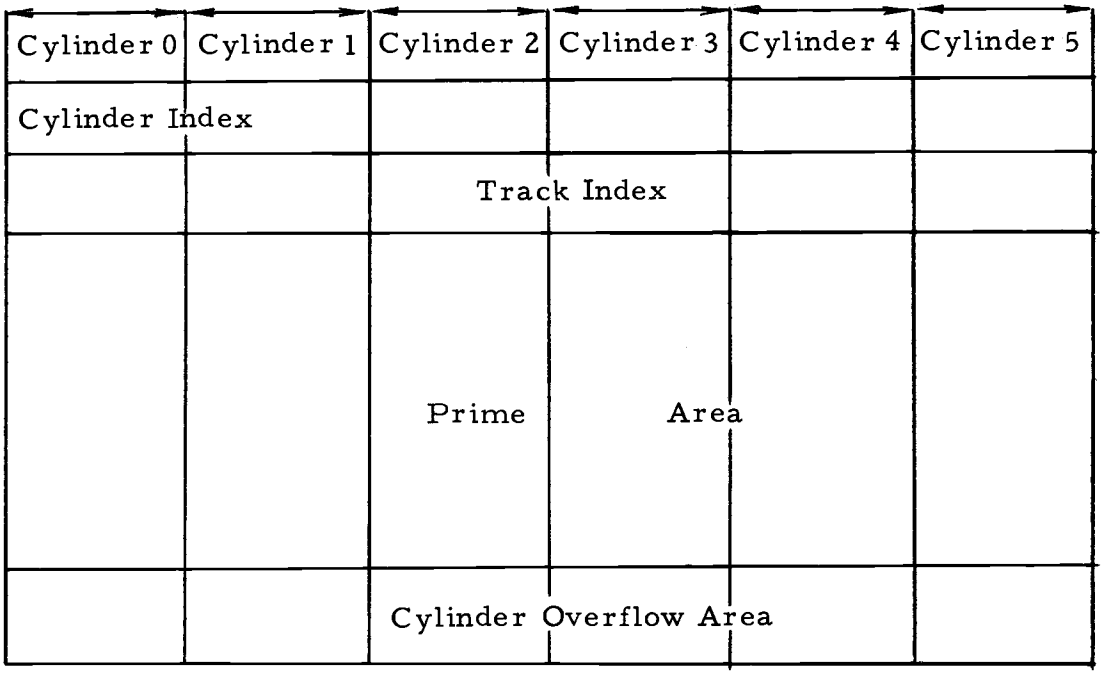


Figure 4.2 Illustration of Cylinder Overflow Area.

The operating system uses the COCR to keep track of the address of the "Last Overflow Record" in the cylinder and the number of bytes left in that cylinder overflow area. In some cases the Operating System uses this COCR for additional information. Two bytes of this COCR are used in case the file has variable-length records. If the file has fixed-length records these two bytes are left blank.

The advantage of having a cylinder overflow area is that additional searches are not required to locate overflow records. The disadvantage is that there will be unused space, if additions are unevenly distributed throughout the file.

b) The Independent Overflow Area provides a means of

placing the overflow records from anywhere in the prime areas in a certain number of cylinders reserved solely for overflows. The size and unit location of the independent overflow area are specified by the user. The area must, however, be on the same disk module as the prime area.

The advantage of having an Independent overflow area is that less space need be reserved for overflows. A disadvantage is that accessing overflow records requires additional searches.

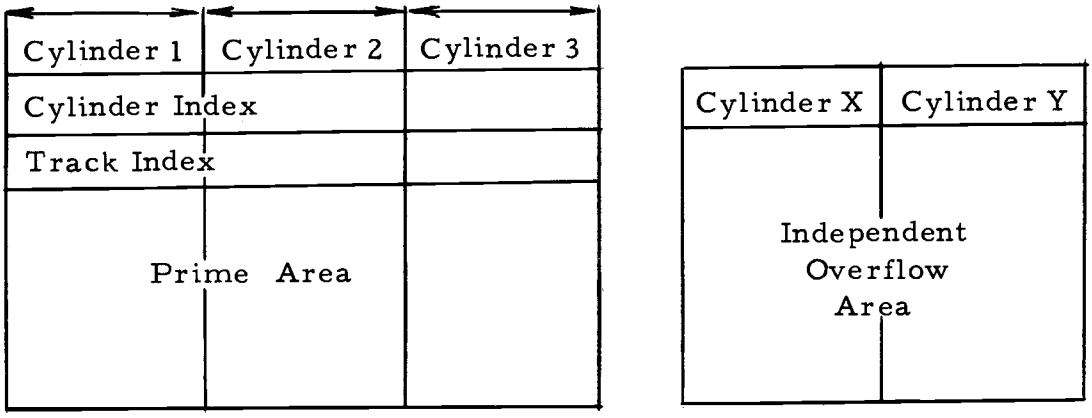


Figure 4.3. Illustration of Independent Overflow Area.

Overflow Records must be unblocked; they must be formatted with keys; they must be of fixed-length or variable-length. If the prime records are blocked, the key of an overflow record is contained in both the key area and the data area so that all logical records have the same format.

The first field of data area of overflow record is a link field, which is used to chain together in key sequence the records that have

overflowed from a prime track. The link field is usually ten bytes long and contains the same type of information as the data area of index entries. If an overflow record is not the last link in a chain, its link field so indicates and contains the address of the next overflow record in the chain. If an overflow record is the last link in a chain, its link field so indicates and points back to the track index. An overflow record has a link field, while a prime record does not. The overflow record is of significance to the user only in that the link field requires space on the disk and in core memory. The operating system presents logical records to the user in such a way that he does not know of the difference in format.

The Use of the On-Line Indexed Sequential Disk File

In the update and maintenance operations of the Indexed Sequential File, the prime record is of a fixed length and blocked. The cylinder overflow area is used to handle the overflow records.

Adding a New Record to the File

There are three cases of adding new records to the file, The standard algorithm of addition is as follows:

A new record is added to a "prime track", to create the file. See illustrated example in Figure 4. 5, page 44.

Step 1. The user supplies the name of the desired record to the system.

Step 2. The operating system decodes the full name of the record into a fixed-length record key name by means of one of the methods to be mentioned in Chapter V, under the section on Directory Decoding Technique. Now the unique record key name of a desired record can be decoded by the directory table.

Step 3. The operating system edits and searches through the cylinder index for a key "high" or "equal" in core memory.

Step 4. When the search is satisfied, the address in the corresponding data area of the record in the cylinder index is read.

Step 5. The reference cylinder is sought (hardware operation).

Step 6. The operating system edits and searches the track index, for a key "high" or "equal" in core memory.

Step 7. When the search is satisfied the address in the corresponding data area of the records in the track index is read.

Step 8. The reference track in that cylinder with key "high" or "equal" is sought.

Step 9. If the proper sequential location on the prime track is found, and it is empty, the new record is added there; the job is done.

Step 10. If the proper sequential location is found, and it is not empty on that prime track, the operating system will move the rest of the records higher on that prime track up one position; some of the higher key records in that prime track may be bumped out.

Step 11. The operating system will automatically continue the search for the first available space in the cylinder overflow area.

Step 12. If the first available space in the cylinder overflow area is found, the bumped records or the new record is written up in the available location in the overflow area, and the linked list concept is introduced. The record is placed in the cylinder overflow area for that cylinder. If it exists and if there is a space in it, the job is done.

Step 13. If the cylinder overflow area is full, the operating system will continue searching for the first available space in the independent overflow area.

Step 14. If the first available space in the independent overflow area is encountered, the bumped record or new record

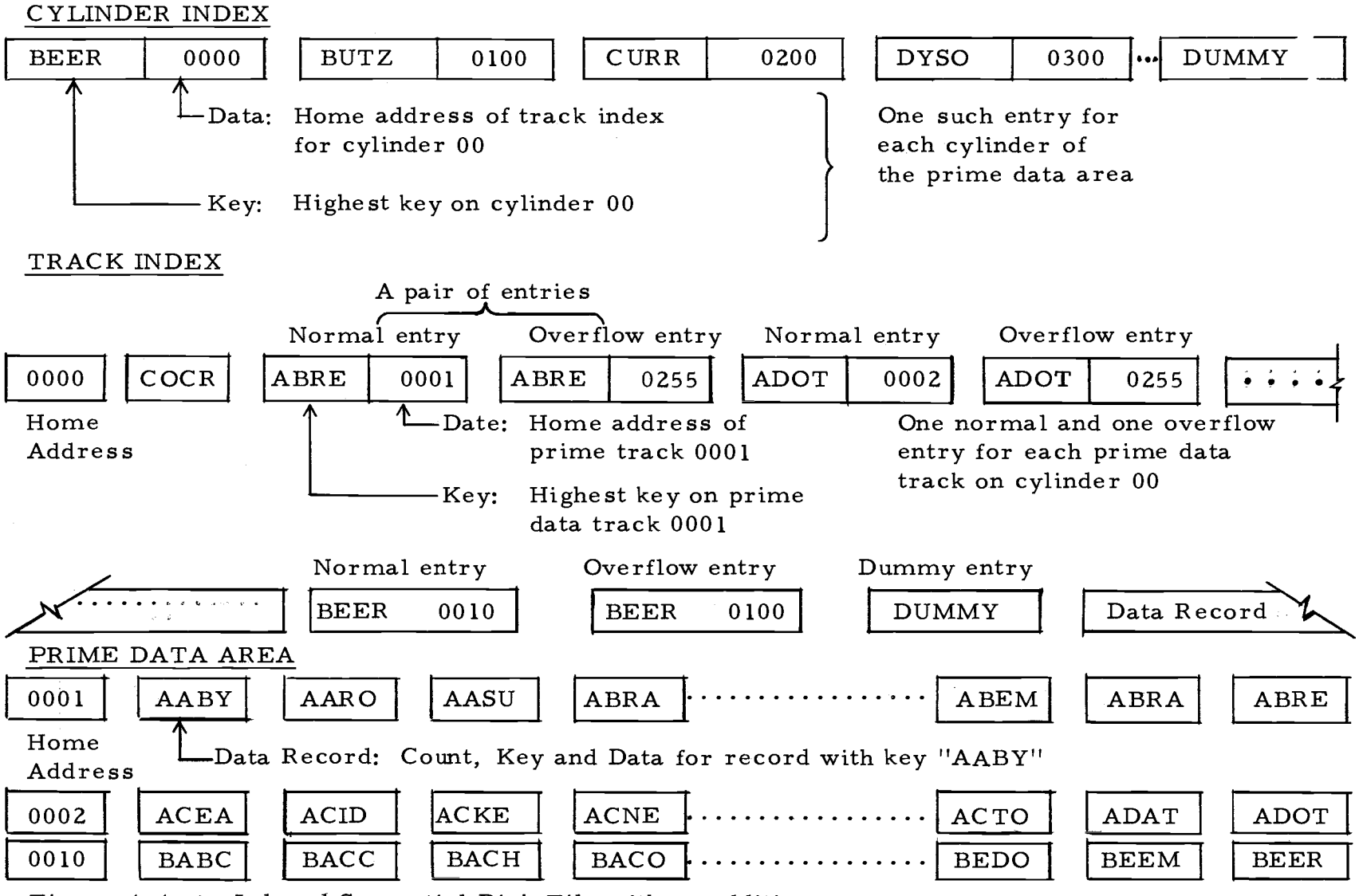


Figure 4.4. An Indexed Sequential Disk File with no additions.

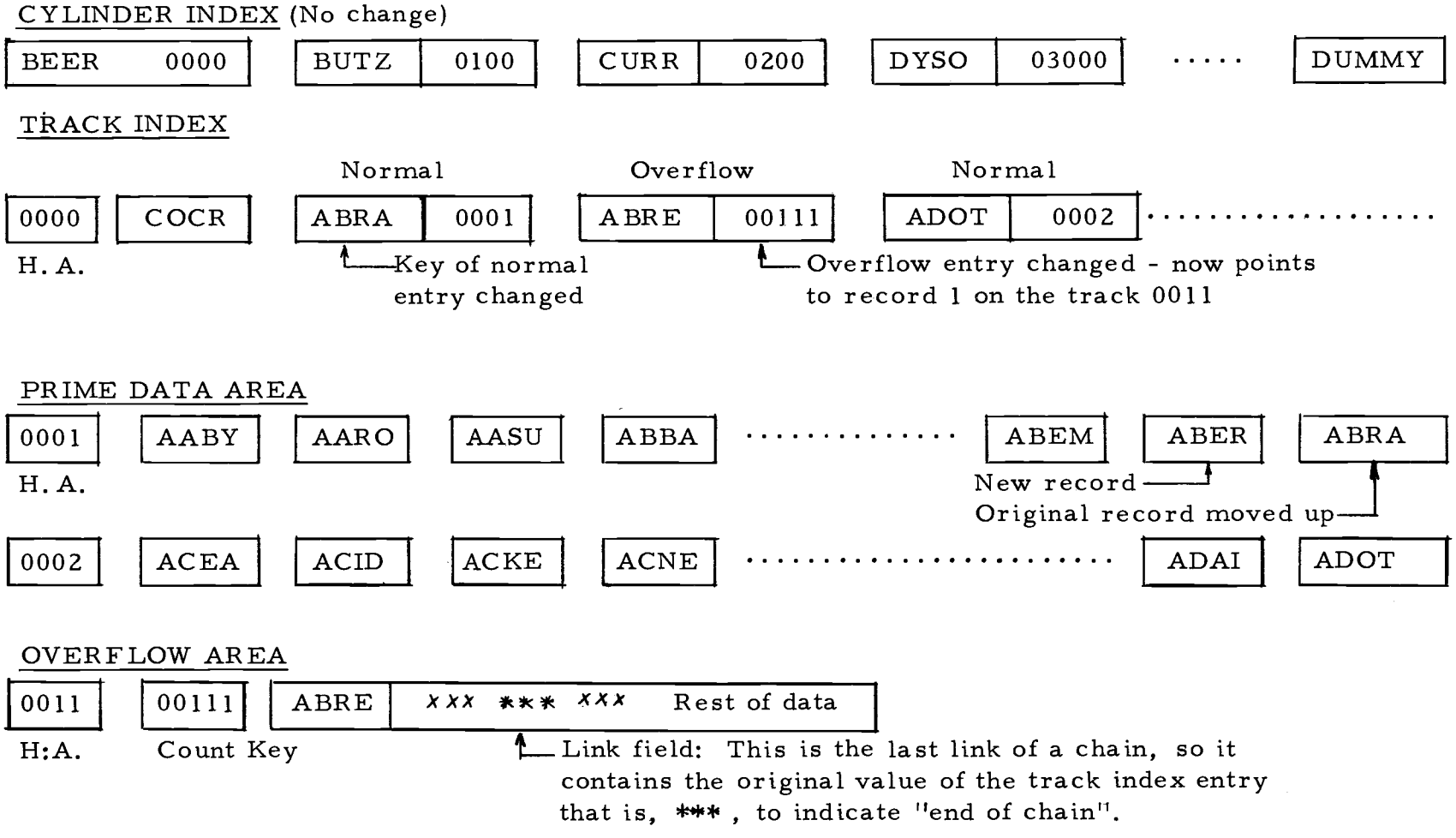


Figure 4.5. An Indexed Sequential Disk File after the first addition to a prime track.

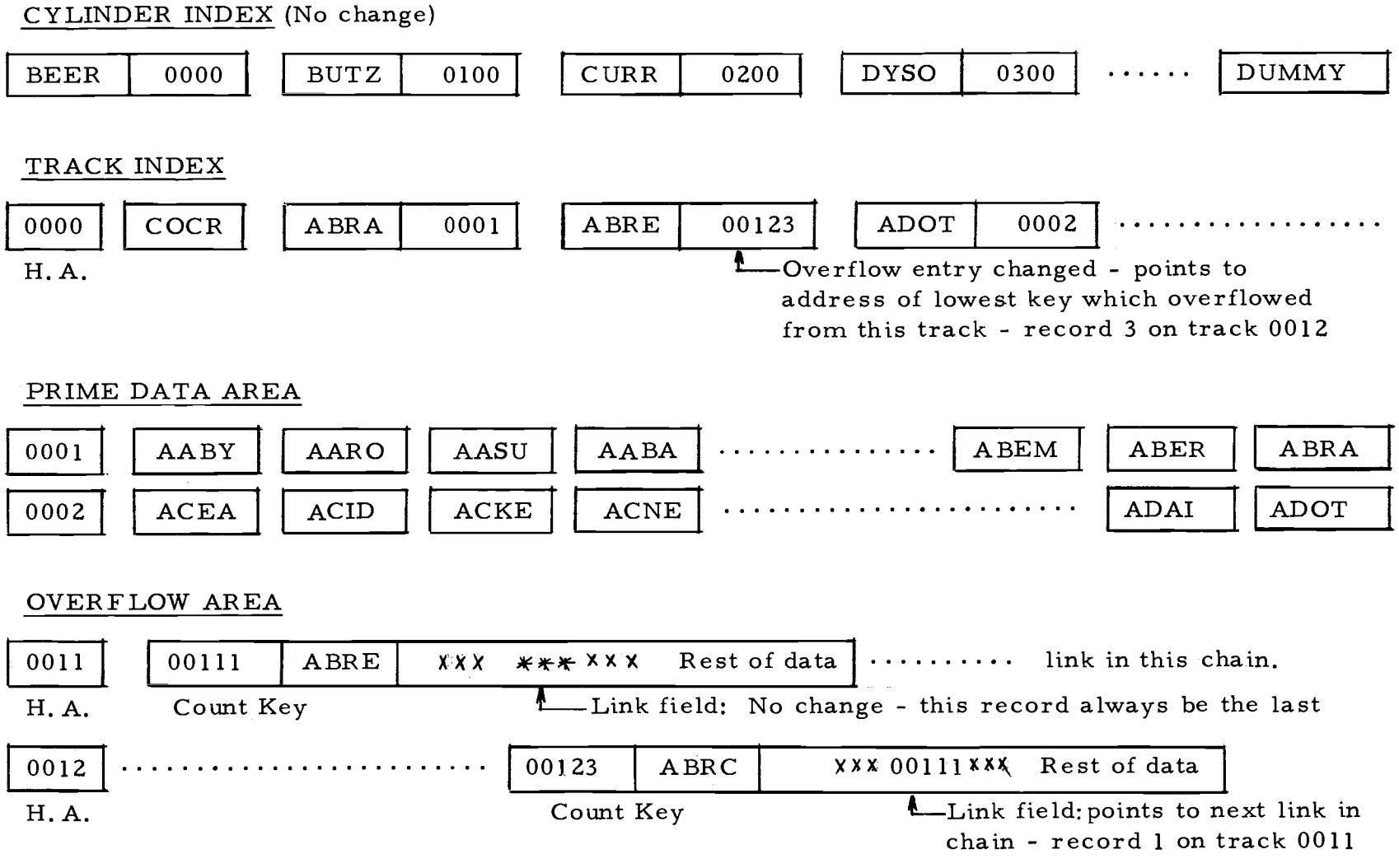


Figure 4.6. An Indexed Sequential Disk File after subsequent additions to a track.

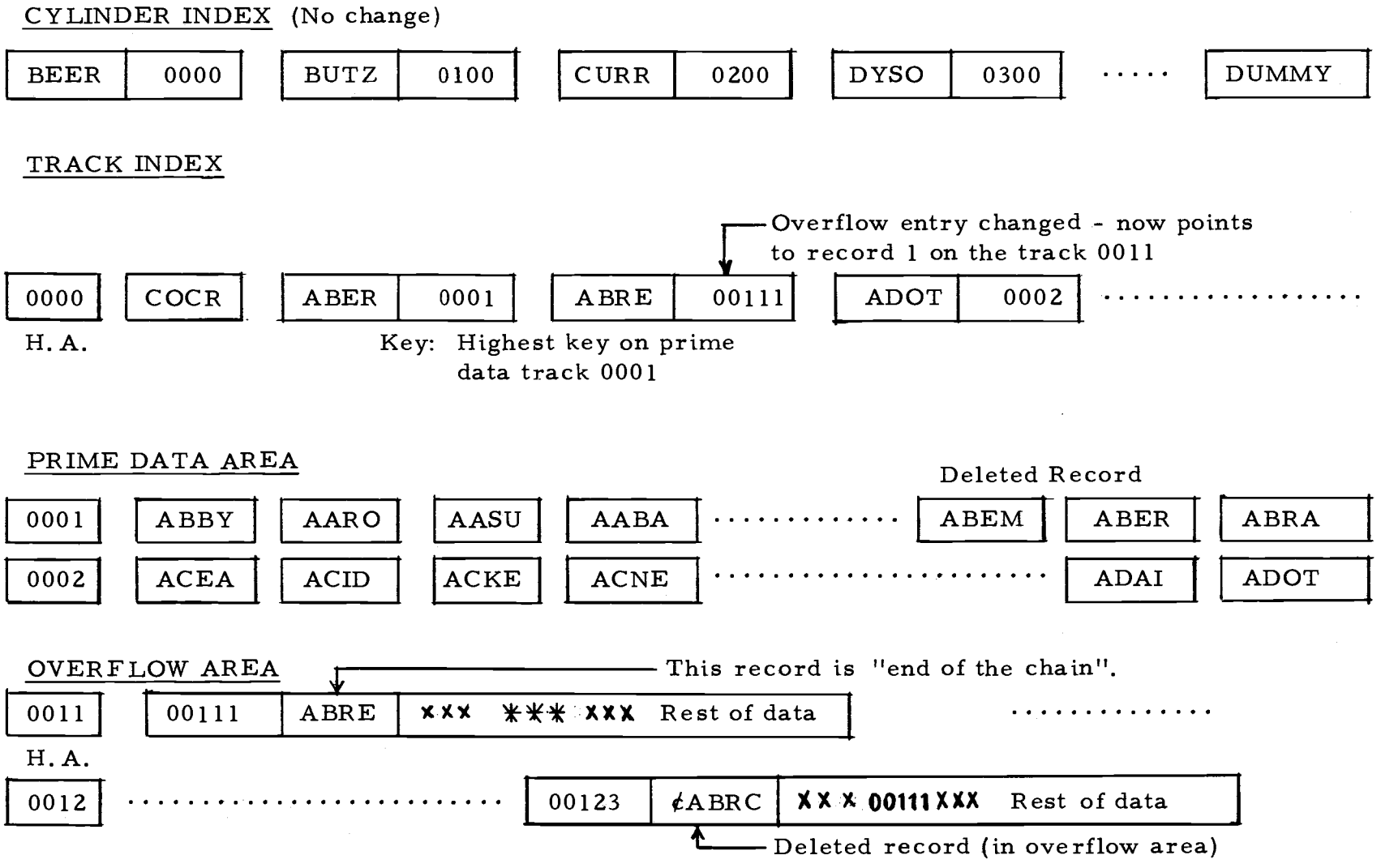


Figure 4. 7. An Indexed Sequential Disk File after deletions of the desired records from File.

is written, the chain is updated, and the job is done.

Step 15. The key area of the normal index entry is updated, containing the highest key in that prime track.

Step 16. In case the overflow exists, the data area of the overflow index entry is also updated; it now contains the address of the overflow record.

Example of Case I, First Addition to a Prime Track. The first addition to a track is always handled in this way. Any record that is higher than the original highest record on the preceding track but lower than the original highest record on this track is written on this track. See Figure 4.5.

Example of Case II, Subsequent Additions to a Track. Subsequent additions are written either on the prime track where they belong or as part of the overflow chain from that track. In case the addition belongs between the last prime record on a track and a previous overflow from that track, as in the case of adding a new record with key name ABRC, which is written in the first available location in the overflow area, with its link field containing the address of the next record in the chain. The link field of a previous overflow may need to be changed; in this example it is not necessary because the Data Area of the overflow index entry always refers to the address of the lowest key in a chain.

If the addition belongs on a prime track as in this case, the

record's key name, ABER, is added in its proper sequential location on the prime track. The bumped record, ABRE, is written in the first available location in the overflow area.

The Key Area of the normal index entry is changed to ABRA. The link field of a previous overflow and the Data Area of the overflow index entry are changed if necessary.

The similarity between the normal and overflow index entries is that the normal entry indicates that a sequence of records starts at the beginning of Track 0011, the last record having a key of ABRA. The overflow entry indicates that a sequence of records chained together by the link field starts with the third record on Track 0012, the last record having a key of ABRE. See Figure 4.6, page 45.

Although the cylinder overflow area may eventually contain overflow records, all prime tracks in the cylinder and the independent overflow area may eventually contain overflow records from anywhere in the file; each prime track has its own chain.

Example of Case III, Addition of High Keys. A record with a key higher than the current highest key in the file is placed on the last prime track containing data records if that track is not full. In case that track is full, the record is placed in the overflow area. The sequence link for this record is chained to the last prime track containing data records. The key area of higher level indexes is

changed to reflect the addition.

Up-Dating or Deleting the Record from the File

There are two conventional types of retrieval of the specified records from this type of file:

Sequential Processing. The logical steps required to retrieve all of the records in the key sequence are as follows, provided that the prime area consists of one contiguous area (since each record in the file is read out, no need to perform cylinder search):

Step 1. The operating system positions the access mechanism at the track index of the first cylinder in the file. It performs read-in operation, an internal search of the track index and picks up two entries from internal track index, the current overflow entry and the normal entry.

Step 2. The operating system reads and presents to the user each record on the specified prime track. If the end-of-file record is read, it goes to end-of-file routine.

Step 3. If the current overflow entry has been changed the operating system reads and presents to the user each record in the overflow chain and then goes to Step 5.

Step 4. If the current overflow entry has not been changed, the operating system goes immediately to Step 5.

Step 5. If the next prime entry is a dummy, that is if all the records in this cylinder have been read the operating system seeks the next cylinder by going to Step 7.

Step 6. If the next prime entry is not a dummy, go to Step 8.

Step 7. The operating system positions the access mechanism at the track index of the next successive cylinder in the file and performs read-in, an internal search of its track index. A track index of this next successive cylinder is read-in and stored at some place in core memory which may be called "track index table". It is the same as Step 1, but not for the first cylinder. Go to Step 8.

Step 8. The operating system performs an internal search through the track index table and picks up the next pair of entries: that is the overflow entry of the next prime track to be processed and the normal entry for the prime track following that, and return to Step 2.

In fact, the cylinder index is used only for the initial positioning at the beginning of the file, and reference to the track index is necessary only once for each prime track.

Random Processing. This typical method of retrieving specified records is more suitable to the nature of an on-line information system.

Step 1. The operating system reads the transaction, the full name of a record or the fixed-length key of a record is used as search argument.

Step 2. The operating system searches the cylinder index for key "high" or "equal".

Step 3. When the search is satisfied, the system reads the address in the corresponding data area.

Step 4. The operating system seeks the referenced cylinder.

Step 5. It searches the track index for the cylinder with a key "high" or "equal" in core memory.

Step 6. When the search is satisfied the system picks up the address in the corresponding data area.

Step 7. If the desired record is on the prime track, it is the normal record. The operating system searches the reference prime track for a key "equal" in core memory and reads and presents to the user the desired record in the main file.

Step 8. If the desired record is not on the prime track, it is an overflow record.

a) The operating system searches the referred overflow track, read-in and an internal search for identifier with equal record address is performed.

b) The system searches the desired record in core along the chain.

c) If the desired record is found, the system presents its data to the user.

d) If it is not found the system repeats Step b) by using the address from the link field until the specified record is found.

e) If the end of the chain is encountered, the system goes to record-not-found-routine which notifies the user "NO SUCH RECORD IN THE FILE."

File Reorganization Criteria

An efficient file must be reorganized periodically for three reasons:

1. The overflow area is eventually full.
2. During additions, increased time is required to locate a record at random.
3. When the prime area contains many deleted records, much space is wasted.

Handling Deletions in the Indexed Sequential File

Most operating systems do not handle deletion in any way. The usual approach is to tag a deleted record in some way by the

same method as that of partitioned file organization and then to omit it when the file is reorganized.

a) Usually the desired record is tabbed by writing all 1-bits in the first byte of the deleted record.

b) If the tagged record is bumped off the prime track by a subsequent addition, it is not rewritten again in the overflow area.

c) When the file is reorganized, any tagged records remaining in the prime area can be omitted from the reorganized file by the user.

d) When the file is processed sequentially, the deletion records (tagged records) are not retrieved for processing.

e) When the file is processed by random sequence, tagged records are retrieved like any other record and thus should be checked for deletion-code by the user's program.

In some cases the operating system can automatically handle deletion records as follows:

a) If the deleted record is the last record in the prime track, the operating system will simple erase this record.

b) In case the desired record is somewhere in the prime track, all the whole records in that prime track are

read-in. Repacking is performed in core memory, and the repacked records are returned to the original prime track. In case there is still an empty space left on this prime track, an overflow record, which is the head of the chain that belongs to this track, is shifted up to be the highest key record of this prime track. The operating system updates data in the overflow entry and in the normal entry which corresponds to this prime track in the track index.

c) In case the desired record is on the cylinder overflow track, deletion by tagging the desired record can be performed.

Variable-Length Records

In some cases the user encounters variable-records. One way to solve this problem is to use trailer records, the extension of the master record with the same operating system used for the fixed-length logical records. The trailer record may be written immediately after the associated master record. The duplicate key is not allowed. It is necessary to add a digit or character to the true key; that is, to use the alphanumeric key. For example 125A, the key of the master record number 125, are added 125B, and 125C, the key of the first and the second trailer record respectively. The trailer

record may be written as a separate file by which link list chain is introduced.

Evaluation of Accessing Characteristics of the Indexed Sequential Disk File

For comparison with other file organization methods, the parameters of the Indexed Sequential File have to be measured. CDC-3300 system with OS-3 is selected as the simulator. The strategy of simulation is as follows:

1. For simulation of the Indexed Sequential Disk File, the data records in the file are assumed to be fixed, and blocked. See Figure 9.2, page 202. Each record is formatted with a key. The overflow records are placed in cylinder overflow area only. The overflow records of each prime track are put in a chain.
2. Data model for simulation uses the selected data model as shown in Appendix B. See simulating block diagram model in Figure 4.8.
3. For accessing the proper location in disk memory, the full name of the record has to be converted into a unique fixed-length key name by the Variable-Length Tree-Decoding Technique. (The detail of Variable-Length Tree-Decoding Technique is shown in Chapter V.)

4. All data records are kept in disk memory, D854.

   a) The Directory Table is read into core memory for initializing and the user supplies the full name of the records to the system. Search of the Directory Table is performed internally.

   b) If the search is satisfied, the operating system will use the fixed length key name obtained as the argument key for retrieval of the desired record from the file system.

   c) If the search is not satisfied, the operating system will notify the user "NO SUCH NAME IN THE FILE".

   d) For accessing the data of the Cylinder Index, the Track Index and the Data of the Main record, the machine has to read into core memory and search for desired item. The user can control the operating system to return the Directory decoding routine to disk memory after file processing task is finished.

5. For measuring the exact accessing characteristics of this file system, the selected record format is considered and the computation is performed. See Example 2, Appendix B.

The Purpose of the Evaluation

The purpose of the evaluation is to measure the following Indexed Sequential Disk File parameters:

1. To compute the access time per random record retrieval when the data file is on the disk.
2. To measure the space storage requirement.
3. To find the characteristics of "average throughput time per record retrieval" and "achievable throughput rate capability".
4. To find the "cost/effectiveness" characteristics, ("user or customer operating cost per call, unit cost").
5. To compare the characteristics of this file method with the other methods of file organization.

Simulating Block Diagram Model

Simulation block diagram model, Figure 4.8 is the illustration of finding the accessing characteristics of a random record form Indexed Sequential Disk File. See details in Appendix B, Example 3.

Results of the Evaluation of Indexed Sequential Disk File

Results of the evaluation of Indexed Sequential Disk File and the comparison of the characteristics of this file with those of the

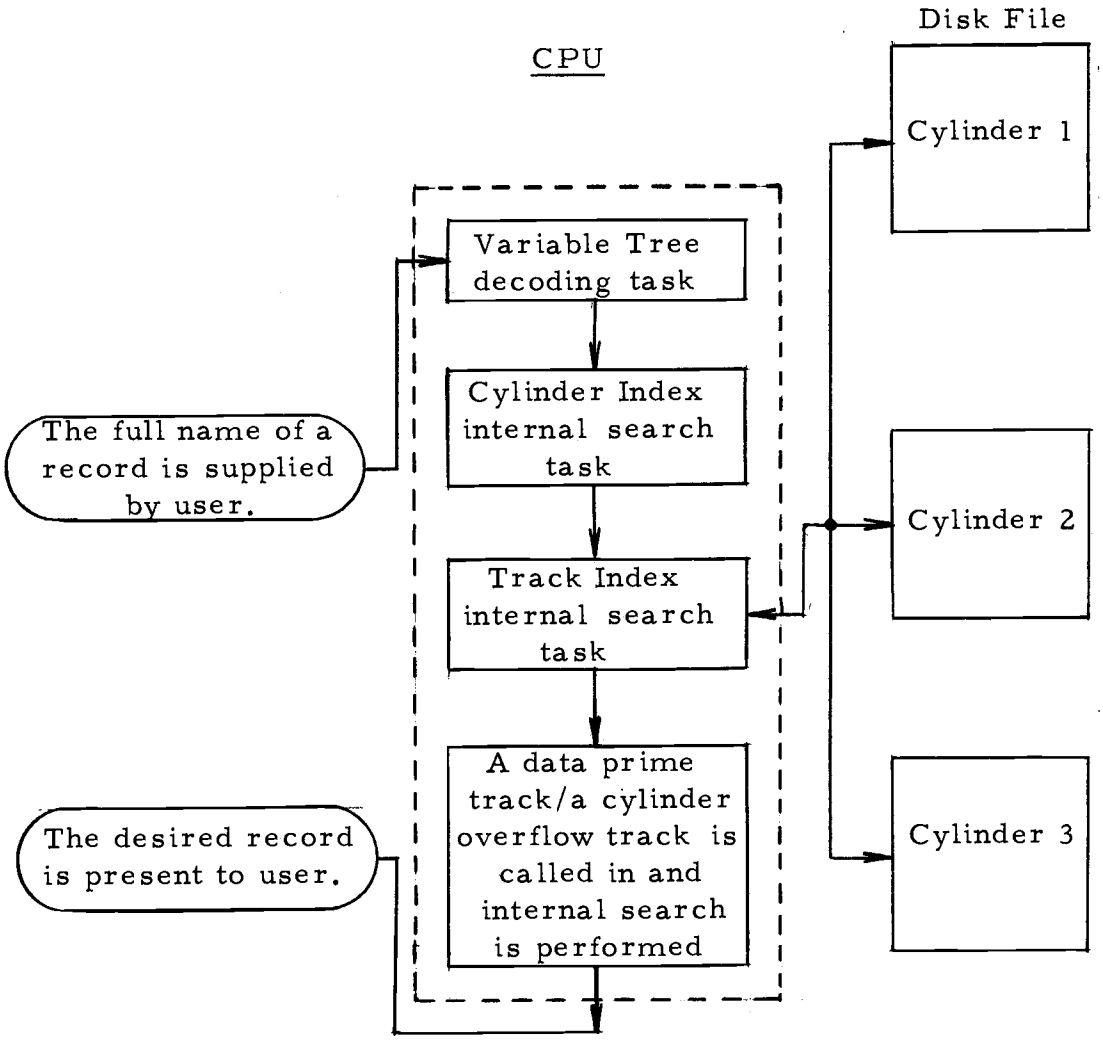


Figure 4.8. Block diagram showing the simulation of accessing a ramdon record from Indexed Sequential Disk File.

other methods of file organization technique are shown in Chapter VII.

V. PARTITIONED FILE STRUCTURE AND USE

Partitioned File Structure

The partitioned file is one of common method of organization for the on-line data processing system. It has a hierarchical file structure which is compounded, and it requires a directory and sometimes even hierarchies of directories for maintaining association and providing access. These directories are of table of content type. Access is most commonly made by attribute. The speed of access depends upon the size and number of directories used. See Figure 5. 1.

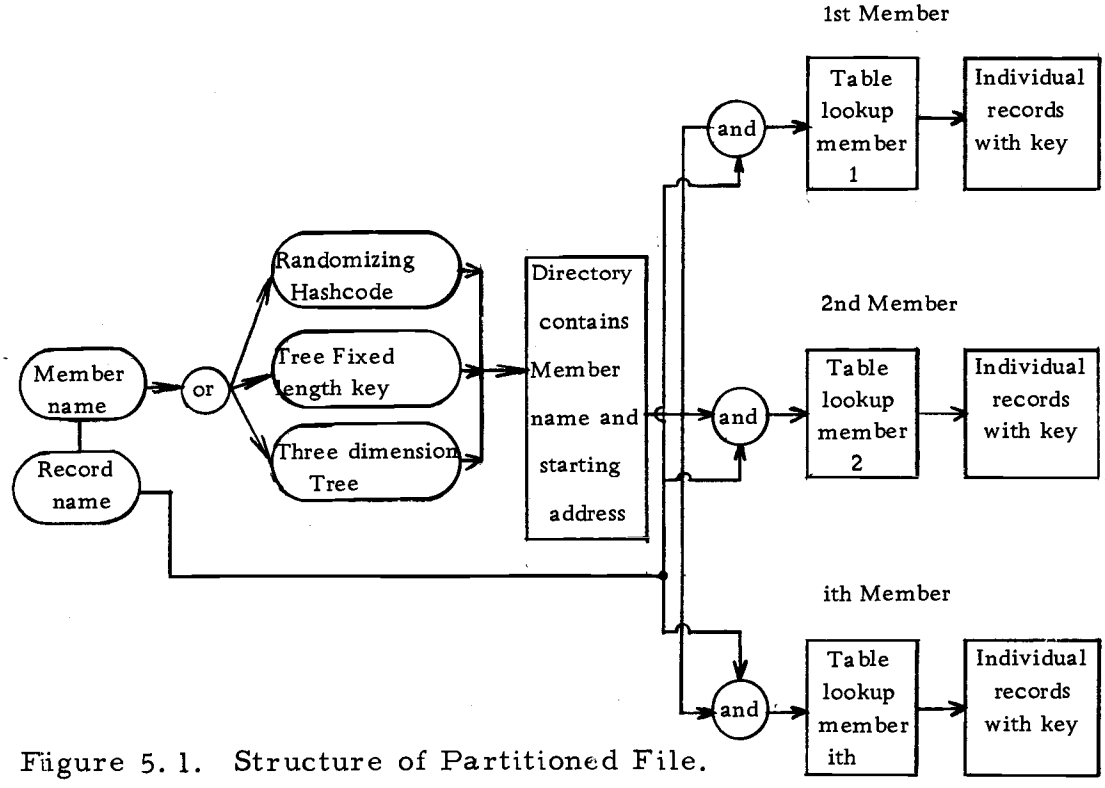


Figure 5. 1. Structure of Partitioned File.

User's Argument Key Name

In the partitioned file, the user's argument key name or record key name is the "key" which is used to communicate the master record of the file, stored on the auxiliary memory units. Due the nature of Directory and Partitioned File Organization, usually an alphabetical string of key names is used. For convenience, the natural language such as a person's name is used as the key name of his own record in the file. Sometimes alphanumeric and numeric keys can be used in the same fashion.

Directory Decoding Techniques

Directory decoding techniques for the partitioned file can be divided into two general classes:

1. Randomizing or Hash Coding. The details of this technique will be illustrated and discussed in Chapter VI.
2. Tree Structure Decoding. This method can be divided into two subclasses:
   a) Fixed-Length Key Decoding. Fixed length keys are generally preferred because the decoding programs are simple to write and fast to execute. There are two general methods for converting a full length name into a Fixed-Length Key, as indicated in

Figure 5.2. One method is "to sample" some of the fixed set of characters or bits from the full-length name. A special method is to "truncate" the key to a given number of characters. This technique is popularly used in many systems. Another method is to apply a randomizing technique to the full name of the record to convert it into a range that is represented by a fixed number of bits.

The disadvantage of Fixed-Length Key decoding is that the method cannot guarantee to offer 0% of redundancy coding.

b) Variable-Length Key Decoding. There are also two ways to obtain the variable-length key.

(1) The full name of the record can be used as the variable key.

(2) A unique sampling can be used for each key so that as little of the key word is retained as is necessary to make it completely different from every other key. In fact this method is applied to a small sized file. It is impractical to implement it on a large scaled file.

The main advantage of the variable length key method is that is provides completely unambiguous decoding.

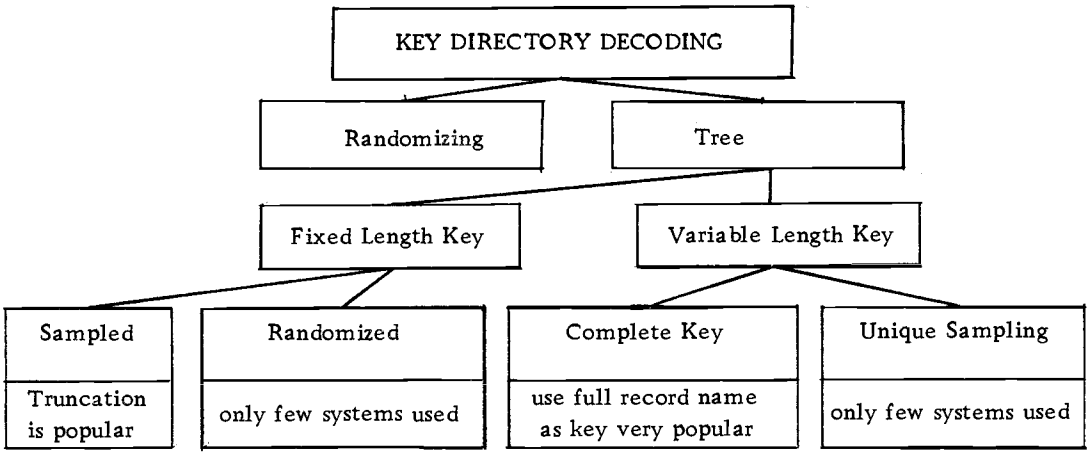


Figure 5.2. Key Directory decoding pattern.

Directory and Main File Organization

Directory. Also called "file directory" has been mentioned before in Chapter II. Partitioned file organization is divided into several parts called "members". A directory contains the "names" of these members and their corresponding "addresses" (location in the main file). The entries of a directory are organized in alphabetical sequence. To provide efficient file accessing a directory can be divided into blocks. Each directory block begins with a Key Area which contains the key name of the last member of the block. Records in the main file are organized the same as directory entries.

Main File Body. In the partitioned file organization, the main body consists of several "sequential subfiles" or "members". In each of these members, the data records are arranged sequentially.

They may or may not be in key sequence, depending on the record-accessing technique. If there are many data records in each member (more than 64 records), data records should be arranged in a key sequence so that binary search can be introduced. This arrangement also provided the possibility of accessing the records from the main file by sequential processing. In case the file system uses hash coding as its directory decoding method, records in the main file are not distributed in any key sequence. See details in Chapter VI.

Use of the Partitioned File

The method of use or maintenance of the partitioned file is dependent on the directory decoding technique used. Three cases will be discussed here.

Case I. The partitioned file system uses "randomizing" or has coding to perform a key decoding task.

The details of hash coding will be discussed in Chapter VI. After the hash address is obtained it is used as the address for the directory table. The address of the first record in the member is kept as an entry in the directory table, which is stored in core memory. But in direct file organization the hash address is directly used as the reference of the data record in the main file. There is no need to search at the directory level.

Standard algorithms of file maintenance are as follows:

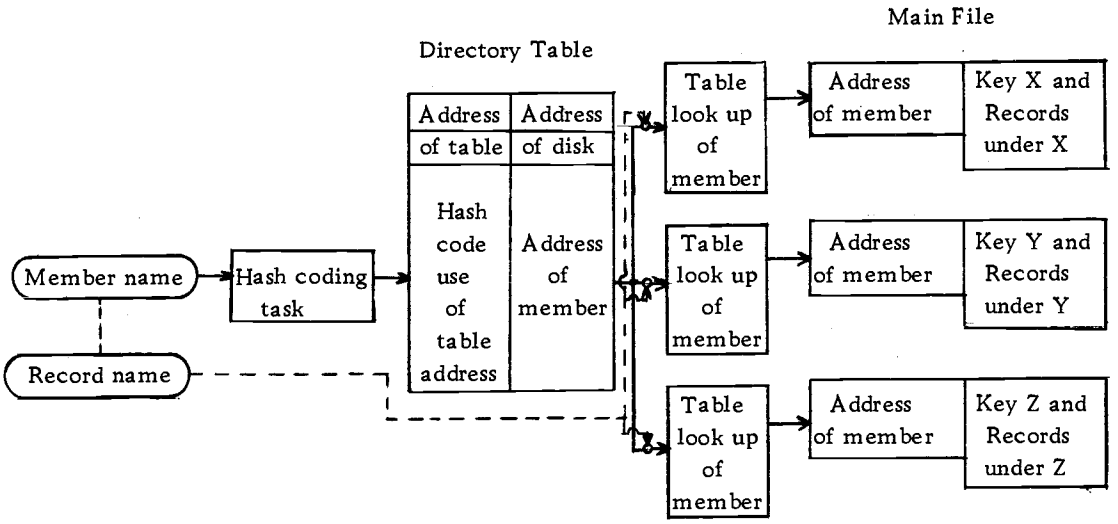


Figure 5.3 Partitioned file using randomizing as directory decoding technique.

Adding a New Record to the File

Step 1. The user supplies full name of a record.

Step 2. The operating system converts the full name of the record being added into hash address to be used for directory table reference.

Step 3. The system fetches the address of the member to which record being added should belong.

Step 4. The operating system searches for the next space available in the member.

Step 5. If the proper sequential location in the desired track of the desired member is found, and it is empty, the new record is added there; the job is done.

Step 6. If the proper sequential location is found, and it is not empty in the track of the desired member, the operating system will move the rest of the records higher in that track (up one position) some of the higher key records in that prime track may be bumped out.

Step 7. The operating system will continue the search for the first available space in the overflow area.

Step 8. If the first available space in overflow area is found, the bumped record or the new record is written. In an overflow area the overflow records which belong to the same track of each member must be organized in the same chain (link list can be applied for overflow records which belong to the same member).

Updating and Deleting the Record from the File

Step 1. The user supplies the full name of the member.

Step 2. The operating system converts the full name of desired member into hash address to be used for directory reference.

Step 3. The operating system fetches the address of the member to which the desired record should belong.

Step 4. The user has to supply the full name of the desired record or unique fixed-length key of the desired record to the system. If it is the full name of the desired record the directory decoding technique should be introduced (in this case the writer suggests to use variable-length tree decoding technique) to convert the full name of record into unique fixed-length key.

Step 5. The operating system searches for the desired record in the member by using the unique fixed-length key of the desired record as the search argument.

Step 6. If the search is satisfied, the data in the desired record may be updated or deleted, as the user may wish. In case of deletion, the deletion-mark is written on the desired record, and this deleted record area cannot be reused until the file is reorganized.

Step 7. If the search is not satisfied in the file, the operating system will notify the user "NO SUCH RECORD IN THE FILE".

Case II. The partition file system uses the tree with a fixed-length key-word by truncation to perform the key decoding task. In this section the terms tree structure, balanced tree and unbalanced tree will be introduced. Since in practice most of the keynames

within the directory possess an unbalanced tree structure, only the technique of unbalanced tree processing will be discussed here. As this method may cause redundancies, the operating system or the processing program has to notify the user every time the new key name is redundant with the existing keys. Some modification has to be performed by the user on the new key. One possible way is to change some of the rightmost characters in the fixed-length redundant key into fixed-length unique key. See example in Table 5.1. When the size of the directory table is large, the number of redundancies will increase, so the above method is impractical. The alternative is to use the full name of a record and convert it into a fixed-length unique key by using the variable-length tree decoding technique to avoid ambiguous decoding. Details will be discussed in Case III, page 75.

Table 5.1. Illustration of full name of record and unique fixed-length key of record.

Argument, Full Name	Fixed Length Key by Truncation				Fixed Length Key After Modifying the key must be unique			
ADMINISTRATION	A	D	M	I	A	D	M	I
AIR COMMAND UNIT	A	I	R	C	A	I	R	C
AIR FIELD CONSTRUCTION	A	I	R	F	A	I	R	F
AIR FIELD LIGHTING SYSTEM	A	I	R	F	A	I	R	L
BABB MARGARET	B	A	B	B	B	A	B	B
BABCOCK DANA	B	A	B	C	B	A	B	C
BABCOCK DONNER	B	A	B	C	B	A	B	O

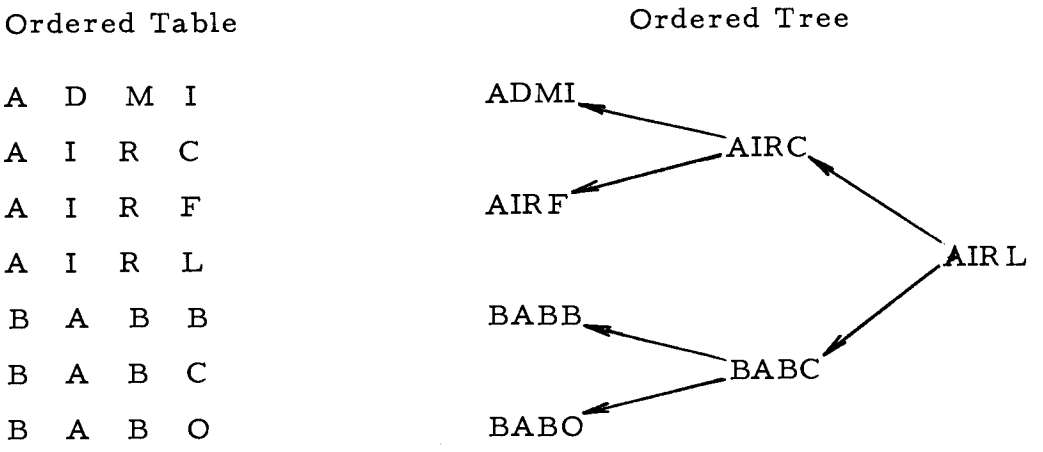
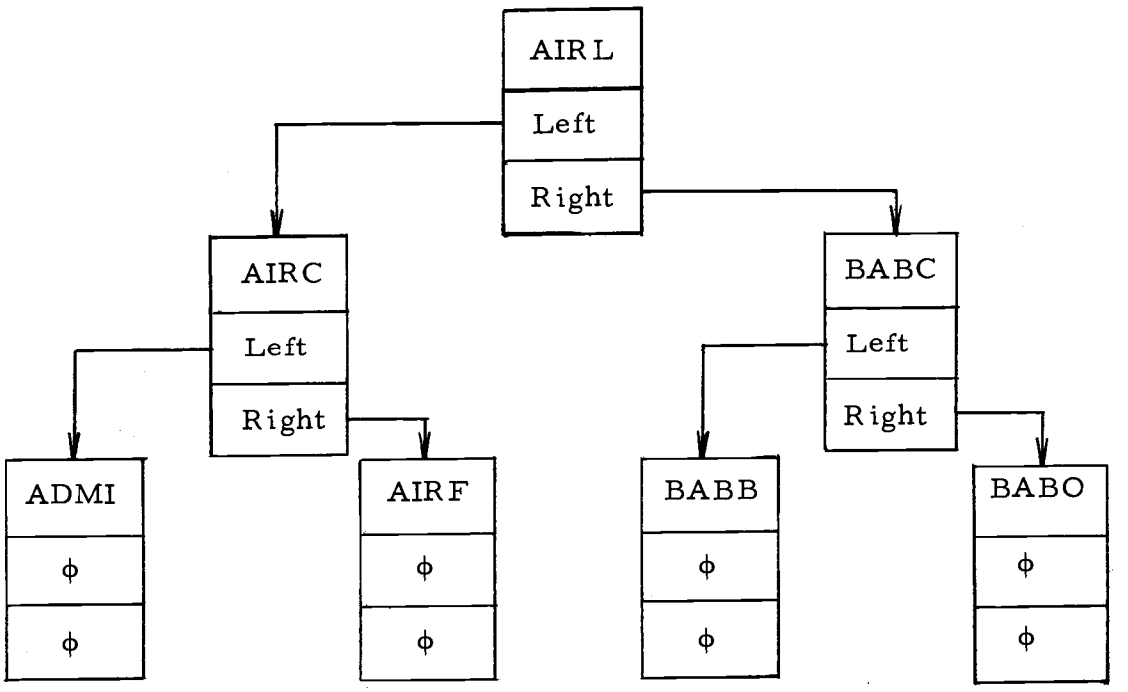


Figure 5.4. Comparison of ordered table and ordered tree.



Where Left = Left pointer: Right = Right pointer

Figure 5.5. Tree structure (balanced tree).

Tree Structure. The entire list structure in Figure 5.5 is called a "tree". Each data item in the tree is called a "node". Node AIRL is the root. It is the base or the beginning of the tree. The order of the tree becomes evident by its relation to the root. All nodes left of root AIRL contain lower keys: ADMI, AIRC, and AIRF. All nodes to the right contain higher keys: BABB, BABC, and BABO. This same order holds for nodes of the tree other than the root. For example, lower key AIDMI is left of node AIRC, and the higher key AIRF is right of it.

Search Mechanism. The algorithm of search of a tree (both balanced and unbalanced tree) is as follows:

1. The search of a tree begins at the root node. The argument search key is compared to the node key.
2. If the argument key is greater, the node right of the root is examined.
3. If the argument key is less, the node left of the root is examined.
4. This process continues until the desired node is found.

Figure 5.6 gives the flow chart form for the search.

Insert Mechanism

The algorithm of insertion of both a balanced and an unbalanced tree is the same as that of searching for a tree. The insert node will

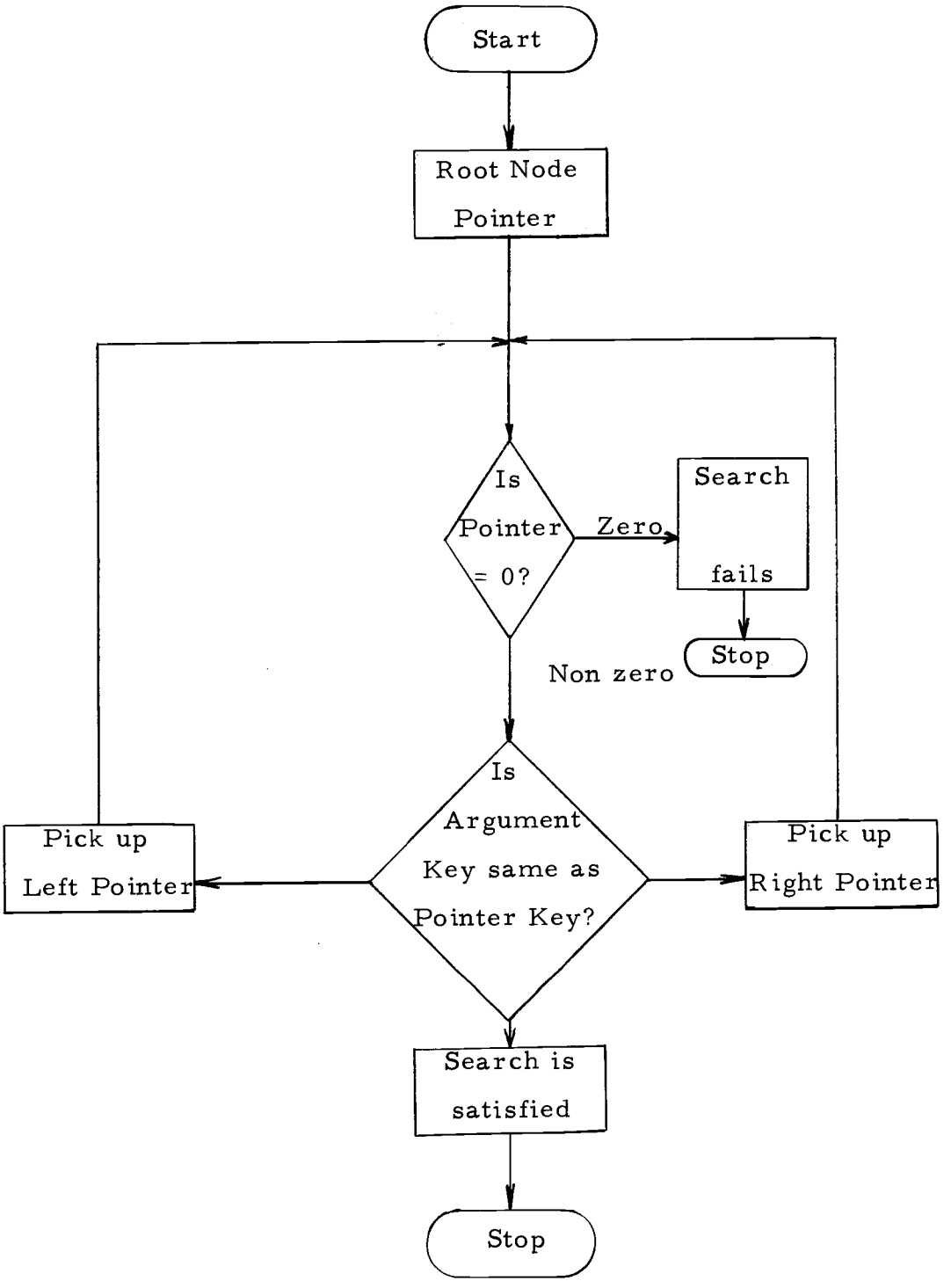


Figure 5.6. Flow chart of tree search.

will always belong at the bottom of the tree. For high efficiency in searching a tree in directory level the tree should be kept balanced as much as possible. Therefore the user has to make some arrangements beforehand. The task is simple, just sorting fixed-length keys in alphabetical sequence, and selecting the middle keyname in the list as the root of the tree.

In Figure 5.5, AIRL should be selected as the root node of the tree, and the user should start loading from this key name. The corresponding sequence nodes left or right AIRC, BABC, ADIM, AIRF, BABB, and BABO can be loaded next.

Delete Mechanism

In contrast to inserting nodes, which are always added to the bottom of the tree, deletion nodes may be taken anywhere from the tree. There are three cases for deleting the tree: the deleted node is either root, internal-node, or leaf-node.

The algorithm of deleting is as follows:

1. A search is made to find the deletion node in the tree.
2. If the deletion is an internal node or the root node,
   -- the search goes down one level to the left of the desired node,
   -- and the search continues along the last of the levels of the tree on the right pointer.

If a node with a right pointer is encountered, the deletion node is replaced by this node and the pointer in the tree is updated. Otherwise, the search goes down, based on the right pointer to the leaf node of the tree (the leaf node does not have both left and right pointers). The deletion node is replaced by the appropriate leaf node, and the pointer in the tree is updated.

3. If the deletion node is a leaf node, it is simple deleted and the pointers of the tree are updated. The illustration examples are shown in Figures 5.7 - 5.9.

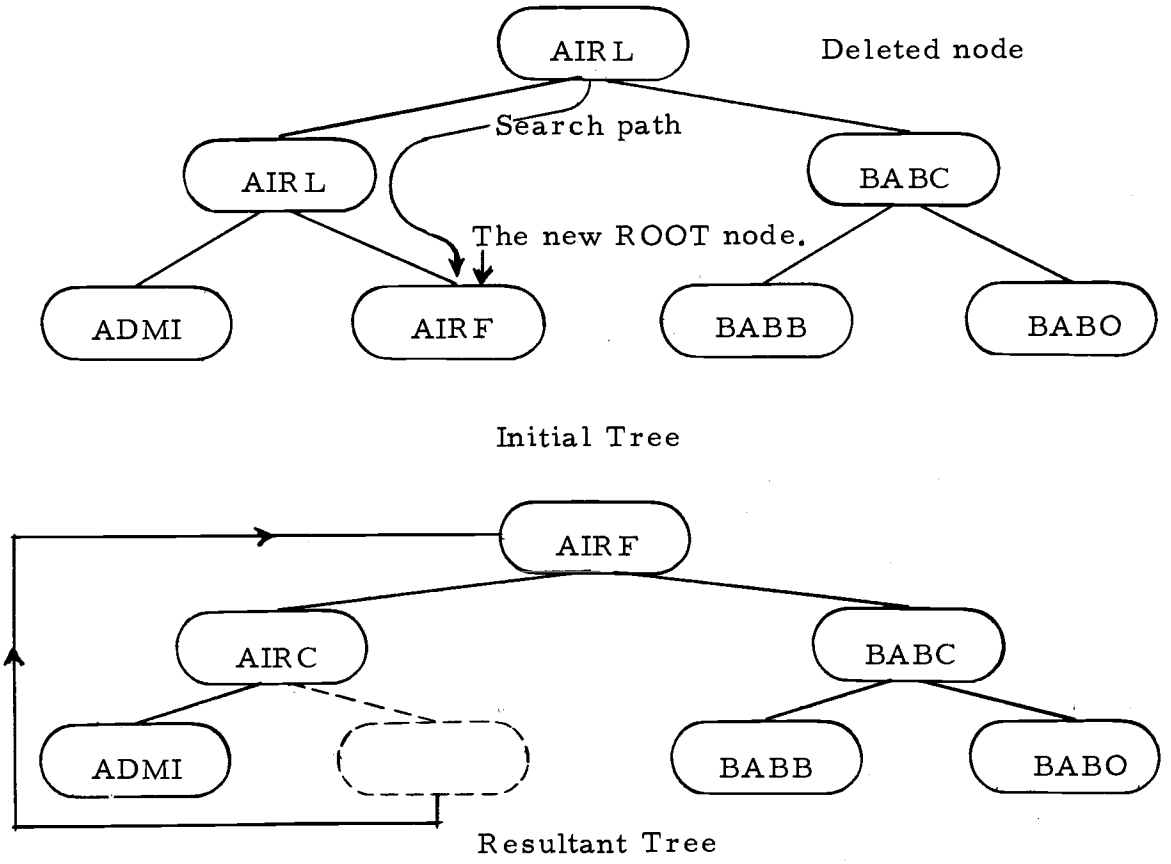
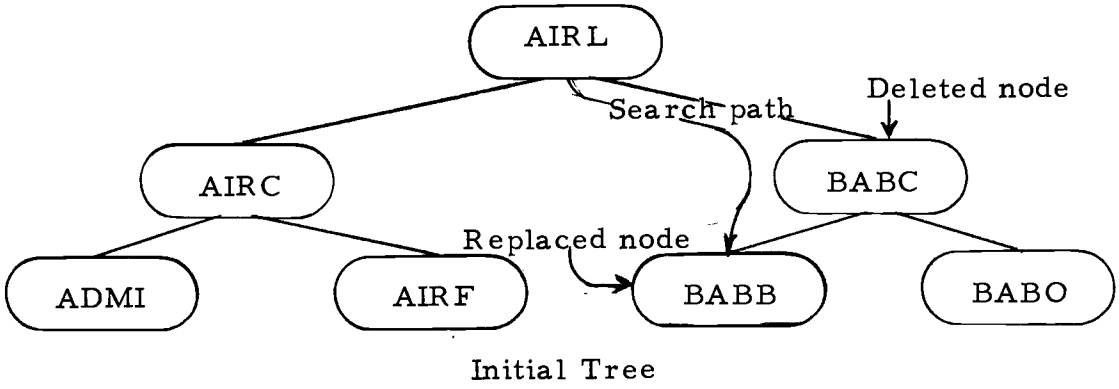


Figure 5.7. Deletion of node AIRL, the ROOT of the Tree.



Initial Tree

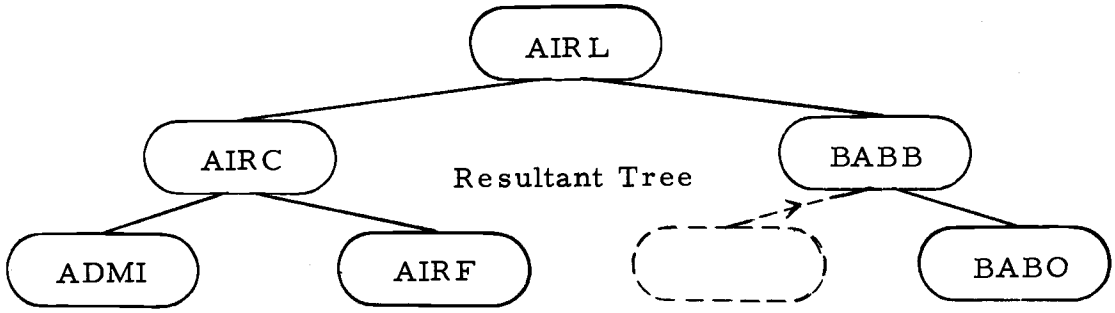
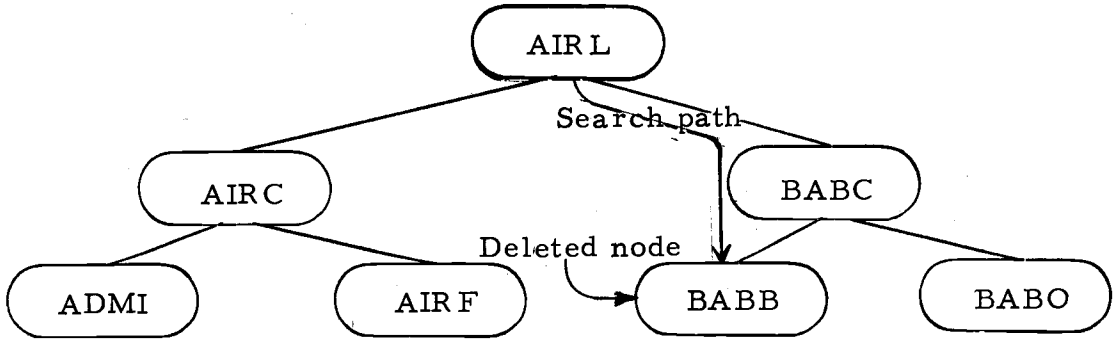


Figure 5.8. Deletion of BABC, the INTERNAL node of the Tree.



Initial Tree

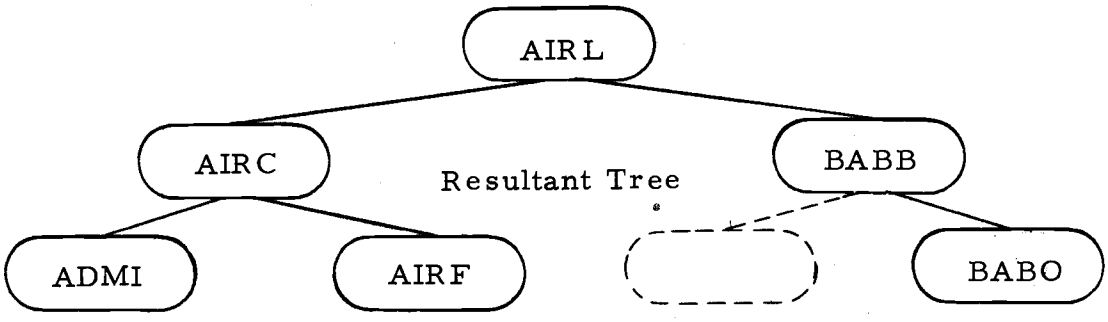


Figure 5.9. Deletion of node BABB, the leaf of the Tree.

The efficiency of the search is dependent on how well the tree is balanced. In general, in the application of the tree search structure, the tree can be rebalanced after each insertion and deletion. The balance mechanisms are shown by Richard L. Gauthier and Stephen D. Ponto in Designing Systems Programs Section 8.6 (12).

Use of the Partition File with Tree
With Fixed Length Key

The maintenance, adding, updating and deleting of the records in the main file are by the same procedure as by those mentioned in Case I. The alternative is to search at directory level using "tree search".

Case III. The partitioned file system uses a tree with full name as a variable-length key in the directory. This technique could be called "varible-length tree decoding" or "three dimension tree".

In this method, the system performs the coding task, by searching for the full name with computer word length, and each fixed-length part of the full name can represent a node of the tree. The searching task is equivalent to decoding the record name by a tree fixed-length key with truncation, as in Case II. If redundants exist, the ambiguity in searching the record in the main file can be avoided if the redundant is of the higher level nodes of the tree share storage space together; i.e. combining subtrees, two-dimension

trees to form three-dimension trees. Each branch of the Three Dimension Tree which represents a different logical meaning is located on different planes, and the corresponding branches in different planes are connected by pointers, which are usually called "successors" (see Table 5. 2). So in each node of a three-dimension tree, the extra space is needed for the successor. See illustrated example in Figure 5. 10.

Use of the Partitioned File with a Three Dimension Tree Directory

The technique of using this typical file, in search at directory level, is the same as that for search in tree directory. The algorithm of adding, updating and deleting the record from the main file is the same as that mentioned in Case II.

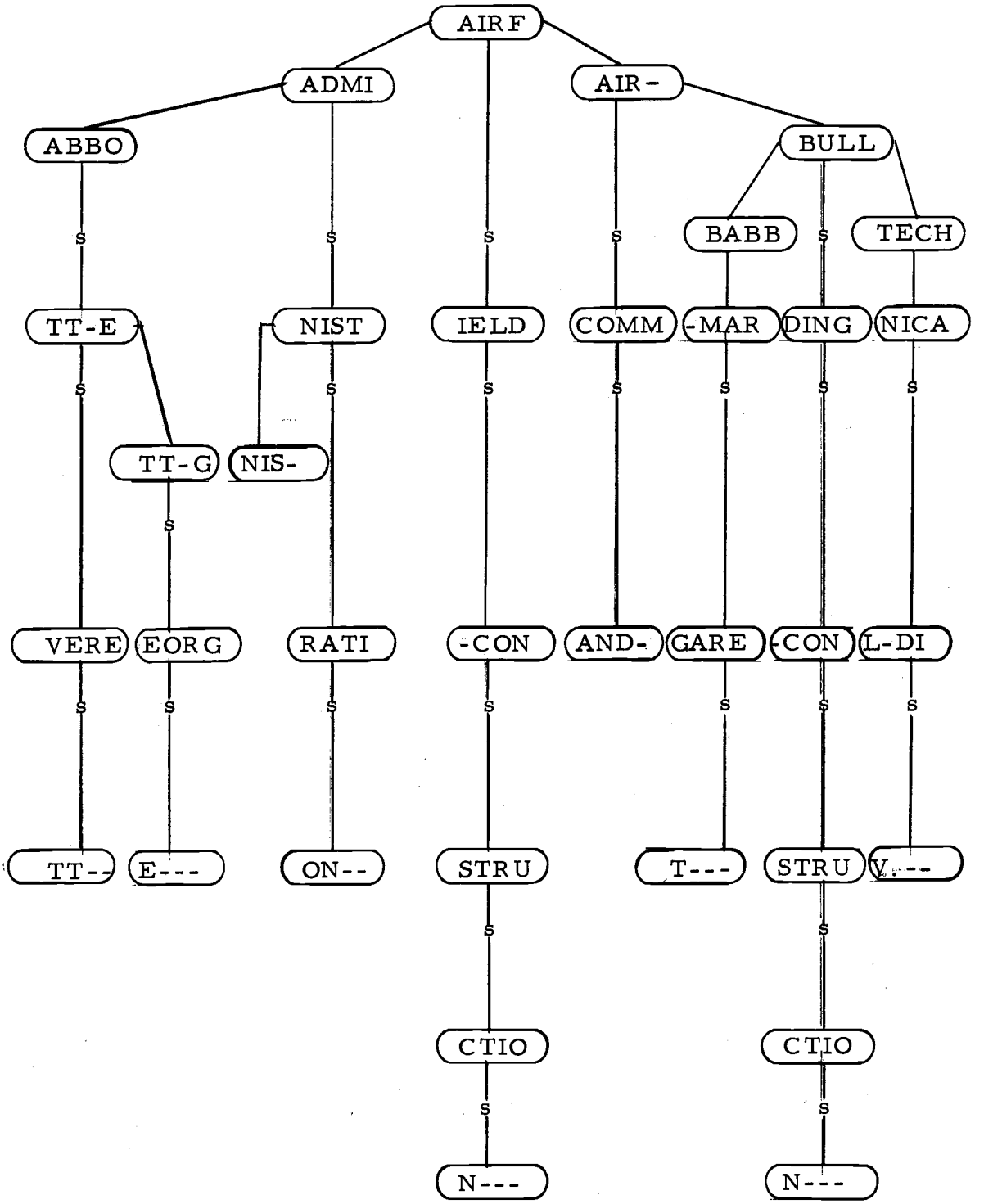
On-Line Partitioned Disk File

As the major structures of the partitioned file system are Directory and Sequence Members of the main file, this file structure can be implemented on disk memory, in one of the following ways:

1. The Directory is permanently kept in main core storage and the Main File Body is kept on Disk Memory. This system is obviously inefficient as the size of the Directory and the Main File is increased.

Table 5.2. Example of Variable length key name.

Complete Key Name	Data	Successive Fixed Length Key
ADMI	0001	ADMI
ADMINIS	0200	ADMI NIS-
ADMINISTRATION	0221	ADMI NIST RATI ON--
AIRFIELD CONSTRUCTION	0004	AIRF IELD -CON STRU CTIO N--
AIRFIEID LIGHTING	0005	AIRF IELD -LIG HTIN G--
AIR COMMAND	0006	AIR- COMM AND-
AIR CONDITION	0007	AIR- COND ITIO N---
BUIL	0008	BUIL
BUILDING	0009	BUIL DING
BUILDING CONSTRUCTION	0010	BUIL DING -CON STRU CTIO N---
TECHNICAL DIV.	0011	TECH NICA L-DI V. --
TECH	0012	TECH
TACTICAL AIR COMMAND	0111	TACT ICAL -AIR -COM MAND
SUPPLY	0014	SUPP LY--
UTIL	0015	UTIL
UTILITIES	0016	UTIL TIES
ABBOTT EVERETT	4205	ABBO TT-E VERE TT--
ABBOTT ETT GEORGE	7065	ABBO TT-E TT-G EORG E--
BABB MARGARET	8693	BABB -MAR GARE T---
BABCOCK DANA	6227	BABC OCK- DANA
BABCOCK DONNER	4934	BABC OCK- DONN ER--



Note: Only some of the complete key names in Table 5.2 are illustrated here. s = successor.

Figure 5.10. Tree structure of Variable-Length Key Names.

2. The Directory is temporarily kept in main core storage only during the processing of the file. After the job is done, both the Directory and part of the Main File are returned to Disk memory. This method is rather complicated in software, but it is popular in usage. Both 1 and 2 are called internal search.

3. Both Directory and Main File are kept in Disk Memory. The searching tasks, both Directory and Main File, are performed in CPU. The advantage is that the majority of main core memory is not required, but the disadvantage is that the nature of this search required more disk-time than do the above two methods.

For comparison with the other type of file organization, the second method above will be discussed and evaluated.

Description of Directory. In practice the simple organization of the directory is similar to that of the strictly sequential file. The directory record, which is in alphabetical sequence by the member name, varies from 12 to 24 bytes in length, depending on how much data is included. For example the directory records are grouped into 256-byte blocks, each containing as many records as will fit into the block. Each directory block has an eight-byte key area containing the name of the last member in the block. If the complete directory is full, no new member can be added, until the file is

Directory

Fixed length Key	Cylinder, track Address	No records/ Member
ADMI	0000	0016
AIRF	0001	0016
BUIL	0002	0016
SUPP	0003	0016
TECH	0004	0016
UTIL	0005	0012
WAST	0006	0002
WEAT	0007	0002
WOOD	0008	0003

a) Directory mapping in paper file.

TECH
ADMI
0000
0016
AIRF
0001
0016
BUIL
0002
0016
SUPP
0003
0016
UTIL
0005
0012

WAST
0006
0002
WEAT
0007
0002
WOOD
0008
0003

b) Directory mapping in core memory.

Shows the beginning of a track or block

COUNT	TECH	0044	ADMI	0000	0016	TECH	0004	0016
COUNT	WOOD	0032	UTIL	0005	0012	WOOD	0008	0003

c) Directory mapping on Disk track; two directory blocks are shown.

Figure 5.11. Example of Directory in Partitioned File.

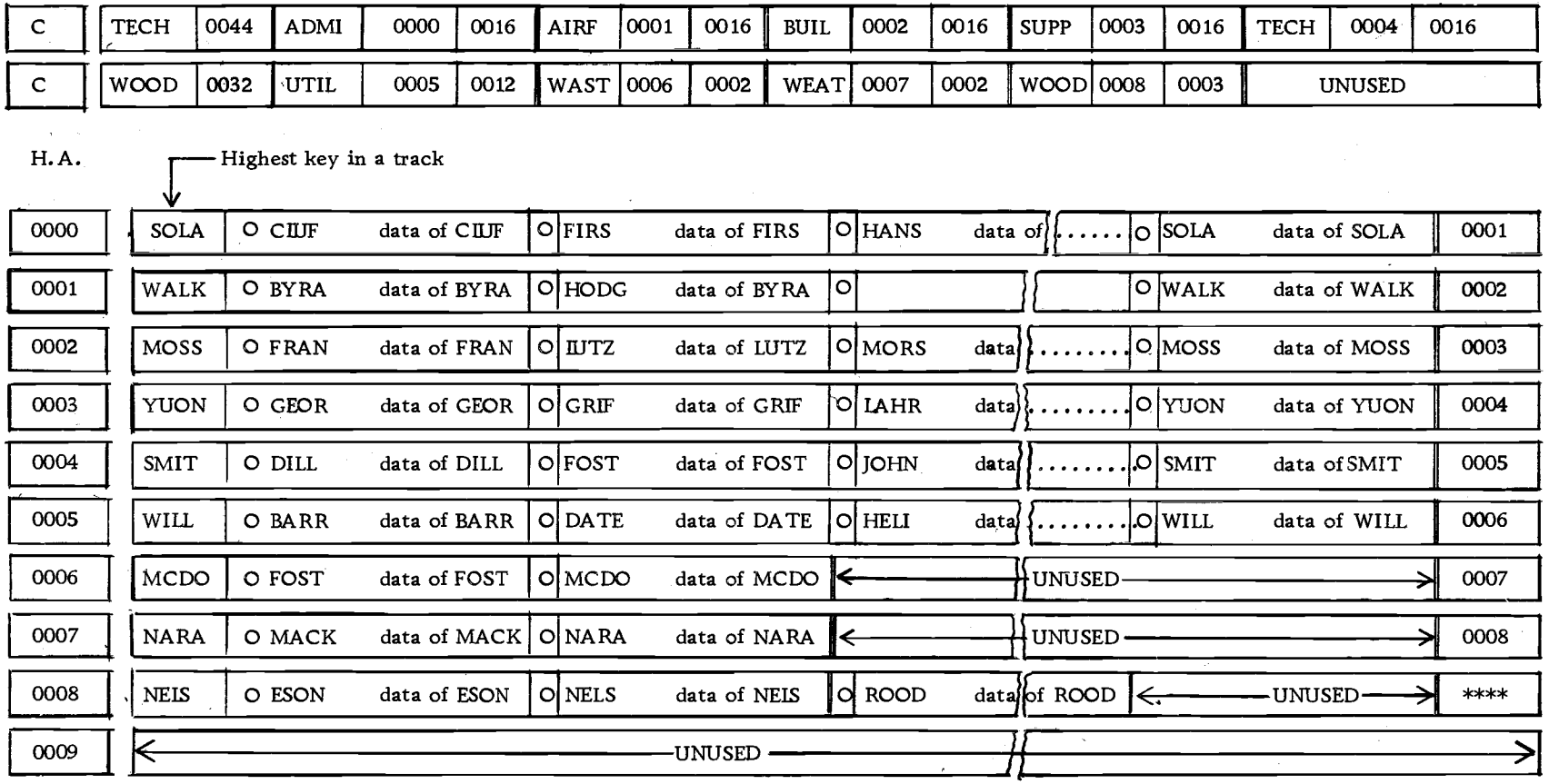


Figure 5. 12. Partitioned file without addition.

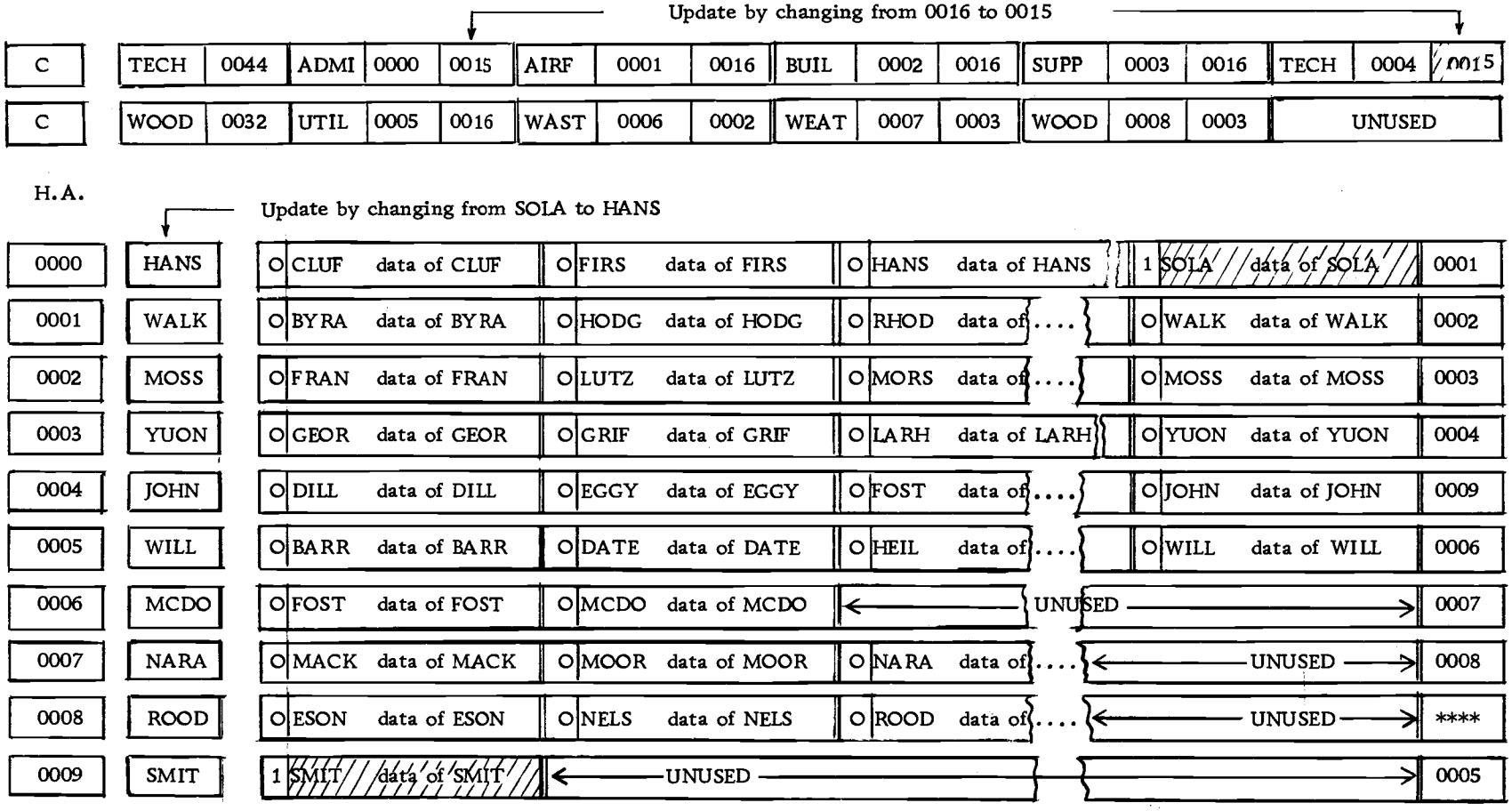


Figure 5.13. Partitioned File after deletion of records "SOLA" and "SMIT".

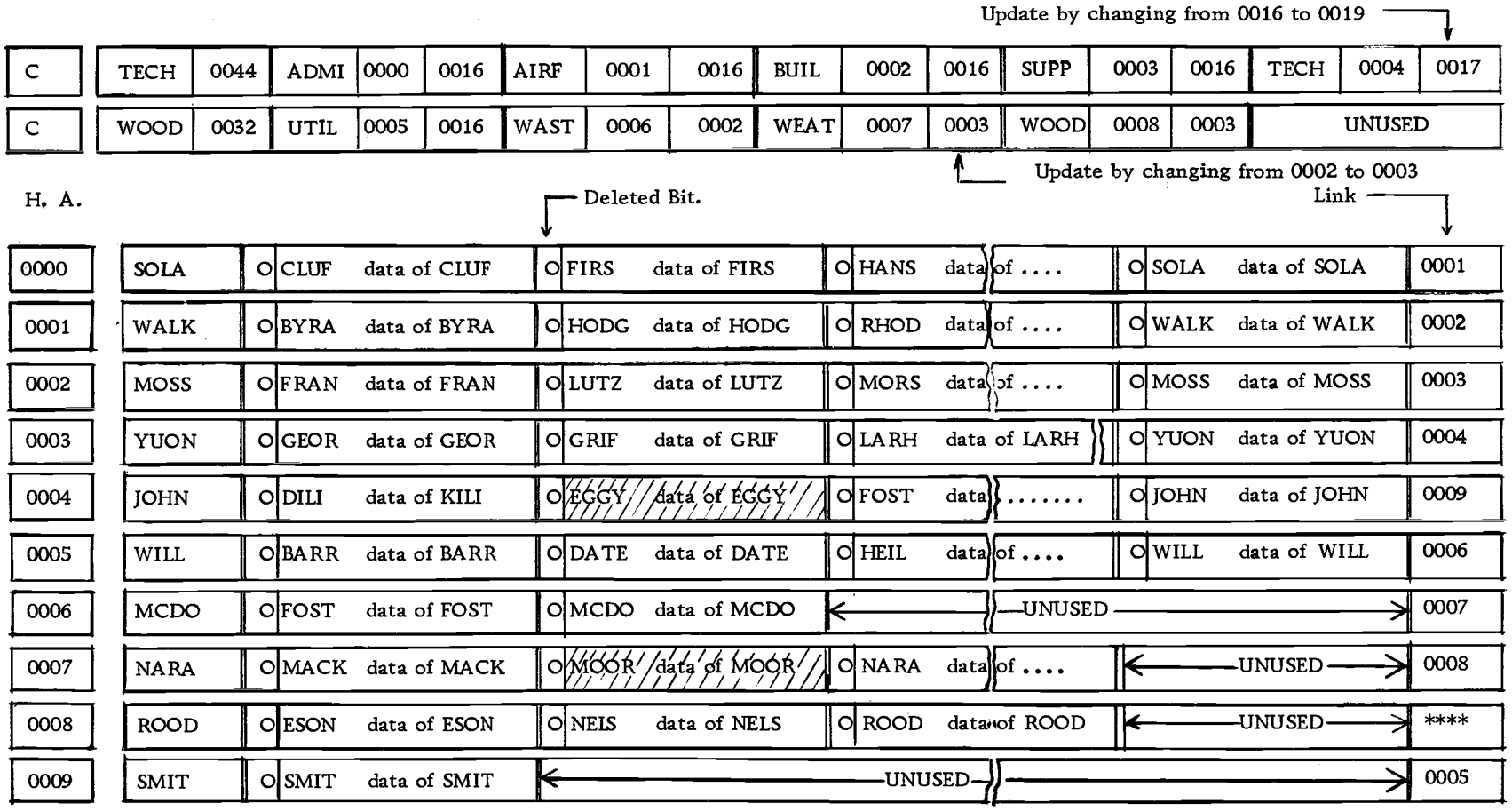
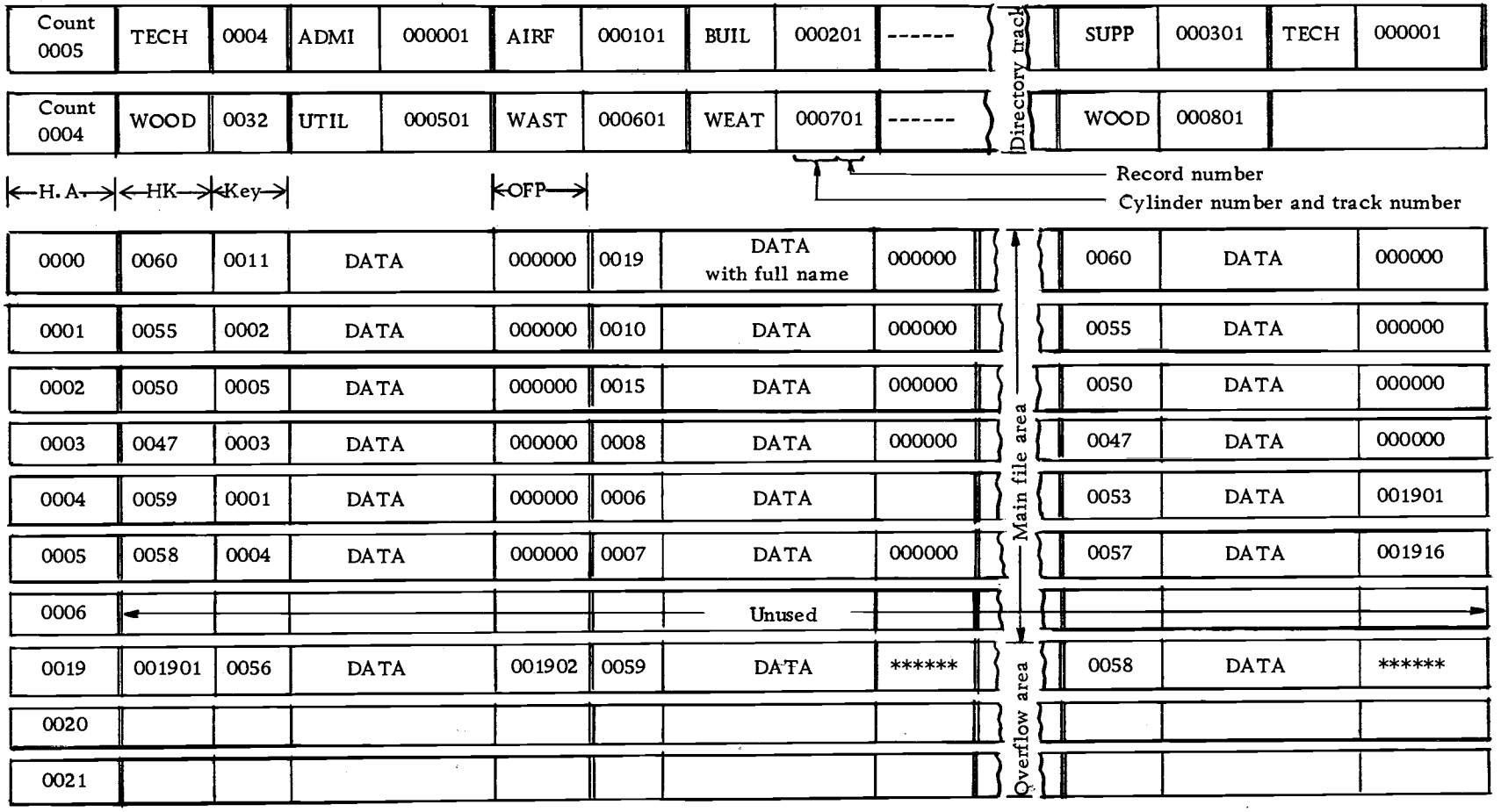


Figure 5.14. Partitioned File after addition of records "EGGY" and "MOOR".



H. A. = Home address; HK = Highest key in the member; DATA = Data of the record with record's full name; OFP = Overflow pointer.
This typical of partitioned disk file is not popular in used for on-line information system; its characteristics are not much different from the file system as shown in Figure 5. 12.

Figure 5. 15. Mapping of Partitioned File on Disk Memory, using Directory and variable length inverted list.

reorganized. Deleted directory entries can be reused. See Figure 5.12-5.15. The above illustrated numbers are based on the IBM File System (15).

Description of Main File Record. In practice the records in the members may be of fixed or variable length, blocked, unblocked, or undefined, and may be formatted with or without keys. See Figure 9.2 on page 202. The records in all the members must have identical formats. Members are stored one after another in the order in which they are written. Deleted member data areas can be reused to implement the Binary internal search. They have to be written in key sequence. See the illustrated example in Figures 5.12-5.15.

Disk Storage Space Requirement. Enough disk storage is required to hold the sequentially organized members and the directory. When new members are added, the Operating System allocates the additional supporting areas if the original area is full.

Use of On-Line Partitioned Disk File

The method used for the on-line partitioned file system is based on the system mentioned before in (2) under the On-Line Partitioned Disk File, page 79. The algorithms are

Adding the New Records to the File

Step 1. The user initializes the operating system by calling the specific name of the file. The operating system will automatically load the directory into main core memory. When the loading procedure is completed, it will notify the user.

Step 2. The user supplies the search argument key. In case the member name and its records have hierarchical relation, the user has to supply the member name and the full name of the desired record to the system.

Step 3. The operating system converts the supplied member into its corresponding address by one of the methods mentioned under Partitioned File Structure, and the internal search, linear, or binary, or tree search is performed.

Step 4. The operating system edits the records of the desired member on disk memory into core memory, effective number of "blocks", "sectors", or "tracks" of data is called in the core memory. In this evaluation assume that one track of data is called into core memory at a time.

Step 5. The internal search for proper location is performed. Searching may be repeated block by block or track by track until the proper location to which the new incoming record belongs is located.

Step 6. When the proper location is found and it is empty, the new incoming record is added there; the job is done.

Step 7. If the proper sequential location is found, and it is not empty in that track, the operating system will move the rest of the records higher in that track up one position; some of the higher key records in that prime track may be bumped out.

Step 8. The operating system will continue the search for the first available space in the overflow area. In this evaluation the cylinder overflow area is to be considered.

Step 9. If the first available space in the cylinder overflow area is found, the bumped record or the new record is written and the pointer is updated. In the overflow area the overflow records which belong to the same prime track of each member must be organized in the same chain.

Updating or Deleting a Record From the File

Steps 1 through 5 are the same as for adding a record to the file (a desired record searching task).

Step 6. When the desired record is found on the prime track of the desired member, the record is updated or its deleted-bit is set to 1 for updating or deleting respectively.

Step 7. If the desired record is not found on the prime track of the desired member, the operating system will continue the search for the desired record in the overflow track by using linked list processing (chain-search). When the desired record is found return to Step 6; the job is done.

Adding a Member Name to the Directory

Step 1. The user initializes the operating system by calling the specific name of the file, and the operating system automatically loads the directory into the main core memory. After the loading procedure is finished, it will notify the user.

Step 2. The user applies some operating system editors; the operating system will call for the member adding

routine. When it is ready the operating system notifies the user.

Step 3. The user supplies the new coming member name and the operating system converts the supplied member name into its corresponding fixed-length key by one of the methods mentioned under Partitioned File Structure.

Step 4. The operating system uses this unique key of the new member as the search argument for searching in the directory: the internal search, linear search, binary search, or tree search is performed depending on which was implemented.

Step 5. For directory searching, if the proper location is found and it is empty space, the operating system stores the fixed-length unique key of the new member there. It also allocates and supplies the address of the first record of this member (in some systems the user has to allocate and supply the address of the first record of the new member of the system).

Step 6. If the proper location is found and it is not empty, the operating system will move the rest of the records in the directory up one position and rewrite in the same fashion as addition of a new record to the strictly

sequential file.

Step 7. When the addition is finished, the operating system will return the updated directory to disk memory by applying an operating system editor. The job is done. Actually the directory needs only one or a few supporting tracks. Then it is not necessary to provide the overflow area for the directory level. In case the extension area is needed for the directory any suitable track in the same cylinder should be assigned and the pointer used to chain the extension track to the last original track.

Deleting a Member Name From the Directory

Steps 1 through 4 are almost the same as the addition of a new member name to the directory. The difference is using deleting operating system editor instead of addition operating system editor.

Step 5. For directory searching, search "equal" is used. If the desired record is found its corresponding data (that is the address of the first data record under this deleted member) is set to all zeros. Now all data records under the name of this deleted member are lost in the file system. Step 1 through 5 are repeated

for all member names to be deleted for the directory. In case the key of the member name is also cited in the file record, the record is accessed, and the key citations is deleted.

Step 6. When the deletion is completed, the operating system returns the directory to disk memory; the job is done.

Evaluation of Accessing Characteristics of the Partitioned Disk File

The Partitioned Disk File can be compared with other types of file organization, when its characteristics are known. The simulation of the partitioned disk file in this thesis is based on CDC-3300 with Disk 854 system. The simulating strategy is as follows:

1. For the simulation of the Partitioned File, the data records are assumed to be fixed, and blocked. Each record is formatted with a key. The records on the disk are organized as "a linked list inverted file" (11). The logical relation between the directory and the records in the main file is the same as that between the "directory inverted file" (11) and its corresponding data record in the main file.
2. Two methods of Directory Organization are evaluated here:

a) Single level Directory Partitioned Disk File. See details and computation in Example 3a, page 268.

b) Double level Directory Partitioned Disk File. See details and computation in Example 3b, page 283.

3. The full name of the record is used for accessing the record from the single level directory partitioned disk file system. See Example 3a. The member key name (the fixed-length unique key of Division names) and the full name of the data record in the main file are used for accessing the desired record from a double level directory partitioned disk file system. See Example 3b.

4. Simulating schemes and descriptions of both single level directory partitioned file and double level directory partitioned file are illustrated by block diagram model as shown in Figure 5.16 and Figure 5.17 respectively.

Purpose of the Evaluation

The purpose of the evaluation is to measure the following parameters of the Partitioned Disk File and to find out its characteristics. The strategies of the evolution are as follows:

1. To compute the access time per record retrieval of the data record on disk.

2. To measure the storage space requirement.

3. To find the characteristics of "average throughput/time record retrieval", and "achievable throughput-rate-capacity" system relative cost.
4. To find the "cost/effectiveness" characteristics (user operating cost per call, unit cost).
5. To compare these characteristics with the other methods of file organization.

Simulating Block Diagram Model

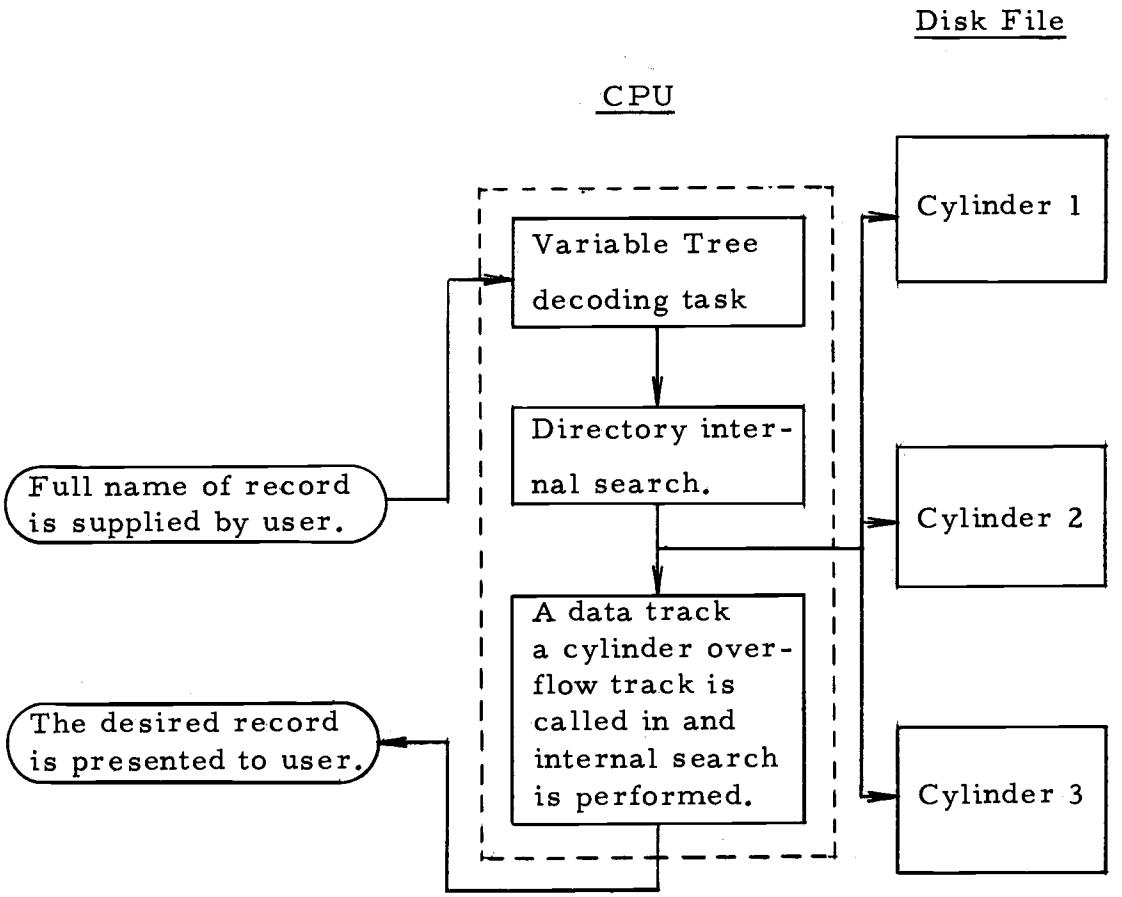


Figure 5.16. Block diagram showing the simulation of accessing a random record from the single level Directory Partitioned Disk File.

The accessing of a random record from a single level directory partitioned disk file is shown in Figure 5.16. The user supplies the full name of the desired record to the system, the operating system will convert this full name into a fixed-length unique key and it uses this key to search the directory for the address of the first data record of the desired member; that is equivalent to roughly determining in which cylinder and track the desired record should be. The access-arm will position on the desired cylinder, and the desired data-track is read-in; the internal search is performed. When the search is satisfied, the operating system will present the desired record to the user; if the search is not satisfied, the operating systime will call for the "unfound routine" and notify the user "NO SUCH NAME IN THE FILE". See more details in Example 3a, page 268.

The method of accessing a random record from the double level directory partitioned disk file is shown in Figure 5.17. The user supplies a unique fixed-length key of the desired member name to the system, the operating system will search for the beginning address of the desired sub-directory and it will notify the user when the job is done. The user has to supply the full name of the record to the system, the operating system will convert the full name of the desired record into fixed-length unique key. The operating system will use this key to search in the desired subdirectory for the desired cylinder and track. The access-arm positions on the desired

cylinder and the desired data track is read-in; the internal search is performed. When the search is satisfied the desired data record is presented to the user; the job is done. If the search is not satisfied, the operating system will call for the "unfound routine" and notify the user "NO SUCH NAME IN THE FILE". See more details in Example 3b, page

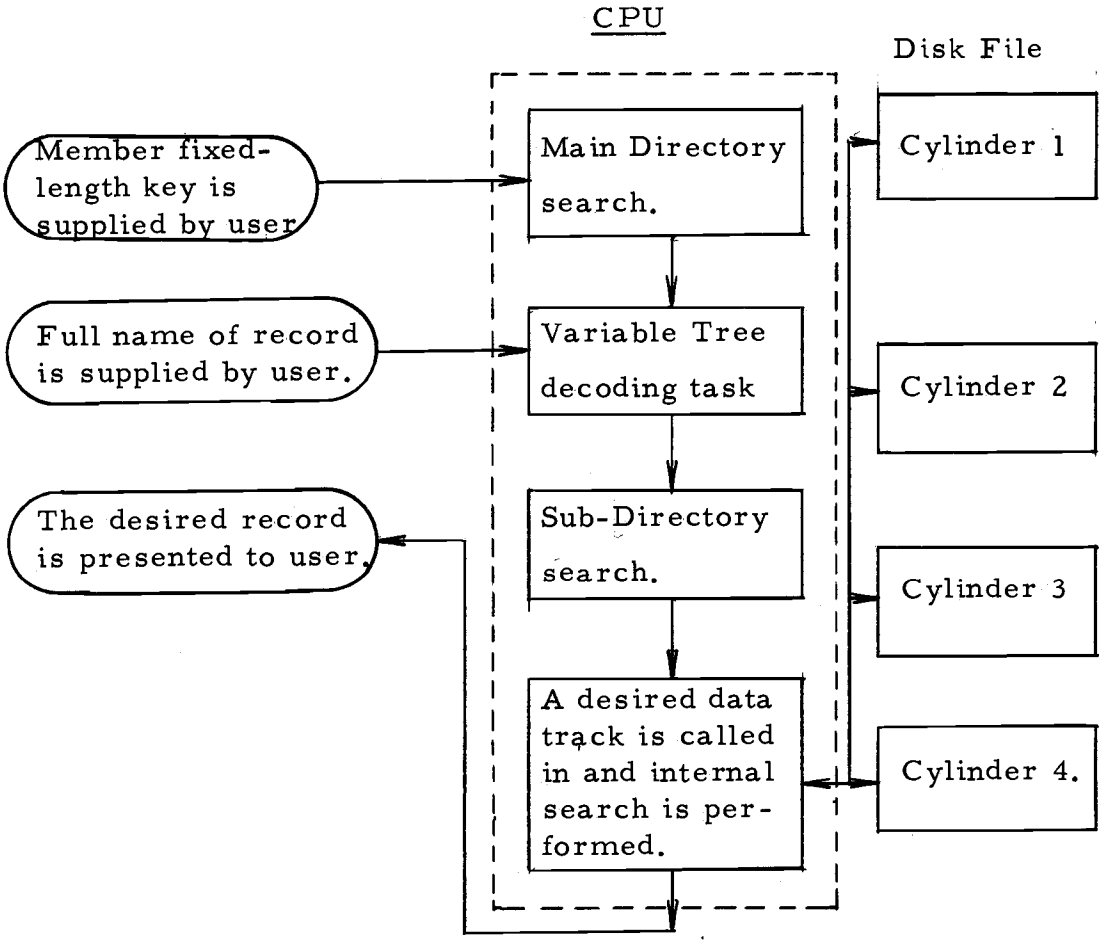


Figure 5.17. Block diagram showing the simulation of accessing of a random record from double level directory Partitioned Disk File.

Results of the Evaluation of the Partitioned Disk File

Results of the evaluation of the partitioned disk file and the comparison of this file characteristics with those of the other method of file organization are shown in Chapter VII.

VI. DIRECT FILE STRUCTURE AND USE

Direct File Structure

This chapter concerns some commonly used methods of direct (random) organization, as well as the access methods provided for files so organized.

General Description

With direct or random file organization, there is a definite relationship between the key of a record and its address. This permits rapid access to any record if the file is carefully organized. The records will probably be distributed nonsequentially throughout the file. To permit the key sequence processing, a preliminary sorting routine is applied.

Addressing

In direct file organization there is no need to use the key name directory. This saves a lot of required memory space. The major components are "mapping function" (Hash Function) and "master file". The graphical representation of direct file organization can be as in Figure 6.1.

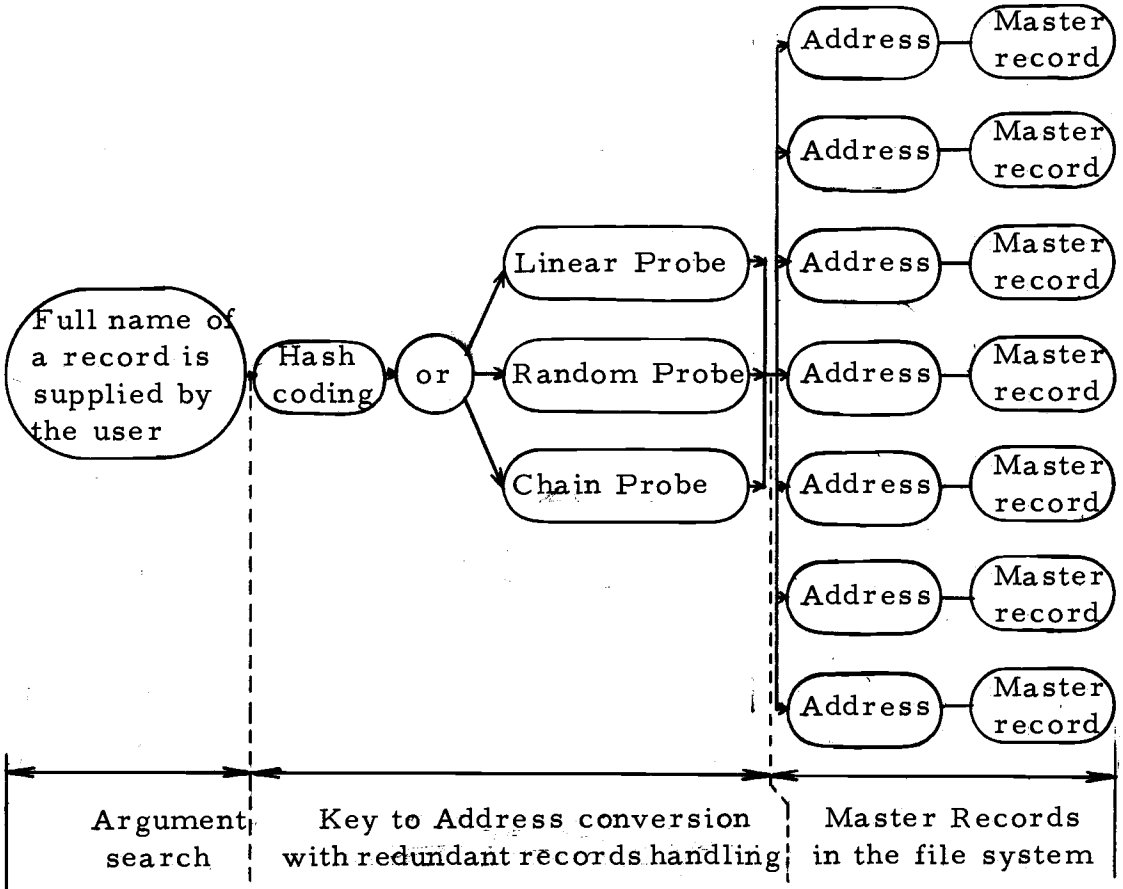


Figure 6.1 Graphical representation of direct file organization.

Mapping Function

The other well known name of the mapping function is "hash function". The technique of converting the key name of a record into the address of a record is known as "randomizing" or "hash coding". In fact, with Direct File Organization the user generally develops a record address that ranges from zero to some maximum address; that is, the mapping area (storage supporting space) may be considered as a single array or table. This storage supporting

space is called "hash table" or "scatter storage table".

Hash code technique is the fastest known method of searching and insertion. It allows either searching or insertion in one step, unless the table is over half full. In a hash table, the position of any item is determined by its hash code, used as its address in the table, which is a number easily obtained from the record name of the item. There are many techniques in performing hash coding, as mentioned by Robert Morris (24), Schay, G., Jr., and W. G. S. Pruth (31), Johnson, L. R. (17) and McIlroy, M. D. (26).

From experiments the writer has found that the simplest hash code is the "division hash code". If the array has length n, that is, there are maximum n entries contained in the table, and the largest integer that may be contained in a computer word is m, the division hash code of a word is found by dividing it as an integer by n. This produces a quotient and a remainder; the remainder is an integer between 0 and n-1, and may be used as the address of the entry in the table or table index.

Items are entered into a table with a table index which is computed from the record name of the item by means of some hash coding method. See detail in Example 4, Appendix B. As long as no two inserted items have the same hash code, searching and insertion are performed each in a single step regardless of the size of the table. When two items have the same hash code, a collision

or redundancy is said to exist. In this case the second item is called the "secondary record" or "overflow record", while the first item is called "first record" or "home record". The overflow record must be put in another place in the table. Although it takes extra time to search or insert the overflow record, if the hash codes are randomly distributed, the average number of searches is less than two, even for a table that is 80 percent full.

The usual hash coding methods involve the calculation of a k-bit field, which is assumed to be a random integer between 0 and 2^k-1. Thus the table size is restricted to the value of 2^k.

Hash Code Redundant Handling

When hash coding is performed, it becomes possible for the computed address of different keys to be the same, causing a collision between the storage locations allocated to each. Some other place in the table must be found for one of the items. It is initially assumed that once an item has been entered, it is never moved or deleted. So another potential place must be found for the new entry. In general, when the table is nearly full, many redundancies or collisions may occur while the table is being probed for an empty slot. Hence some procedure is needed which generates additional calculated addresses until an empty slot is found. Of course, the same procedure for generating additional calculated addresses

must be used when the item is later looked up.

In practice, when a hash coding routine is called for it is not necessary to specify whether an item is being entered or being looked up. What is required of the routine is to determine the address at which the offered key belongs and to notify the user whether the key has already been entered. Then the calling routine can make the entry or extract the information, as appropriate. The procedure will be to generate successive hash addresses until encountering either a slot that contains the desired key or a empty slot. In the latter case, the key is entered in the empty slot, if it is entered at all.

Many methods of resolving the collisions problem have been suggested by Robert Morris (27). The particular method to be used in a specific application should be chosen carefully, since the method of handling collisions will affect the efficiency of the technique and the difficulty of the programming task. The three conventional methods of handling the collision problems are briefly mentioned here.

Linear Probing. In linear probing, which is also called the "open method", the filing algorithm is as follows:

1. Calculate the address of the record to be entered in the file.
2. Input the record in that location if it is empty.
3. If the previous step is not successful, add i to the address considered in step two, where i is an integer such as

   i = 1, 2, 3, ... etc.

4. Repeat step two.

In figures 6.2-6.4 it is assumed that the size of the table is N=8 and the following key names are going to be entered in the hash table: MOORE, SMITH, JONES, BLACK, BROWN, JOHNSON, BARONE, and OWEN (Details on pages 109 - 110).

In this method, upon collision, a search is made forward or backward from the nominal position, the initial calculated addresses, until the desired entry is found or an empty space is encountered. The search is made circularly past the end of the table to the beginning, if necessary. If an empty space is encountered, that space becomes the home for the new entry.

The efficiency of the linear probing method can be analyzed by techniques similar to those used by Schay, G., and W. G. Spruth (31) to evaluate a related method. The result is that, to within suitable approximation, as shown in Equation 6.1, if E is the average number of probes necessary to look up an item in the table

Then $E = (1-\alpha/2)/(1-\alpha)$ (6.1)

where α is the loading factor of the table.

Random Probing. With random probing, the algorithm of generating successive calculated addresses to handle collisions is as follows:

1. Calculate an address in the table by using some transformation (hash coding) of the key as a table index or address.
2. If the item is already at this address, or if the place is empty, the job is done.
3. If some other key is there, call a pseudorandom number generator for an integer offset . Make the next probe at location $1 + \rho$ and go to step 2.

In practice the random number generator must generate every integer from 1 to N-1, N being the size of the table, exactly once. When the generator runs out of integers, the table is full and the entry cannot be made. See details of the random number generator in Example 4, Appendix B, and an example of random probing in Figures 6.5-6.6 on pages 113 - 114.

The efficiency of the random probing method is expressed in terms of E, the average number of probes necessary to retrieve an item from the table. It happens to be equal to the average number of probes which were required to enter the items originally. The value of E depends on the fraction α of the table which is occupied but not on the size of the table. In case N is the size of the table, and k represents the number of items in the table.

Since E is equal to the average of the expected number of probes to retrieval on record from the table, by mathematical manipulation and approximation, as in equation 6.2

$$E = -(\frac{1}{\alpha}) \log (1 - \alpha) \qquad (6.2)$$

where

α is the loading factor. See the details in Example 4, Appendix B.

Deletion of entries by using this scheme is a somewhat complex process. One cannot mark an entry as empty in order to delete it because other entries may have collided at that place, and they would become unreachable. The hash addresses for every entry in the table would have to be recomputed and some of them moved in order for the gap caused by the deleted entry to be closed up. A much more convenient method of deletion is to reserve a special sign for a deleted entry. When a search is made for the proper place of a key, the search continues if a deleted entry is encountered. A new item can be stored in place of any deleted entry. The disadvantage of this method is that the lookup time is not reduced when the entries are deleted; only the lost space is reclaimed.

Direct Chain Probing. Another method of resolving collisions is called direct chaining, and is considerably more efficient in terms of the number of probes per record-retrieval. In this method, part of one of the words or extra spaces in each entry is reserved for a pointer to indicate where additional entries with the same calculated addresses are to be found, if there are any. So all the same

calculated addresses are to found on a chain (or linked list) starting at that address. The last entry on each chain must be distinguished in some way, such as by having "***" as a zero pointer, and end of the chain.

The standard algorithm of direct chain probing is as follows:

1. When a key is to be looked up, its hash address is computed, and then
   — if that address is empty, the key has not been entered
   — if that calculated address is occupied, search down the chain starting from that address; if the key is not encountered, it is not in the table.
2. When a new item is to be entered in the table, its hash address is computed and then
   — if that address is empty, the item is installed there.
   — if that address is occupied by the item which is the head of the chain, the next available space (or unallocated cell) in the table is found by a "search for available space routine", and the new item is placed in the new available space. Then the new entry is inserted in the chain, the pointers are updated both for the previous successive entry and the new-coming entry in the same chain, starting from the calculated address.
   — if that address is occupied by an entry which is not the

head of the chain, i.e., by an entry which is not at its own calculated address, then the old entry must be moved to another slot and the new entry inserted in its place. Search for the next available space is required for installing the old entry, and updating the pointers of the chain it is on.

The disadvantage of this method is that the entries must be moved in the storage, which makes the programming of handling this task more complicated than by other methods. Robert Morris (27) suggested that when a newly entered item is to be placed on the chain, it is usually more profitable to place it near the head of its chain rather than at the end of the chain. The other inefficiency of the direct chain method is that it requires more space.

An attractive feature of this method, from the result of an experiment made by the writer, shows that when the table is almost filled up or completely filled up, the new items can be placed in the table (or even overflow area) with no change in the strategy of making entries of looking them up. The efficiency of this method is still quite good even after overflow has occurred.

The average number E of probes necessary to find an item using this scheme has been calculated by Johnson, L. R., (17) as shown in Equation 6.3.

$$E = 1 + \alpha/2 \qquad (6.3)$$

where α = k/N, the loading factor

k = number of entires

N = table size.

See the values of Equation 6.3 by computation and by simulation on pages 304 - 309.

The standard rule for deleting items entered by chaining is as follows:

1. An entry not stored at its calculated address may be set as empty and its former chain joined around it (no link connected to deleted entry).
2. An entry stored at its calculated address, but with no chain starting from it, may merely be set as empty.
3. An entry stored at its calculated address with a chain starting from it must either be marked or deleted, or one of the items on its chain must be moved to the calculated address and the chain properly set up.

See the illustrated examples on pages 116 - 118 and the computation of simulation in Example 4, Appendix B.

Direct File Maintenance

Maintenance of the direct (random) file is influenced by the mapping function and the technique of handling the secondary

records selected by the user.

Suppose an effective mapping function is selected.

Case I. When the linear probing method is chosen to handle the secondary records.

To add a new record to the file:

Step 1. Compute the calculated address of the record, l, to be added to the file.

Step 2. Look up the calculated address location and input the record in that location if it is empty. The operating system furnished the addition for that record. Otherwise, go to Step 3.

Step 3. Add i (i=1, in the example on page 109) to its calculated address forming the new address, l + i, where i is an integer such that i = 1, 2, 3, etc., depending on the user's solution and go to Step 2.

When a record in the file is deleted:

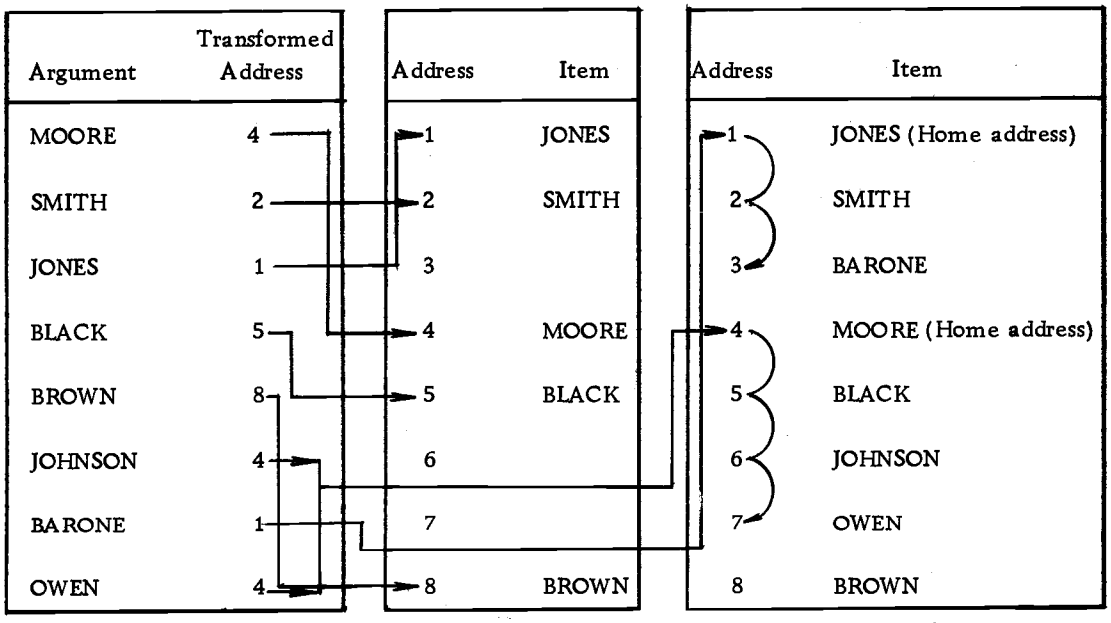
Step 1. Compute the calculated address of the record, l, to be deleted from the file.

Step 2. Look up the calculated address location and if the key name of the record is encountered, delete it and the deletion is finished. If that location is empty, the operating system will notify the user that there is no such record name in the file. Otherwise go to Step 3.

Step 3. Compute the new address by 1 + i (i = 1 in the example in Figure 6.2) where i has the same meaning as in the adding procedure, and repeat Step 2.

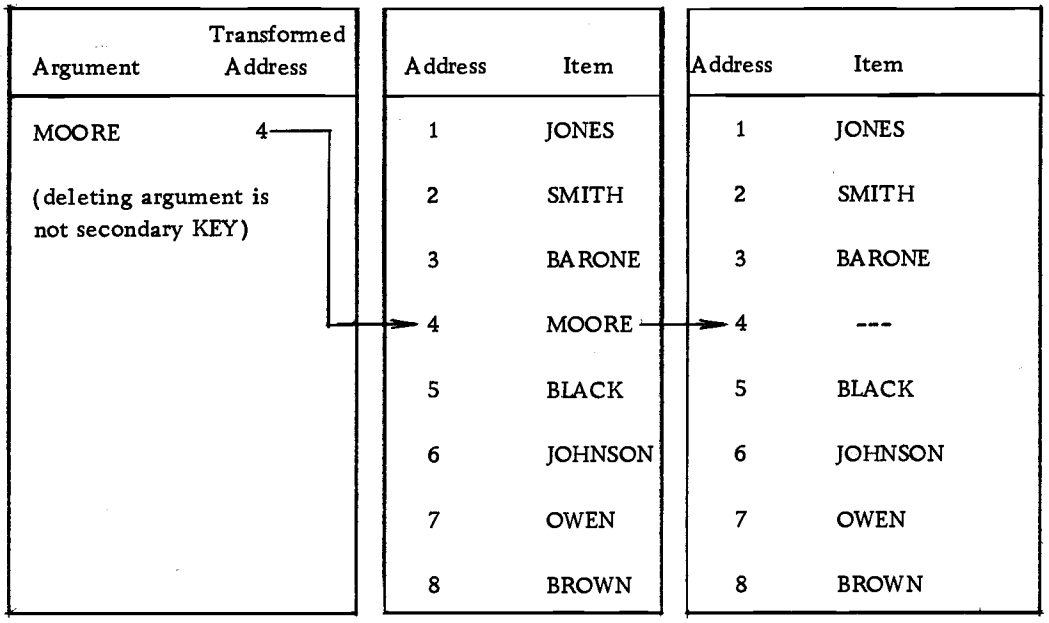
The procedure for updating the data of a record in the file is the same as for deleting it, but when the search is matching, read out the desired record instead of deleting it.

See the illustrated examples in Figures 6.3 and 6.4 on page 110.



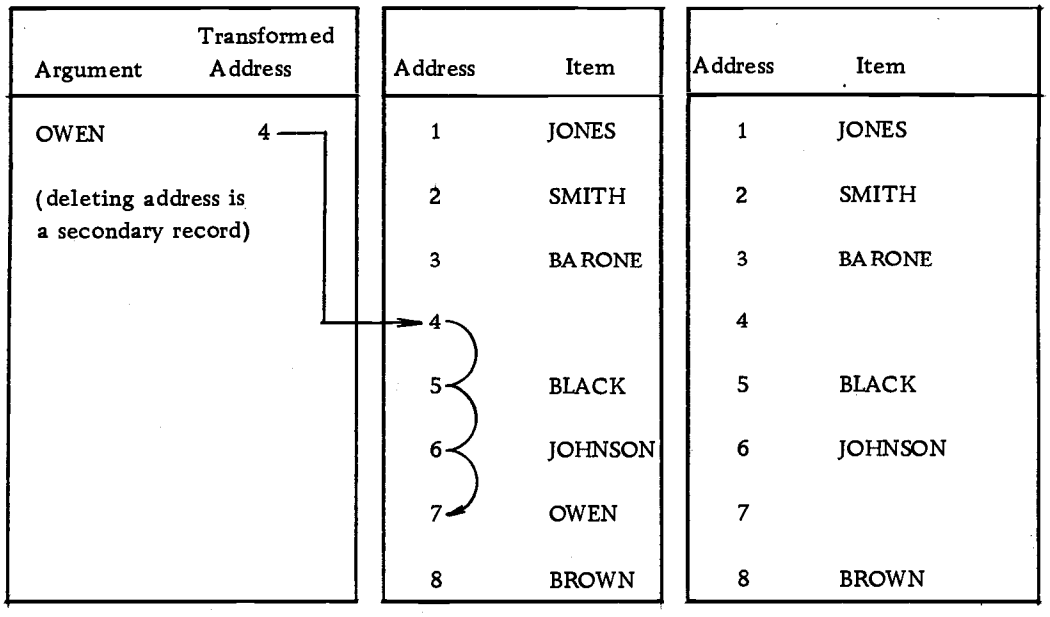
a) List of Arguments and Transformed Addresses b) File after Step 1 and 2 c) After Steps 3 and 4

Figure 6.2. Addition of the new items into the file with Linear probing.



a) Deleting Argument and its Transformed Address. b) File after Steps 1 and 2 c) File after Steps 3 and 4

Figure 6.3. Deletion of the items which are not secondary records from the file with Linear probing.



a) Deleting Argument and its Transformed Address b) File after Steps 1 and 2 c) File after steps 3 and 4

Figure 6.4. Deletion of the items which are secondary records from the file with linear probing.

Case II. When the random probing method is chosen to handle the secondary records.

To add a new record to the file:

Step 1. Compute the calculated address of the record, l, to be added to the file.

Step 2. Look up the location of the calculated address. Input the record in that location if it is empty; then the task for adding that record is finished; otherwise go to Step 3.

Step 3. Call pseudorandom number generator routine to generate random number, ρ, compute the new address by $l + \rho$, and repeat Step 2.

To delete a record from the file:

Step 1. Compute the calculated address of the record, l, to be added to the file by selected mapping function.

Step 2. Look up the location of the calculated address. If the key name of the argument record and the key name of the calculated address location are matching, delete the recoded, and the deletion is finished. If that location is empty, the operating system will notify the user that there is no such record name in the file, otherwise go to Step 3.

Step 3. Call for random number generator routine again to

generate the new random number , compute the new address location by $l + \rho$, and repeat Step 2.

The procedure for updating data of a record in the file is the same as that for deletion, but when matching occurs the record is read out instead of being deleted.

See the illustrated example of addition and deletion of records for the file with random probing, Figure 6.5.

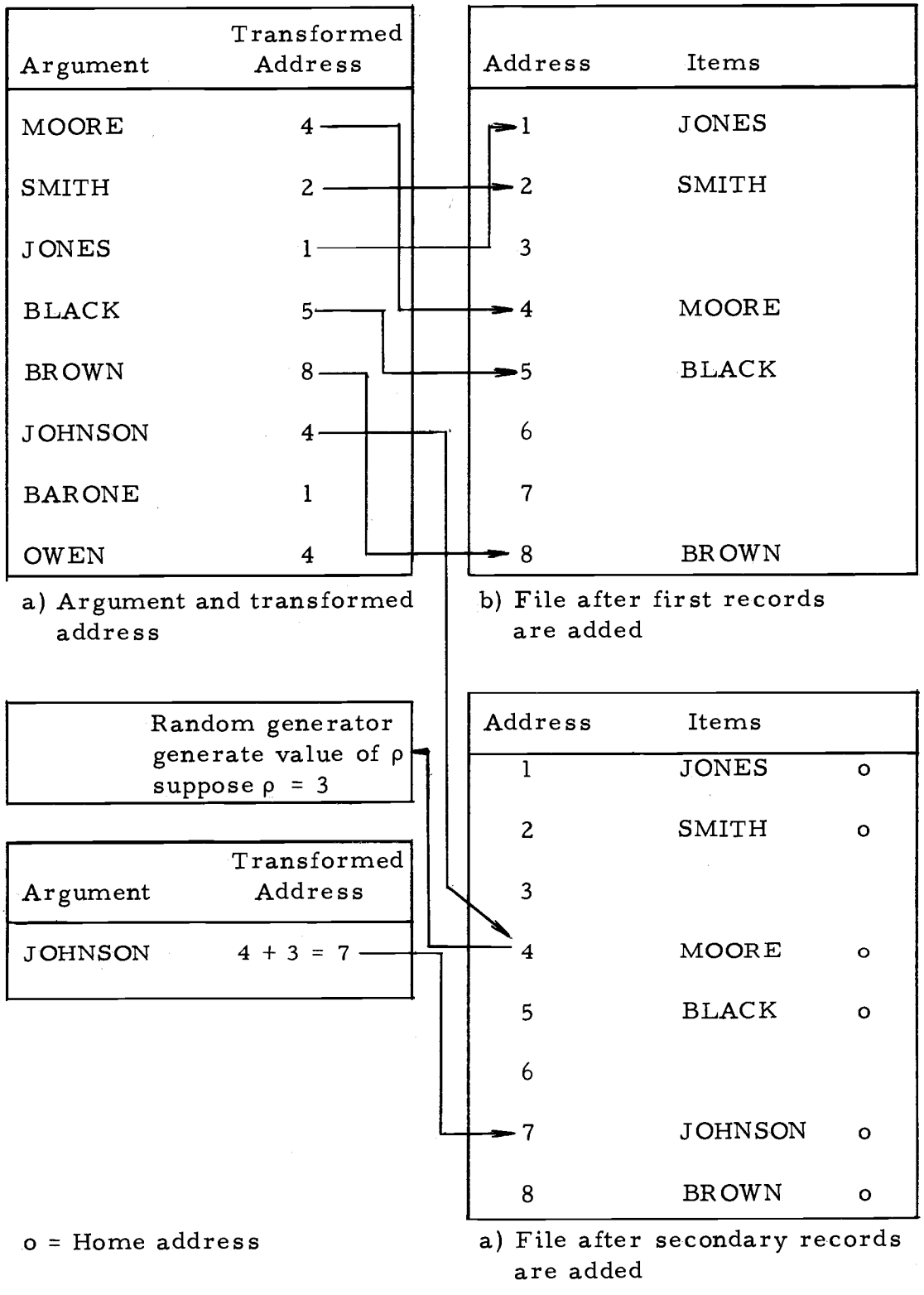
Case III. When the direct chain probing method is chosen to handle the secondary records.

To add a new record to the file:

Step 1. Compute the calculated address of the argument record, l.

Step 2. Look up the location of the calculated address. Input the new record in the location immediately after the hash address of the key if that location is empty; i.e., this record is the first record, and is also the head of the chain. If the location of the calculated address is not empty, go to Step 3.

Step 3. Call for the routine to find the first available space in the table, depending on the result of the lookup from Step 2. If the hash address of the argument record and the hash address of the installed record are matching, the new record becomes the secondary



a) Argument and transformed address

b) File after first records are added

o = Home address

a) File after secondary records are added

Figure 6.5. Addition of records to the file with Random probing.

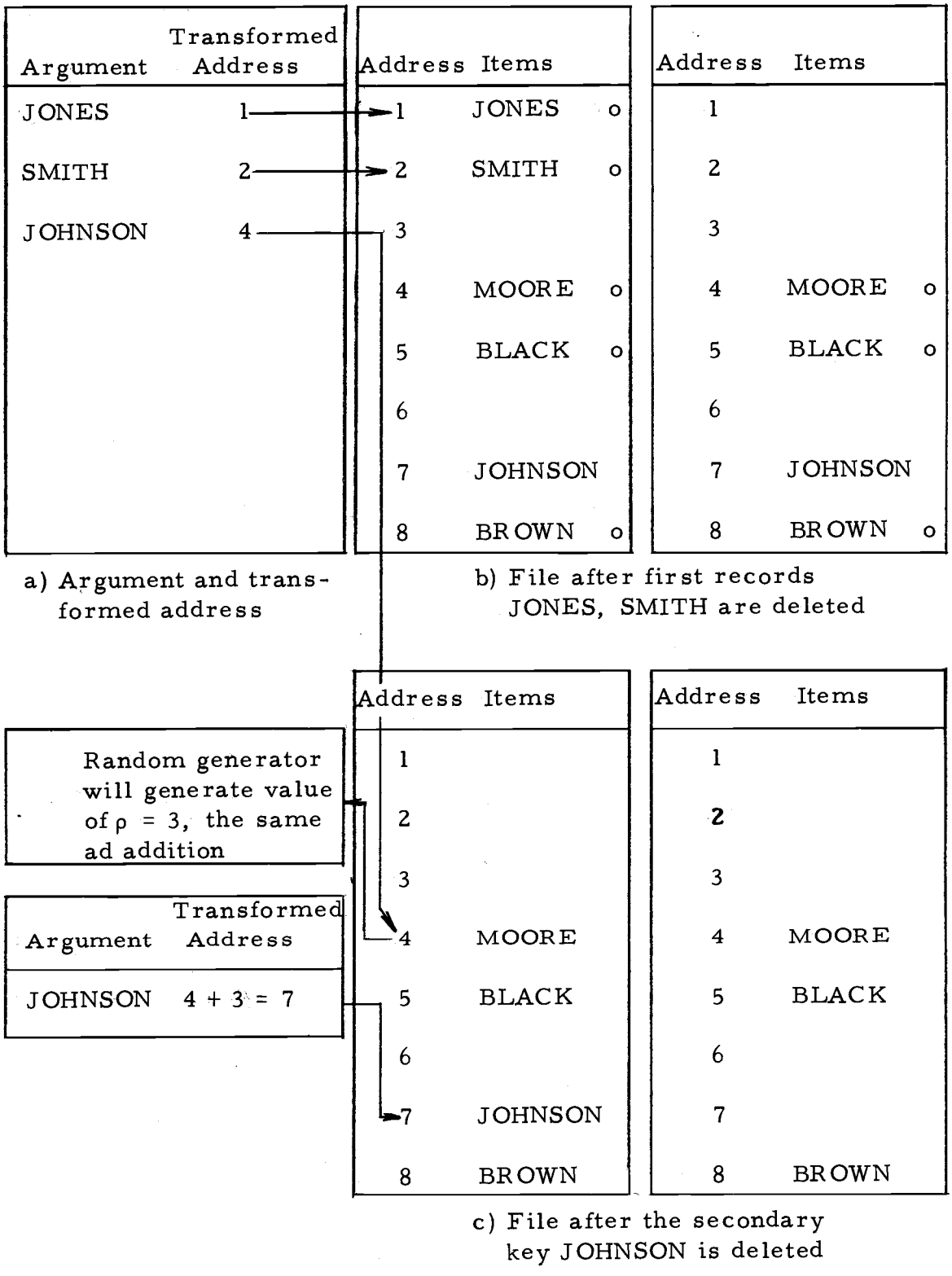


Figure 6.6. Deletion of records in the file with Random probing.

record. Install this new record in the new available space, preceded by the hash address of the key. If the above comparison is not matching, the new record is the first record and is also the head of the chain at that lookup location. Move the successive secondary record in that chain into lookup location and adjust the pointers in that chain. The deletion is complete. Otherwise go to Step 4.

Step 4. If the key of the desired record is encountered at some place down the chain, delete it, and the deleted record is the secondary record in the chain. Update the pointers of the chain. The deletion is finished. For updating the data of the record in the file, the steps are the same as for deletion, but when the desired record is encountered it is read out instead of being deleted. There is nothing to change, not even the desired record at the head of the chain.

To delete a record from the file:

Step 1. Compute the calculated address of the key name, 1, to be deleted.

Step 2. Look up the location of the calculated address, 1. If it is empty, the operating system will notify the user that there is no such record name in the file. If it is

occupied, search down the chain starting from this location. If the key is not encountered, the operating system will notify the user that there is no such key in the file. Otherwise go to Step 3.

Step 3. If the key of the desired record is encountered at its calculated address, delete it and the deletion is complete.

See the illustrated example of addition and deletion of records in the file with direct chain probing in Figures 6.7, 6.8, 6.9, and 6.10. Note * is the end of the chain.

Argument	Transformed Address
MOORE	4
SMITH	2
JONES	1
BLACK	5
BROWN	8
JOHNSON	4
BARONE	1

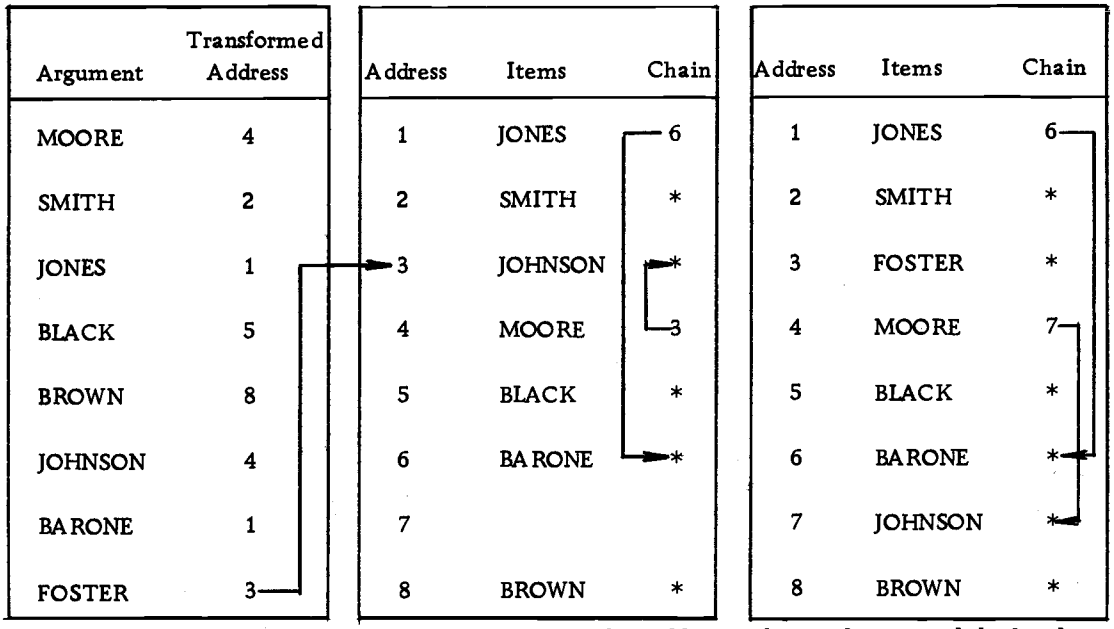
a) Listing of arguments and transformed addresses.

Address	Items	Chain
1	JONES	-
2	SMITH	-
3	-	-
4	MOORE	-
5	BLACK	-
6	-	-
7	-	-
8	BROWN	-

Address	Items	Chain
1	JONES	6
2	SMITH	*
3	JOHNSON	*
4	MOORE	3
5	BLACK	*
6	BARONE	*
7		
8	BROWN	*

b) File after addition of first records and secondary records; the pointers are updated.

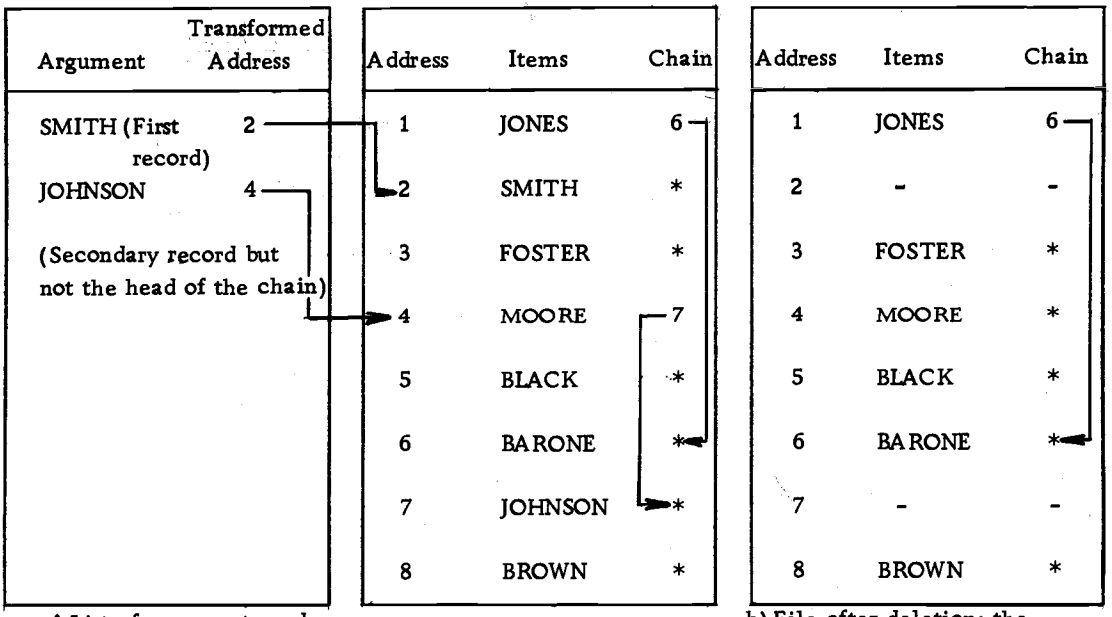
Figure 6.7. Addition of items in the file with direct chain probing.



a) List of addition arguments and transformed addresses

b) File after addition of secondary record the head of the chain; the records, and pointers are updated.

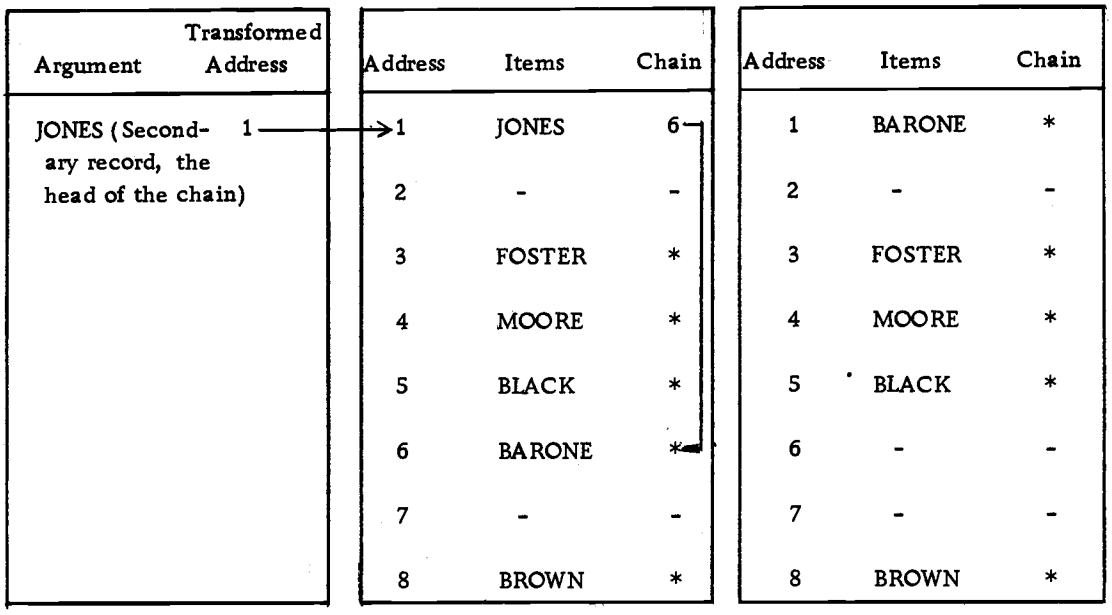
Figure 6.8. Addition of secondary record, the head of the chain when the calculated address of the new coming item is occupied.



a) List of arguments and transformed addresses being deleted.

b) File after deletion; the chain is updated.

Figure 6.9 Deletion of the items which are first record and secondary record, not the head of the chain.



a) Deleted argument and transformed address.

b) File after deletion; the head of chain and the pointer are updated.

Figure 6.10. Deletion of the item which is the secondary record, the head of the chain.

Direct Disk File for On-Line System

As direct file organization permits rapid access to any record from the file, on-line or realtime system designers have considered this attractive feature of direct file organization. There are several computer manufacturing companies which design their own on-line or realtime information system, using direct file organization to handle the information of the individual system.

Direct File Organization Supported by Disk

When the direct organization file is supported by Disk, according to the nature of the mapping function, the records will probably

be distributed non-sequentially throughout the file. See illustrated example in Figure 6.11 on page 121. The mapping of master records on the disk surface is the same as that in the sequential file. The user has to figure out beforehand how many words are required for one record (the space in core memory), and how many records can be contained in one track, in one cylinder, or in one disk unit. The user has to keep in mind that even the address of the block of the master records in the disk is assigned, but the disk hardware system as designed can be addressed only by cylinder number, head number, and track number. Only the portion of a track (an effective number of blocks of records transferred between core memory and disk, one at a time), is written as read into core memory. The exact location of the record in the file can be located directly in internal core memory.

Therefore to address a record from disk the following strategies have to be used:

1. The approximate location of the desired record has to be known as to what cylinder or what head is to be used to read or write the desired information, and the number of the track in that cylinder, in which it is to be located.
2. The disk control program will set the specified head on the desired cylinder and the desired track.

3. The reading or writing begins. If it is a reading process, the number of effective blocks of information are read into the main core memory. That is the numbers of the records, containing in the desired track, are copies and mapped into the core memory.
4. The desired record is located by some means (Linear search, or Binary search if the successive records are in sequence of keys, because sort routine has been introduced beforehand).

Addressing

With direct file organization, the user generally develops a record address that ranges from zero to some maximum track address. However, the addresses are noncontiguous. For example, the address of the last track on the first cylinder of an IBM disk unit 2302 is 0045, while the first track on the second cylinder is 0100. Furthermore, the file may start at other than the first track of a device, and it may occupy several nonadjacent areas. According to the nature of the Disk Operating System (DOS), the user is permitted to refer to a relative cylinder and a relative track address. Suppose N tracks are allotted to a file. The user refers to relative track 0 through N-1. The input-output control system will convert this number to the corresponding absolute track address.

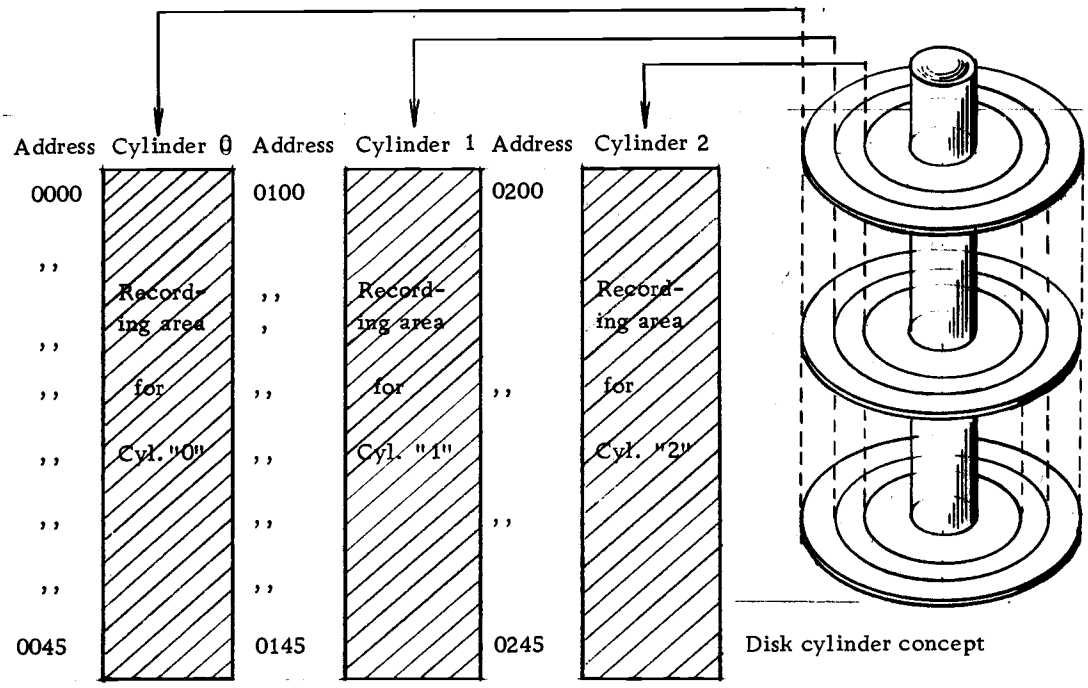


Figure 6.11. Address pattern on Disk memory.

With some operating systems such as the Basic Operating System (BOS) of IBM, the user programs the steps to convert the relative track address to an absolute track address of the format shown in Figure 6.12 on page 122 which illustrates an absolute track address format of D854, disk memory. Each byte (1 byte = 8 bits) in the address is a binary number. When the user wishes to refer to a particular record, he must supply either its key or its identifier (i.e. cylinder number, head number, and record number) as well as the track reference.

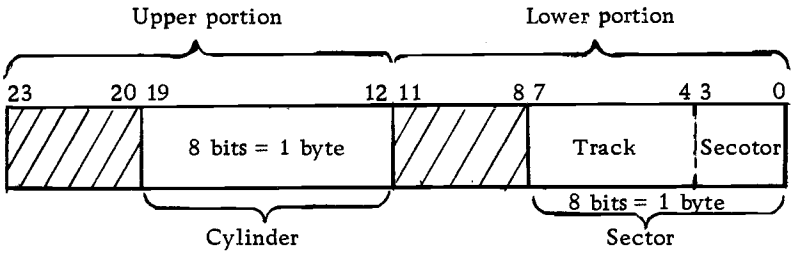


Figure 6.12. Disk storage drive address format, based on D854, disk memory.

Directly Addressed File for Disk. For direct addressing, every possible key in the file converts to a unique address. This makes it possible to locate any record in the file with one search and one read. The conventional technique of direct addressing is a follows:

1. Using the key as the address. In order to be able to use the key of a record directly as its address, the record must be of a fixed length, and the key must be numeric. One computation is required. Divide the key by the number of records per track; the quotient equals the relative track address, and the remainder plus one equals the record number (where record 0 is used as a capacity record). This method of direct addressing not only allows minimum disk time when processing at random, but also provides for sequential processing, since the records are written in key sequence. The disadvantage is that there may be a

large amount of unused direct access storage. A location must be reserved for every key in the file's range, even though many keys are not used. Furthermore, this method is similar to hash code, and uses the numeric character as the record's key name. For example, the user may use the student number or social security number, worker number or customer number as the key name of the record.

2. Using a Cross-Referenced List. With this method, each record in the file is assigned an address and a cross-reference list of keys and assigned addresses is maintained. Since the address must be looked up, then the list, as well as the file, must be kept up-to-date. The list may itself be a file, recorded on a disk. Although any record can be located directly when its address is known, time is required to look up the address in the list. The index sequential file is a variation of this method.

Indirectly Addressed File for Disk. Indirect addressing is generally used when the range of keys for a file includes such a high percentage of unused records that direct addressing is not feasible. For example, customer numbers range from 0001 to 9999, but only 3000 of the possible 9999 numbers are assigned. Indirect addressing is also used for alphabetic keys.

With indirect addressing, the range of keys for a file is

compressed to the smaller desired range of addresses by some sort of computation. This technique is called "randomizing". It inevitably causes collision, redundant or secondary records.

Two objectives must be considered in selecting a randomizing technique (or mapping function) for disk memory.

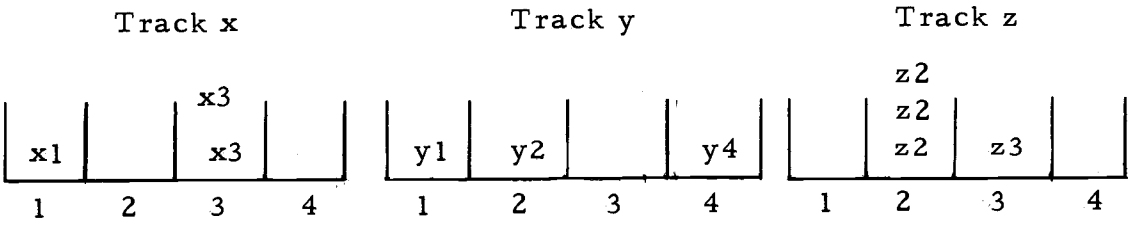
1. Every possible key in the file must randomize to an address in the allotted range, and
2. The addresses should be distributed evenly across the range so that there are few redundancies.

With disk memory, a record that is written where it "belongs" (at the address to which its key randomizes, or its calculated address) is called a "home record". Any other records whose keys randomize to this address are "overflow records". The overflow records should be kept to a minimum because of the additional time required to locate them.

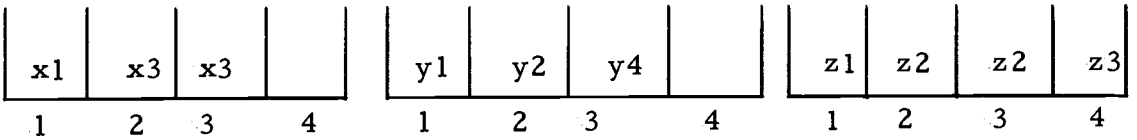
A way to minimize overflow records is to provide more space for the file than is actually required to hold all the records. The term "packing factor" or "loading factor" means the percentage of allotted location that is actually used. For the indirectly addressed file, an initial packing factor of 80-85% is suggested by IBM (15). For example, a 10,000 record file packed 83% would be allotted space for 12,000 records.

The technique to minimize overflow records is to randomize

to track address rather than to record address. In case we randomize to record addresses, all redundant records cause overflows. That is, the disk system simply cannot provide the alloted space for the redundant records that the core memory can. See Figure 6.13a. Thirty percent of the records are redundant and 30% are overflows. Randomizing to track address, causes many redundancies, but no overflow until a track is full. As shown in Figure 6.13b, 70% of the records are redundancies (the redundant records are two x3, y2, y4, two z2, and z3). If randomizing to record number, the commands to locate the desired record are Seek, Search Identifier Equal, Read Data. If randomizing to track number, the commands are Seek, Search Identifier Equal, and Read Data. It is evident that both sets of commands take the same amount of time.



a. Randomizing to record address.



b. Randomizing to track accress.

Figure 6.13. Redundants and overflows in disk memory.

Randomizing Techniques Used for Disk

There are many randomizing techniques to convert the key of a record into its address in disk memory. Selecting a good one for a particular file may require some trial and error. IBM (15) has suggested that an effective randomizing technique should cause no more than 20% redundancies in excess of the number of records per track.

The most popular randomizing technique which is simple and often gives good results is called "Division/Remainder Method". The key is divided by a prime number, a number evenly divisible only by itself and by one, that is close to the number of addresses alloted to the file. The remainder is used as the address.

Example 1. The problem is to load 8000 200-byte records on a disk with randomizing to track address with 80% packing.

1. With 80% packing, 10,000 locations are required.
2. Only 16 records can be loaded per track, so 625 tracks are required.
3. A prime number close to 625 is 619.
4. Divide the key by prime number 619.
5. The remainder (000 to 618) equals the relative track address.

Example 2. Same as above, but randomizing to record address.

1. A prime number close to 10,000 is 9973.
2. Divide the key by 9973.
3. Divide the remainder in Step 2 by the number of records per track, which equals 13 in this case.
4. The quotient equals the relative track address; the remainder plus one equals the record number.

This method can also be used both for numeric keys and non-numeric keys. Using binary arithmetic will probably give better results than using decimal arithmetic, since the uniqueness of the letter and special character in the key is retained.

The division/remainder method automatically achieves the first objective, to have all keys converted to addresses within the allotted range. Whether it achieves the record objective for a particular file, that is, to have few (not more than 20%) redundancies is determined by testing (simulating) it. See details in Example 4, Appendix B.

Description of a Direct Disk File

With direct organization, the records may be fixed-length records, variable-length records or undefined-length records. See Figure B.2, page 202. They may be formatted with or without keys. In case the file is indirectly addressed and randomizing to track

address[2/] for efficiency the records have to be formatted with keys. If not, each record on the track must be read to determine if it is the desired record or not. The records may be blocked or unblocked. The access method for directly organized files in the operating system handles physical records rather than logical records. So if the file is indirectly addressed (by use of an alphabetical key), the records are probably unblocked.

In most direct organization files, the record number zero, RO of each track is used as a capacity record. It contains the address of the last record written on the track and is used by the operating system to determine whether a new record will fit on that track. The capacity records, which are originally written for a file by a utility program, are updated by the operating system as records are added to the file. They do not account for deletions. When a track is full, it remains full as far as the operating system is concerned, until the file is reorganized, even though the user deletes records.

An indirectly addressed file generally consists of just one logical area, main file area, which may actually be several non-adjacent physical areas. The location of overflow records and secondary records is up to the user, but they are generally put in an

---

[2/] Randomizing to track address: key of records are converted into their corresponding track address and count only the synonyms in excess of the number of records per track. See Figure 6.13b on page 125.

unused location in the main (and only) file area. Secondary records can be put in a separate area if the user desires. The disadvantage of doing this is that each overflow record will require an additional seek. One file area should be used and a good randomizing technique selected so that the file is not being packed too tight. The overflow records are likely to be in the same cylinder as the home record. This will eliminate the need for an additional seek.

On-Line Direct File Maintenance

As the user has complete freedom in deciding where records are to be located in a direct file, he is free to select the mapping function and the technique to handle the secondary records of the file.

When the randomizing technique is chosen, the maintenance of the file can be performed by the following strategies.

1. When creating or adding records to the file, the user may specify the location for a record by supplying the track address, or he may supply just a track address and let the operating system find a location for the record. That is, the user supplies the key name or the full name of the record, and the operating system converts the key name of the record or its full name into track address, and finds the location for the record. If ther is room on the specified track, the operating system writes the record,

and updates the capacity record for files constructed with capacity records. If the specified track is full, the operating system continues searching for first empty space on successively higher tracks until a first emply space location is found. This search continues for as many tracks as the user has specified, to a maximum of the entire file. If a maximum search is specified and the end of the file is reached, the search for the first empty space will return to the beginning and continue until a location is found or until the original track is reached. In some operating systems such as BOS or Disk Operating System (DOS) of IBM, during the adding operation, if the specified track is full, the user must supply another track address.

2. When reading or updating the file, the user must supply a key for the desired record. The operating system converts it to a proper track address. The operating system searches for that key and, in case the search is satisfied, reads or writes back the corresponding Data Area. If a key is not found, the operating system so indicates to the user. A search by key is provided with two options, a restricted or an extended one. On a <u>restricted search</u>, only the track specified by the user is searched. On an

extended search, the operating system continues searching on successively higher tracks for as many tracks as the user has specified. Actually the operating system and Disk Control System will continue searching to the end of the cylinder.

Furthermore, with indirect addressing, the logic of creating, maintaining, and processing the file depends mainly on the technique that the user has selected to handle the overflow records in the file area.

Evaluation of Accessing Characteristics of the Direct Disk File

The parameters of the Direct Disk File have to be measured so that this type of file can be compared with others. For the CDC-3300 with OS-3, the simulator provides the following strategy to be performed for the simulation and computation of the Direct Disk File.

1. For all cases of simulation with the Direct Disk File, records in the file are assumed to be fixed and blocked, formatted with keys, and randomized to track address. Overflow records are placed in unused locations in the main area only.
2. In measuring the efficiency of mapping functions (hash coding) the alphabetic key name system is more desirable.

In fact the alphabetic key name, the numeric key name, and the alphanumeric key name can be interpreted equally well by the computer, under the same mapping function. However, it is simpler and more convenient to select the alphabetic key name (the full name of the record) for use in the simulation.

3. In evaluating the efficiency of the selected hash functions, the writer has performed experiments using only selected names of people of the United States. Results of these experiments appear in Example 4, Appendix B.

4. A random record is accessed from the direct disk file system. All data records are stored on disk memory (D854) and distributed according to the nature of the selected hash function. The user has to supply the full name of the desired record to the system. By user's editing, the operating system will call for the selected hash coding routine to convert the record full name into the hash address. The operating system will use this hash address for accessing the desired record in direct disk file. The procedure of accessing a record from the file system is dependent on the method selected to handle redundant records. In this evaluation three well known methods, linear probing, random probing, and direct chain probing

are to be considered. See details in Example 4, Appendix B.

Purpose of the Evaluation

The purpose of the evaluation is to determine the following parameters of the Direct Disk File, and its characteristics.

1. To evaluate the efficiency of the selected hash function with three methods of handling overflow records: linear probing, random probing, and direct chain probing, in terms of the average number of search length per record retrieval, based on testing programs (expected number of searches or compares until the desired record is found for accessing a random record from File).
2. To compute the access time when the record is located on the disk memory, based on the selected hash coding technique.
3. To measure the storage space requirement.
4. To find the characteristics of "average throughput/time per record retrieval" and "achievable throughput-rate capability".
5. To find the "cost/effectivemess" characteristics (user or customer operating cost per call, unit cost).

6. To compare its characteristics with other methods of file organization.

Simulating Block Diagram Model

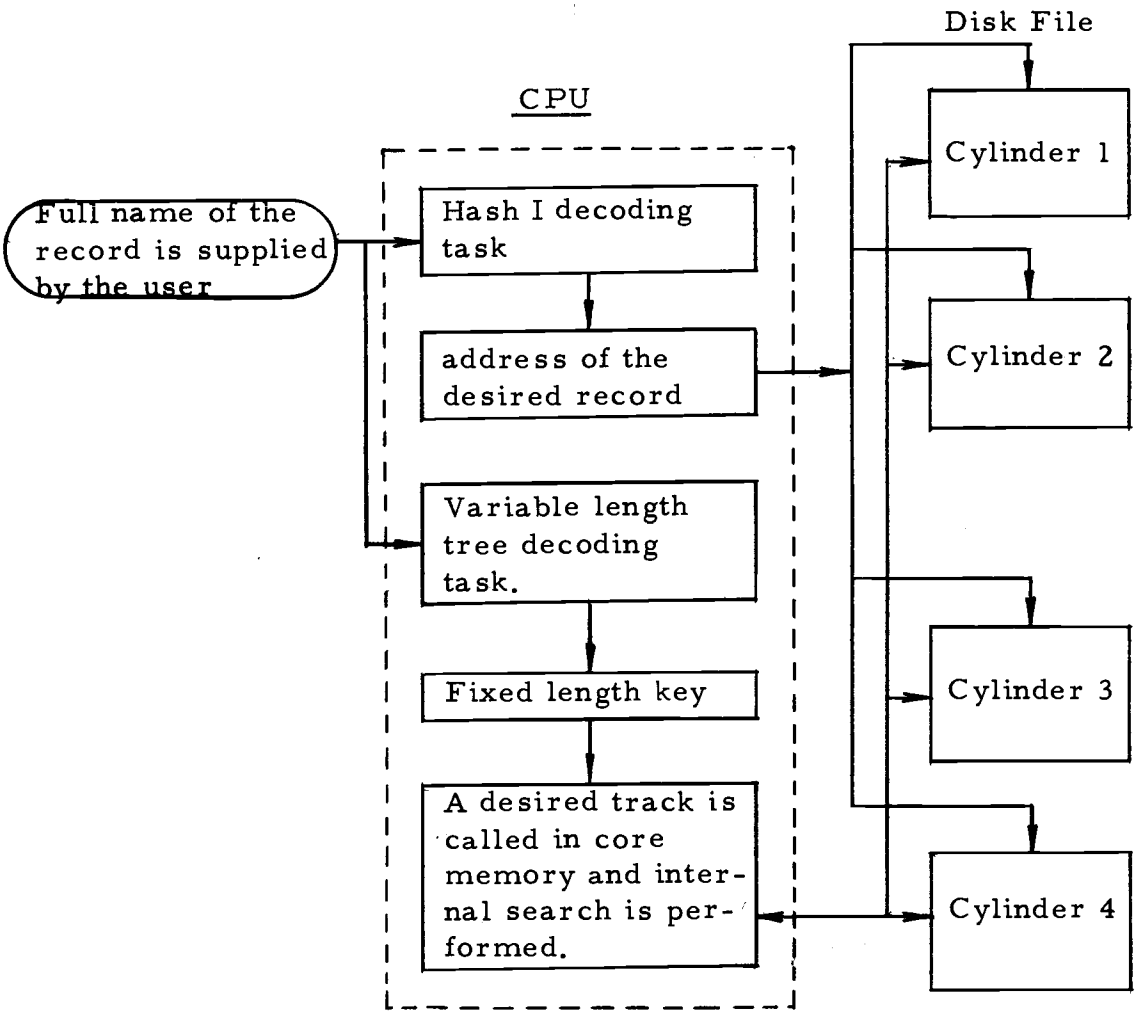


Figure 6.14. Block diagram showing the simulation of accessing of a random record from Direct Disk File.

The simulation of accessing a random record from a Direct Disk File can be illustrated as in Figure 6.14. Each logical record in the file is fixed, blocked, and associated with a key and a pointer.

The preliminary evaluation of five hash functions is determined by internal simulation: Hash 1-5, with three methods of handling redundant records. The results of the evaluation are shown in Figures 9.12-9.15. Hash 1, which possesses the best search characteristics has been selected to demonstrate the results of direct disk file accessing. The direct file is recorded on D854, disk memory, one read/write head per disk surface by using the cylinder concept. See Figure 2.10, page 15. The records can be stored in contiguous areas track by track, cylinder by cylinder, not in a key sequence, but distributed according to the nature of mapping function (hash function). For accessing the desired record from the direct file, the full name of the record is supplied to the system. The tree with variable-length key directory decoding routine is called, and converts the full name of the desired record into one word, a unique fixed-length key. The operating system can use this fixed-length key as search argument. Since each record of the direct disk file needs extra space for a pointer there are 63 logical records for each track of this file. From the results of evaluating the five selected hash functions, it is more efficient to use Hash 1 as the selected mapping function associate with linear, random, or direct chain probing one at a time to compare the results with other file accessing methods. By controlling of operating system one track (desired track) of direct disk file is called in and the internal search by linear probing,

random probing, or direct chain probing is performed. If the desired record is found; the operating system will present it to the user. In case desired record is not found, the operating system will notify the user that it is not there. See details in Example 4, Appendix B.

Results of the Evaluation of the Direct Disk File

Results of the evaluation of the direct disk file and comparison with the other methods of file organization are shown in Chapter VII.

VII. RESULTS AND CONCLUSIONS

Summary of Investigation and Evaluation

One objective of this thesis is to present a possible method which the writer expects to be useful for people in data-processing or digital-information-systems. This technique reduces time and effort in the calculation of meaningful numerical values to be used as references in determining the economic performance. To meet this objective, the following technical terms, average throughput time per record retrieval, achievable throughput-rate-capability, and operating cost per performance, are defined and evaluated. A certain system-operating assumption is also defined. In addition, technical terms, their definitions, and equations in which they appear are shown in Appendix B. The extensive use of curves and tables in this chapter is intended to help the system designer and the evaluator to better understand the trade-offs available. The reader has to keep in mind that in all cases of the investigation the system is assumed to be an on-line only operation and no-error with fixed-length message arriving at a uniform input rate.

For the evaluation of the performance and operating cost of an on-line data file system, the schematic diagram of investigation and evaluation is presented in Figure 7.1, page 142. The following data

file characteristics have to be evaluated and compared:

1. Average throughput time per record retrieval as the function of file loading factor, for each typical file. The description and results of the investigation are discussed on pages 146 - 151.
2. Achievable throughput-rate-capability as the function of file loading factor for each typical file. This is computed by using general formular equation (7. 1), page 141 . This description and results are discussed on pages 152 - 161.
3. File operating cost per call (unit cost) as the function of file loading factors for each typical file at selected rates of call (calls per hour), to help the evaluator visualize better the trade-offs available. Details of the description and results of the evaluation are covered on pages 162 - 175.
4. Two common methods of internal search, linear search and binary search, as the preliminary work for data file internal search evaluation and compare. See details and results of evaluation on pages 193 - 197.
5. Five common hash methods selected for investigation of the values of average search-length per record retrieval (expected number of searched per random record retrieval) as the function of file loading factor, as the preliminary work for selecting the best method to be used in evaluating

direct disk file organization. From the results of evaluation and comparison, Hash I is the best hash function, and is selected for the evaluation of the accessing characteristics of direct disk file organization. See details and discussion on Example 4, Appendix B, pages 294 - 311.

In making a decision as to which method of data file organization is the most suitable one, the following criteria are to be considered:

1. Among techniques (types of files) with equal cost, total cost of the system, or in some cases only operating cost, is to be considered, depends on the objective of the problem. The technique with the greatest effectiveness (the one providing the maximum achievable throughput-rate capability) is best.
2. Among techniques (types of files) with equal effectiveness, the one with the least cost is best.

Thus it is reasonable to evaluate and compare the average throughput time per record retrieval, achievable throughput-rate-capability (the maximum rate of use of each typical file), and file operating cost per call as the function of the file loading factor. In fact, when a decision is made on a set of data file organization methods, file-system parameters have to be specified:

a) File loading factor or file size.

b) Rate of use of file (calls per hour).

The algorithims of making decisions are:

1. Checking which type of file can be operated at the specified rate of use, by using the curve or the table of achievable-rate-capability, to find a set of file organization methods (at least one) which satisfies the specified rate of use.
2. Checking with type of file from a set of files obtained from 1. can be operated at lowest cost, by using the curves and the table of file operating cost per call, unit cost. If there exists more than one type of file giving the lowest operation cost per call, go to 3.
3. Checking again a set of files obtained from 2 by using the achievable-rate-capability curves of Table 7.3 on page and selecting the one providing the highest achievable-throughput-rate-capability as the typical file for decision making, to preserve the capability of increasing rate of use (call per hour) of the file system.

Throughput Time Per Record Retrieval is the response time of the data file system starting from the entry of the last character of the full name of a desired record or from the unique fixed length key of desired record, and the receipt of the first character of a reply. The throughput time of an automated savings account system

is illustrated in Figure 7. 3; t2-t3 is a throughput time per record retrieval. In practice the throughput time varies depending on the size of file and the method of accessing. Then to compute the throughput time of a random record retrieval the average throughput time per record retrieval is considered.

Achievable through-put-rate capability must be considered in developing criteria for evaluating cost, and performance in a specific data file system. It is the maximum through-put-rate at which the system can meet such an applicable specification as response time. A system, meeting all specifications at the acheivable through-put-rate should be considered to have achievable through-put-rate capability. It may be viewed to measure the expansion capability of the system, and at times to indicate that the desired system may be more powerful than required. The general form of the relationship between achievable through-put-rate-capability and average through-put time is:

Achievable throughput-rate-capability, (calls per hour)

$$= 3600 \div \text{average throughput time per record retrieval, (sec)} \qquad (7.1)$$

Results of the Investigation

The writer has examined and presented the accessing of a desired record by each of the four methods of file organization for an

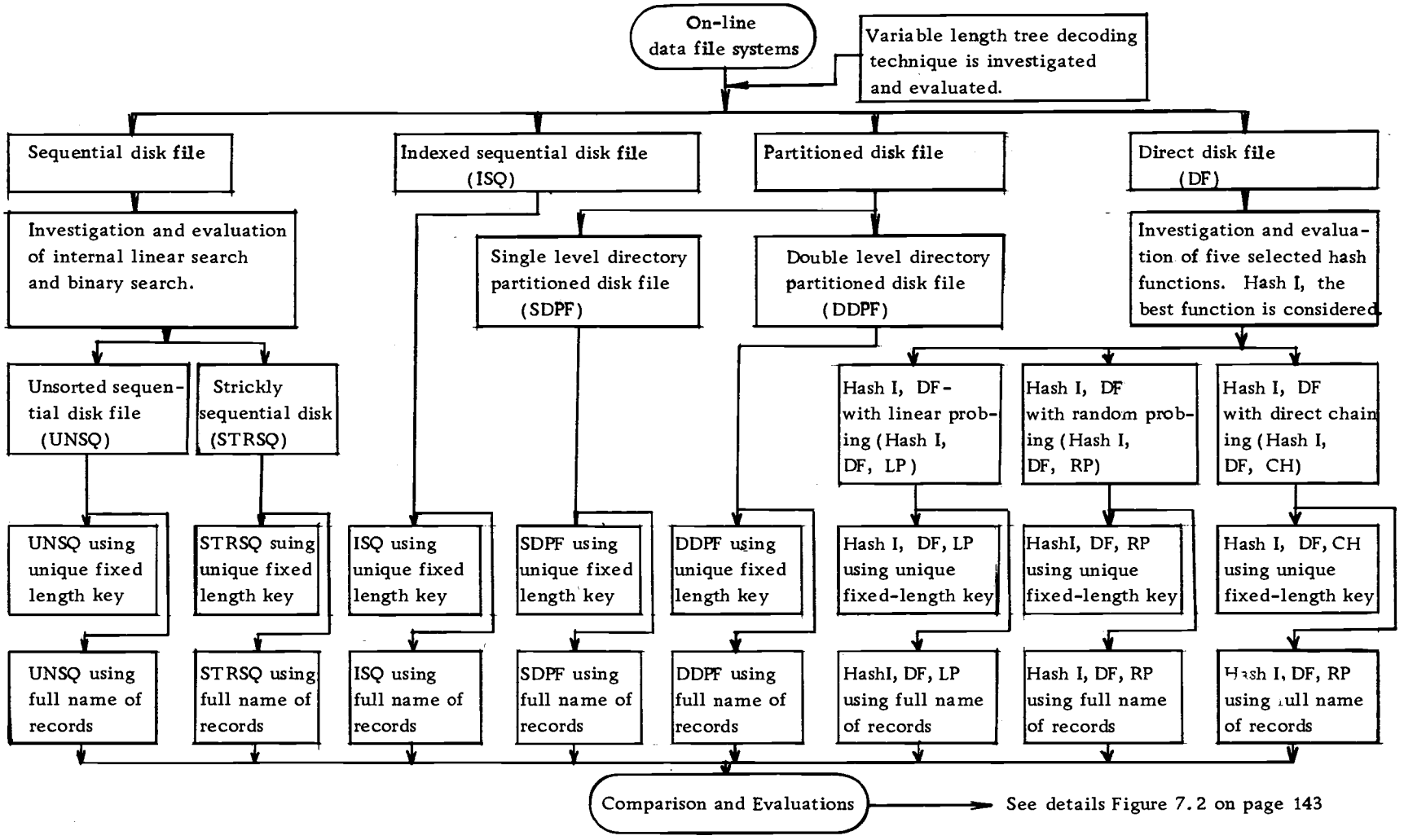


Figure 7.1. Schematics diagram of investigation and evaluation of on-line data file systems.

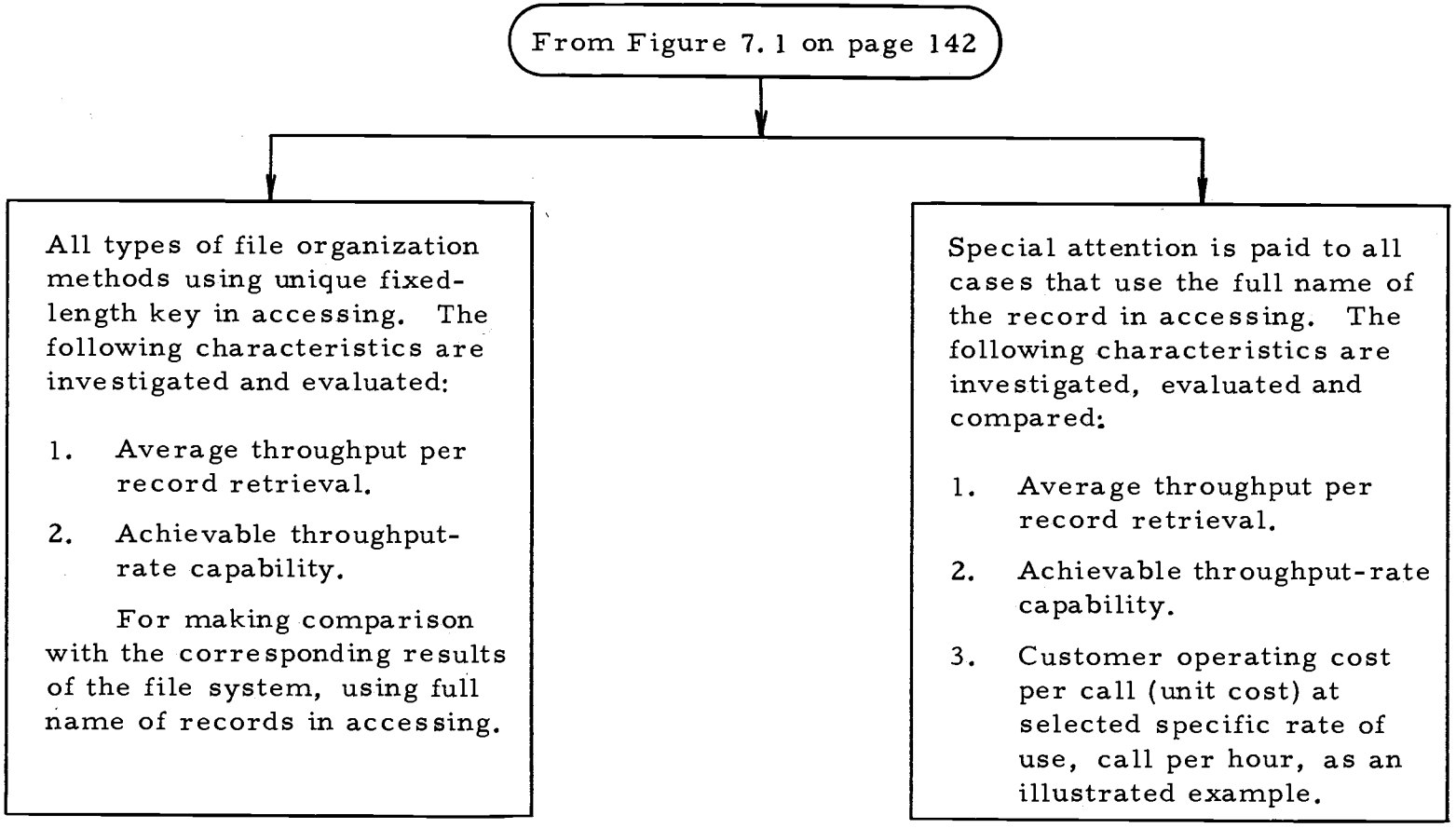


Figure 7.2. Investigation and evaluation of characteristics of on-line data file system.

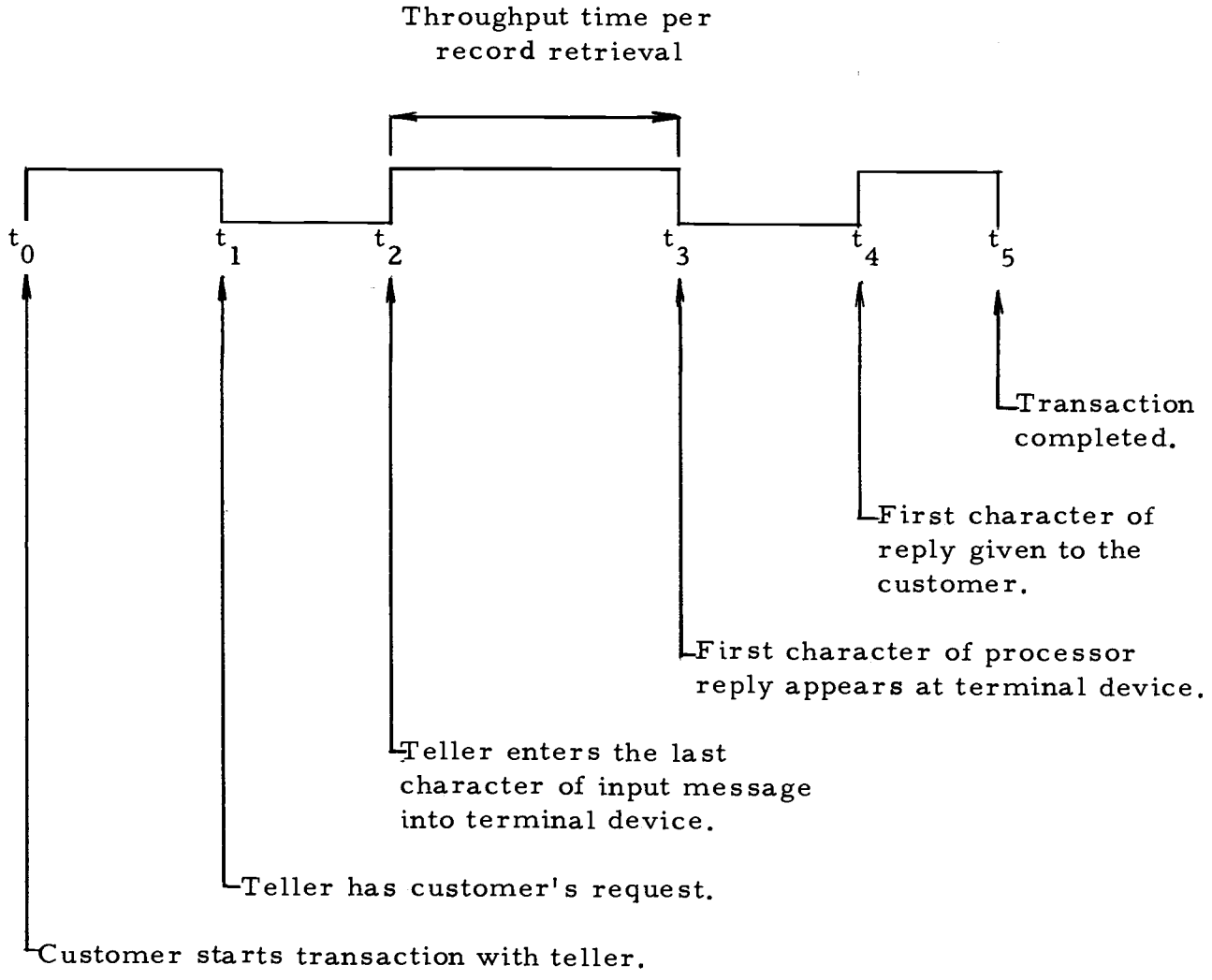


Figure 7.3. Throughput time per record retrieval.

on-line information system, supported by disk. The parameters of the technical system are based on the CDC-3300 computer system at Oregon State University Computer Center. The method of file organization for an on-line information system has been shown to have superior to average throughput time per record retrieval, and also operating cost per call (unit cost). The comparative results from simulations and computations are shown in the following graphs and tables.

RESULTS OF THE EVALUATION AVERAGE THROUGH-PUT TIME PER RECORD RETRIEVAL FOR EACH TYPICAL FILE

Table 7.1. Data results of computation of average throughput time per record retrieval as the function of file loading factors, for each typical file organization method, using the full name of records in accessing.

File loading factor, α	0.0156	0.0372	0.0625	0.2500	0.5000 half	0.7500	1.0000 full
Number of records in file	128 records	512 records	1024 records	4096 records	8192 records	12288 records	16384 records
Typical files	← Average throughput time per recprd retrieval as the function of file loading factors →						
	ms	ms	ms	ms	ms	ms	ms
Unsorted sequential file	227.399	377.479	577.519	1905.599	3632.605	5360.130	7087.645
Strickly sequential file	232.371	397.510	617.582	2065.351	4050.113	5937.890	7922.660
Indexed sequential file	232.286	232.374	232.414	232.498	232.551	232.608	232.656
Single level directory partitioned file	182.358	182.473	182.513	182.663	182.705	182.765	182.780
Double level directory partitioned file	182.360	182.476	182.516	182.631	182.672	182.732	182.748
Direct file with linear probing	177.470	177.550	177.590	177.671	178.573	179.714	183.519
Direct file with random probing	177.484	177.565	177.605	178.929	180.248	183.606	191.746
Direct file with direct chain probing	177.468	177.549	177.589	177.769	178.572	179.708	183.420

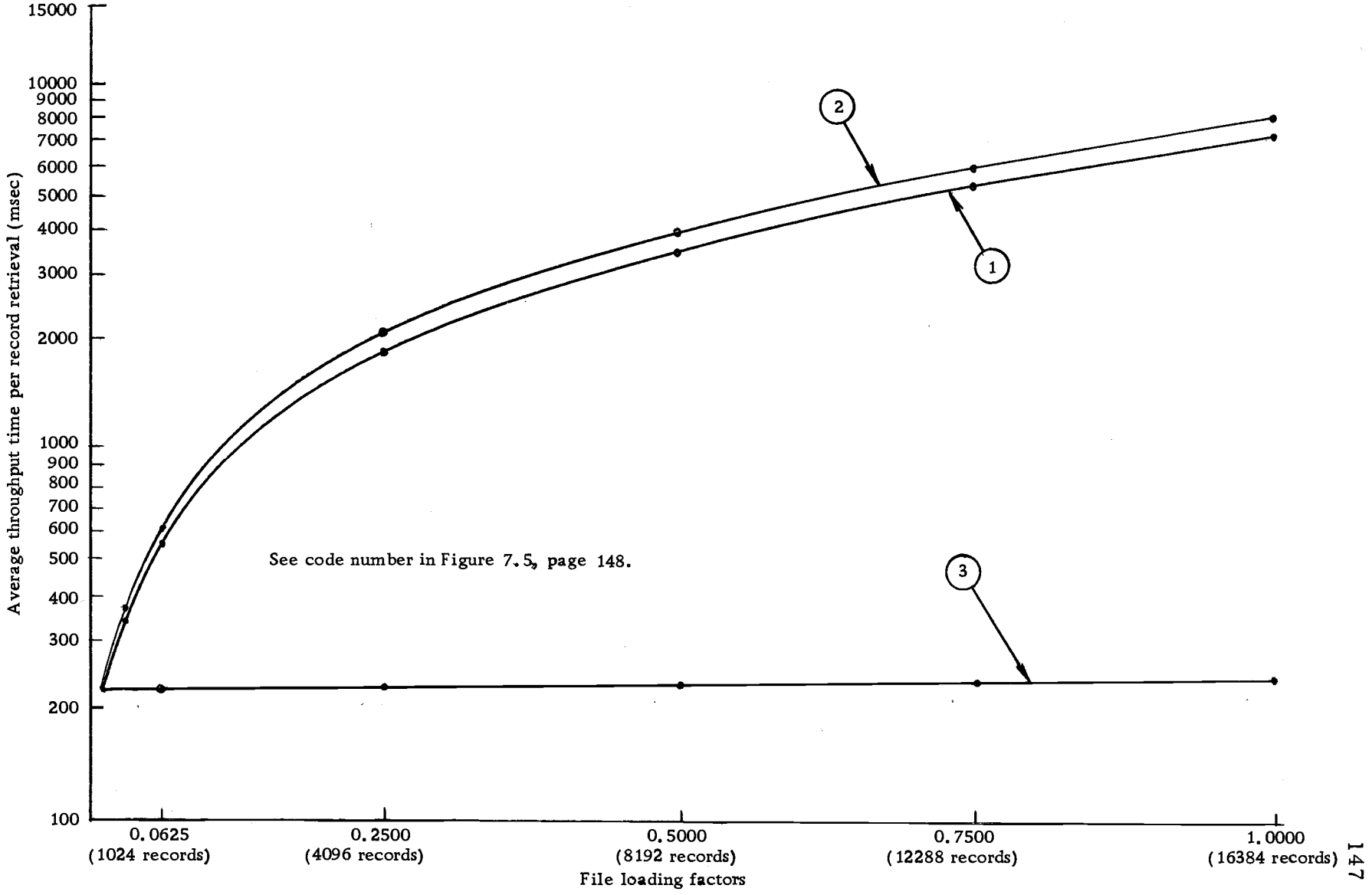


Figure 7.4. Average throughput time per record retrival as the function of file loading factors for each typical file, code numbers 1 to 3, using full name of records in accessing.

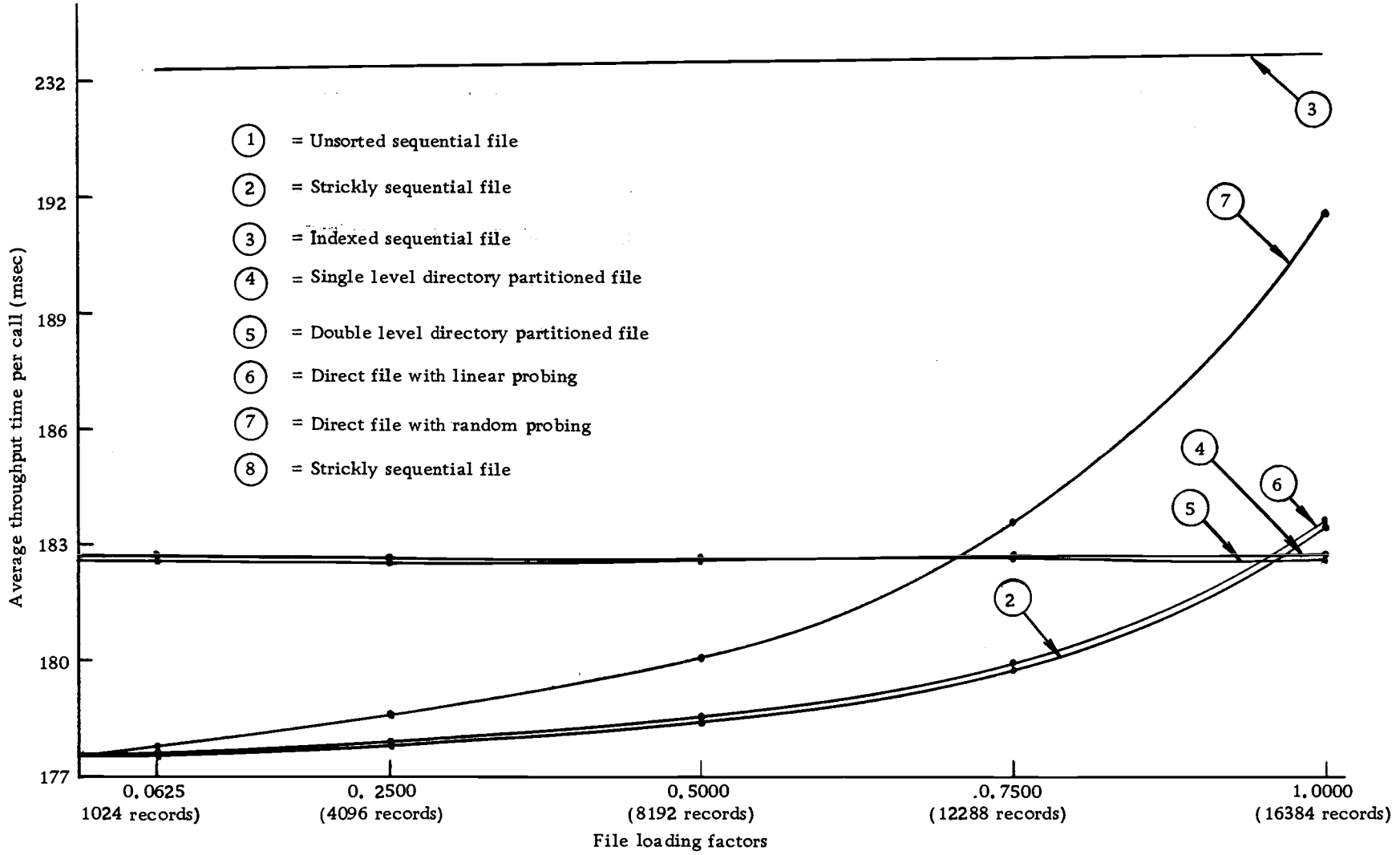


Figure 7.5. Average throughput time as the function of file loading factors for each typical file organization method using the full name of records in accessing.

Table 7.2. Data results of computation of average throughput time per record retrieval as the function of file loading factors for each typical file organization method using unique fixed-length key in accessing.

File loading factor,	0.0156	0.0372	0.0625	0.2500	0.5000	0.7500	1.0000
Number of records in file	128 records	512 records	1024 records	4096 records	8192 records	12288 records	16384 records
Typical files	← Average throughput time per record retrieval as the function of file loading factors →						
	ms	ms	ms	ms	ms	ms	ms
Unsorted sequential file	219.004	369.004	569.004	1897.004	3624.004	5351.504	7079.004
Strickly sequential file	223.976	389.035	609.067	2056.756	4041.512	5929.289	7914.019
Indexed sequential file	223.891	223.899	223.899	223.903	223.950	223.982	224.015
Single level directory partitioned file	173.963	173.998	173.998	174.068	174.104	174.139	174.139
Double level directory partitioned file	173.965	174.001	174.001	174.036	174.071	174.131	174.107
Direct file with linear probing	169.075	169.075	169.075	169.076	169.972	171.113	174.878
Direct file with random probing	169.089	169.090	169.090	170.334	171.647	175.005	183.105
Direct file with direct chain probing	169.073	169.074	169.074	169.174	169.971	171.107	174.779

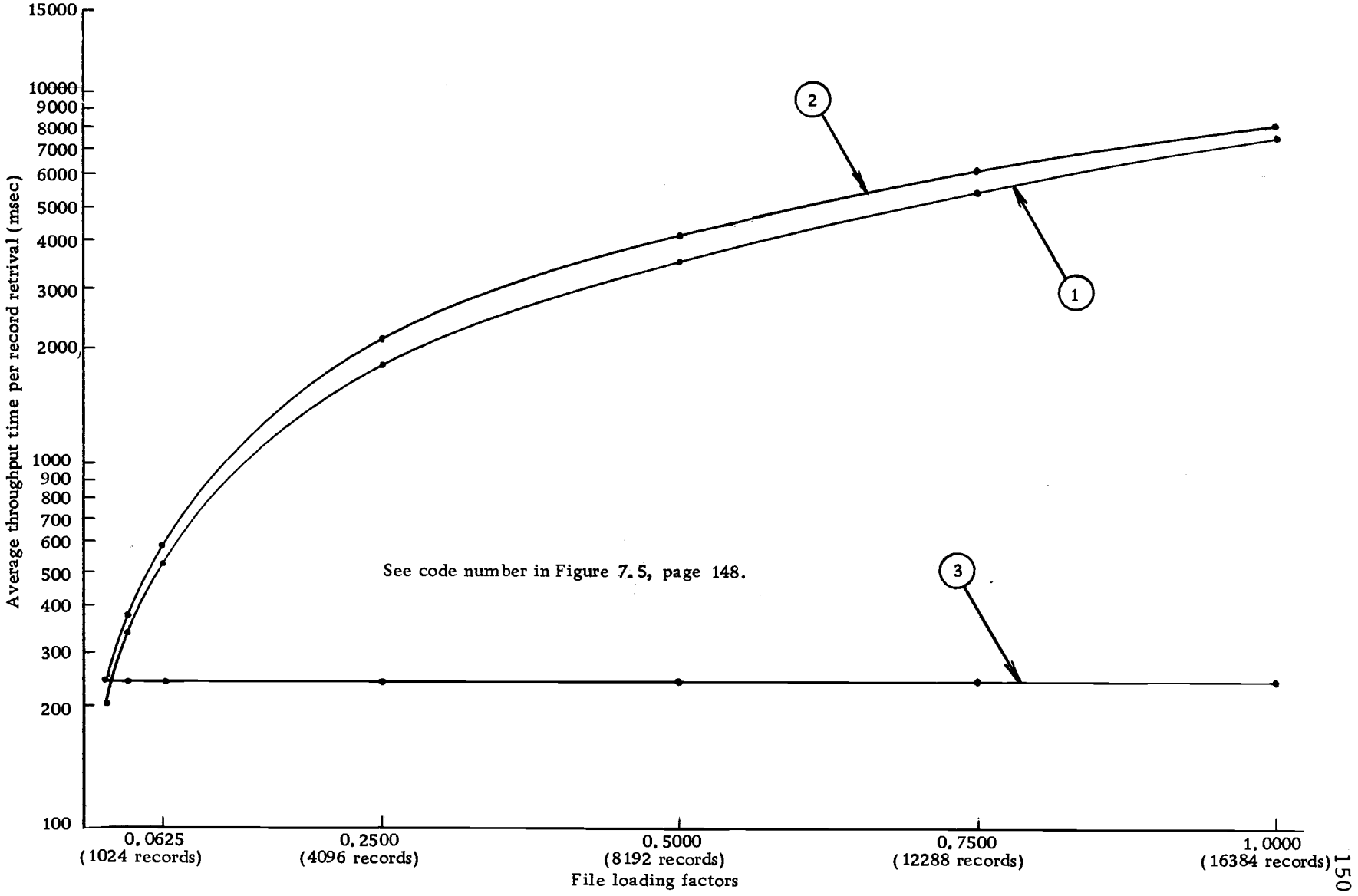


Figure 7.6. Average throughput time per record retrieval as the function of loading factors for each typical file code numbers 1 to 3 using unique fixed length key in accessing.

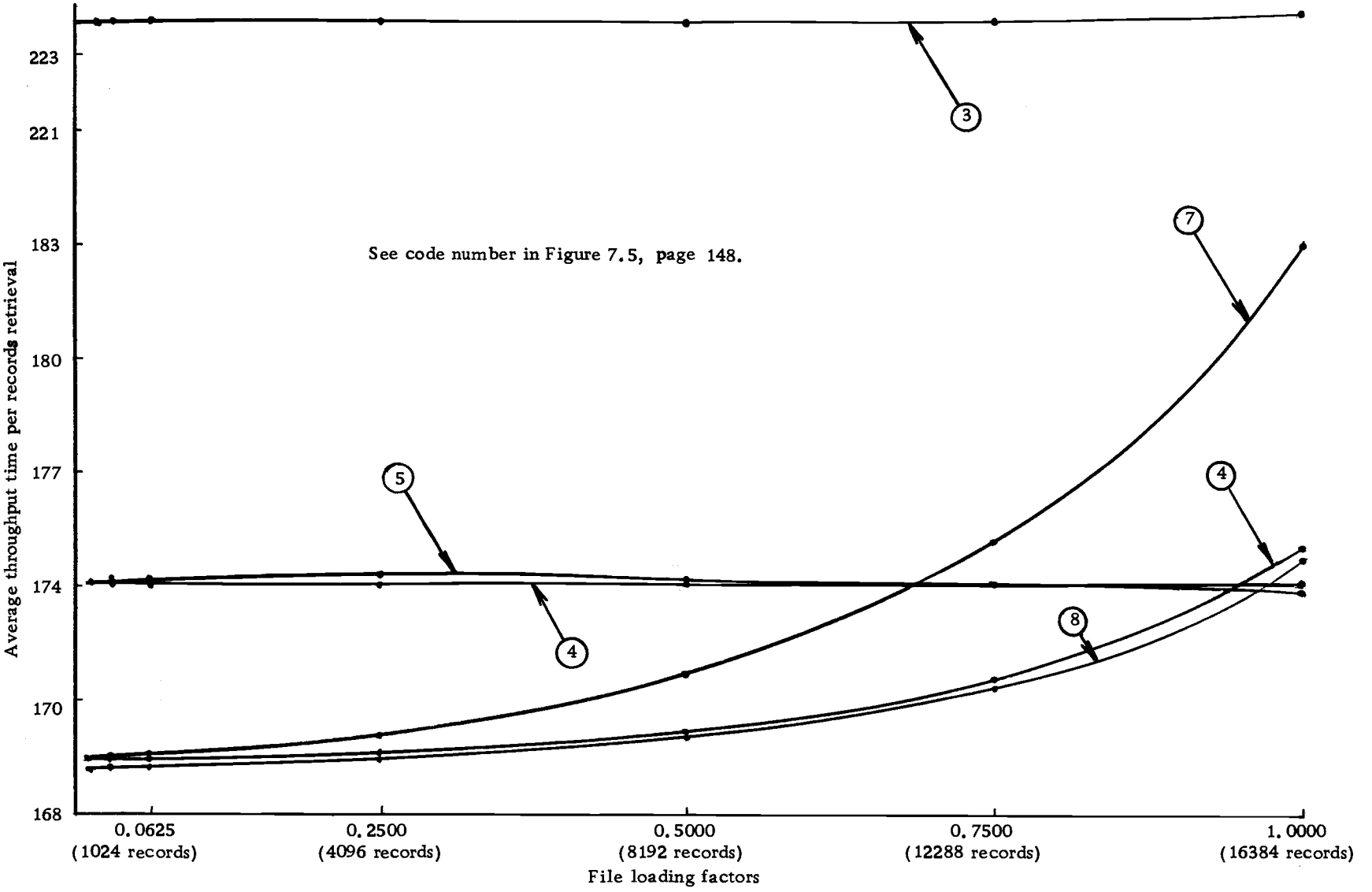


Figure 7.7. Average throughput time as the function of file loading factors for each typical file organization method, code numbers 3 to 8, using unique fixed-length key in accessing.

RESULTS OF THE EVALUATION

ACHIEVABLE THROUGHPUT-RATE CAPABILITY

FOR EACH TYPICAL FILE

Table 7.3. Data results of computation of achievable throughput-rate capability as the function of file loading factors for each typical file organization method using the record's full name in accessing.

File loading factor, α	0.0156	0.0372	0.0625	0.2500	0.5000	0.7500	1.0000
Number of records in file	128 records	512 records	1024 records	4096 records	8192 records	12288 records	16384 records
Typical files	← Throughput-rate capability as the function of file loading factors →						
	calls/hr.	calls/hr.	calls/hr.	calls/hr.	calls/hr.	calls/hr.	calls/hr.
Unsorted sequential disk file	15831	9537	6234	1889	991	672	508
Strickly sequential disk file	15493	9056	5829	1743	889	606	453
Indexed sequential disk file	15498	15492	15490	15484	15481	15477	15472
Single level directory partitioned disk file	19741	19729	19725	19708	19704	19697	19696
Double level directory partitioned disk file	19741	19729	19724	19712	19708	19701	19699
Direct desk file with linear probing	20285	20276	20271	20262	20160	20032	19617
Direct disk file with random probing	20284	20274	20270	20120	19973	19607	18232
Direct disk file with direct chain probing	20285	20276	20272	20251	20160	20033	19627

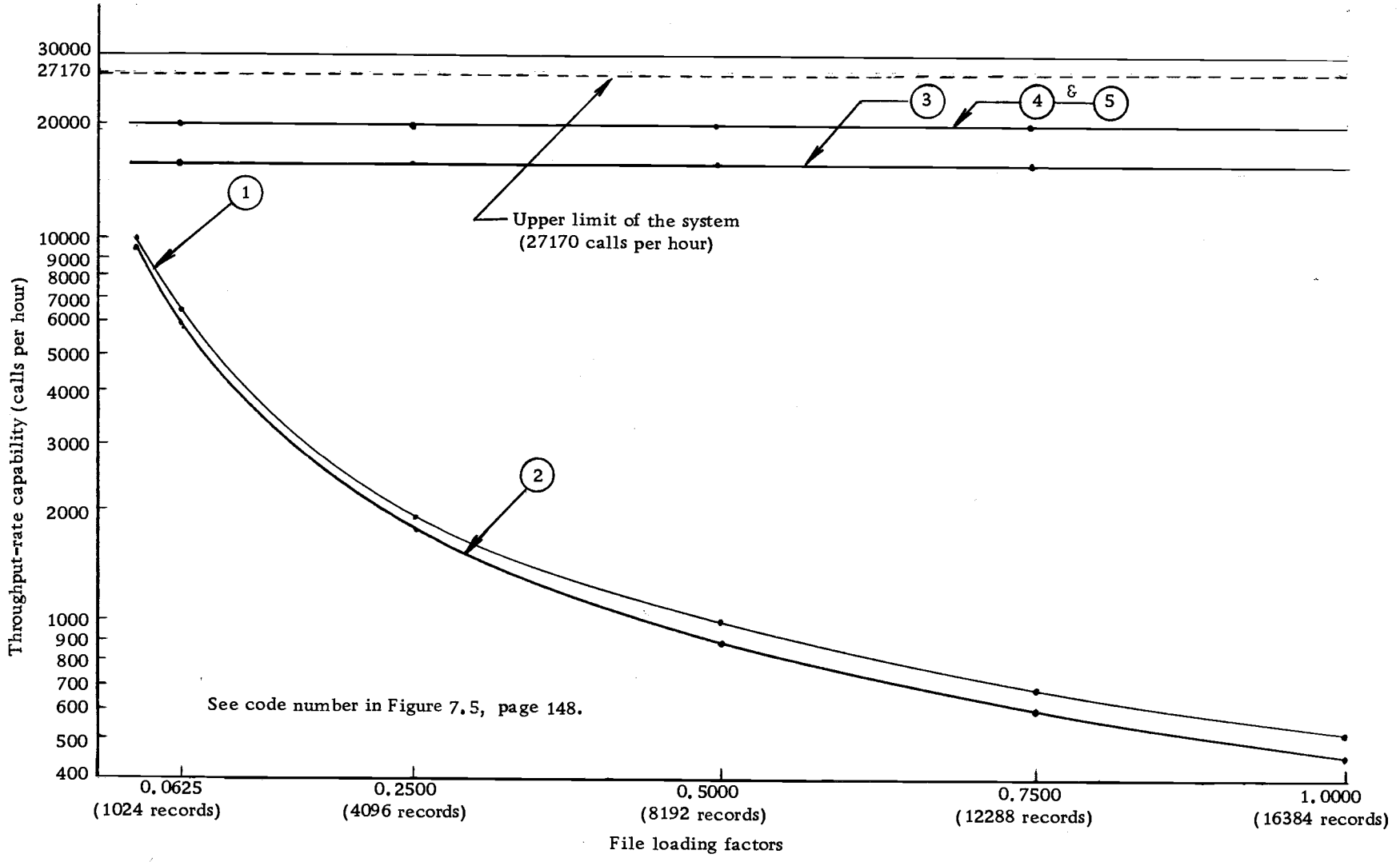


Figure 7.8. Achievable throughput-rate capability as the function of file loading factors for each typical file, code numbers 1 to 5, using full name of records in accessing.

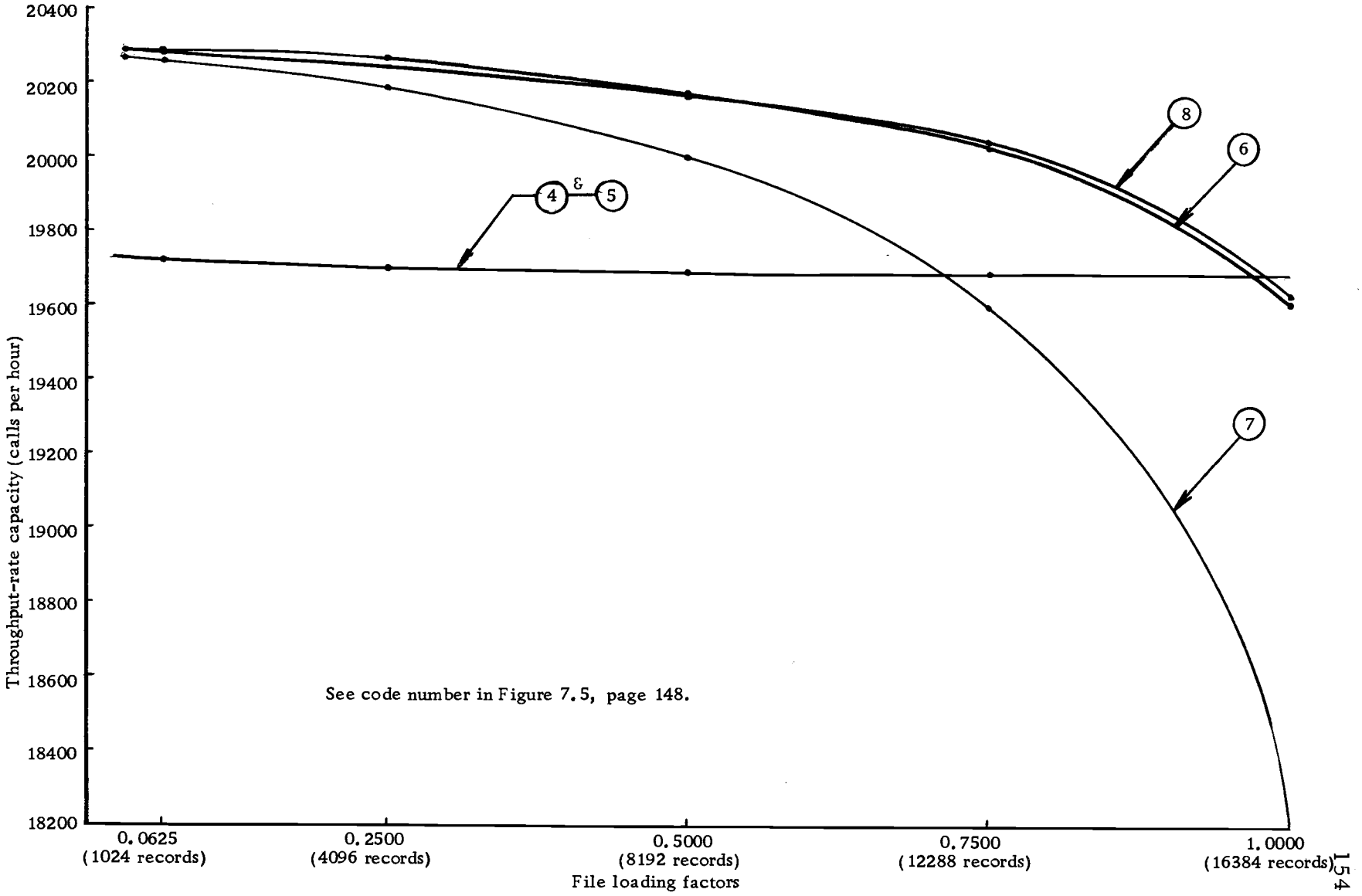


Figure 7.9. Achievable throughput-rate capability of file loading factors for each typical file organization method, code numbers 4 to 8, using full name of records in accessing.

Table 7.4. Data results of computation of achievable throughput-rate capability as the function of file loading factors of each typical file organization method, using unique fixed-length key in accessing.

File loading factors	0.0156	0.0372	0.0625	0.2500	0.5000	0.7500	1.0000
Number of records in file	128 records	512 records	1024 records	4096 records	8192 records	12288 records	16384 records
Typical files	← Throughput-rate capability as the function of file loading factors →						
	calls/hr.	calls/hr.	calls/hr.	calls/hr.	calls/hr.	calls/hr.	calls/hr.
Unsorted sequential disk file	16438	9756	6327	1898	993	673	509
Strickly sequential disk file	16073	9254	5911	1750	89'	607	455
Indexed sequential disk file	16079	16079	16079	16078	16075	16073	16070
Single level directory partitioned disk file	20694	20690	20690	20682	20677	20673	20673
Double level directory partitioned disk file	20694	20690	20690	20685	20681	20674	20677
Direct disk file with linear probing	21292	21292	21292	21292	21180	21039	20586
Direct disk file with random disk file	2129	21290	21290	21135	20973	20571	19661
Direct disk file with direct chain probing	21293	21292	21292	21280	21180	21039	20597

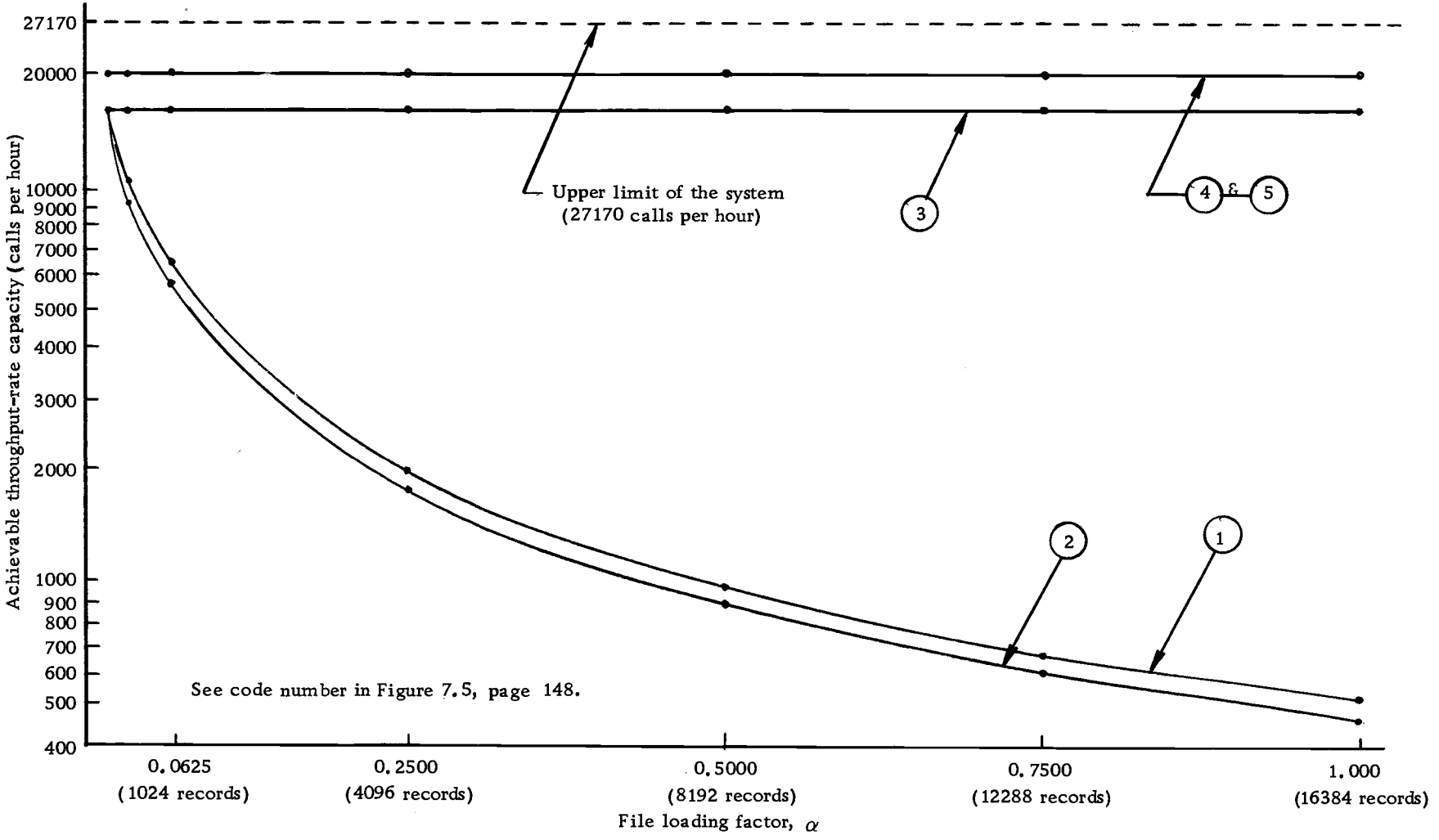


Figure 7.10. Achievable throughput-rate capacity as the function of file loading factors for each typical file, code numbers 1 to 3, using unique fixed-length key in accessing.

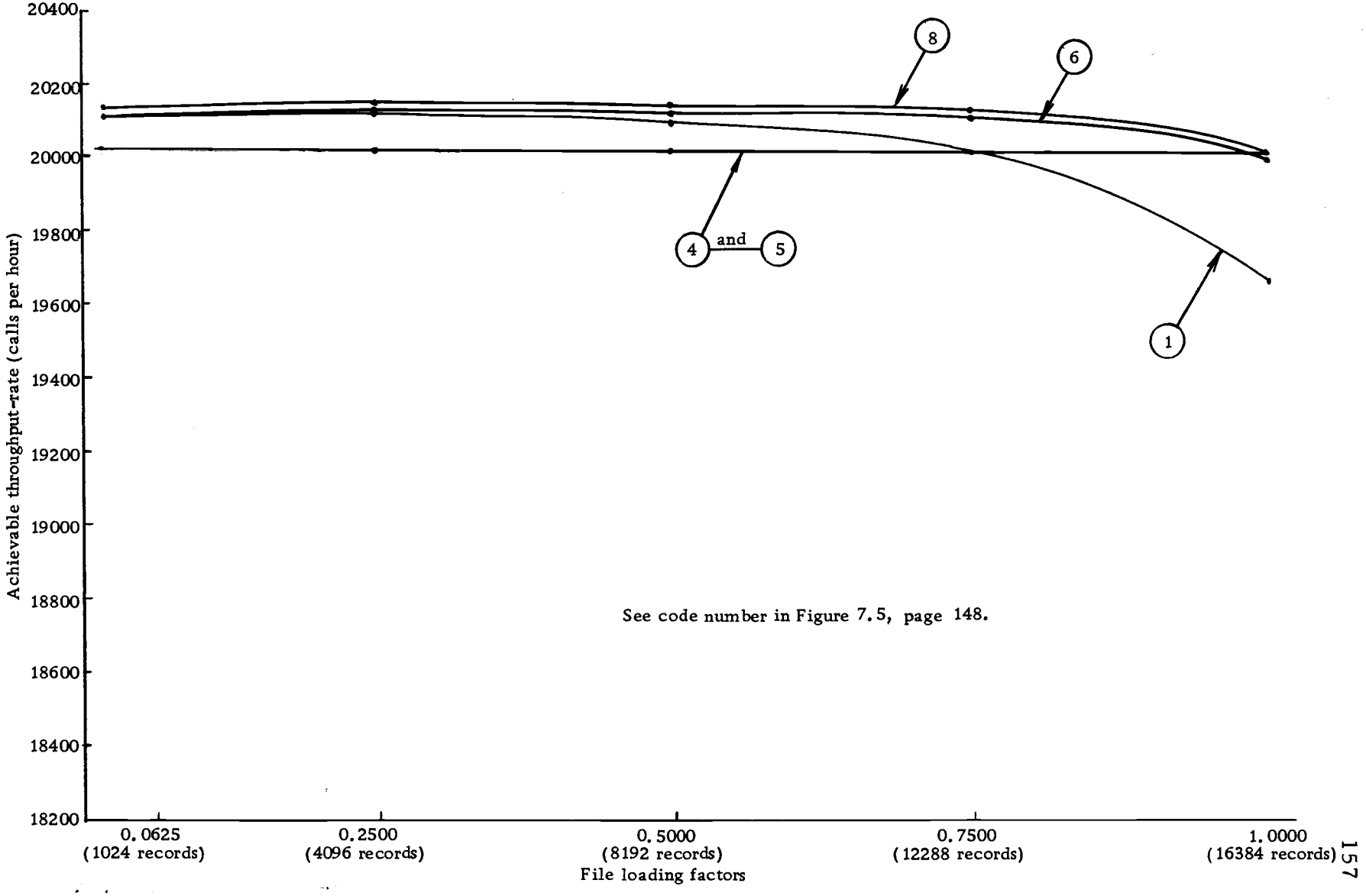


Figure 7.11. Achievable throughput-rate capacility of file loading factors for each typical file organization method, code numbers 4 to 8, using fixed length key in accessing.

Table 7.5. Approximate formulas of average throughput time and achievable throughput-rate capability for each typical file.

Typical file	Approximate average throughput time	Approximate achievable throughput-rate capability	Description
Unit	m sec.	calls/hr.	
Unsorted sequential file	$T_1 \simeq 0.635\ N + 97.453$	$C_1 \simeq \frac{3600\ (Sec)}{T_1\ (Sec)}$	for $0 \leqslant \alpha \leqslant 1.00$
Strickly sequential file	$T_2 \simeq 0.635\ N + 102.451$	$C_2 \simeq \frac{3600\ (Sec)}{T_2\ (Sec)}$	for $0 \leqslant \alpha \leqslant 1.00$
Indexed sequential file	$T_3 \simeq 232.3878$	$C_3 \simeq \frac{3600\ (Sec)}{T_3\ (Sec)}$	for $0 \leqslant \alpha \leqslant 1.00$
Single level partitioned disk	$T_4 \simeq 35.00\ (\log_2 N) + 182.4268$	$C_4 \simeq \frac{3600\ (Sec)}{T_4\ (Sec)}$	for $0 \leqslant \alpha \leqslant 1.00$
Double level partitioned disk file	$T_5 \simeq 35.00\ (\log_2 \frac{N}{4}) + 182.4268$	$C_5 \simeq \frac{3600\ (Sec)}{T_5\ (Sec)}$	for $0 \leqslant \alpha \leqslant 1.00$
Direct disk file with linear probing	$T_6 \simeq 177.545$ $T_6 \simeq 177.545 + 0.007875\ E_{LP} + 0.5\ E_1 + 0.675\ E_2$	$C_6 \simeq \frac{3600\ (Sec)}{T_6\ (Sec)}$	for $0 \leqslant \alpha \leqslant 0.25$ for $0.25 \leqslant \alpha \leqslant 1.00$
Direct disk file with random probing	$T_7 \simeq 177.604$ $T_7 \simeq 177.604 + 0.0346\ E_{RP} + 0.5\ E_3 + 0.675\ E_4$	$C_7 \simeq \frac{3600\ (Sec)}{T_7\ (Sec)}$	for $0 \leqslant \alpha \leqslant 0.25$ for $0.25 \leqslant \alpha \leqslant 1.00$
Direct disk file with direct chain probing	$T_8 \simeq 177.573$ $T_8 \simeq 177.573 + 0.019875\ E_{DCH} + 0.5\ E_5 + 0.675\ E_6$	$C_8 \simeq \frac{3600\ (Sec)}{T_8\ (Sec)}$	for $0 \leqslant \alpha \leqslant 0.5$ for $0.5 \leqslant \alpha \leqslant 1.00$

From Table 7.3, the following deductions can be made:

1. Both T_1 and T_2 increase excessively when number of records in a file or file loading factor is increased. T_2 increases most of all, so that

   $T_2 > T_1 > T_3, T_4, T_5, T_6, T_7$ and T_8 for $0 < \alpha < 1.0000$ as shown in curve in Figures 7.3 and 7.4 pages 144 and 147.

   $\Rightarrow C_2 < C_1 < C_3, C_4, C_5, C_6, C_7$ and C_8 for $0 < \alpha < 1.0000$.

2. T_3, from the approximate formula, is almost a constant function. Its value is not increased much when the file loading factor is increased, as in Figure 7.4.

   T_3 - 232 ms/call

   $T_2 > T_1 > T_3 > T_4, T_5, T_6, T_7$ and T_8 for $0 < \alpha < 1.0000$ see Figure 7.5 and 7.6 pages 148 and 150.

   $\Rightarrow C_2 < C_1 < C_3 < C_4, C_5, C_6, C_7$ and C_8 for $0 < \alpha < 1.0000$

3. T_4 and T_5, from the approximate formula, are almost constant functions. But when the number of records in a file, N, is increased,

   $T_4 \alpha g \times 10^{-3} (\log_2 N)$ ms, to small

   and $T_5 \alpha 0.25\ g \times 10^{-3} (\log_2 N)$ ms, to small

   g = average search time per record in second level directory, μsec. in Figure 7.6 page 150

$T_4 = T_5 = 182$ msec. $T_4 < T_5$ for $0 < \alpha < 0.25$,

$T_4 > T_5$ for $0.25 < \alpha < 1.00$

$T_2 > T_1 > T_3 > T_4 > T_5 > T_6$, T_7, and T_8 for α 1.0000

$\Longrightarrow C_2 < C_1 < C_3 < C_4 < C_5$, C_6, C_7, and C_8 for α 1.0000

4. T_6, T_7 and T_8, from the approximate formula, are:

$$T_6 \cong 177.4588 \left(7.875 \frac{1 - \alpha/2}{1 - \alpha}\right) + 165.625) \times 10^{-3} \text{ ms}$$

for $0 < \alpha < 0.50$

$$T_7 \cong 177.4588 (3.4 (-\alpha^{-1} \log_e (1 - \alpha)) + 137.375) \times 10^{-3}$$

ms for $0 < \alpha < 0.25$

$T_8 \cong 177.4588 (19.875 (\alpha/2) + 164.25)$ ms for $0 < \alpha <$

0.50

Then in the range of $0 < \alpha < 0.50$

$T_2 > T_1 > T_3 > T_4 > T_5 > T_7 > T_6 > T_8$

$\Longrightarrow C_2 < C_1 < C_3 < C_4 < C_5 < C_7 < C_6 < C_8$

5. Again T_6, T_7 and T_8 from the appropriate formula, are:

$T_6 \cong 177.4588 + 30 (E_1) + 50 (E_2)$ for $0.55 < \alpha <$ 1.000

$T_7 \cong 177.4588 + 30 (E_3) + 50 (E_4)$ for $0.25 < \alpha <$ 1.000

$$T_8 \cong 177.4588 + 30\,(E_5) + 50\,(E_6) \qquad \text{for } 0.5 < \alpha < 1.000$$

where E_1 = Average search across cylinder for linear probing.

E_2 = Average search across track for linear probing.

E_3 = Average search across cylinder for random probing.

E_4 = Average search across track for random probing.

E_5 = Average search across cylinder for direct chain probing.

E_6 = Average search across track for direct chain probing.

As the variation of E_1 to E_6 is empirical, the results from the simulation are used for evaluation in this thesis. Then in the range of $0.5 < \alpha < 1.000$

$$T_2 >> T_1 >> T_3 > T_4 > T_5 > T_7 > T_6 > T_8$$

$$\Longrightarrow C_2 < C_1 < C_3 < C_4 < C_5 < C_7 < C_6 < C_8$$

The above discussion is based on the file system with the full name of the record used in accessing. For the file system using the unique-fixed length key in accessing, the discussion is the same. The difference is the omission of the full name record to the fixed-length key conversion time, about 8.55 ms/record.

RESULTS OF THE EVALUATION

USER TOTAL CHARGE PER MONTH

FOR EACH TYPICAL FILE AT

A SELECTED SPECIFIC RATE OF CALLS

FOR ILLUSTRATION OF THE EVALUATION

File Operating Cost per Call (Unit Cost)

In general, when cost per performance of digital computing system is mentioned, there are two options to be considered. One is that the user plans to have his own computer system (buying or renting from manufacturer). In this case cost per performance has to be calculated based on: CPU cost, terminal devices cost, communication cost, and operator's cost. See details of an example in Stimler, Saul (33) page 149-160. Another is that many users are planning to rent only CPU time from a time-sharing computer system. In this case cost per performance per month has to be calculated based on only CPU, busy time, cost per month, storage space rental cost per month, terminal devices rental cost per month, communication line charge per month, and operator's cost per month. This investigation is concerned with only the second option. From now on, all the terms of cost per performance have to be replaced by customer operating cost per call, unit cost, for the evaluation of the on-line data file.

Let the reader consider the following situation: suppose that a designer or an evaluator works with one on-line time sharing computer system. He is assigned to investigate and evaluate the performance of typical file organization methods with the specified maximum capacity (the number of records when the file is full) and

current capacity (the current number of records in the file), with a certain rate of use (number of calls per hour) of the file. He must help the new customer (the user) to make a decision as to which type of file is the most suitable for him at a certain capacity and rate of use of the data file system. In this case an evaluator needs to know only which type of file organization method is supported by his computer system under a certain processing assumption, and which provides the most economical performance (lowest unit cost).

The following formula is introduced as a means for measuring the unit cost performance averaged over the life of the file project when the specified rate of use of the file (number of calls per hour) is processed over the active life of the file. Then

$$\text{customer operating cost/call (unit cost)}, \quad U_{c(i)}$$

$$= \frac{\text{project cost}}{\text{total specified calls}}$$

$$= \frac{M.\ T_{(i)}\ R_{cM}.\ 300 + N_{T(i)}\ (0.3)\ M}{R_{cM} M} \quad \text{for time sharing customer}$$

$$= \frac{T_{(i)}\ R_{cM}.\ (300) + N_{T(i)}\ (0.3)}{R_{cM}} \times 100 \quad \text{cent/call} \quad (7.2)$$

where

$T_{(i)}$ = CPU busy time per call in hour, i = file code numbers, 1, 2, 3, ..., 8.

R_{cM} = Rate of calls per month, for illustrated example,

52500 calls per month (250 calls/hr) for low-rate

210000 calls per month (1000 calls/hr) for medium rate

420000 calls per month (2000 calls/hr) for high rate.

These selected rates are based on real-world problems.

300 = CPU charge per hour in dollars

0.3 = Disk rental charge in dollar per track per month.

These figures are based on Oregon State Computer Center, Corvallis, rate of charge for time sharing customer.

$N_{T(i)}$ = Disk space required tracks for each type of file

M = File project active life time in term of months.

The data results of computation of the unit cost, $U_{c(i)}$ equation (7.2) as the function of file loading factor are tabulated, plotted and compared as shown in Table 7.6 page 173 and in Figures 7.13 to 7.16 pages 169 - 172. See the details of computation of each typical file in Appendix B, Examples 1 to 4. The terminal device rental-cost for each selected rate of use is illustrated on page 168.

Transmitted characters = full name of a record + data of a record

= 16 + 64 = 80 characters/call

Then

Transmitted characters for 250 calls per hour

= (80 x 250) $\div$ 3600 = 5.556 characters/sec.

= 5.556 < 10 character per sec

It requires one teletype with 10-character per sec. with 10 characters, modem 1 unit.

Transmitted characters for 1000 calls per hour

= (80 x 1000) $\div$ 3600 = 22.22 character/sec.

= 22.22 < 30 characters/sec

It requires three, 10-character per sec teletypes with a 10-character data modem 1 unit.

Transmitted characters for 2000 call per hour

= (80 x 2000) $\div$ 3600 = 44.44 character/sec

= 44.44 < 50 characters/sec

It requires five, 10-character per sec teletypes with a 300-character modem 1 unit.

Terminal Device Cost for 250 Calls Per Hour

Description	Monthly rental
1, 10-character-per-sec teletypewriter	$100.00

1, 10-character-per-sec modem	25.00
1 operator (he can do both as teller or programmer)	600.00
20 miles, 10-character-per-sec channel rates, communication line	35.00
Total	$760.00

Terminal Device Cost for 1000 Calls Per Hour

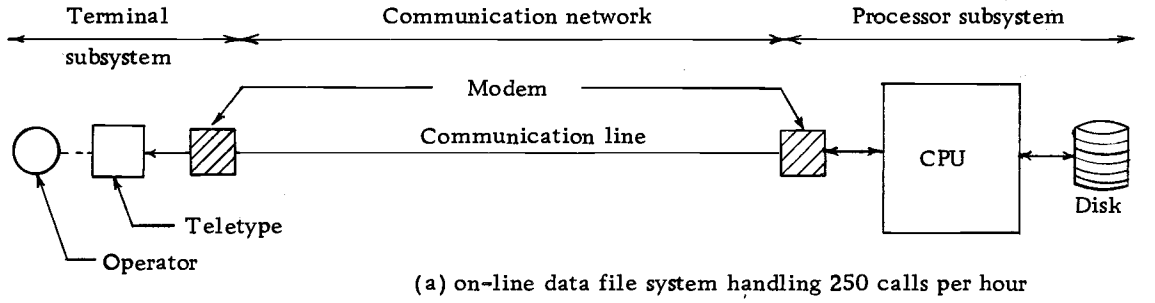
Description	Monthly rental
3, 10-character teletypes	$300.00
1, 300-character-per-sec time multiplexor	500.00
1 up-to 300 character-per-sec modem	40.00
3 operators (salaries)	1800.00
1, 10-character-per-sec 20 mile communication line	35.00
Total	$2675.00

Terminal Device Cost for 2000 Calls Per Hour

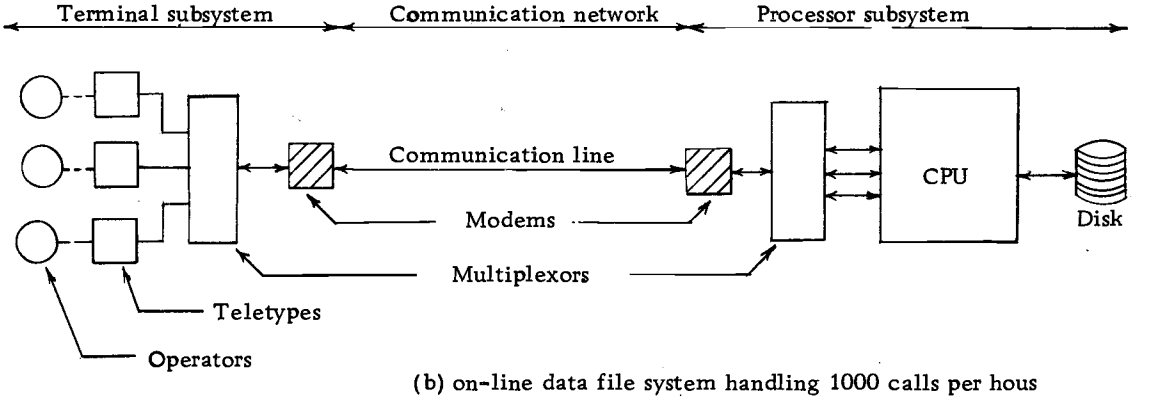
Description	Monthly rental
5, 10-character teletypes	$ 500.00
1, 300-character-per-sec line multiplexor	500.00
1 up-to 300-character-per-sec modem	40.00
5 operators (salaries)	3000.00
20 miles, 10-character-per-sec, communication line	35.00
Total	$4075.00

See the details of on-line data file terminal devices required for each specific rate of use of the records in the file, in Figure 7.12 which follows.

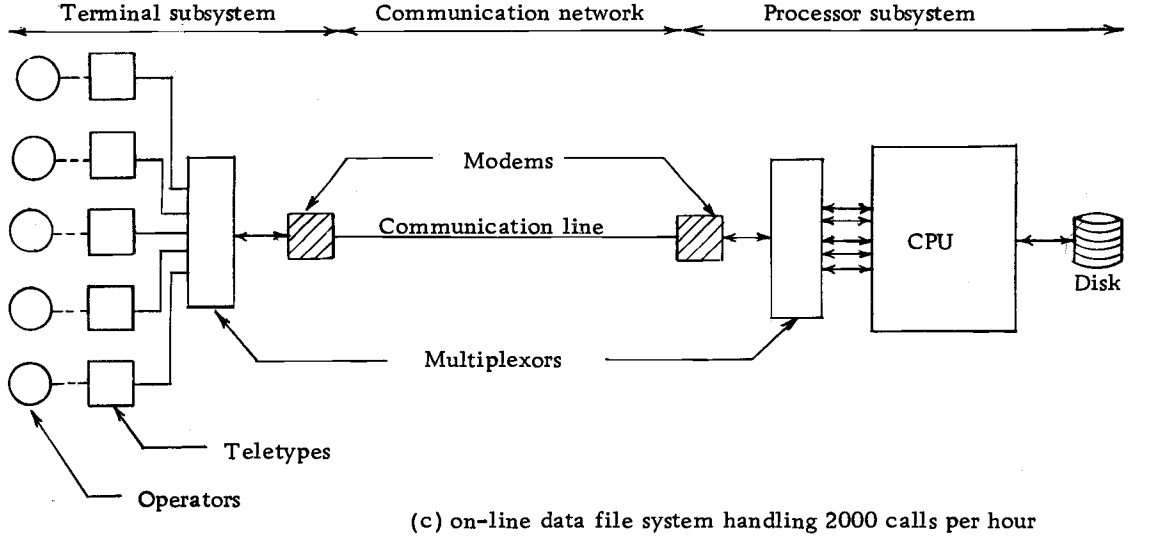
Figures 7.13, 7.14 and 7.15 illustrate how customer operating cost per call (per unit cost) of each typical file increases as the value of file loading factors, α, or the number of records in the file increases when the rate of use are 250 calls per hour, 1000 calls per hour and 2000 calls per hour respectively. For each specific rate of use the degree of increase depends upon the value of average throughput-time per record retrieval and the number of required tracks (disk storage space). As shown before in Figures 7.3 to 7.6 at a specific file loading factor, α, each type of file takes a different average throughput-time per record retrieval; the degree of variation between the different type of file depends upon their methods of accessing. Each typical file needs two types of disk storage space. One is to support the processing program. This type of disk storage space does not vary much for each type of file. The other storage space is to support the data records of the file. In case the file system uses File 1 to File 5 the required disk space for these types of files, varies directly as the number of data records increases. In case the file system uses File 6 to File 8, direct disk files using hash function as their directory decoder, the required disk storage space at any value of loading factor, is the same as when $\alpha = 1$ (the



(a) on-line data file system handling 250 calls per hour



(b) on-line data file system handling 1000 calls per hous



(c) on-line data file system handling 2000 calls per hour

Figure 7.12. On-line data file system configurations for handling 250, 1000, and 2000 calls per hour.

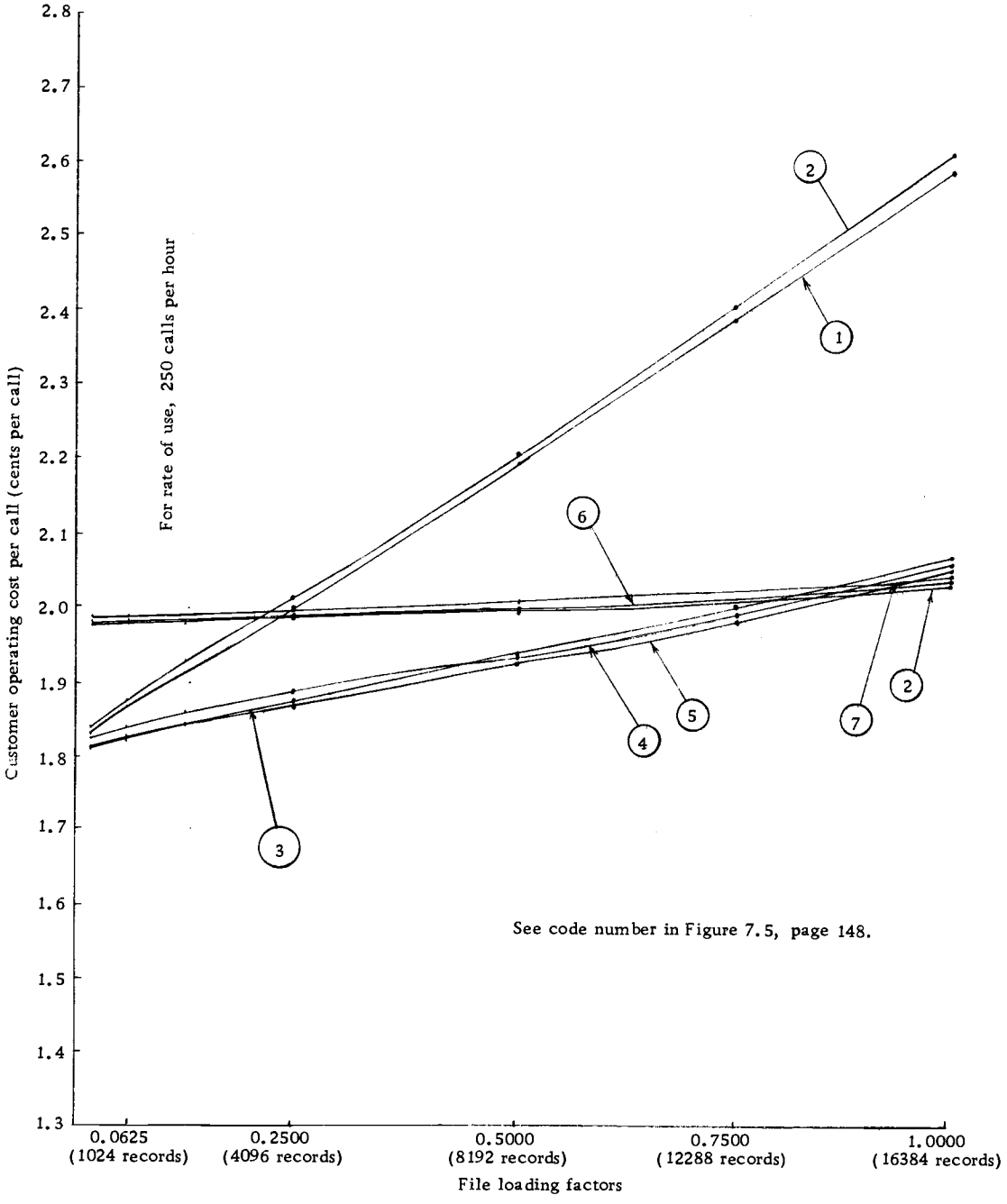


Figure 7.13. Customer operating cost per call (unit cost) as the function of file loading factors, for each typical file, code numbers 1 to 8, using full name of record in accessing.

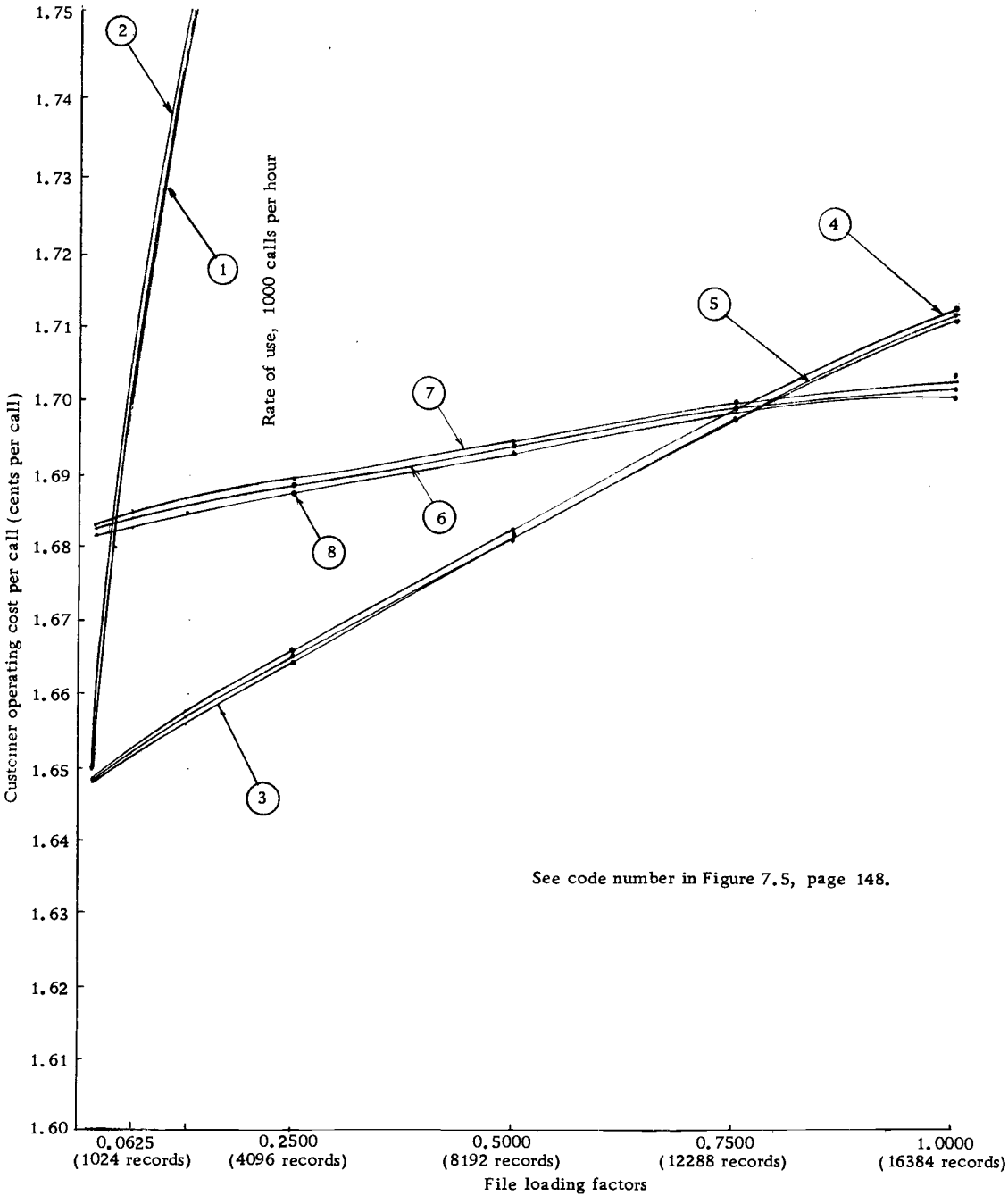


Figure 7.14. Customer operating cost per call as the function of file loading factors, for a typical file, code numbers 1 to 8, using full name of record in accessing.

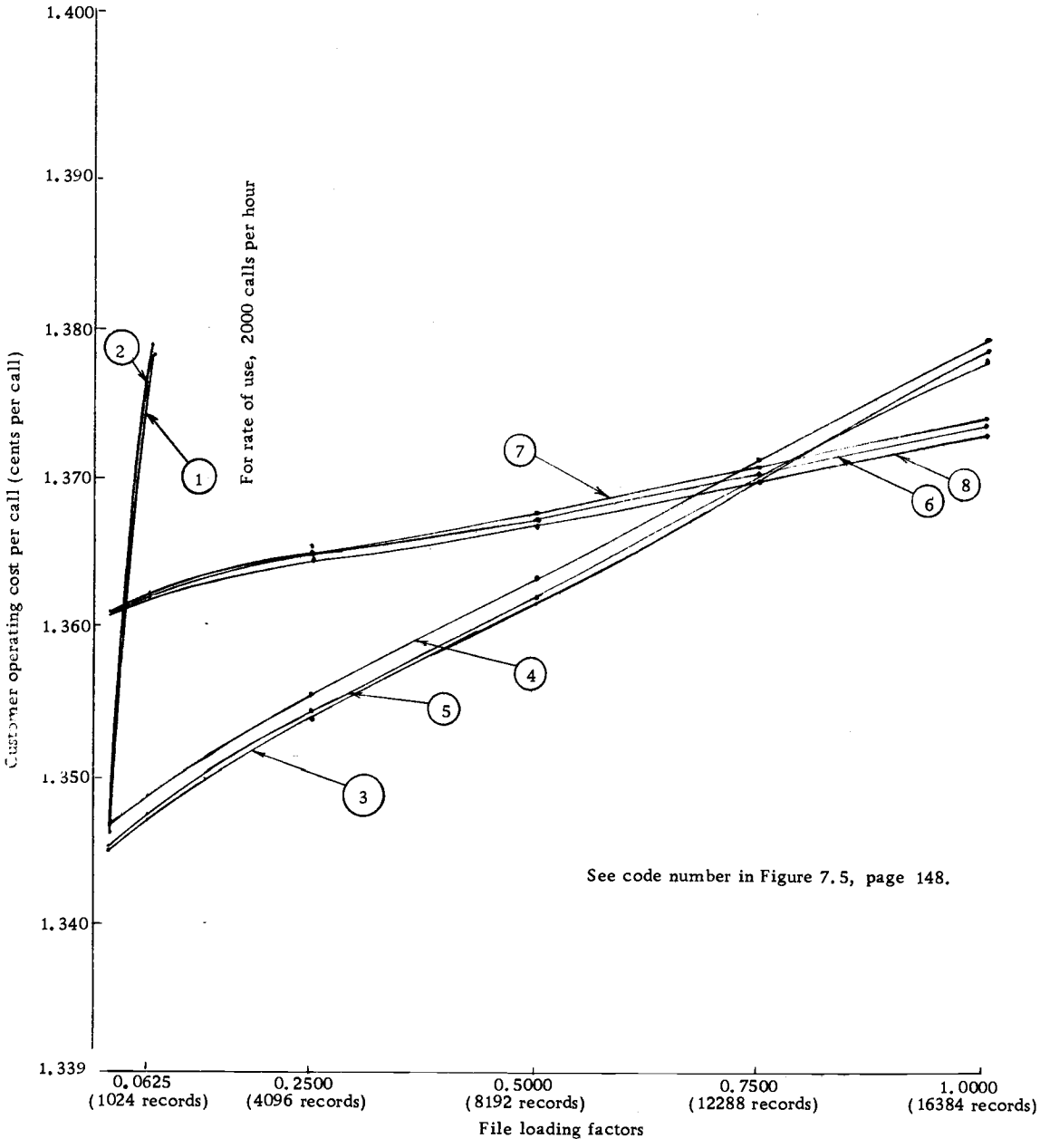


Figure 7.15. Customer operating cost per call as the function of file loading factors, for each type of file, code numbers 1 to 8, using record's full name in accessing.

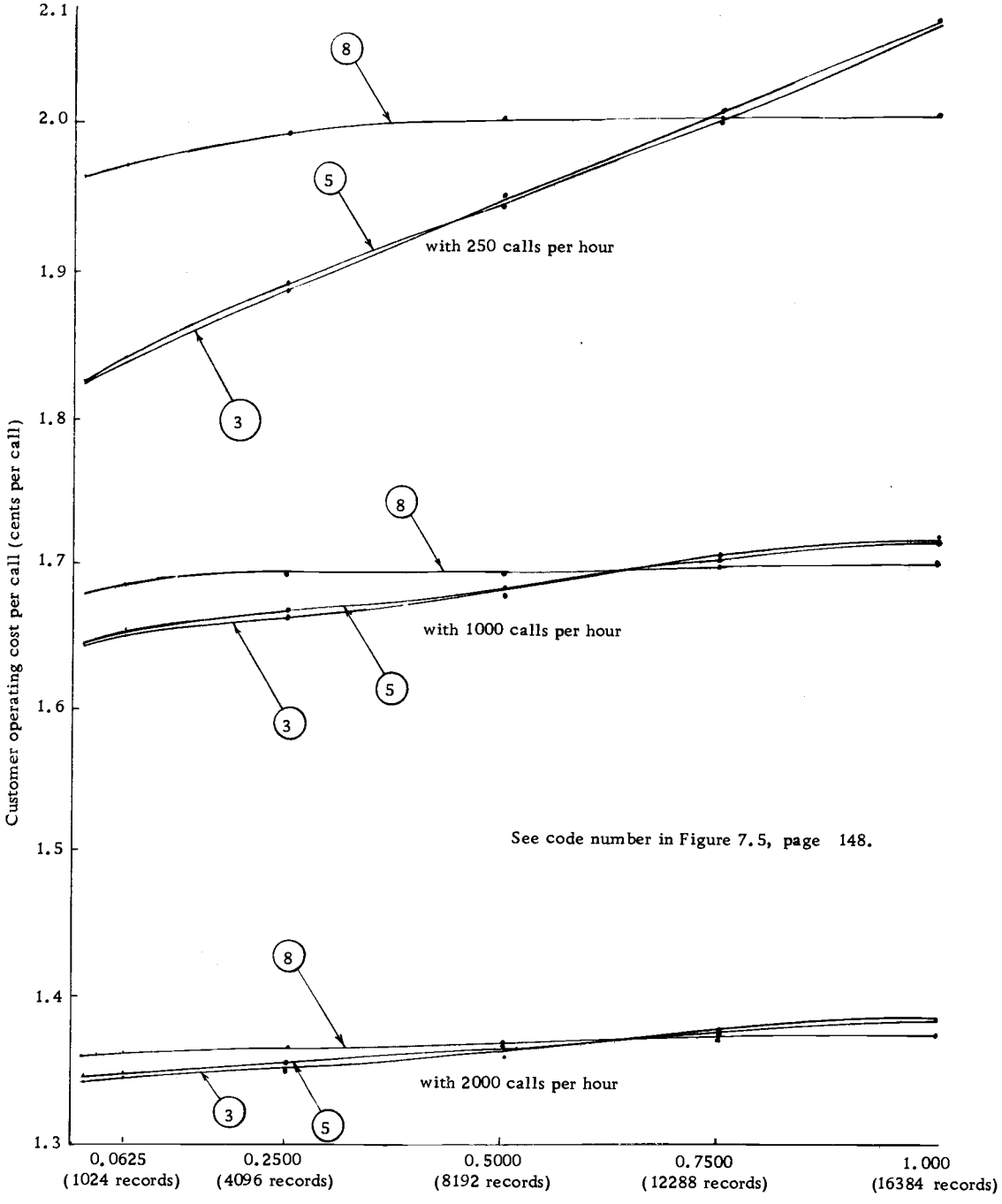


Figure 7.16. Customer operating cost per call (unit cost) as the function of file loading factors for the outstanding files, code numbers 3, 5 and 8 using full name of records in accessing.

Table 7.6. Data computation of customer operating cost per call (unit cost) as the function of file loading factors of each typical file organization method of specific selected rates of use, using full name of records in accessing.

File loading factor,		0.0156	0.0372	0.0625	0.2500	0.5000	0.7500	1.0000
Number of records in file		128 records	512 records	1024 records	4096 records	8192 records	12288 records	16384 records
Typical files	Rate of use calls per hr.	cent	cent	cent	cent	cent	cent	cent
Unsorted sequential disk file	250	1.82523	1.84384	1.86880	1.98994	2.20476	2.39668	2.58874
	1000	1.65000	1.66472	1.88378	1.80000	-	-	-
	2000	1.34619	1.36025	1.37851	-	-	-	-
Strickly sequential disk file	250	1.82615	1.84487	1.86984	2.01828	2.21489	2.41417	2.60813
	1000	1.65041	1.66522	1.68445	1.79847	-	-	-
	2000	1.34224	1.36066	1.37893	-	-	-	-
Indexed sequential disk file	250	1.82392	1.83150	1.83933	1.88662	1.9480	2.01017	2.0 967
	1000	1.64811	1.65086	1.65331	1.66589	1.68146	1.69736	1.71300
	2000	1.34421	1.34615	1.34771	1.35450	1.36243	1.37062	1.37861
Single level directory partitioned disk file	250	1.82464	1.83198	1.83950	1.88677	1.94728	2.00906	2.07040
	1000	1.64849	1.65139	1.65359	1.66642	1.68182	1.69762	1.71306
	2000	1.34453	1.34734	1.34800	1.35508	1.36298	1.37111	1.37890
Double level directory partitioned disk file	250	1.82434	1.83160	1.83927	1.88647	1.94693	2.00880	2.07013
	1000	1.64817	1.65096	1.65333	1.66606	1.68143	1.69736	1.71276
	2000	1.34420	1.34625	1.34773	1.35473	1.36258	1.37085	1.37860
Direct disk file with linear probing	250	1.96960	1.97209	1.97480	1.99013	2.00961	2.02933	2.04967
	1000	1.68336	1.68417	1.68542	1.68973	1.69475	1.69979	1.70558
	2000	1.36104	1.36144	1.36233	1.36507	1.36757	1.37027	1.37362

Table 7.6. Continued.

File loading factor,		0.0156	0.0372	0.0625	0.2500	0.5000	0.7500	1.000
Number of records in file		128 records	512 records	1024 records	4096 records	8192 records	12288 records	16384 records
Typical files	Rate of use calls per hr.	cent	cent	cent	cent	cent	cent	cent
Direct disk file with random probing	250	1.96970	1.97222	1.97499	1.99026	2.00980	2.02958	2.05055
	1000	1.68347	1.68460	1.68553	1.68988	1.69487	1.70007	1.70644
	2000	1.36116	1.36206	1.36270	1.36522	1.36776	1.37054	1.37449
Direct disk file with direct chain probing	250	1.96960	1.97211	1.97487	1.99007	2.00961	2.02927	2.04883
	1000	1.68344	1.68448	1.68540	1.68973	1.69466	1.69975	1.70475
	2000	1.36102	1.36193	1.36256	1.36507	1.36756	1.37023	1.37280

file is full). This causes the customer's operating cost per call (unit cost) of File 6 to File 8 to be a little higher than that for File 1 to File 5 within the range of $0 < \alpha < 0.75$. However, as for $0.75 < \alpha < 1$, this range causes the unit cost of File 6 to File 8 to be less than the unit cost of File 1 to File 5. Furthermore, the average throughput time per record retrieval of File 1 and File 2 varies as the function of $\frac{(N+1)}{2}$, where N is the number of records, in the file system. Therefore, when the value of N is increased, the average throughput time per record retrieval of File 1 and File 2 is enormously increased. This causes the unit cost of File 1 and File 2 to be excessively high and the achievable throughput rate capability lower than that of the other files, when file loading factor, α, is increased. So for the specific rate of use 1000 calls per hour and 2000 calls per hour, File 1 and File 2 cannot operate in the range of $0.500 < \alpha < 1.000$. From the results of computation and shown in Table 7.6 and Figure 7.15 File 3 and File 5 give good result of unit cost for $0.0078 < \alpha < 0.75$. File 8 gives the best result for $0.75 < \alpha < 1.000$. File 2 gives the worst result for $0.0078 < \alpha < 1.000$, as the unit cost is excessively high.

Conclusions and Recommendations

From the preceding comparative results it has been shown that eight specific types of file can be grouped into four groups; (1) sequential disk file (Files 1 and 2), (2) Index Sequential Disk File (File 3), (3) Partitioned disk file (Files 4 and 5), and (4) Direct disk file (Files 6, 7 and 8). Because of the results of the investigation and the methods of organization there is not much difference between the typical files in the same group. The following conclusions and recommendations may be useful to the designer or the evaluator in deciding which typical file is the most suitable for his specific problem.

1. In case the data file is rather static, having a low percentage of additions of data records to the file system, the most suitable method of organization is dependent on its initial file size:
   a. If the initial file size is in terms of hundred-logical-records, or equivalent to $0 < \alpha < 0.0156$ in this investigation, the Indexed Sequential File seems to have the lowest customer operating cost per call, unit cost. If the rate of use of the file required by the customer is less than its maximum-achievable-throughput-rate capability, 16438 calls per hour, it is

recommended that the Indexed Sequential File be used. For simplicity, it is also recommended the Unsorted Sequential File to be used.

b. If the initial file size is in the range of thousand-logical records to ten-thousand-logical records, i.e. $1000 < N < 10,000$ or $0.0625 < \alpha < 0.7500$. The Indexed Sequential File and the Partitioned File especially the Double-directory partitioned disk file, have less unit cost than other types of files. If the rate of use of the file required by the customer is less than its maximum achievable-throughput-rate capability, 19701 calls per hour, it is recommended that the Index Sequential File or the Double-directory partitioned disk file or some other type of Multi-level-directory partitioned disk file be used.

c. If the initial file size is in the range of ten-thousand logical records to sixteen-thousand-logical-records or more, i.e., $10,000 < N < 16,000$ or more, or $0.7500 < \alpha < 1.0000$. The Direct disk file, especially the direct disk file using chain probing, has the lowest unit cost and the greatest speed of accessing. If the rate of use of the file required by the customer is less than its maximum achievable-throughput-rate

capability, 20597 calls per hour, it is strongly recommended direct disk file with direct chain probing be used.

d. From the results of the investigation it is not recommended that the strictly sequential disk file be used for the on-line-data file system.

2. In case the data file is a dynamic one having a high rate of increase of records in the file, i.e., it takes only a short period of time for the file to be full, $\alpha = 1.0000$, it is recommended that the direct disk file with direct chain probing be used.

3. In this evaluation all results obtained may be considered as the optimum results, although the writer cannot guarantee 100% perfection, since in each specific on-line data file system there are many factors (or constraints) to be considered. In case the reader wishes to analyze and optimize any specific system it is recommended that the optimizing procedure be carried out as that in the same manner as in this investigation. Hopefully the results obtained will still be within the span of the writer's conclusions and recommendations.

4. This investigation indicates that for the on-line data file system using disk memory as the storage device, the

speed of accessing or retrieval is limited by the following factors:

a. operating speed of disk unit (disk RPM)

b. read-write-head positioning time.

This limitation could be solved by a higher speed disk, if one could be designed. The user will wish to select and use the highest speed and least read-write head positioning time possible, if there is not too much trade off between its operating performance and its cost.

5. The results of the preliminary work of this investigation shows that of two common types of internal search, linear search and binary search, illustrated in Appendix A, in the range of $0 < N < 64$ linear search has a higher speed and also needs less storage space than binary search. However, in the range of $64 < N < \infty$, although binary search requires more storage space because it has a higher speed, it provides a lower unit cost of operation. Then it is recommended that the internal search be used as follows:

   Linear search for $0 < N < 64$ approximately

   Binary search for $64 < N < \infty$

6. The results of the preliminary work show that the Hash decoding technique has the highest speed among the

directory decoding techniques which have been mentioned in Chapter V. More work has been investigated and evaluated for Hash decoding methods and the results obtained indicate that the efficiency of Hash decoding is strongly dependent on the expected searched records per random accessing, E, ($E \rightarrow 1 \Longrightarrow$ high efficiency; $E > 1 \Longrightarrow$ low efficiency). Results of the investigation show that the trade-off of E is available among the different hash functions with the same method of probing at a specific file loading factor, α. See Figure B. 19 page 302. The trade-off of E is also available among the different probing methods (linear probing, random probing and direct-chain probing) for a specific hash function. See Figure B. 20 page 304.

In this investigation Hash I with direct-chain probing has proved to be the best hash decoding method in the long run. Therefore, Hash I is selected as the mapping function for the evaluation of the direct disk file. See more details in Example 4, page 294.

BIBLIOGRAPHY

1. Bell, James R. The Quadratic Quotient Method: A Hash Code Eliminating Secondary Clustering. Communication of Association Computing Machinery 13, 2 (February 1970). 107-109.

2. Brightman, Richard W., Bernard J. Luskin and Theodore Tiltion. 1968. Data Processing for Decision-Making. The Macmillan Company, New York 252-255, 352-357.

3. Chapin, Ned. A Comparison of File Organization Techniques.

4. Chapin, Ned. Common File Organization Techniques Compared. Fall Joint Computer Conference, 1969. 414-421.

5. Coffmann, E. G., Jr., and Eve J. Coffman. File Structure Using Hashing Functions. Communication of the Association for Computing Machinery. 13, 7(July, 1970).

6. Control Data. 3228 Disc Controller Training Manual. First Edition. 1968.

7. Control Data. 1969. Computer System Compass Reference Manual.

8. Filler, William. 1958. An Introduction to Probability Theory and its Applications. John Wiley and Sons, Inc. New York.

9. Flores, Ivan. 1966. Computer Programming. Prentice-Hall, Inc. Englewood Cliffs, New Jersey. 246-287.

10. Flores, Ivan. 1969. Sorting. Englewood Cliffs, New Jersey. Prentice

11. Flores, Ivan. 1970. Data Structure and Management. Prentice-Hall, Inc. Englewood Cliffs, New Jersey. 52-54, 225-241, 247-263, 269-279, 320-327.

12. Gaulthier, Richard L. and Stephen D. Ponto. 1970. Designing Systems Programs. Prentice-Hall, Inc., Englewood Cliffs, New Jersey. 112-156.

13. Hslio, David. Formal System for Information Retrieval From File Communication of the Association for Computing Machinery 13, 2 (February, 1970). 67-73

14. Hopgood, F. R. A. 1969. Compiling Techniques. American Elsevier, Inc. New York. 4-27.

15. IBM Student Text Introduction to IBM System/360 Direct Access Storage Device and Organization Methods. International Business Machines Corporation. 1969.

16. Jame, N. Meeker, N. Ronald Crandall, Fred A. Dayton and G. Rose. OS-3: The Oregon State Open Shop Operating System. Spring Joint Computer Conference, 1969, 241-248.

17. Johnson, L. R. An Indirect Chaining Method for Addressing on Secondary Keys. Comm ACM 4(1961), 218-222.

18. Landauer, Walter I. 1963. The Balanced Tree and Its Utilization in Information Retrieval. IEEE. Transaction on Electronic Computers. December 1963. 863-871.

19. Lefkovitz, David. 1969. File Structures for on-line System New York. Washington, Spartan Books. 27-36, 92-105.

20. Lombardi, Lionello. 1960. Theory of Files. Proceedings of the 1960 Eastern Joint Computer Conference New York IRE, 1960. 137-141.

21. Lowe, Thomas L. Direct-Access Memory Retrieval Using Truncated Record Names, Software Age, September 1964. 22-23

22. Lum, V. Y., P. S. T. Yuen and M. Dodd. Key-to-Address Transform Techniques - A Fundamental Performance Study on Large Existing Formatted Files. Comm. ACM 14, 4 (April 1971), 228-239.

23. Maurer, W. D. An Improved Hash Code for Scatter Storage Comm. ACM 77, 1 (Jan. 1968), 36-38.

24. Maurer, W. D. 1969. Programming. An Introduction to Computer Languages and Techniques. Holden-Day, Inc. San Francisco, 94-109. 116-132.

25. McGee, William C. 1962. The Property Classification Method of File Design and Processing. Communication of the Association for Computing Machinery 8. 450-458.

26. McIlroy, M. D. A Variant Method of File Searching Communications of the Association for Computing Machinery.

27. Morris, Robert. Scatter Storage Techniques. Comm. ACM 11, 1 (Jan.1968), 38-43.

28. Parkhill, D. F. 1966. The Challenge of the Computer Utility. Addison Wesley Publishing Company, Reading, Massachusetts. 121-144.

29. Peterson, W. W. Addressing for random-access storage. IMB Journal. April 1957, 130-146.

30. Rothstein, Michael F. 1970. Guide to the Design of Real-Time System. John Wiley and Son, Inc. New York. 8-18, 42-55, 116-137, 187-206, 225-232.

31. Schay, G., Jr., and Spruth, W. G. Analysis of a File Addressing Method Comm. ACMS, 8(Aug. 1962), 459-462.

32. Sharpe, William F. 1969. The Economics of Computers. The Rand Corporation. 279-363.

33. Stimler, Saul. 1969. Real-Time Data-Processing Systems. A Methodology for Design and Cost/Performance Analysis. McGraw-Hill Book Company. 17-36, 37-76, 149-170.

34. Sussenguth, Edward H. Use of Tree Structures for Processing Files. Comm ACM 6, 5.

35. The Rand Corporation. 1955. A Million Random Digits. The Free Press. Glencoe, Illinois.

APPENDICES

APPENDIX A

The Analysis of Typical File

Evaluation of Internal Linear Search and Binary Search

For evaluation of the characteristics of internal linear search and binary search, the two basic methods of internal search the following strategies are used:

1. The model of the Test File is created in internal core memory. The Test File should look like a directory file. Each record of this Test File consists of two parts, key name of the record and its address in the main file. The operation is equivalent to simulation of the directory file. The selected names for the Test File are shown in Table A. 1.
2. In the arrangement of the format of the Test File record, each record in the Test File is a "fixed length record", and its supported by two successive computer words, (1 word = 24 bits). The first word contains four characters which is the fixed-length unique key of the record in the main file. The second word contains its corresponding address in the main file.

Table A. 1. Directory File.

Name selected as the record name in the main file	Key Name (Name Area)	Address Location (Data Area)
Administration Division	ADMI	0001
Air Field Construction Division	AIRF	0014
Building Construction Division	BUIL	0027
Material Division	MATE	0025
Supply Division	SUPP	0040
Technical Division	TECH	0053
Utilities Division	UTIL	0066

Search argument item = BUIL

output = 0027

Search argument item = BIUL

output = BIUL NO SUCH KEY IN DIRECTORY

Illustrated example of accessing a record from directory file.

Due to the fact that the CDC-3300 can handle only 4 characters for a 24-bit word, it is impossible to put the full-name in the one word name area of the Test File; only 4 characters of the full name have to be selected. The group of 4 selected characters from the full name is called key name of the record the main file. There are many ways to select the 4 characters from the full name to be used as the key name. The reader has to keep in mind that the effective way of selecting 4 characters from the full name should provide 0% (or slightly more) redundance of key names. Otherwise the redundant

key name causes ambiguity in processing the file.

In this experiment the writer has chosen to form the key name by truncating the trailer characters of the full name. The method is usually preferred by users, but 0% redundancy can not be guaranteed. The details of conversion of the full name into a fixed-length key name are mentioned in Chapter V.

3. The symbolic program for sequential search has been performed as Test 1. The desired item is searched for in the Test File by means of linear search. The search function flowchart is equivalent to the flowchart of sequential cylinder indexed search as shown in Appendix C, page 343.
4. The symbolic program for binary search has been performed as Test 2. The desired item is searched for in the Test File by means of binary search. The binary search flowchart is shown in Appendix C, page 368.

Symbolic Program for Sequential Search: (Compass)

Location	Operation Code	Address + Index	Comment
IDENT	Test 1		Test 1 is NAME OF PROGRAM
ENTRY	START		Initial Statement to start program
START	ENI	0, 1	0 → B1
	ENI	11, 3	11 → B3
LOOP	ENQ	1	

Location	Operation Code	Address + Index	Comment
	ENA	A	(A) → Acc
	AIA	1	
	READ	60	
	INI	1, 1	(B1)+1 → B1
	IJD	LOOP, 3	If (B3) >0; (B3) - 1→B3, go to LOOP
	ENQ	1	
	ENA	KEY	(KEY) → Acc.
	READ	60	
	ENI	12, 1	12 → B1
	ENQ, S	-0	Set Q = (0000000)
	LDA	KEY	(KEY) → Acc.
	MEQ	A, 2	
	UJP	LOST	
	INI	1, 1	(B1) + 1→B1
	LDA	A, 1	(A + (B1)) → Acc.
	STA	KEY	(A_{cc}) → KEY
	ENQ	2	
	ENA	BLANKS	
	WRITE	61	
	UJP	END	
LOST	ENQ	9	
	ENA	BLANKS	
	WRITE	61	
END	SBJP		
BLANKS	BCD	1,	
KEY	BSS	1	
MESSAGE	BCD	7,...NO SUCH KEY IN DIRECTORY	
A	BSS	12	

END START

FINIS

T_{ALS} = Average linear searching time per record retrieval from array, file

$$= t_{LSPA} + \left(\frac{1+N}{2}\right) t_{LS}$$

$$= 1.375 + 1.375 + 2.750 + \left[\left(\frac{1+N}{2}\right) 4 \times 2.5 + 4.125\right] + 1.375 + 1.375 + 3.125 + 2.79$$

$$= 5.0\,N + 9.125 + 14.125$$

T_{ALS} = 5.0 N + 23.25 μsec. (based on tested-program. (8.1)

where t_{LSPA} = Linear search program auxilliary time per record retrieval

t_{LS} = Logical path Linear search time per second retrieval

Symbolic Program for Binary Search: (Compass)

Location	Operation Code	Address + Index	Comment
	IDENT	TEST2	
START	ENI	0, 1	0 ⟶ B1
	ENI	11, 3	11 ⟶ B3
LOOP	ENQ, S	1	Set (Q) ⟶ all bits are ones
	ENA	A	A ⟶ Acc.
	AIA	A	(Acc) + (B1) ⟶ (Acc)
	READ	60	
	LDA	A, 1	(A + B1) ⟶ (Acc)
	SCA	MASK	Selective complement A_{cc}
	STA	A, 1	(Acc) ⟶ (A + B1)

Location	Operation Code	Address + Index	Comment
	INI	1, 1	(B1) + 1 ⟶ B1
	IJD	LOOP, 3	If (B3) > 0; (B3)-1⟶B3, go to LOOP
	ENQ, S	1	Set (Q) ⟶ all bits are ones
	ENA	KEY	KEY ⟶ Acc
	READ	60	
	SCA	MASK	
	STA	KEY	(Acc) ⟶ KEY
	ENI	0, 1	0 ⟶ B1
	ENI	1, 2	1 ⟶ B2
	LDQ	KEY	(KEY) ⟶ Q
	LDA	INC	(INC) ⟶ Acc
OVER	IAI	1	(Acc) + (B1) ⟶ Acc
	LDA	TOP	(TOP) ⟶ Acc
	SBA	BOT	(TOP)-(BOT) ⟶ Acc
	ASG, S	3	Check to see if A_i < KEY < A_{i+1}
	UJP	LOST	
	LDA	INC	Computes NEXT 1/2 of Binary Search interval
	SHA	-1	
	AJA	2	
	STA	INC	
	LDA	A, 1	(A + B1) ⟶ Acc
	AQJ, EQ	FOUND	
	AQJ, LT	LESS	Binary search loop in upper portion of search Table 2 + KEY < a_i
	TIA	1	
	STA	TOP	
	LCA	INC	
	UJP	OVER	

Location	Operation Code	Address + Index	Comment
	TIA	1	Binary search loop in lower portion of search Table If KEY > a_i
	STA	BOT	
	LDA	INC	
	UJP	OVER	
FOUND	INI	1, 1	If search is satisfied and prepare for output
	LDA	A, 1	
	SCA	MASK	
	STA	KEY	
	ENQ	2	TTY OUTPUT
	ENA	BLANKS	
	WRITE	61	
	UJP	END	
LOST	ENQ	9	NOTIFIED THE USER
	LDA	KEY	
	SCA	MASK	
	STA	KEY	
	ENA	BLANKS	
	WRITE	61	
END	SBJP		
TOP	OCT	13	
BOT	OCT	-1	
INC	OCT	6	
MASK	OCT	4	
BLANKS	BCD	1,	
KEY	BSS	1	
MESSAGE	BCD	7, --NO SUCH KEY IN DIRECTORY	

Location	Operation Code	Address + Index	Comment
A	BSS	12	
	END	START	
	FINIS		

T_{ABS} = Average binary search-time per record retrieval from array, file.

$$= t_{BSPA} + t_{BS} \times \frac{1}{N} \left[2^{K}(K-1) + 1 + (K+1) \times (N - 2^{K} + 1)\right] - 10.25$$

$$= (8.250 + 9.995 - 10.250) + (35.30) \cdot N_{ATBS}$$

$$T_{ABS} = 7.995 + 35 \cdot 30 \times N_{ATBS} \quad \mu sec. \qquad (8.12)$$

where

t_{BSPA} = Binary search program auxiliary time per record retrieval

t_{BS} = Logical path Binary search time per record retrieval.

Comparison of Storage Space Used for Internal Search

Typical of required memory location	LINEAR SEARCH No. of required locations	BINARY SEARCH No. of required locations
Initiation and read-in both master record and key	13	19
Search	15	40
Reserve Spaces for Master file	2N	2N
Reserved space for program variables	11	15
Total required space (word)	2N + 40	2N + 14
Total space requirement (in Bit)	48 N = 960	48 N + 1776

Lets N = Number of items in array or file.

S_{LS} = Number of storage spaces requirement in core memory for Linear search, in words.

S_{BS} = Number of storage spaces requirement in core memory for Binary search, in words.

Then the general formulas are

$$S_{LS} = 2N + 40 \qquad (8.3)$$

$$S_{BS} = 2N + 74 \qquad (8.4)$$

The plotting results are shown in Figure B. 4, page 204.

Table A. 2. Results of average search time, linear and binary search per record vs. file size.

Number of items, N	$N_{ATLS} = \frac{(1+N)}{2}$		$N_{ATBS} = \frac{1}{N}[2^K (K-1) + (N - 2^K K - 1)$	(K + 1)*	T_{ALS}(μsec)	T_{ABS}(μsec)
1	1	1	1.000	1	28.25	43.295
2	1.5	2	1.500	2	30.75	60.945
4	2.5	3	2.000	2	35.73	78.595
8	4.5	5	2.625	3	45.75	100.657
16	8.5	9	3.375	3	65.75	127.132
32	16.5	17	4.278	4	105.75	159.008
64	32.5	33	5.125	5	185.75	188.907
128	64.5	65	6.073	6	345.75	222.371
192	96.5	97	6.714	7	505.75	245.000
256	128.5	129	7.039	7	665.75	256.471
320	160.5	161	7.437	8	825.75	270.309
384	192.5	193	7.693	8	985.75	279.557
448	224.5	225	7.895	8	1145.75	286.688
512	256.5	257	8.021	8	1305.75	291.136
576	288.5	289	8.241	8	1465.75	298.902
640	320.5	321	8.417	8	1625.75	304.868
704	352.5	353	8.561	9	1785.75	310.198
768	384.5	385	8.681	9	1945.75	314.434
832	416.5	417	8.738	9	2105.75	376.446
896	448.5	449	8.869	9	2265.75	327.070
960	480.5	487	8.924	9	2425.75	323.718
1024	512.5	573	9.011	9	2586.75	326.083
2048	1024.5	1025	10.006	10	5145.75	367.000
4096	2048.5	2049	11.003	11	10265.75	396.295

*Value of $\log_2 N$.

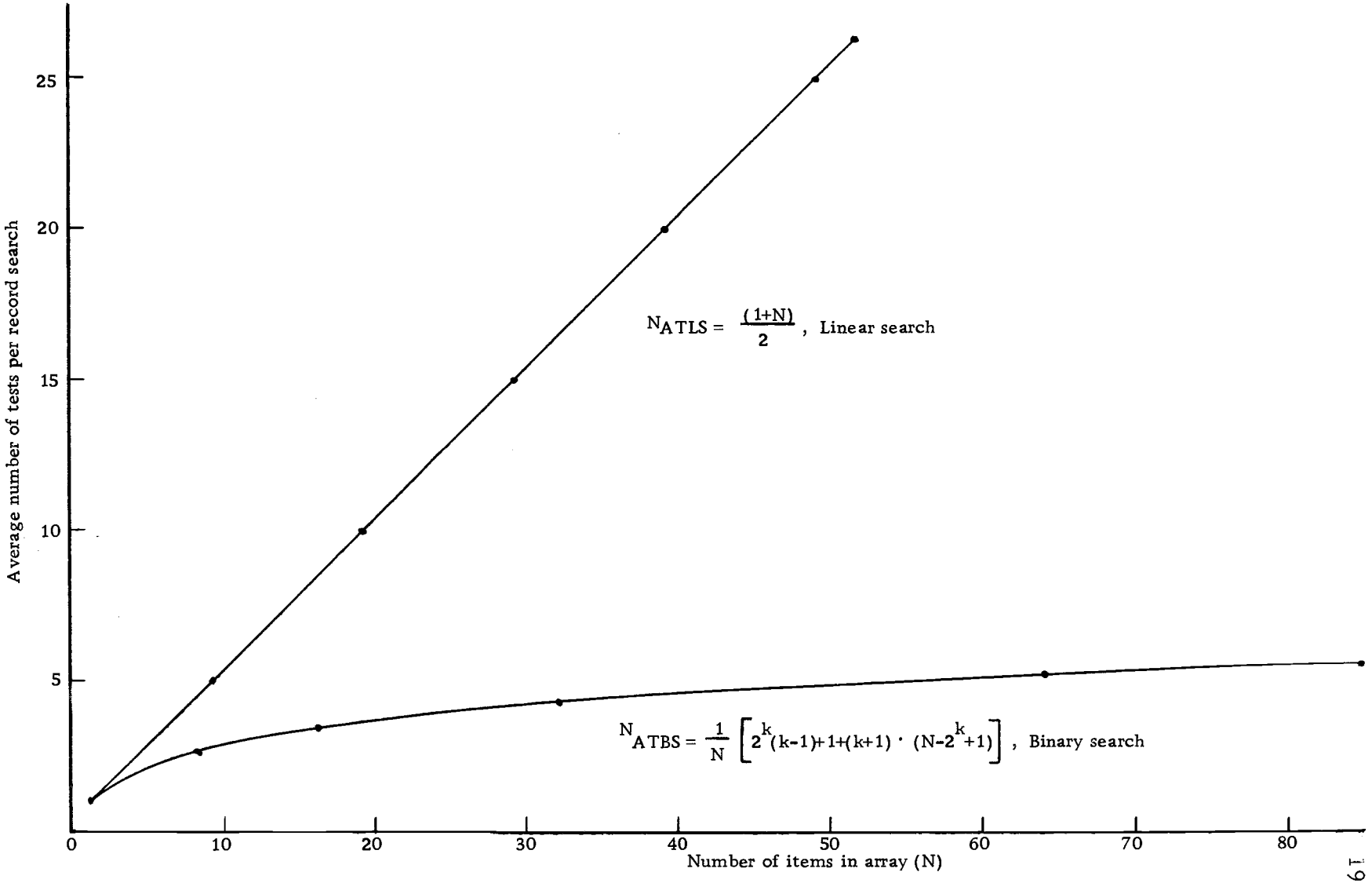


Figure A.1. Average number of tests per record search as a function of number of the items in the array.

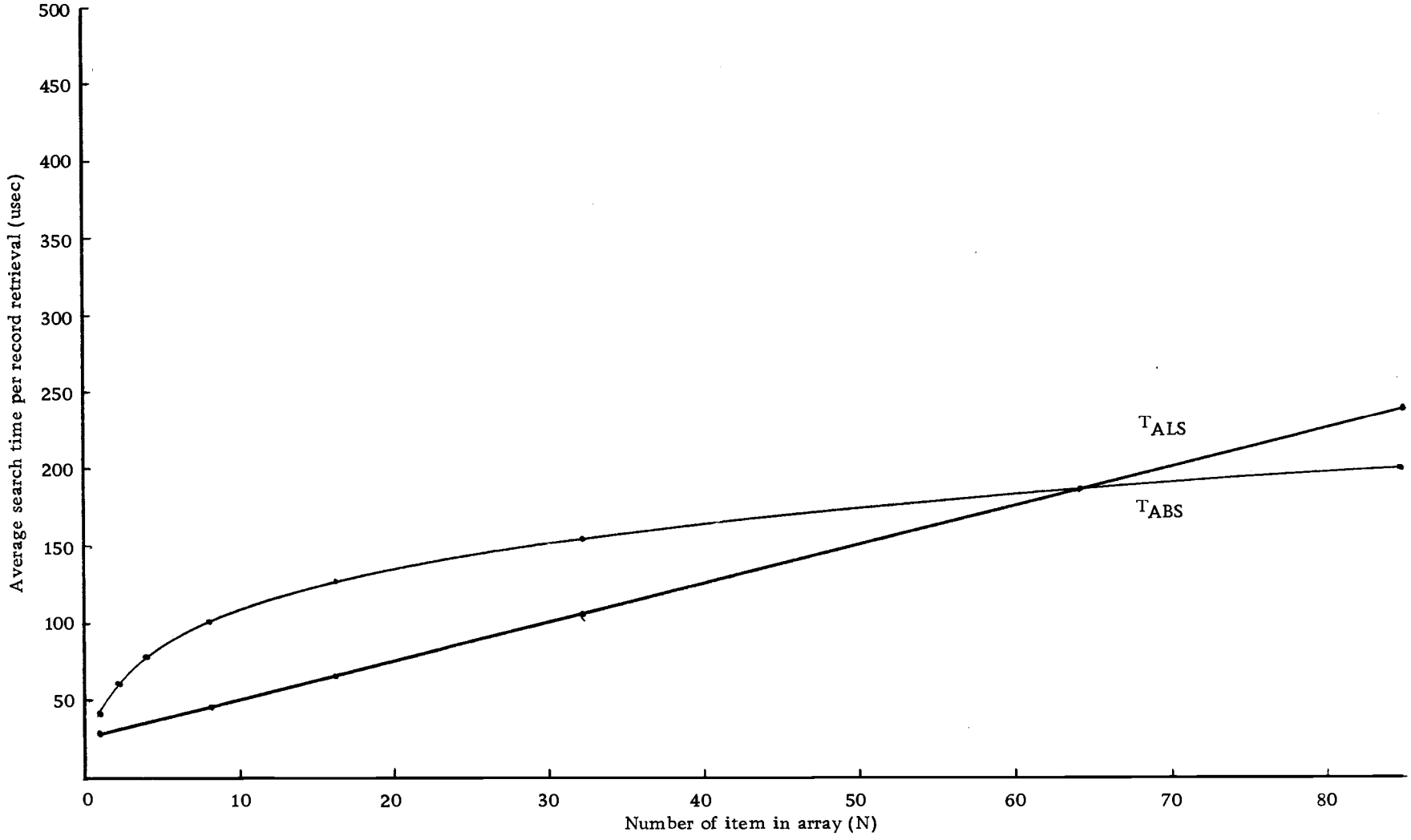


Figure A.2. Average search time per record retrieval both linear and binary search as a function of items in the file.

Figure A. 1 makes a graphical comparison between the number of average search per random record retrieval of internal linear search and internal binary search, and also shows how the curves rise as the number of records in the file increase. For internal linear search the degree of increase of number of average search per record retrieval, N_{ATLS}, varies directly as $\frac{(N + 1)}{2}$. When $N \rightarrow \infty \Longrightarrow \frac{(N - 1)}{2} \rightarrow \infty \Longrightarrow N_{ATLS}$ is enormously increased. For internal binary search, N_{ATBS} varies directly $\frac{1}{N}$ $[2^K (k - 1) - (K - 1) \times (N - 2^K - 1)] \simeq \log_2 N$ when $N \rightarrow \alpha \Rightarrow N_{ATBS} \simeq \log_2 N \Rightarrow N_{ATBS}$ is increased very little, so that the curve of N_{ATBS} is quite flat, providing a great diversion between N_{ATBS} and N_{ATLS}.

Figure A. 2 makes a graphical comparison between the average search time by linear search, T_{ALS} and the average search time by binary search, T_{ABS}. The degree of increase of T_{ALS} and T_{ABS} when N is increased, is the same as that described in Figure A. 1. The curves in Figures A. 1 and A. 2 indicate that for $0 < N \leqslant 64$, internal linear search is better than internal binary search. For $64 < N < \infty$ binary search is better than linear search, with only a little difference between their storage space requirement. See Table A. 3 and Figure B. 4 for how the number of core storage spaces increases for both linear and binary search as N, number of records in the file increases.

Table A. 3. Core storage space required for internal linear and binary search.

File size, N	S_{LS}	S_{BS}
Items	Words	Words
1	42	26
2	44	78
4	48	82
8	56	90
16	72	106
32	104	138
64	168	202
128	296	330
192	424	458
256	552	568
320	680	714
384	808	842
448	936	970
512	1064	1098
576	1192	1226
640	1320	1354
704	1448	1482
768	1576	1610
832	1704	1713
896	1832	1866
960	1960	1994
1024	2088	2122
2048	4136	4170
4096	8232	8266

S_{LB} = Core storage space for linear search.

S_{BS} = Core storage space for binary search.

One item = 2 successive computer records.

APPENDIX B

Analysis of Disk Access Time for Disk 854

Disk Access Time

Access time = the cylinder position time + the rotational latency time

then: Average access time = average position time + average latency time.

Disk Storage Drive (CDC· DISK 854 Unit)

Maximum positioning time	= 165	msec.
Average positioning time, (1/3 of max. move)	= 95	msec.
Cylinder to clyinder positioning time	= 30	msec.
Maximum latency	= 25	msec.
Average latency	= 12.5	msec.
Maximum access time	= 190	msec.
Average access time	= 107.5	msec.
Full rotation time	= 25	msec.

Data Transfer (CDC Disk 854 Unit)

The nominal data transfer rate, disk drive = 100,000 bytes/ second.

The data transfer is of the following nature:

1. Data is addressed and written in a storage unit in discrets (0, 1, 2, 3, ect) blocks (sectors).
2. Data channel can read or record as little as one byte or as many bytes as necessary to reach the end of file.
3. Reading or writing, the operation must commence at the start of a sector.
4. When writing, if less than a full sector is written, the remainder of the sector is automatically filled with zeroes.

Record Format

The record format for a test file is shown in Table A. 1, page 185 and Figures B. 2 and B. 3, pages 202 - 203.

1 logical record	=	384 bits
1 sector, disk 854, data field	=	1536 bits
1 sector, disk 854	=	4 logical records
1 track	=	64 logical records
1 cylinder (10 tracks)	=	640 logical records
1 unit (100 cylinders)	=	64000 logical records.

Data Read/Write Time, $T_{R/W}$

$$T_{R/W} = \frac{\text{Full rotation}}{\text{Number of records per track}} \times (\text{Number of records R/W})$$

$$T_{R/W} = \frac{25 \text{ msec.}}{64} \times N \quad \text{for } N < 64$$

Data Track Read in Time, T_{RINDT}

$$T_{RINDT} = 25 \text{ msec.} \tag{9.1}$$

Read/Write Head Positioning Time, $T_{AR/WHPT}$

$$T_{AR/WHPT} = 90 \text{ msec.} \tag{9.2}$$

Average Waiting Time Per Track, T_{AWT}

$$T_{AWT} = 12.5 \text{ msec.} \tag{9.3}$$

Checking and Connecting Logical Unit Time, T_{RCLU}

$$T_{RCLU} = 0.008 \text{ msec.} \tag{9.4}$$

Checking and Connecting I/0 Unit Time, $T_{RCI/0}$

$$T_{RCI/0} = 0.008 \text{ msec.} \tag{9.5}$$

Record Full Name of Fixed-Length Key Search Time, $T_{CFNFK})$

To obtain the fixed length key from the supplied record full name, the program called "Test 8" is used (see Appendix C, and the following parameters are defined and computed.

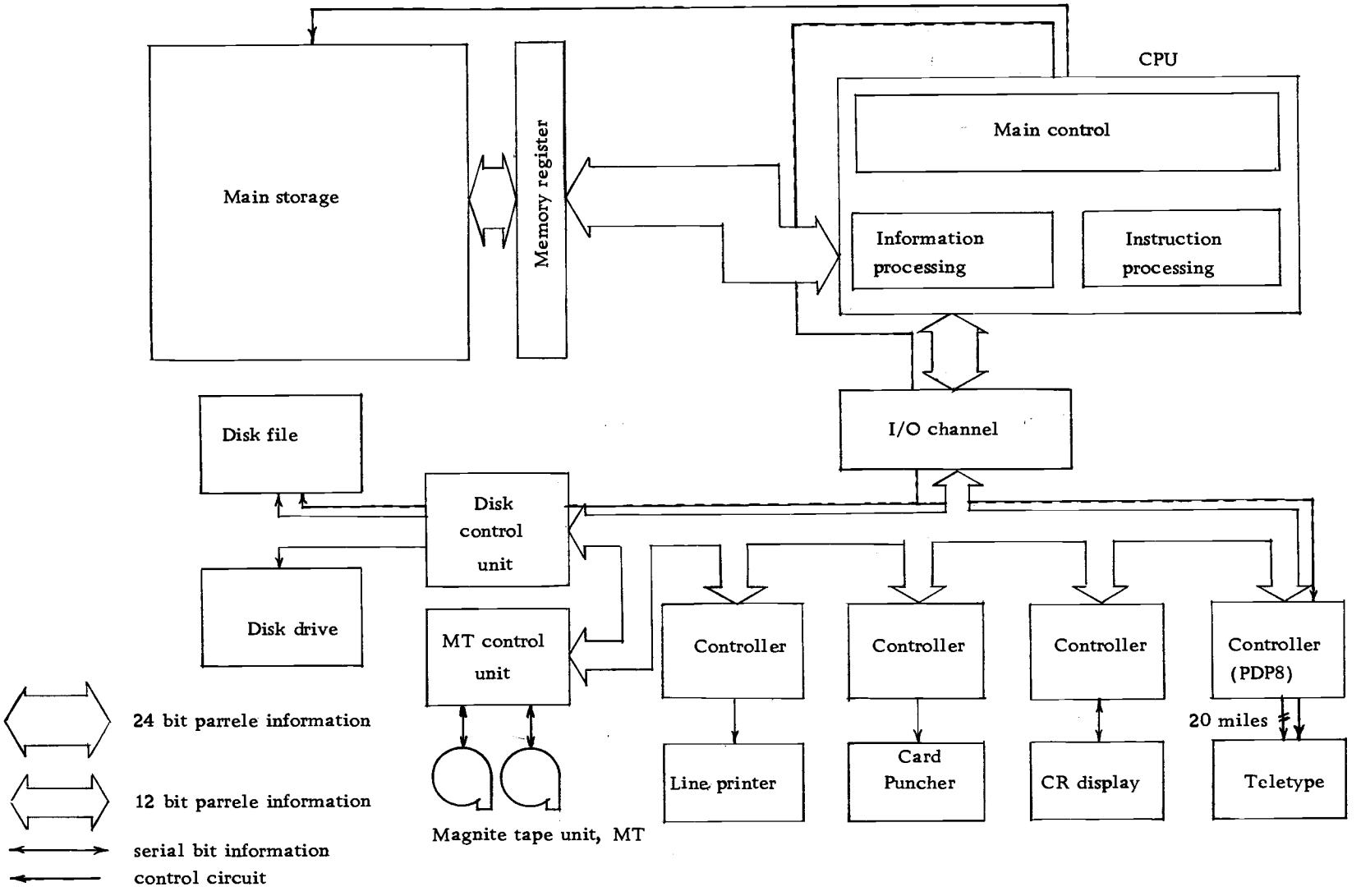


Figure B. 1. System block diagram of the simulator.

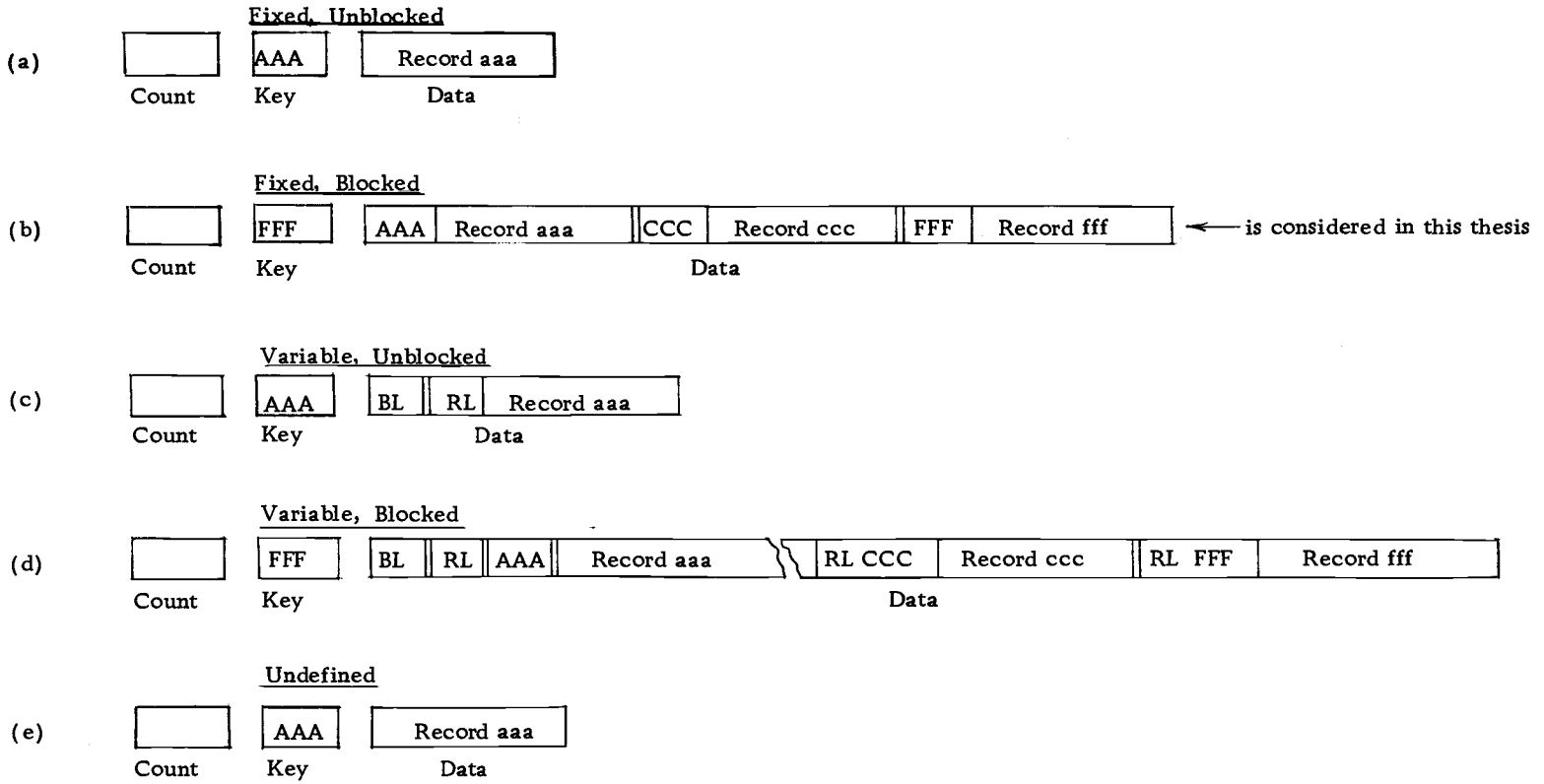


Figure B.2. Conventional record formats.

Items	Character	Computer word
(or key of a record)		
1. Employee number	4	1
2. Name	16	4
3. Social Security and age number	9+3	3
4. Starting date M/D/Y	8	2
5. Qualifications	3	1
6. Salary per month	4	1
7. Dependents	2	1
8. Net income per year	7	2
9. Accumulate saving account	4	1
Total	60 (spare 4)	16

Illustration of logical format in a file

Location	Core Memory			
L	0	0	3	2
L + 1				
L + 2				
L + 3				
L + 4				
L + 5	5	4	3	6
L + 6	4	7	1	7
L + 7	3	A	3	5
L + 8	1	2	/	1
L + 9	8	/	6	4
L + 10	B	S	C	
L + 11	8	0	0	·
L + 12	0	0	0	5
L + 13	0	1	0	0
L + 14	·	0	0	
L + 15	5	0	5	·

The mapping of a logical record in core

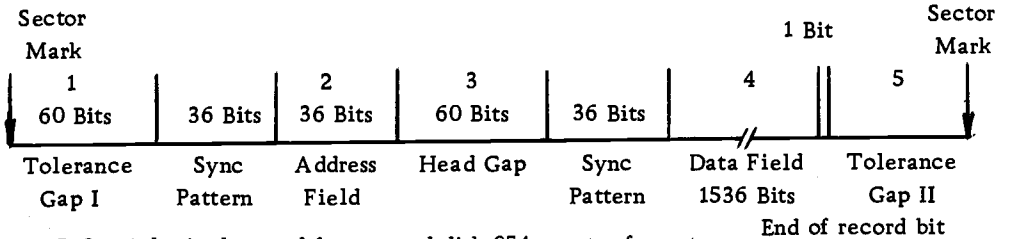


Figure B.3. A logical record format and disk 854, sector format.

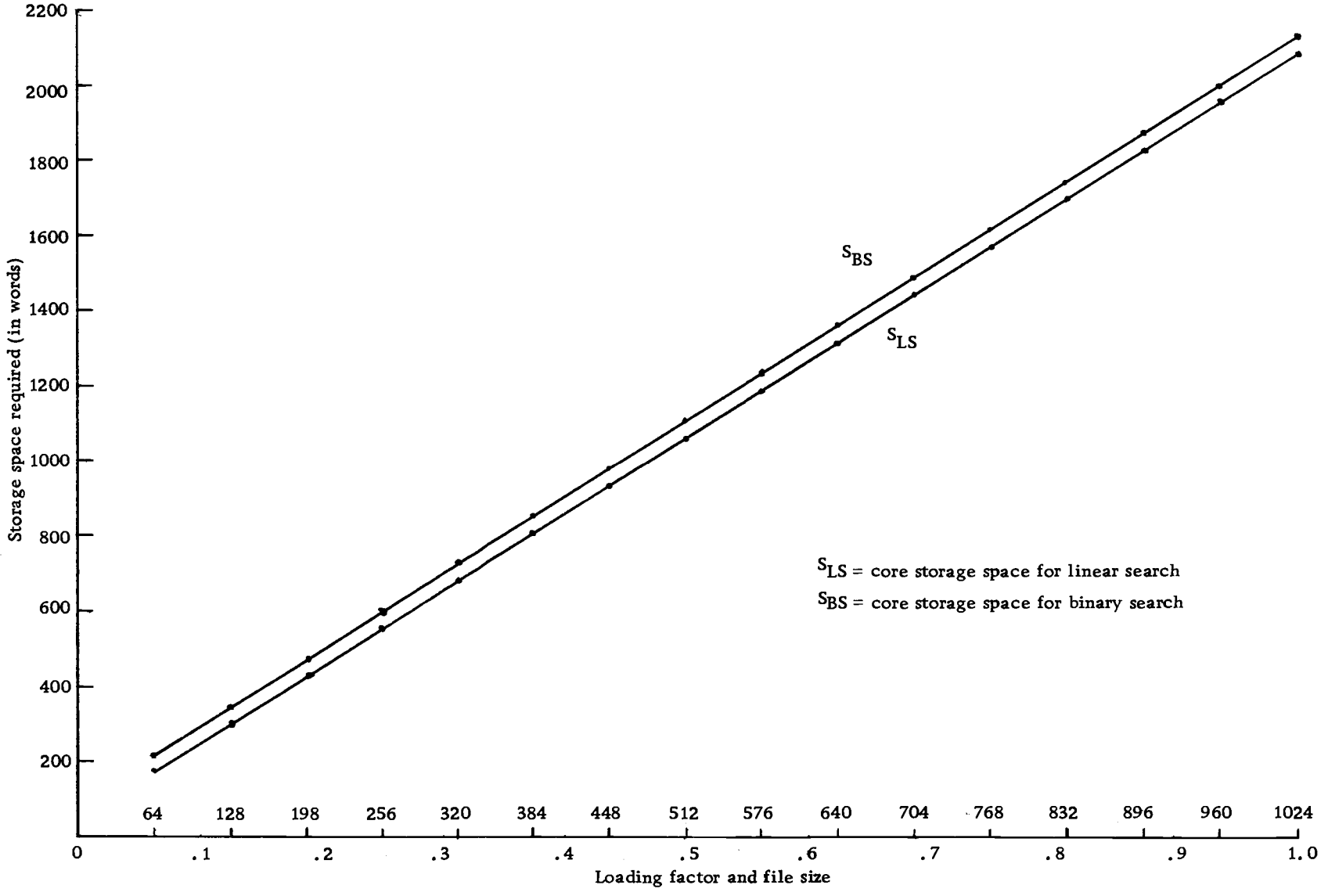


Figure B. 4. Storage space required for test program as a function of file size.

T_1 = (Read in the NAME time) +(compute no. of input-word time) + (using the no. of word control loop time) +(check to see if NAME = END time)

= (32.250 + a) + (2.75) = (35.00 + 8000) = 8035 μsec.

a = average read-in a record name time = (0.5 ms/character) (16 characters) = 8000 μsec.

T_2 = Compliment the first bit of each word in NAME time

= 77.625 μsec.

T_3 = (Average search time in each successive plane)

= $\frac{1}{2}(0 + \log_2 N \times 20.125)$ μsec.

T_4 = Time required for searching each successor

= 7.875 μsec.

T_{SVT} = search time of three dimension tree.

= $T_1 + 4 \cdot T_2 + 4(\frac{1}{2} \log_2 N \times 20.125) + T_4$

= $(8000) + 4(11.625) + 4(\frac{1}{2} \log_2 N \times 20.125 + 7.875$ μsec.

T_{SVT} = $8077.9 + 40.25 \times \log_2 N$ μsec. (9.6)

T_{AVT} = Adding time for variable tree

= (T_{SVT}) + (adding routine time)

= $(T_{SVT}) + 4 \times T_5 + T_8$

T_5 = (Item adding loop time + search for free space loop time)

T_6 = search for free space loop time

= [(UJP** → ENI 0, 1) + (average number of search free space) x (LAD A, 1 → UJP* -4) + (TIA 1 → UTP FREE, time)]

= $(6.875 + \frac{1024}{5.2} \times 77.580 + 8.715)$

= 6.875 + (102.4) x 11.580 + 8.715)

= 1201.382 μsec.

T_7 = Adding item loop time

= 34.525

T_8 = [RTJ INDATA + (UJP** → UJP INDATA) + UJP REDO]

= 2.75 + 102.375 + 1.375

= 106.5 μsec.

T_{AVT} = $(8077.9 + 40.25 \times \log_2 N) + 4(1201.382 + 34.525) + 106.5$

= $13128.028 + 40.25 \log_2 N$ μsec.

T_{DVT} = Deleting time of variable length key name tree

= [Search time - (STA DATA, instruction time) + (ENA,S -1, instruction time) + (STA A + 1, 1 instruction time)

= $[T_{SVT} - 2.75 + 1.375 + 3.215]$

= $8079.75 + 40.25 \log_2 N$ (9.7)

Communication Time

In this simulation assume the user (teletype terminal) is located twenty miles from the computer center. The delay time of the selected communication cable is 1.524 nsec. per foot.

communication time = (20 miles) x (1760 x 3) (1.524 nsec.)

= 160934.44 nsec.

= 0.16093444 msec.

If T_{CBF} = back and forth communication time.

= 0.1609 x 2

= 0.3218 msec. (9.8)

Transfer Time of the Desired Out-Put Record Informations From Core

For all cases of the simulation, the fixed-length logical record 16 computer words (64 characters) is used and presented back to the user during retrieval.

The output of logical record transfer time = 16 x 4 x0.5625 msec.

$$T_{RWOUT} = 36 \text{ msec}$$

User's Operating Cost Per Call

The general formulae of the computation for the user's operating cost per call in our evaluations for all of the tested files can be derived as follows:

1. Disk space required for supporting file system.

If $N_{T(i)}$ = number of tracks on the disk required to support the file and processing program, for each typical file,

N = number of records in the file, file size.

Then $N_{T(i)} = \frac{N}{64}$ + Disk-track required for supporting processing program in case 1 track contains 64 logical records (9.9)

$N_{T(i)} = \frac{N}{63}$ + Disk-track required for supporting processing program in case 1 track contains 63 logical records. (9.10)

2. Disk storage space rental charge per month for user.

Since in this computation the rate of charge for on-line disk storage is \$0.30 per track per month, based on Oregon State University Computer Center's rate, if

R_D/M = amount of charge per month for disk file in \$

then, in general

$$R_D/M = 0.30 \times N_{T(i)} \quad (9.11)$$

3. The charge per month for CPU busy time.

Since the CPU busy time is not only dependent on the CPU access time per record retrieval but also on the frequency of using the file. So the general formulae should be:

If C_{CPU}/M= amount of charge per month for CPU busy time \$

T_{CPU}/R = CPU busy time per record retrieval in seconds as shown in average throughput computing table of all files.

F/hr = number of calls per hour, rate of use

= 250/hr., 1000/hr., 2000/hr are to be considered in these computations.

In case the computer system is operated 7 hours a day, and considering 30 days in a month, the rate of charge of CPU busy time is \$300/hr.

F/M = number of calls per month

= 30 x 7 x (F/hr)

Then, in general

$$C_{CPU}/M = 30 \times 7 \times (F/hr) \times (T_{CPU}/R) \div 3600 \times 300$$

$$= 17.5 \times (F/hr) \times (T_{CPU}/R) \qquad (9.12)$$

C_T/M = total amount of charge per month

C_T/M = amount of charge per month for disk file + amount of charge per month for CPU busy time.

Then, in general

$$C_T/M = 0.30 \times N_{T(i)} + 17.5\ (F/hr)\ (T_{CPU}/R) \qquad (9.13)$$

The result of equation (9.13) is tabulated as a function of call/month and for each file size, are shown in computing table of

customer operating cost per call (unit cost computing table) of each typical file. See details on pages

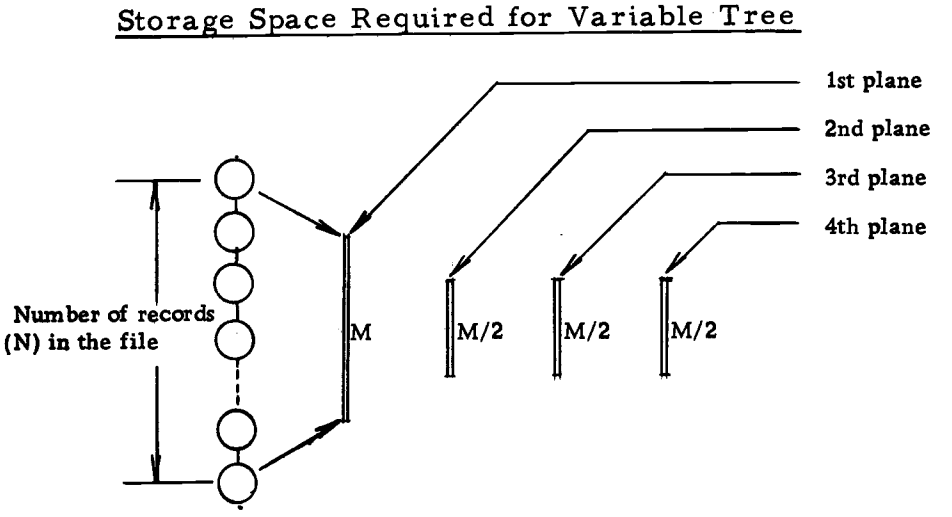


Figure B. 4. Average storage space uses for Variable Tree.

1. If N = number of records in the file

   M = number nodes of different fixed-length key, 1st four characters in the first plane, after trunicated

   $\frac{M}{N}$ = Empirical ratio

which has been measured by Thomas L. Lowe (21)

$$\frac{M}{N} = 0.85 \qquad (9.14)$$

2. The maximum nodes in the 2nd, 3rd, or 4th plane = M

   The minimum nodes in the 2nd, 3rd, or 4th plane = O

   The average of nodes in each plane = $\frac{1}{2}$ (M + O) = M/2

If one node required 4 words.

3. The average storage required for variable length tree, S_{AVLTR}

$$S_{AVLTR} = (0.85N + \frac{0.85N}{2} + \frac{0.85N}{2} + \frac{0.85N}{2}) \times 4$$

$$= 8.5N \text{ words} \qquad (9.15)$$

Selected Sample Key Sets

In this evaluation, 15,805 names of persons from the Corvallis Telephone Directory have been considered as a "parent group", and 4,096 names have been randomly selected from this parent group by use of the standard tables of random digits, A Million Random Digits, by the Rand Corporation (35), as a sample group or "working group".

From now on, the 4,096 selected names are used as the Test-model for evaluation of the system Data File.

Computation of Typical Files

Example 3a - The Unsorted Sequential Disk File

For accessing a random record from a sequential file on a CDC 854 disk storage drive unit, assume that there are N logical records, each 16 computer words (384 bits or 32 bytes) long, to be formatted with keys. The data disk file is read into internal core memory one track (64 records) at a time. The linear search is

NARASIMHAM M N L
MOORE ARCHIE
HODGE D M
PAUL ALEX
KOPPES LARRY
O≠DONNELL DAVID
SMITH A PETE
DATERMAN GARY
YORK BARBARA
SCHECTER LARRY
WAKER BARBARA AN
OMEALY TODD
LAHREN S L JR
WARREN CHARLES E
FOSTER CHRIS
LUTZ JIM
NASSAR ALEXANDER
STOKER DEBORAH
GEORGE ANTHONY J
CLUFF JOHN ROBER
RAMP JEFF
WILLARD EDYTH
RHODS STANLEY C
MCDONALD BILL
SCHEIDEGGER KEN
HANSON C C
PULLEY M A
DILL MARY ROSALI
BARR ABDELSALAM
MORSE CHRUSTY
MACK CHESTER G
HEATH EDWARD H
SWEAT JEAN SISTE
TUCK MARJORY
HAUXWELL GERALD
SOLANDER ROSALYN
MCDANIEL JAMES M
MOSS JAS
GRIFFIN BRIAN
ROBEY S L
RANDALL ARTHUR
HART CHARLA
BYRAM KENNETH
WILLARD H KIRK
WALL JAMES H
HIATT GALE
FRANCE J C
YOUNG ALAN
HELIN WM J
JOHNDOHL GAIG
SEAL EDWARD H

SMITH ALBERT R
NARASIMHAM M N L
ROOD RODNEY
OWERN GEORGIA FAY
SCOTAI THERES E
ESON NELS
SCHMALL RODNEY A
MALONE HOMER C
CORNCCHIA SHERRY
FIREY WILLIAM J
KING ARTHUR
OKBY HUSSEIN
MARTEL DONALD J
WOJCIECHOWSKIE W
NELSON LYOTA
TAM MARILYN
ROBERSON MARY E
SHERBERT JAMES L
STONE LOUIS N
LIND GORDON
OKATA RON
SHUFF FLORENCE
SHUFF FLORENCE L
HANISH C
SHUHART TOM
PHELPS DAVID W
DE HAAS CHARLES
SMITH ALEX
FORRER EUGENE
TOPE LORY
WILSON AL
DEDOLPH ROBERT E
LUTTON DONALD L
HANKE CHRIS
MCGIBNEY MICHELL
SHANAHAN MICHAEL
REGELE DAVID
RICE ALLEERTA L
GRABE DON F
TAYLOR ALICE
SCHAAD DAVID
GOLD JUDY
WOLBERG FLOYD B
ENGE JOHN
HUNT ANITA
MCNURLIN BESSIE
MYER A R
YEE WAIMUN
SAMIN HELEN
FAULHABER JUDY
MULLBOCK MARTA

KANG HENRY
EIDE STUART A
HALE CORA M
MARTENS ANITA M
PROBER DENNIS
OLIVER A W
DIMMICK GREGORY
CHIU DEREK
HARBAUGH DAVE
SNOOK DAVID P
O≠CALLAGHAN DENI
POPE CARLENE
LUND FER E
LAWRENCE BESSIE
PAARMANN BILL
REYNOLDS C W
WALL MARY JANE
JOHNS JUDY
COOK ALFRED J
WOMACK CHARLES W
JONES A L
MOORE ARCHIE M
LEHNERT H M
STUART CHARLES E
OTTUM MARGARRET
JOYCE BERNICE
MCNIVEN SCOTT
GATHERCOAL FORES
HAMM PEG
WOOD BENJAMIN W
TRUMBO JOHN A
TURNNELL BLAINE
HEINDL A L
TILLEMAN PAUL
LOWRIE A C
POOLE A R
POND KEITH E
KAVANAGH ROSS
COOPER A L
HIGGINS ARTHUR
HOLCOMB G W
COATS LELAND F J
JOHNS TOM
VARS R CHARLES
OVER TOM
HOWARD ART
NORED FRANCIS
LAMB F M
LUEBBERT EDWIN G
MYER BRUCE
NOREN CHRISTINE

HOSTETLER DONAL
CORWIN GERALD L
SCICK LAWRENCE
WADE BERTHA
GLASER MARGARET
HARVEY ELIZABETH
CAHILL DON
JOHANSEN DANIEL
KESSEL JUDY
MICHAEL NEAL D
CONVERSE PAUL T
RAYMOND A F
HAITH A E
HULTBERG H D
STROBEKATHY
STROBEL KATHY
PHELPS DOUG
HERY BARBARA
EDLUND LANCE E
PEDEN MYRNA
ALLEN ARTHUR H
GWINNER DONALD B
LOWE ARTHUR L
WASHBURN F E
WILSON BONNIE
SHELDON MICHAEL
MONSON GIN
JUNG PHILLIP
GARRARD JAS L
HEATH ELIZABETH
CALL JOHN
MUCKEY BEVERLY
MCCLAFLIN JOHN
SORENSEN ANN
LIBBRECHT MARIAN
DERY RUTH
REVELS ALIC L
CUMMING STELLA
HOPER B R
RAINERT LINDA
BAILEY BOB
STOVALL FRANK R
WILLBERGER GALE
HALL BILL
BRETHANER MARSHA
FESSEL WILLIAM C
CROVER STEPHEN E
HALL PENNY
MOORE BEN A
MIDDAGH JAMES E
COSLER DENNIS

REYNOLDS CHARLEN
KLIEWER JEAN
LIND HELEN
GRIESSE CHERYL
LIND RICHARD C
CRAIG A W
HAIGHT BETSY
WARREN DAVID R
COUPER ALAN
LIND RONALD E
KANTOR JOSEPH
THARP CHARLEY
MCGRATH B W
LEPPER CHARLES
HER LEV
OBERG KALERVO
WIKMAN CARL
ELIASON DON
WOLF DAVE L
MASSEY MARY
JOHNSON ALBERT W
KENNEDY CHARLES
KILMARTIN JIM
JOHNSON ANDREW A
MORGAN ARTHUR
HAAK BRUCE A
EISEMAN DAVID
ROTH BEN A
KAUFFMAN A J
KALDAHL NORMAN E
WILSON BRUCE A
MISKOWIEC C V
LIBBY HARLEY
CROSLEY BELTY
EVES S DAVID
KATO DIANE
LEHR EDGAR I
OAKS DENNIS E
CRAWFORD CAMERON
ENDERSON FRANK J
PAYNE BOB
PASSMORE LARRY W
DORN HAROLD
TILLERY JERRY O
TSUDA ALBERT H
ENLOWS HAROLD E
NUNNELLEY LEWIS
MARTIN ARNOLD
BAKER DAN A
HUECKMAN CAROL
HITCHCOCK P A

MASON C R
NUGENT DAVID D
TODD ALLAN
HSIUNG KOU YING
KAUSCHE LOUISE
GARLAND DENVIS
MACK GUY
FITZGERALD CHRIS
SEARCY JAMES T
HAY DAN
MINER GARY
EMBANKS ROYCE
PRICE B O
LAMB N J
SIPP EDWARD M
MCALISTER JAMES
TEST CHERYL
LAWRENCE FRANCIS
SCOTT ALLEN B
PATANA MERRIT
KRAUS HOWARD G
BRANT JOHN L
HUGHES ARTHUR D
CRAWFORD CHARLES
VARADY CLARA
LEE LARRY J
SOUTEN DAVID R
HUFF JIM
WEBB ALAN
SHUDY MICHAEL
JOHNSON ARTHUR C
MOBERG MARCIA
MARK DIANA
HEYER ALBERT
HAMILL BILL
PERIN C A
GRETZ HAROLD
HORTON HAZEL
STELLA AL
MARIMAN DAVID
MCMULLEN EARL F
TROSETH STEVE
MERCER BARBARA
WILSON BURTON
MORE GARY KIELY
WILLBERGER GREG
CHEN JENNY
WILL PEGGY JO
EDWARDS A HURCIE
WEST DENNIS E
FOYLE JO ELLEN

WILLE CHRISTOPHE
GATTMAN LEWIS
MOORE BERNARD J
MILBRATH JOHN A
MOYLE B L
GRIFFIN CYNTHIA
SOLTE ERNEST P
HELM CRICKET
SORENSEN CLIFFOR
CHRISMAN LEON D
PAHRE R E
FOSTER D L
DOTY DELLA
SWAN ANDREW
HILL A S
WASHLEURN HERBER
LEHRMAN J K
MASS SALLY
KAMPFER LAWRENCE
HALSE E A
GRISCHKOWSKY OSC
EISENBRANDT EDIT
WALLACE ARDEN H
EVANS BRIAN F
WUNDERLICH MARTH
SEWELL HAROLD RA
ROCK JOHN
WORKINGER MAY
CRAMER BRAD
STEIGER JACK
JOSI TIMOTHY
VOSS CHARLES R
HAMADA SPENCER
LACEWELL DAVID A
MEDLEY FLORENCE
JONSON ARTHUR P
HILL CHRIS
TESTER JOHN
LACH JOHN
HOOD ALPHA F
LYNCH ANNA
FRYE DANIEL E
TERREL MARK H
SAGER CHARLES E
WILKEN CALVIN
DOUGHERTY KENNET
CHARLES ARTHUR
WIEBE MIKE
HABERMAN J R
MCQUEEN MIKE

DOND PHIL
HUNDLEY GARY
MEDANIEL L W
RUDD MARIAM M
CULL APUL
LUSTO H E
RICE ARCHIE H
HAYES C W
LOTT BOB
VALLEY IRENE
VOGEL ALLAN
EVANS DARLENE
CRUDELE ANTHONY
HATHAWAY EARL
ROLOW W J
HAYDEN A A
OLSON ARTHUR E
SMITH ALEXANDER
HUNT DEBBIE
CARLILE CAROL
ROLLER S P
MCGUINESS ERNA
MILLARD ALLEN L
O/CONNELL K
HAWES HAROLD
MCKIBBAN ROBERT
LANE CHRISTOPHER
NESS D M
HALLENBECK VERNO
JOSSIS ROBERT G
ALLEN BARBARA
PAINE HOWARD
MASON DALE F
STUBBERT DAVID
VENDETTE ALLAN
JENKIN CLIFFORD
HASSOUN HUSSEIN
NANSON ARTHUR
GARNER CARY A
PERREARD GEORGE
ROSE BOB
MURRAY AILUN
MATTHEWS JOHN
NOLL CARROLL
PRATER LAUREN
CUTTING GEORGE
DEAN CHARLES
YENNE HERBERT
STELZER MILTON J
KERTH D H

EWART ROBERT B
KENNEDY D C
MEIER GERALD A
GANNON SUE
SCHUDEL DAVID
SWAN R O
WOLHOWE HANS
ARMSTRONG BOB
DYSERT NELL
SLEGEL DAVID L
HOUSE CRAIG
HARRELL EVERETT
HUGHES BUDDY LEE
HENDERER CHARLES
SHEEHY HUGH F
OELKE RAYMOND A
O/DONNELL JOHN J
COUCH DICK
KING BERTHA
MEYER BARBARA
HOTCHKISS RON
GREEY EWARD A
SCULLEN HERMAN A
WILSON CARL C
SNYDER D L
HAMILTON DWIGHT
JOHNSON BERTHA
WELL LARRY W
REIMAN BILL
COX JOE D
YODER MAX
LECKIE MICHAEL
PUCKETT BILL
HUSTED FRANK
KALBERER KATHLEE
SMITH BEVERLEY
GRIM DON
HOPPE JIM
VANDECOEVERING J
TRIPP G R
KULM LAVERNE
WHYLER RICHARD
LARSE LLOYD
RAO P S
YOSS JAMES K
LINDAHL DONALD G
TITMAN JOHN E
HANEL MICHAEL P
RECTOR MAVVIN C
DOHERTY DAVID T

STRODE LELAND L
KELLAR DEBBIE
HORN DON
FORD DOROTHEA
CISAR JOHN O
COPELAND BARB
TOWNE TOM
LARSE ROBERT W
JAFARI JEFF
DURHAM JOANNE
HENNEBRY H M
CRUSON JON JAY
MOSEMANN JAMES
PEDERSEN D I
LESTER L JUSTIN
POTTER A W
REYNOLDS DENNIE
VAIA SCOTT F
STREEBY LARRY
HAGOOD NANCY
MCWADE J H
HAMM RON W
GALLAGAR JILL
MILLARD MIKE
LAWRENCE GILBERT
ROBIDART GABRIEL
MILLEMANN R E
MUMPER JERRY
WOOLERY LAWRENCE
GOSHU CARL
SMITH CAROL
LEY YVONNE
ODOM DEBORAH
HANSON DEAN
VETICH JAY
RYAN ANNE
HAWK DIANE
STONE KATHU
WHEELER DENNIS
CURREY DEAN L
CARR BRUCE F
THRELKELD CURT
PHELPS HAROLD E
KNAPP DAVID
SAITO AL
LARSELL DAN
LENAER MIKE
JOLMA ROGER E
BROOKES VICTOR
KALK PETER A
PECK DENNIE

DRAPER JOHN W
ROZENDAL PER H
DALRYMPLE GARY
ENGEL JOHN HOWAR
STOCK DAVID E
PETER JAMES R
IVERS ALVIN
JACKS CLINTON
LUBIN J M
LEHMAN HAL W
LUCK JOHN
KOBLITZ GORDON F
KING CHARLES S
TOWNER HERBERT B
PURDOM E E
ARNESEN MICH
SUHR GENE
SUMMERS DANIEL
RICH ANDY
HAY ROBERT
PRICE BARBARA
OLSON ARTHUR H
WICK RON
GALLAGHER DONALD
CHADWICK E B
LISS BILL
FESSENDEN PETER
GOODE D M
MARCH GUY
GOWAN ENID L
LOCKE NESON
URBAN BOB
OLSEN ANTHONY
TERHAAR JOHN C
SWEARENGIN E B
PETERKORI JACK
HORTON HOWARD F
SECHER ARNOLD
GOLDBAUM FRANCES
ORDEMAN D T
HALTER A N
PAQUIN JAMES E
O≠CONNER KATHLUN
HILL CYNTHIA
SMITH BOG
GRAY ALDEN K
WEBER ALISON L
MCAULIFFE DEBBIE
SCHNEBLY WILLIAM
KELLER CHARLES
HAGEL PATRICK

WILBORN JACK G
HULL DANIEL F
KAKIMOTO
SACKETT HARRY A
DALGAS C M
ROBLEY ASA A
WILT ALMA
STANGE STEVEN
ROSS ARTHUR M
HISE FRANK
HEALD JAMES R
RICKABAUGH U S
DALRYMPLE LINDA
PAPRATER ROY
GROSHART JODY
SCHMALTZ JAMES N
SCHUDEL STEVE
WALSH JAMES W E
ASHBY MERLA
EGHBALT HASSAN
CRAWFORD GEORGE
CUTHBERT ANN
JOHNSON BILLY F
SIGMA CHI
SIDLES KARES A
PHILBIN JANET
KIRBY DAVID N
HOERLING DERALD
KESTLER HULDA
KLIPPEL E A
MEAD TOM
FIEBER LESLIE E
FINSETH L C
CONVERSE RICHARD
HOLT ELEANOR
SETO FELIX
MILLER BRUCE H
SPELBRINK ROBERT
YUNG TONY
HALL BOB
MENDEL W C
HAWLEY CRYSTAL
FOUNTAIN S L
PRIC CECIL K
NORD PHILLIP M
KAPLAN E L
KENDALL JOHN
DEATHERAGE DENNI
LOOMUS BOB
HODGE WILLIAM J
SHAW C G

LAWRENCE H SAM
JONES ADORA A
JOHNSON BOB
PETERS ALLEN R
STROEMPLE JAN
DUNN CHARLES
JOHNSON BRAD
WELLER L J
FINE DARWIN
JORDAN ANNA
ANDELL ROBERT
RASMUSSEN MARK
HAYES CLARK
SHUM EDDY
NEVILLE ARTHUR S
HATHAWAY ELMER C
CARR DONALD F
SCHMALTZ MARGARE
BOLAND DALE H
SCHURMAN DONALD
LEAPTROTT JOHN
LEE ALAN DALE
KEELER ADELLE H
KUYKENDALL A L
CULLEN GARY
GUNTHER EMIL B
PHIPPS KEN
TAYLOR C EDWARD
RICH JACK
LUND FERN E
CROWE HUGH L
STARK CECIL M
WOLSKE DAVID
RIVERA D
ROESER MARION A
REDSHAW JAMES
KENT WAYDE
COWAN BETSTY
ANDERER DAVID A
MCCLAIN DICK
MCKILLIP GREG
TORHEIM ROBERT E
TICE EDWARD
BEHRENS ROBERTS
MOE RON
WALLACE BARRY J
TERHAAR RON
COPELAND CHAMP
STARTEN MIKE
DAY JERRY
KIENLE CLARENCE

GENTZ KEITH R
GOSS LARRY D
CHART NONA L
SCHEIDLER DAVE
CLEM C S
THOMA GALE L
DOGGET TOM M
RICH LARRY L
WHITE ANNA N
BARBER CARL
QUINLAN DAVID C
WILLE JERALD
POST LULU
REESE H DARWIN
JOHNSON CARL
LITTLE B R
MULLEN HAZEL
OWEN JOYCE S
MINER HELEN
KRAUSE D G
NEIL FORREST A
MARTIN BOB
HIATT PETER
DORT JAMES B
FRENCH J R
UNGER STEVE
WALES J H
STEVENS ANN
YOUNG B DIANE
LEE JAMES K
SKINNER D J
DECKER FRED
TURNBULL
AMERICAN LEGION
GRANDATAFF MAURI
HARDEN IRVING
PAYGR STEVE
ROGERS CHARLES W
HOLDEN ARNOLD G
GAROIAN LEON
HATHAWAY JAMES C
GILSON JOHN A
JACOB PHILIP
SPEES EARL
DOUGLAS DAVETTE
TAYLOR CARL BEN
PETERS BARBARA
MOGAN DAVID
GARCIA EDMUNDO
MCDONALD D LYNN
ROGERS D L

MILLER BRUCE H
FISHER BILL
ANDERSON JACK R
ELDRED CAROLYN
LUCKER CATHERINE
ENGLAND DAVID C
HASHITANT GLENN
HOYT HAROLD P
GULAN MICHAEL P
RATH ALBERT C
MATHEWS LAMONT
EMIGH ANDREY L
RUKKE DIANE
LEONARD RUTH E
VANGENT COR
STOCHR CAROL
ROBERSON MICHAEL
ZUR WILLIAM
O≠ROURKE CHARLES
STAVE CHARLES
RILEY ADELLA
MCCOLD LANCE
OKANO BOB
LONG CARL
YOST GAROLD
ROBINSON ALAN H
WATENPAUGH FRANK
SMITH DEBRA MARI
HAAS KERRY
TOY MARY THOMPSO
NOBLE DONALD
RODLEND RICK
OLIVER AVERY W
ENDICOTT STEVE
LOOP DAVID A
CARPENTER CARL
WEIS DORA
HOLDEN CHRISTINE
MCALISTER WANDA
HAMLIN LOUIS W
SCHOPPERT KENTON
PETRIE DAVID
HOSTETTER I M
RHONE W T
REINERT DAVID
DEEGAN DON
RHODEN J
HAMMACK DICK
RICHARDS CLYDE G
URE ROBERT VAN
TETZ DENNIS

HAWKES STEPHEN J
BENNETT C V
EDWARDS ANNE
HARDAGE ROGER
NICHOLS DARLA
COMVAY MARK
MCCLANAHAN RALPH
STEINBRUGGE DICK
KOTUO PEARL
EDELSON JOAN
BROOBECK RALPH J
ROBERT ALOYTH
DUDLEY STEVEN R
WILSON DENNIS L
GRAIG CHARLES
PARK DAN
PUGSLEY DAVID W
WHITE CLARENCE R
REID RICHARD
STEAGALL MARY
WILSON GARALD R
SMITH EWARD
VANEIKEN HANS
DALRYMPLE W
HENDRICKS JEANNI
LANKFORD DONALD
LYMAN RICHARD E
GARREN RALPH
MCKINLEY STEVEN
WIESE FRED
IWANAGA PAUL M
STUFFLEBEAM EUGE
KNEHTA THOMAS J
WALTER AUSTIN F
POMEROY LYLE J
MINGLE J G
BRYAN D F
MCMACKIN TERRY
TRIPP RANDY
STONE ARCHIE
HOLM BIRDIE
LONG DAVID L
DUBOIS MAY
RICHARDS FRANK B
ECKMANN BARBARA
GRAHAM COYNE
FERNANDEZ ENRIGV
HORN GREGORY
JACKSON BILL
ROYCE ROBERT A
WESSBECHER HOWAR

MORGAN DON E
CRUM CONNIE
EDDY HELGA
HILL DALE
COSBY H B
THOMAS QUEBIIN V
BYERS BILL
MARTIN CONNIE
SPINK THOMAS J
WEGNER ALBERT
KEASEY GILMAN
CUNNINGHAM DONNA
WHITE FRANCIS
FROTHINGHAM PHYL
YORK GEORGE
CRAVEN RICHARD
WILSON HOWARD L
HEISE JOHN
THOMAS SUZANNE
MAIER ROGER
MCNAIR ALFRED B
SIEGMUND WALT
TRICE DENICE ANN
SHIOSHI SARA
HAMBLIN DON
LANG JOHN L
REIMAN LYNN E
ELLIS BEN
WARNATH CHARLES
EBERHARDT CLIFFO
TOBABEN CLARENCE
STUFFELBEEN CHAR
GARLAND JAMES E
FONATAN PETER R
PASLEY FLOY
HEINE DELORES
HARRICK CYNDY
MACK HARRY J
HENDRICKSON DAVE
ENYART CAROL
CRUM GARLAND
PATTERSON ALICE
JONES RICHARD A
MYBENGA DANIEL E
KANOUSE DANIEL
LIBBY RICHARD
ECCLES TERRY L
MICHAEL ROBERT
PUN LINDA
VANDCHEY JAMES A
JORDAN GUY W

LEELAND ALBERT L
ROBERTS CLARENCE
PARSONS JEANETTE
MOOTHART GORDON
RICHTER EDWARD W
DEWEY GEORGE W
TODD GEORGE F
GREENWOOD RICHAR
NELSON DULCIE
WALLACE H WAYNE
FLEMING DOROTHY
IRWIN ROGER
FOOS DAVID
FOWLER GERALD A
WHITAKER CONNIE
HARMAN ALBERT L
BROWN BETTY
BARNEBURG BRENT
HOWARD BONNIE O
MILLIKEN MARGARE
SIHTO GEORGE
TURNER LAWRENCE
GATE ROBERT D
COOPER KENNETH G
KENNEDY KATHY
MASER K
KING JOHN PHILLI
LYDA JOHN H
STADSVOLD CYARCH
GODARD RUSSELL H
JOYCE JAMES H
CHRISTENSEN BENN
CROBOK EDNA M
HISAKA CAROL
REEVE DANIEL G
WILLEY DALE H
HOLLAMON RICHARD
LUSKY CHUCK
ELLIKER ANNE
MORGAN SALLU
NELSON MILTON
WEST GRANTON
DAVID FORBES
PARMENTER ROBERT
PHILIPPI SANDRA
FOWLER MARILYN
BUNTING JAMES
HARRISON BEN
ROESER THOMAS
DEDEURWAERDER CH
WALETICH MARK

MOON CARROLL C
DEARBORN RICHARD
WARE BARBARA
FELL ROSEMARY
WALSH NANCY
SCOVILLE JACK A
OLLEMAN ROGER D
GITHENS JENNIFER
JAGER DUANE
KHASHOGJI EMAD
SINNARD H R
KATTER V
KNUTH LINDA
RHYNARD WAYNE E
HEATH LARRY
HOFFMAN DATE L
GUY GREG
HENDRIX MIKE
LINDBERG MARVIN
PLYER DOUGLAS C
BUCK DARRELL K
EMERICK DONALD J
DAVISON EUGENE V
DUNN ELLEN
MECHAN SIDNEY
OSBORN FRED P
WILCOX R C
LITCHFIELD A
WHEELER TERRY
BOYD FREDDIE
AUSTIN CYRUS W
GRAHAM CRAWFORD
GILBERT MIKE
STOKES DONALD B
FERRAN FRANCISCO
BURGE ANNA
WOOD EDWARD
HIGBEE BRIAN
SMITH FREDERICK J
PERKINS H A
PANDEY N N
FATO GABRIEL
WUSTRACK MIKE
VIMMERSTEDT JO
DEDRICK M C
GARG HARE PRASAD
BUNKER ALBERTA
NELSON WILLIAM R
ZENOR DANNY
CRAIG ROBERT V
GRIFFIN STEWART
WOLF DONALD S
WRIGHT GEORGE C
IRONS L M
KLINE DOUGLAS J
JANS FRED C
RYDELL ROBERT A
MATTHEWS MILDRED
BUSBY HAROLD L
GRUBB STEVE
SCHMID RICHARD A
HANSEN KENNEDY
NG DINA
WALKER RALPH DEA
OVERACKER CLAIRE
JOYNER W B
FLYNN JOAN
DONOHUE PAT
GOULD BEULAH
SMOUSE CHERILYN
WONG PATRICK
TABOR CLAYTON
HANT DONALD
NESBITT CANDICE
LEWIS ANNIE
CRUSE HOWARD
SMITH HAROLD E
NUNLEY M KIRBY
SCHWANKE WALTER
WATSON CHARLES R
HORNER GAMMY
ZOBEL DONALD B
PIECE KENNETH
WILLIAMS CLARENC
ROBINSO RAYMOND
RICE KENNETH L
FRY LOYD L
GULLEDGE MARY
FREEMAN GEORGE F
LEHAMAN GEROLD O
HOY MAE R
MC NUTT JIM
FIREY WILLIAM J
MERRICK STEPHEN
HEPNER RALPH
DOLBY MICHAEL W
KUROVOSKY DENNIS
SANDER GARY H
SEMONES RONALD R
RONNING MARYLEE
PETERS EMERSON C
GUERBER J RICHAR
SCHOTH H A
FAMILY BILLIARDS
NOVAK RAYMOND P
SAWYER REBECCA
HOPSON CHERLYN
NELSON KERMIT

performed in internal core memory. Compute the average throughput time per record retrieval in random sequence; assume the user terminal is 20 miles away from CPU. The concept of computations are illustrated in Figures B. 5, B. 6, and B. 7 on pages 228, 229, 230, and respectively.

1. Computation of the number of required tracks and required cylinders

   Number of required tracks = i

   Number of required cylinders = Z.

   Where i and Z are the smallest integer such that

   $$i \geqslant \frac{N}{64} \quad \text{and} \quad Z \geqslant \frac{N}{640}$$

   Where numbers of records per track = 64

   Number of tracks per cylinder, D854 = 10

   Number of records per cylinder = 640

2. Computation of the average accessing time of a random record from a sequential disk file. The concept of using average cylinder, average track is considered. See details in Figure B. 5 on page

   Since average number of required looking-up records $= \frac{N + 1}{2}$

   Average number of required cylinders = Za

Where Za is an integer such that $Za \geqslant \frac{(N + 1)}{2 \times 640}$

Then the records contained in the last average cylinder = $[\frac{N + 1}{2} - 640 (Za - 1)]$ and the number of looking-up tracks in the last average cylinder, t_{LLAC}

Where t_{LLAC} is the smallest integer such that $t_{LLAC} \geqslant \frac{1}{64} [\frac{N + 1}{2} - 640 (Za - 1)]$

The number of looking-up records in the last track in last average cylinder, N_{ALT}

$$N_{ALT} = [\frac{(N + 1)}{2} - 640 (Za - 1) - 64 (t_{LLAC} - 1)]$$

a) Total cylinder to cylinder positioning time (t_{TCTCP})

t_{TCTCP} = 30 ms x (number of full cylinders)

= 30 ms (Za - 1)

b) Total track average waiting time (t_{TAWT})

t_{TAWT} = 12.5 x number of average full cylinders + (12.5, initial waiting time) ms

= 12.5 ms (Za - 1) + 12.5 ms

c) Total previous tracks read-in time (t_{PTRIN})

t_{PTRIN} = (a track read-in time + one disk r.p.m. waiting time) x (number of full tracks used in file)

= (25.00 + 25.00) ms x (number of average tracks in the file)

= 50 ms x (number of average full tracks in the file)

= 50 ms (t_{LLAC} -1)

d) Average search CPU busy time per random record retrieval for the main file ($t_{CPUFSQF}$)

$t_{CPUFSQF}$ = (fault-loop linear search time) x (number of average full tracks) + [(fault loop linear search time) (N_{ALT} -1) + (correct loop linear search time) (1, a desired record)]

where N_{ALT} = Average number of records in last average track.

N_{AFT} = Average number of full tracks = (t_{LLAC} -1)

$t_{CPUFSQF}$ = [5.5 + 7.875 x 64) x N_{AFT} + [7.875 N_{ALT} -1) + 165.625 (1)] 9] x 10^{-3} ms

= [509.4 N_{AFT} + [7.875 (N_{ALT} -1) + 165.625]] x 10^{-3} ms

e) Average search time per random retrieval for desired track (T_{ASFSQF})

T_{ASFSQF} = (Fault loop linear search time) (N_{ALT}-1)

correct loop search time (1, desired record)

$$= [7.875 (N_{ALT}-1) + 165.625] \times 10^{-3} \text{ ms}$$

f) Average disk access time per random record retrieval ($T_{AVACFSQF}$)

$$T_{AVACFSQF} = (\text{initial positioning time}) + t_{TCTCP} + t_{TAWT} + t_{PTRIN} + T_{RINDT} + T_{ASFSQF}$$

$$= 95 \text{ ms} + 30 \text{ ms } (Z_a - 1) + 12.5 \text{ ms } (Z_a - 1) + 12.5 \text{ ms} + 50 \text{ ms } (t_{LLAC} - 1) + 25 \text{ ms} + [7.875 (N_{ALT} - 1) + 165.625] \times 10^{-3} \text{ ms}$$

$$= 107.5 \text{ ms} + 42.5 \text{ ms } (Z_a - 1) + 50 \text{ ms } (t_{LLAC} - 1) + 25 \text{ ms} + [7.875 (N_{ALT} - 1) + 165.625] \times 10^{-3} \text{ ms} \qquad (9.16)$$

Illustration computation for N = 1024 records in the file

$$Z_a \geqslant \frac{(1024 + 1)}{2 \times 640} = 1$$

$$t_{LLAC} \geqslant \frac{1}{64} \left[\frac{N + 1}{2} - 640 (Z_a - 1)\right] = \frac{1}{64} \left[\frac{1024 + 1}{2} - 640 (1-1)\right]$$

$$= 9 \text{ tracks}$$

$$N_{AFT} = (t_{LLAC} - 1) = 8 \text{ full tracks}$$

$$N_{ALT} = \frac{(N + 1)}{2} - 640 (Z_a - 1) - 64 (t_{LLAC} - 1)$$

$$= 513 - 0 - 512 = 1$$

$$T_{AVACFSQF} = 707.5 \text{ ms} + 42.5 \text{ ms } (1-1) + 50 \text{ ms } (9-1) + 25 \text{ ms} + [7.875\ (1-1) + 165.625] \times 10^{-3} \text{ ms}$$

$$= 107.5 \text{ ms} + 400 \text{ ms} + 25 \text{ ms} + 0.166 \text{ ms}$$

$$= 532.666 \text{ ms}$$

The results of computation of $T_{AVACFSQF}$, Equation (9.16) are shown in Table B.1, page 231.

3. Computation of the average throughput time per record retrieval of unsorted sequential disk file:

   a) Average throughput time per random record retrieval of unsorted sequential disk file $T_{ATHRPSQUN}$

   From the time diagram on page 2 the following equation can be set up:

$$T_{ATHRPSQUN} = T_{CBF} + T_{CFNTK} + T_{RCLU} + T_{AVACFSQF} + T_{TCI/O} + T_{RWOUT} \qquad (9.17)$$

$$= 0.3218 + 8.515 + 0.008 + 532.666 + 0.008 + 36.00 \text{ ms}$$

$$= 577.159 \text{ ms for N = 1024 records.}$$

The results of the computation of $T_{ATHRPSQUN}$, Equation (9.17) as the function of file size are shown in Table B.2 on page 232.

b) Approximate formula of $T_{ATHRPUSQF} = T_1$

Only the major components of search time (msec) are considered; the minor components of search time (μsec) are omitted.

Since $T_{AVACFUSQF}$ = (initial positioning time) + (cylinder search time) + (track read time) + (read-in the desired track)

and $44.8878 \text{ ms} = T_{CBF} + T_{CFNTFN} + T_{RCLU} + T_{RCI/O} + T_{RWOUT}$

then $T_1 \simeq T_{AVACFUSQF} + 44.8878 \text{ ms}$

$$\simeq 107.5\,(1) + 30\left[\frac{N+1}{2 \times 630} - 1\right] + 50\left[\frac{N+1}{2 \times 630} - 1\right] + 259) + 44.8878$$

$$T_1 \simeq 0.635\,N + 97.453 \text{ msec.}$$

4. Computation of the average CPU busy time per record retrieval fro unsorted sequential disk file using both unique fixed-length key, and full name of a record in accessing.

a) Total CPU busy time per record accessed using unique fixed-length key ($t_{TCPUTSQFUN}$)

Since $t_{CPUTSQFUN} = [509.4\,N_{AFT} + [7.875\,(N_{ALT} - 1) + 165.625]] \times 10^{-3}$ ms

$$= [509.4 \times 8 + [7.875\,(1 = 1) + 165.625]] \times 10^{-3}$$

$$= 4.242 \text{ ms} \quad \text{For N = 1024 records}$$

$$\text{then } t_{TCPUTSQFUN} = T_{RCUL} + t_{CPUTSQFUN} + T_{RCI/O} + T_{RWOUT} \qquad (9.18)$$

$$= 0.008 + 4.242 + 0.008 + 36.00$$

$$= 40.258 \text{ ms} \quad \text{For N = 1024 records.}$$

b) Total CPU busy time per record access with using full name of a record ($t_{TCPUSQFREUN}$)

$$\text{Since } t_{TCPUTSQFRUN} = T_{CENTFK} + F_{RCUL} + t_{CPUTSQFUN} + T_{RCI/O} + T_{RWOUT} \qquad (9.19)$$

$$= 8.515 + 0.008 + 4.242 + 0.008 + 36.00$$

$$= 48.773 \text{ ms} \quad \text{For N = 1024 records}$$

Equation (9.18) and (9.19) as a function of file size are computed as shown in Table B.2 columns 9 and 10 respectively.

5. Computation of achievable-throughput-rate capability of unsorted sequential disk file.

a) Achievable-throughput-rate capability of unsorted file, C_1

can be computed by using the general formula which has been mentioned on page 141. Equation (7.1) can be rewritten. Throughput-rate capability of unsorted sequential disk file, C_1, (calls per hr.)

$$= 3600 \div [\text{average throughput time per call (sec), } T_{(1)}] \qquad (9.20)$$

or $$C_1 = \frac{3600}{T_{(1)}} \quad \text{where } T_{(1)} = T_{ATHRPSQUN}$$

For illustration, from Table B.2, $T_{(1)} = 577.519$ ms = 0.577519 sec. for file loading factor, $\alpha = 0.0625$ or N = 1024 record in the file. The file system using the full name of record in accessing.

$$C_1 = \frac{3600}{0.577519} \simeq 6234 \text{ calls per hr.}$$

The results of calculation of Equation (9.20), Achievable-throughput rate-capability of each typical file organization method using both the full name of record in accessing and the unique fixed-length key can be computed in the same manner as shown above. See results of computation in Table 7.3, page 152 and Table 7.4, page 155.

6. Computation of customer's operating per call (unit cost) of unsorted sequential disk file.

   a) Customer operating cost per call (unit cost) is based on rental cost as the objective of the evaluation in this thesis. By application of the general formula, Equation (9.21) can be rewritten as follows:

Customer operating cost/call of unsorted sequential file,

$$U_{C(1)} = \frac{[T_{(i)} R_{cM} (300) + N_{T_{(i)}} (0.3)] \times 100}{R_{cM}} \qquad (9.21)$$

by using the equation (9.13); page 209, then

$$U_{C(1)} = \frac{[17.5 \text{ (F/hr)} (T_{CPU}/R) + N_{T_{(i)}} (0.3)] \times 100}{R_{cM}} \qquad (1.5)$$

For illustration, from Table B.2, page 232, $T_{(i)}$ = 0.48773 sec, from Table B.4 page 234, Disk required space, $N_{T_{(i)}}$ = 24.755 tracks. For N = 1024 or α = 0.0625. The selected rate of use is 250 calls/hr (R_{cM} = 52500 call/month); 1000 calls/hr, (R_{cM} = 210000 calls per month), 2000 calls/hr, (R_{cM} = 420000 calls/month). Then

$$U_{C(1)} = \frac{[17.5 (250) (0.048773) + (24.755 \times 0.3)] \times 100}{(52500)}$$

= 1.86880 cent per call.

The results of computation of unit cost of each typical file organization method using the full name of record in accessing can be computed in the same manner as shown above. The results of computation are tabulated and compared in Table 7.6, page 173. The graphical comparison of the results is shown in Figures 7.13 - 7.16 pages 169 - 172 respectively.

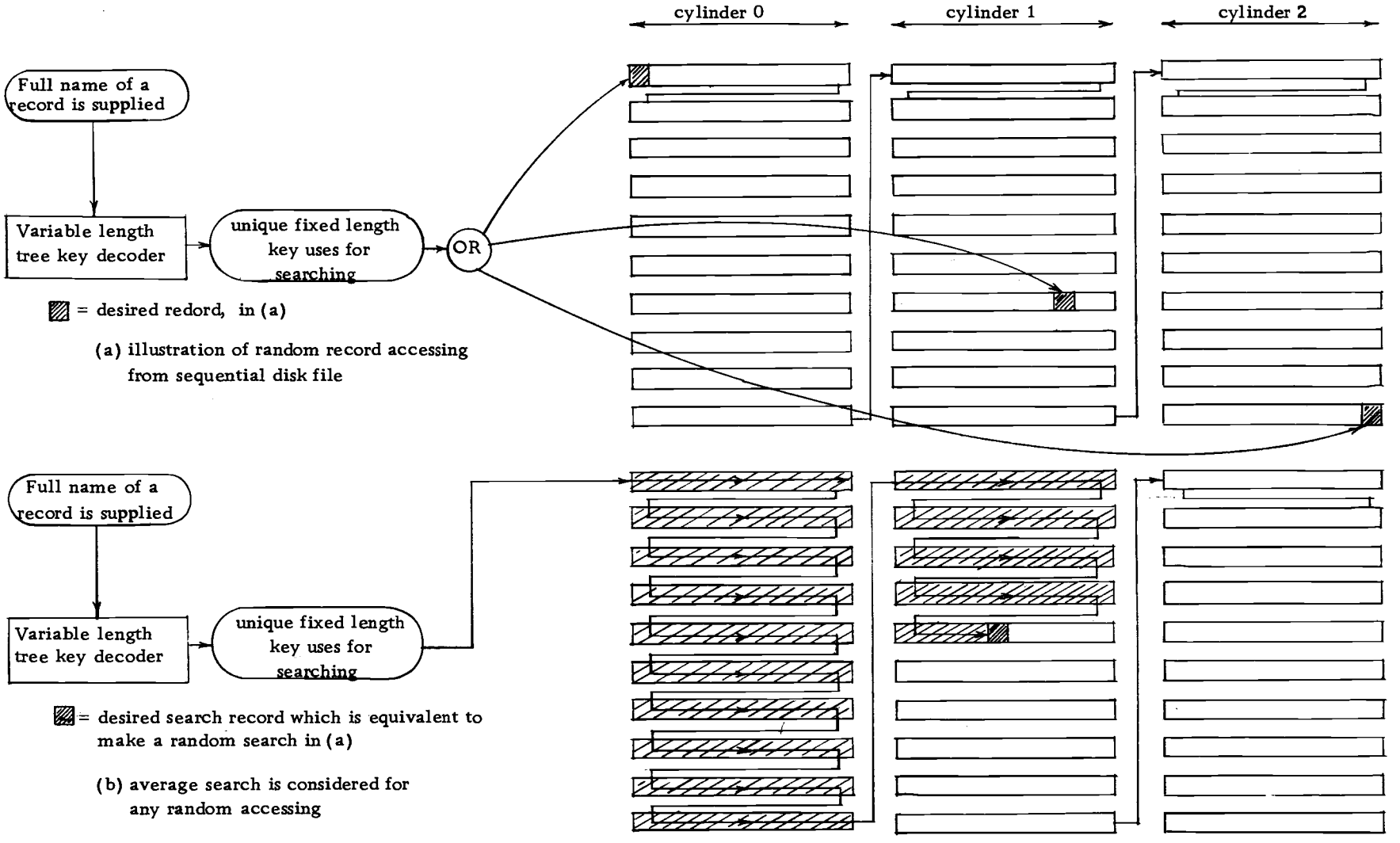


Figure B .5. Random access a record in a sequential disk file in (a) is equivalent to average access of a record in (b). Average looking-up records in the file: average cylinders, average tracks have to be considered.

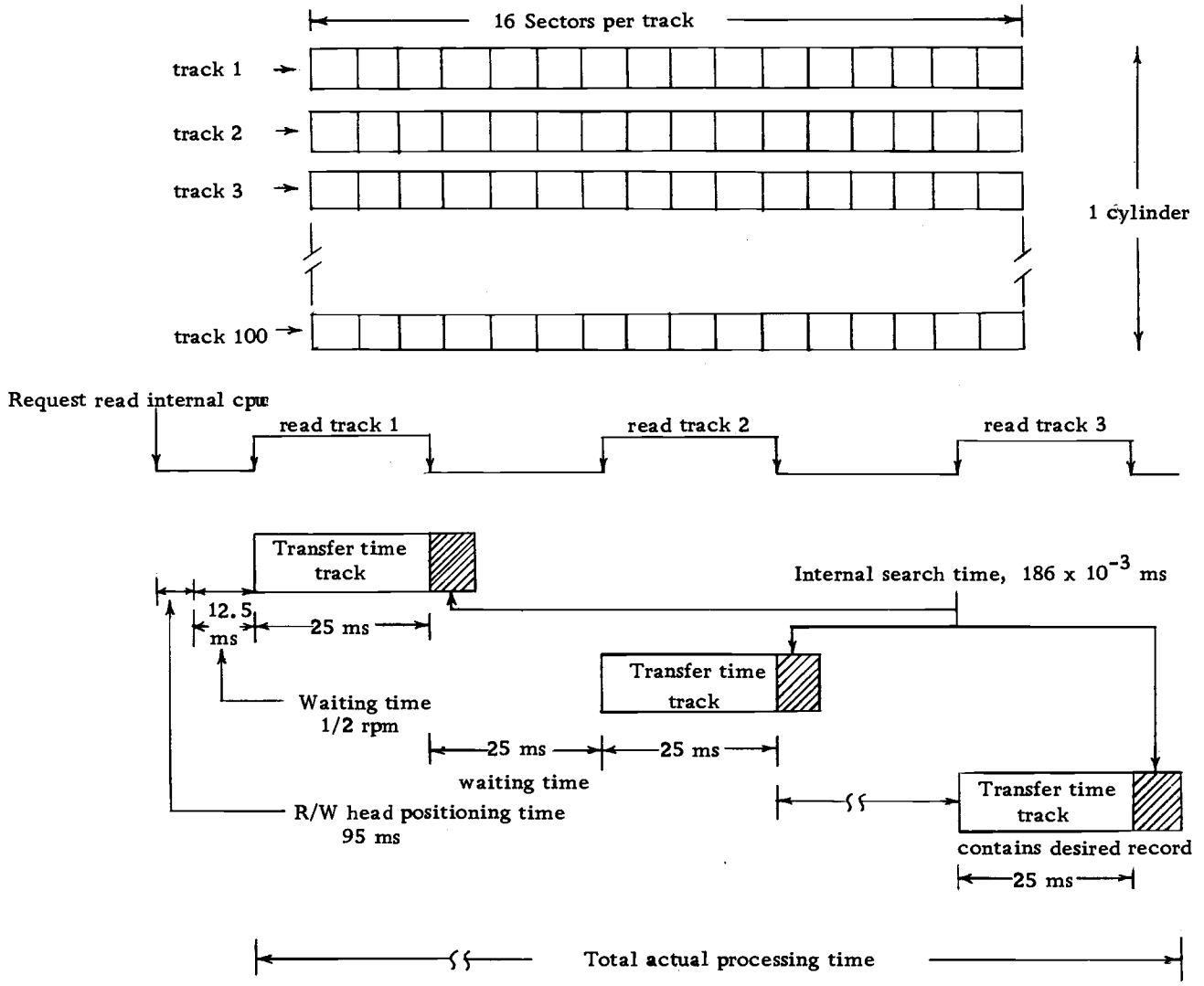
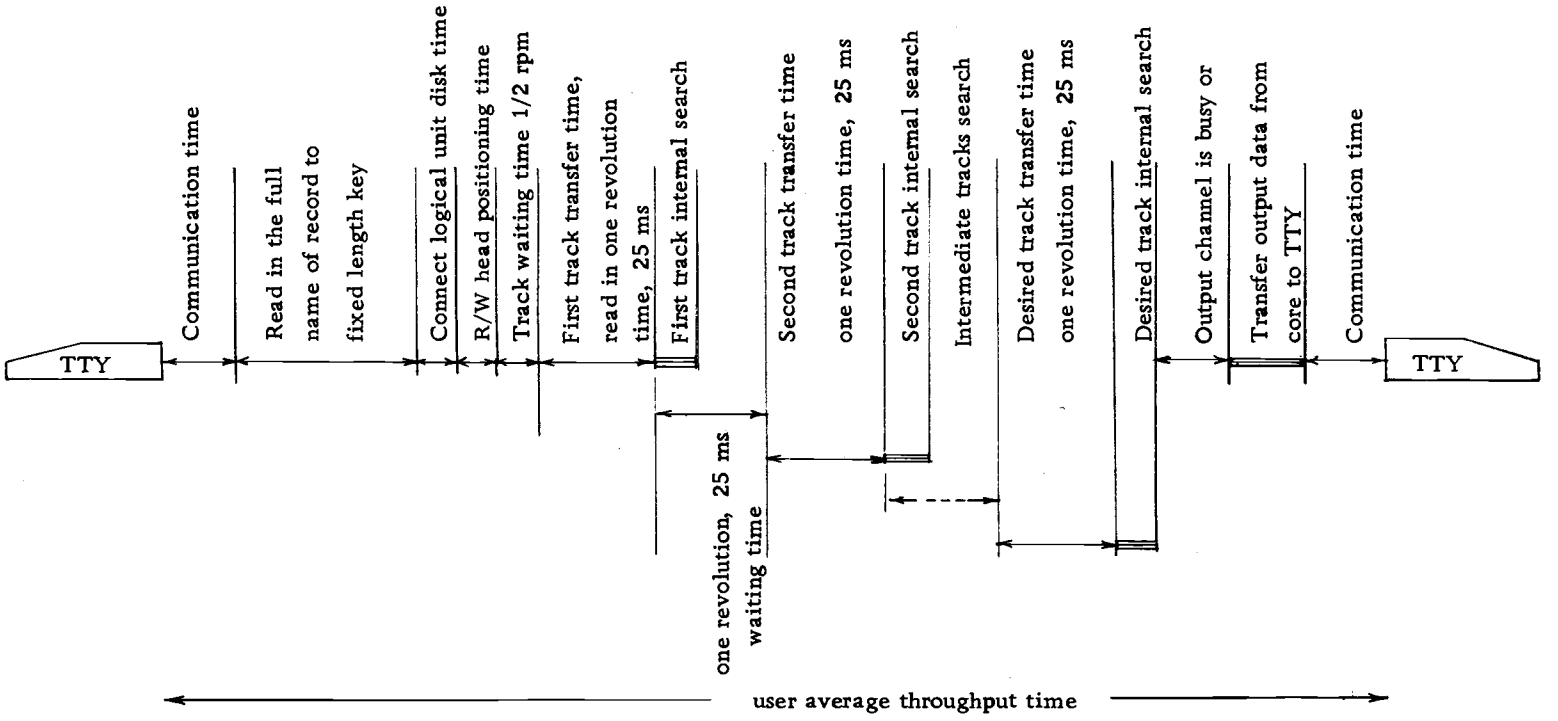


Figure B. 6. Time diagram for accessing a record from sequential disk file.



NOTE ⟺ Billing time (ms)
TTY billing time, period of ON-OFF to the system (hr)
File space charge, depends on number of file blocks the user uses in disk memory (block/month)

Figure B.7. Time diagram for random accessing a record in a one-cylinder sequential disk file.

Table B. 1. Results of computation of an accessing time per random record retrieval as the function of file size for unsorted sequential disk file.

Size of full file in record	Average number of searched records	Average number of supporting cylinders	Average number of full cylinders required	Number of tracks required in the last cylinder	Number of full tracks required in the last cylinder	Number of records in the last track	Initial R/W head positioning time	Cylinder to cylinder positioning time	Track average waiting time	Previous read-in time	A desired track read-in time	A desired track internal search time	Average access time of a random record in SQ disk file
N	(N+1)/2	Z_a	$(Z_a - 1)$	t_{LLAC}	$t_{LLAC}-1$	N_{ALT}							
unit	records	cylinders	cylinders	tracks	tracks	records	ms	ms	ms	ms	ms	ms	ms
128	65	1	0	2	1	1	95	-	12.5	50	25	0.166	182.666
512	257	1	0	5	4	1	95	-	12.5	200	25	0.166	332.666
1024	513	1	0	9	8	1	95	-	12.5	400	25	0.166	532.666
4096	2049	4	3	3	2	1	95	90	50.0	1600	25	0.166	1860.166
8192	4097	7	6	5	4	1	95	180	87.50	3200	25	0.166	3587.666
12888	6145	10	9	7	6	1	95	270	125.00	4800	25	0.166	5315.166
16384	8193	13	12	9	8	1	95	360	162.50	6400	25	0.166	7042.666

Note: Unsorted sequential disk file -- 64 records per track

-- 640 records per cylinder.

Table B.2. Results of computation of average throughput time per random record retrieval and CPU billing time per record accessing with file system using both unique fixed-length key and full name of records in accessing.

Size of full file in records	File looding factor	Communication time back and forth	Full name of a record to fixed-length key conversion time	Connecting logical unit time	Average disk access time per record	Time required for checking I/O	Time required to write out	CPU busy time per record access using fixed-length key	CPU busy time per record access using full name of a record	Average throughput of a random access from SQ disk file
N	α	T_{CBF}	T_{CFNTFK}	T_{RCLU}	$T_{ASFSTSQF}$	$T_{RC\,I/O}$	T_{RWOUT}	$t_{TCPUSQF}$	$t_{CPUTSQFRFU}$	
unit		ms	ms	ms	ms	ms	ms	ms	ms	ms
128	0.0078	.3218	8.395	.008	182.663	.008	36	36.691	45.086	227.399
512	0.0312	.3218	8.475	.008	332.663	.008	36	38.220	46.695	377.479
1024	0.0625	.3218	8.515	.008	532.663	.008	36	40.258	48.773	577.519
4096	0.2500	.3218	8.595	.008	1860.663	.008	36	52.486	61.081	1905.599
8192	0.5000	.3218	8.607	.008	3587.663	.008	36	68.789	77.390	3632.605
12288	0.7500	.3218	8.626	.008	5315.166	.008	36	85.084	93.710	5360.130
16384	1.000	.3218	8.641	.008	7042.663	.008	36	101.398	110.039	7087.645

Table B. 3. Results of computation of storage space required as the function of file size of unsorted sequential file with both fixed length key and record full name.

File size in records	Full name of a record program S_{CFNFK}		Main file search program S_{CPRS}		Total required core memory with S_{CFNFK}, in	Total required core memory without S_{CFNFR}	Disk space supporting the
unit	words		words		words	words	track
	(a)	(b)	(a)	(b)			
64	200	544	61	1024	1829	1085	1
128	200	1088	61	1024	2373	1085	2
256	200	2176	61	1024	3461	1085	4
512	200	4352	61	1024	5637	1085	8
1024	200	8704	61	1024	9989	1085	16
2048	200	17408	61	1024	18693	1085	32
4096	200	34816	61	1024	36101	1085	64
8192	200	69632	61	1024	70917	1085	128
12288	200	104448	61	1024	105733	1085	192
16384	200	139264	61	1024	140549	1085	256

Note: Column a = Number of computer words used for processing program.

Column b = Number of computer words used for supporting data of one track.

Table B.4. Results of computation of customer operating cost per call (unit cost) as the function of file loading factor of unsorted sequential file with using the full name of record in accessing.

File looding factor	Disk space for data file	Disk space for accessing program	Disk space charge per month	TTY, line and operator cost per month	Rule of use data file	CPU-time charge per month	CPU-line and disk charge per month	CPU-line and disk charge per month per call
Unit	Tracks	Tracks	$	$	calls per month	$	$	$
.0078 (128 records)	2	1.317	0.995	760 2675 4075	52500 210000 420000	197.25 789.01 1578.00	958.25 3465.01 5652.00	1.82523 1.65000 1.34614
.0312 (512 records)	8	4.505	3.752	760 2675 4075	52500 210000 420000	204.27 817.17 1634.34	968.02 3495.92 5713.09	1.84384 1.66472 1.36025
.0625 (1024 records)	16	8.755	7.427	760 2675 4075	52500 210000 420000	213.39 853.53 1707.03	981.12 3535.96 5789.76	1.86880 1.68378 1.37851
.2500 (4096 records)	24	34.255	17.477	760 2675 4075	52500 210000 420000	267.24 1068.90 2137.83	1044.72 3780.00 —	1.98994 1.80000 —
.5000 (8192 records)	128	68.255	28.877	760 2675 4075	52500 210000 420000	338.58 1354.32 2708.64	1157.46 - -	2.20467 - -
.7500 (12288 records)	192	102.255	88.28	760 2675 4075	52500 210000 420000	409.98 1639.92 3279.84	1258.26 - -	2.39668 - -
1.000 (16384 records)	256	136.255	117.68	760 2675 4075	52500 210000 420000	481.41 1925.67 3851.37	1359.09 -	2.58874 -

Example 1b. Strictly Sequential Disk File

In case the file is strickly sequentially organized and the cylinder overflow concept is used to handle the overflow records, assume half of each cylinder overflow area to be full and the accessing of an overflow record use the direct chain method as in the Direct file.

1. Computation of the required tracks and required cylinders. In this type of file the concept of using cylinder overflow track is introduced; the methods of calculation are as follows:

   Assume that the number of searches in the cylinder overflow track is 10% of the number of retrieval records from the main file (approximately 9% of total access times).

   The data records are organized in such a way that:

   The number of prime track per cylinder = 9 tracks

   The number of records in a prime track = 63 records

   The number of cylinder overflow track = 1 track

   The number of records in a cylinder overflow track = 56 records.

   Since the computation is made when the cylinder overflow track is half full

Then the number of looking-up in overflow track

$$= \frac{56}{2} = 28 \text{ records.}$$

assumes the overflow records which are the head of each chain, are stored sequentially in the first nine successive spaces, but not necessarily in any key sequence (the same as unsorted sequential file). See detail in Figure B. 8, page 245.

Then average number of looking for the head of the chain records $= \frac{9 + 1}{2} = 5$

The average number of fault looking up for head of the chain records = 5-1 = 4

The average number of correct looking up for head of the chain record = 1

The number of overflow records which are not the head of the chain = 28 - 9 = 19

Then average number of looking up these records = $\frac{19 + 1}{2} = 10$

a. The average search time in cylinder overflow track (T_{ASOFT})

T_{ASOFT} = (initial time) + [(Fault check count loop time) (5 - 1) + correct checking count loop time (1)] + [(Fault checking chain loop time) (10 -1)

+ correct checking chain loop time (1)] (9.22)

= (5.5) + [10.625 (5 - 1) + 7.875 (1)] + [(19.25 (10 - 1) + 4.725) + 158.75 (1)]

= 5.5 + 50.375 + 336.125 = 392 μsec.

= 0.392 ms << 25 ms; then 50 ms, waiting and read-in time per track is considered.

2. Computation of the average access time per record retrieval of strickly sequential disk file.

a) The average disk access time per random record retrieval of strickly sequential file ($T_{AVACFSTSQF}$)

According to the concept of accessing a random record from strickly sequential disk file as shown in Figure B.8, page 245. The $T_{AVACFSTSQF}$ can be computed as follows:

In strickly sequential disk file organization, an overflow exists when each track which is supporting file is full, so each times when searching is performed over a full track 10% of T_{ASQFT} is to be considered. The the following equation can be set up:

$$T_{AVACFSTSQF} = (\text{initial positioning time}) + t_{TCTCP} + t_{TAWT} + t_{PTRIN} + \frac{1}{10}\ (50\text{ ms}) (t_{LLAC} - 1) + T_{RINDT} + T_{ASFSQF} \quad (9.23)$$

All parameters on right hand side of Equation (9.23) have the same meaning as in Example 1a, but the computation is now based on 9 prime tracks per cylinder; each prime track contains 63 records. The result of computation of Equation (9.23) are shown in Table B.5, page 241.

3. Computation of the average throughput time per record retrieval of strickly sequential disk file. The methods computation are as follows:

   a) Average throughput time per record retrieval of strickly sequential disk file ($T_{ATHRPSTSQF}$)

   From the time diagram on page 230 the following equation can be set up.

$$T_{ATHRPSTSQF} = T_{CBF} + T_{CFNTFK} + T_{RCLU} + T_{AVACFSTSQF} + T_{RCI/O} + T_{RCWOUT} \quad (9.24)$$

$$= 0.3218 + 8.515 + 0.008 + 572.729 + 0.008 + 36.00$$

$$= 617.582 \text{ msec.}$$

   when $N = 1084$, $\alpha = 0.0625$ for the illustration. The results of the computation of $T_{ATHRPSTSQF}$, Equation (9.24) as the function of file size or file loading factor, α, are shown in Table B.6 on page 242.

b) The approximate formula of $T_{ATHRPSTSQF} = T_2$

The major components of search time (msec.) are considered; the minor components of search time (μsec) are omitted.

$T_{AVACFSTSQF} \simeq$ (initial positioning time) + (cylinder search time) + (track search time) + (read-in the desired track) + $\frac{1}{10}$ (overflow track search time)

$44.8878\ (ms) \simeq T_{CBF} + T_{CPNTFK} + T_{RCLU} + T_{RCI/O} + T_{RWOUT}$

$$T_2 \simeq T_{AVACFSTSQF} + 44.8878 \text{ msec.}$$

$$\simeq 107.5\,(1) + 30\,[\frac{N+1}{2 \times 630} - 1] + 50\,[\frac{N+1}{2 \times 630}] + 25\,(1) + \frac{1}{10}\,(50)$$

$$T_2 \simeq 0.635\,N + 102.451 \text{ msec.} \quad (9.25)$$

4. For the computation of the achievable throughput-rate capability of strickly sequential disk file, $C_{(2)}$, the method of computation is the same as that illustrated in Example 1a. See page 226.

Results of the computation of $C_{(2)}$ are shown in Table 7.3, page 152 and plotted in Figure 7.3 for comparison with other files.

5. For the computation of the customer operation cost per call of strickly sequential file, $U_{C(2)}$, the method of computation is the same as that shown in Example 1a. See page 227. Results of the computation of $U_{C(2)}$ are shown in Table 7.6, page 173 and plotted in Figures 7.12 to 7.14 at the specific rates of use 250 calls/hr., 1000 calls/hr and 2000 calls/hr for comparison with other files.

Table B. 5. Results of computation of an accessing time per random record retrieval from strickly sequential disk file.

File size in records	Average number of looking-up records	Average number of supporting cylinder	Average number of full cylinder required	Number of track in the last cylinder	Number of full track in the last cylinder	Number of records in the last track	Initial R/W head positioning time	Cylinder to cylinder positioning time	Track average waiting time	Previous track read-in time	A desired track read-in time	A desired track internal search	10% overflow track search	Average access time per record of STSQF
N	(N+1)/2	Z_a	$(Z_a - 1)$	t_{LLAC}	$t_{LLAC}-1$	N_{ALT}	$t_{PR/WH}$	t_{TCRCTP}	t_{AWT}					
unit	records	cylinders	cylinders	tracks	tracks	records	ms	ms	ms	ms	ms	ms	ms	ms
128	65	1	0	2	1	2	95	-	12.5	50	25	0.174	5	187.674
512	257	1	0	5	4	5	95	-	12.5	200	25	0.197	20	352.697
1024	513	1	0	9	8	9	95	-	12.5	400	25	0.229	40	572.729
4096	2049	4	3	6	5	33	95	90	50.0	1600	25	0.418	160	2020.418
8192	4097	8	7	3	2	2	95	210	100.0	3250	25	0.174	325	4005.174
12288	6145	11	10	8	7	34	95	300	137.5	4850	25	0.426	485	5892.926
16384	8193	14	14	5	4	3	95	420	187.5	6500	25	0.181	650	7877.681

Note: Strickly sequential disk file -- 63 record per track, 9 track per cylinder

-- 56 record in one cylinder overflow track

-- both 10% cylinder overflow search and internal search track by track are effecting the average throughput time.

Table B.6. Results of computation of average throughput time per random record retrieval and CPU busy time per record accessing with the file system using both unique fixed-length key and full name of records in accessing.

Size of full file in records	File looding factor	Communication time back and forth	Full name of a record to fixed-length key conversion	Connecting logical unit disk time	Average disk access time per record	Time required for checking I/O	Time required to write out	CPU busy time per record accessing using fixed-length key	CPU time per record accessing using full name of record	Average throughput of a random access from STSQ
N	α	T_{CBF}	T_{CFNTFK}	T_{RCLU}	$T_{ASFSTSQF}$	$T_{RCI/O}$	T_{RWOUT}	$t_{TCPUSQF}$	$t_{CPUTSQFRFU}$	
unit		ms	ms	ms	ms	ms	ms	ms	ms	ms
128	0.0078	.3218	8.395	.008	187.674	.008	36	36.722	45.117	232.371
512	0.0312	.3218	8.475	.008	352.697	.008	36	38.259	46.734	397.510
1024	0.0625	.3218	8.515	.008	572.729	.008	36	40.297	48.812	617.582
4096	0.2500	.3218	8.595	.008	2020.418	.008	36	52.526	61.121	2065.351
8192	0.5000	.3218	8.601	.008	4005.174	.008	36	68.835	77.436	4050.133
12288	0.7500	.3218	8.626	.008	5892.926	.008	36	85.496	94.086	5937.890
16384	1.0000	.3218	8.641	.008	7877.681	.008	36	101.447	110.008	7922.660

Table B. 7. Results of computation of required storage space as the function of file size of Strickly sequential disk file with using both fixed-length key and full name of record in accessing.

File size in record	Full name of record to fixed-length conversion S_{CFNFK}		Search programs S_{CPRS}			Total required core memory using full name of a record	Total required core memory using fixed-length key	Disk space supporting D_s
Unit	Words		Words			Words	Words	Track
	d	e	a	b	c			
128	200	1088	61	72	1024	2449	1161	3.032
512	200	4352	61	76	1024	5713	1161	9.127
1024	200	8704	61	76	1024	10065	1161	18.254
4096	200	34816	61	76	1024	36101	1161	73.016
8192	200	69632	61	76	1024	70993	1161	145.032
72288	200	104448	61	76	1024	105809	1161	217.040
16348	200	139264	61	76	1024	140625	1161	289.064

Note: Column a = Number of computer words used for processing search routine.

Column b = Number of computer words used for search overflow track routine.

Column c = Number of computer words used for supporting data of one track.

Column d = Number of computer words used for variable length tree search.

Column e = Number of computer words used for variable length tree supporting.

Table B. 8. Results of computation of customer operating cost per call (unit cost) as the function of file loading factor of strickly sequential disk file using the full name of record in accessing.

File looding factor	Disk space for data file	Disk space for accessing program	Disk space charge per month	Terminal devices cost per month	Rule of use data file	CPU-time charge per month	CPU-line and disk charge per month	CPU-line and disk charge per month per call
Unit	Tracks	Tracks	$	$	calls per month	$	$	$
.0078 (128 records)	3.032	1.392	1.33	760	52500	197.40	958.73	1.82615
				2675	210000	789.55	3465.88	1.65041
				4075	420000	1579.11	5637.44	1.54244
.0312 (512 records)	9.127	4.579	4.11	760	52500	204.45	968.56	1.84487
				2675	210000	817.86	3496.97	1.66522
				4075	420000	1635.69	5714.80	1.36066
.625 (1024 records)	18.254	8.829	8.13	760	52500	213.54	981.67	1.86984
				2675	210000	854.22	3537.35	1.68445
				4075	420000	1708.41	5791.54	1.37893
.2500 (4096 records)	73.016	34.255	32.18	760	52500	267.42	1059.60	2.01828
				2675	210000	1069.62	3776.80	1.79847
				4075	42000	-	-	-
.5000 (8192 records)	145.032	68.329	64.01	760	52500	338.79	1162.82	2.21489
				2675	210000	-	-	-
				4075	420000	-	-	-
.7500 (12288 records)	217.047	102.329	95.813	760	52500	411.63	1267.44	2.41417
				2675	210000	-	-	-
				4075	420000	-	-	-
1.0000 (16384 records)	289.064	136.329	127.62	760	52500	481.65	1369.27	2.60813
				2675	210000	-	-	-
				4075	420000	-	-	-

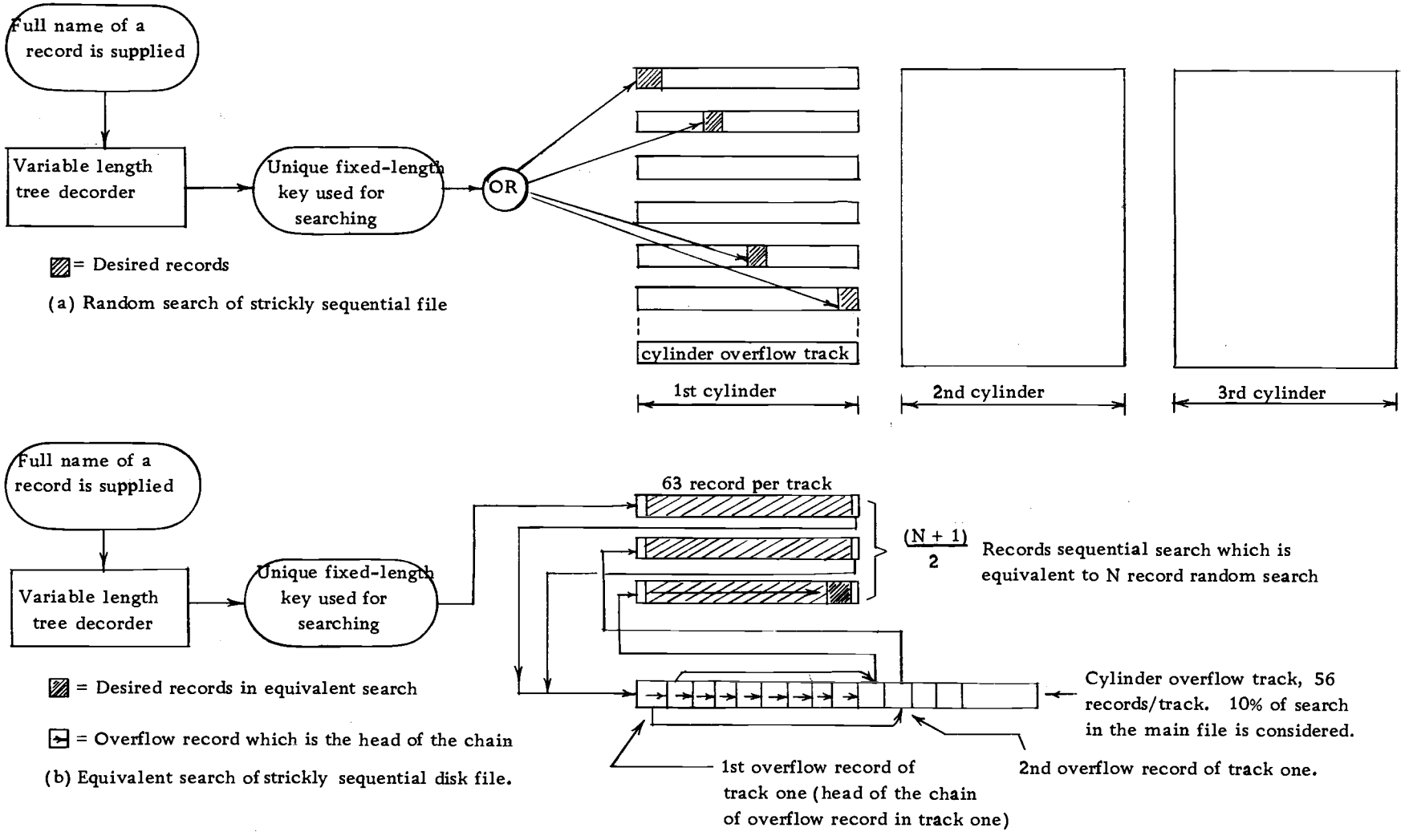


Figure B. 8. Random access of a record from strickly sequential file in (a) is equivalent to search of a desired record in (b).

Example 2. The Indexed Sequential Disk File

For accessing a random record from an Indexed Sequential File, supported by CDC Disk 854 unit, the following assumptions are applied:

- There are N, the current records in the file, each record containing 384 bits (32 bytes = 16 words, including 2 bytes, 4 character keys), in continguous area.
- First data track, prime track, contains cylinder index, track index and data area, respectively. There are 55 records in the first track, 63 records in the successive tracks and 56 records in cylinder overflow tracks.
- The last track in each cylinder is reserved as an overflow area. Then there are 9 prime tracks and 1 overflow track for the 10 track-cylinder disk file unit.
- To access a random record from the file the full name of the record is used. The variable-length tree decoding technique is used to convert it to a unique fixed-length key. See details on page 75.
- File system processing programs and cylinder index are kept in internal core memory only during operating hours. They are kept permanently at some place in disk memory.

1. For the computation of the number of required tracks and required cylinders. The following parameters are introduced.

   a) The number of used cylinders = Z

      Where Z is the smallest integer such than $Z \geqslant \frac{N}{N_c}$

      N_c = number of records in each cylinder of this typical file

   Furthermore

   Z = number of records in Cylinder Indexed

   For illustration, according to the writer's desired

   N_c = 496 for first cylinder

   N_c = 559 for the latter-cylinder

   (First track = 55 records; the latter track = 63 records)

   Z can be rewritten in the following form

$$Z \geqslant \frac{496}{496} + \frac{(N-496)}{559} = 1 + 0.945 = 1.945 \qquad (9.25)$$

   $\Longrightarrow$ Z = 2 cylinder. For N = 1024

   $\Longrightarrow$ The number of record in Cylinder Index = 2

   b) The number of used tracks, N_{UT}

$$N_{UT} \geqslant \frac{N}{63} \qquad \text{For } N < 496 \qquad (9.26)$$

   where N_{UT} is the smaller integer such that $N_{UT} \geqslant \frac{N}{63}$

$$N_{UT} = 9 + 9\,(Z-2) + N_{TLC}, \text{ for } N > 496 \qquad (9.27)$$

where 9 = Number of used tracks in first cylinder

$9\,(Z-2)$ = Number of used tracks in successive full cylinders

N_{TLC} = Number of used tracks in the last used cylinder.

$$\geq \frac{(N - 496 - (Z-2)\,559)}{63}$$

where N_{TLC} is the small integar satisfies the above inequality.

For illustration for N = 4096

Z = 8 cylinders, by Equation (9.25)

$$N_{TLC} = \frac{(4096 - 496 - (8-2)\,559)}{63} = 3.143$$

= 4 tracks

$N_{UT} = 9 + 9\,(8-2) + 4 = 67$ tracks.

2. For the computation of cylinder Index Entries and the average search time per random entries retireval of Cylinder Index Table. The following parameters are introduced and compute:

a) For indexed sequential disk file

The number of cylinder index entries, $N_{CI} = Z$

b) Cylinder Index average search time, T_{LSCYLI}, based on tested program. See page

$$T_{LSCYLI} = 3.9375\, N_{CI} + 14.3125 \ \mu sec. \qquad (9.28)$$

Since in this designed problem, $N_{CI} < 63$, internal linear search is better than binary search. See Figure A. 2 page 195.

For illustration

$$T_{LSCYLI} = 3.9375\,(8) + 14.3125$$

$$= 0.046 \quad msec.$$

3. For the computation of Track Index Entries and the average search time per random entries retrieval of Track Index Table in core memory. The following parameters are introduced and compute:

The following reserved bits are provided in this Indexed Sequential disk file for:

a) Home address (H. A.) Each track in prime area and a track index reserve 24 bits (24 bits = 1 word = 2 bytes) for the home address of a track.

b) Control Overflow Cylinder Record (COCR)

(1) Reserve 24 bits (1 word) for address of the last overflow record in the cylinder.

(2) Reserve 24 bits (1 word) for indicating the left

bytes (6 bits = 1 bytes) in the cylinder overflow area. Both (1) and (2) are located in data file at the beginning of Track Index; There reserved bits will be used by System Operating program.

c. Normal, Overflow and Dummy Entries. Reserve 48 bits (2 words: first word for 4 character-key and second word for its corresponding address) for each Normal, Overflow, and Dummy entry.

Then the number of reserved bits per Track Index

= 48 ($2P_T$ + 1, dummy)

= 48 ($2P_T$ = 1, dummy)

= 48 (2 x 9 + 1) = 912 bits = 152 bytes.

where P_T = Number of pair of Normal and Overflow entries

= Number of data tracks in the prime area, ($P_T \leq 9$ in this case).

Then the number of reserved bits for data records in Track Index =

= (one track data bit capacity) - 912 bits

= 24,576 - 912 = 23664 bits = 986 words.

Since this system is designed with 55 data file records in the first track (Track Index): 55 records = 880 words

Cylinder index = 60 words

Then (60 + 880) > 986 $\Longrightarrow$ (986 - 880 - 60) = 45 words as spare storage space which might be needed for extra information used by the operating program in accessing of the data file.

d) In computing the average search time per random entry retrieval, the number of average records per Track Index is considered and computed. See details on page 266, and Figure B.12. If

$P_{T(i)}$ = Number of required Track Index entries (pairs) in Cylinder ith

P_{AT} = Average number of pairs in each Track Index

$$P_{AT} = \frac{1}{Z} \sum_{i=1}^{i=Z} P_{T(i)} \qquad (9.29)$$

where Z = Number of used cylinder = number of used Track Index.

For illustration N = 1024 Z = 2; $\sum_{i=1}^{i=Z} P_{T(i)} =$ 20 pairs

By Equation (9.29)

$$P_{AT} = \frac{1}{2} (20) = 10 \text{ pairs.}$$

See results of computation in Table 9.9, page

There are three cases of searching entries in the Track Index:

(1) Search time spent in satisfied path search, T_{SSFP}

(2) Search time spent in overflow path search, T_{SOVFP}

(3) Search time spent in fault path search, T_{SFLP}

The assumption is that the number of searches in the overflow cylinder is 10% of the number of searches in the prime area (9% of total search).

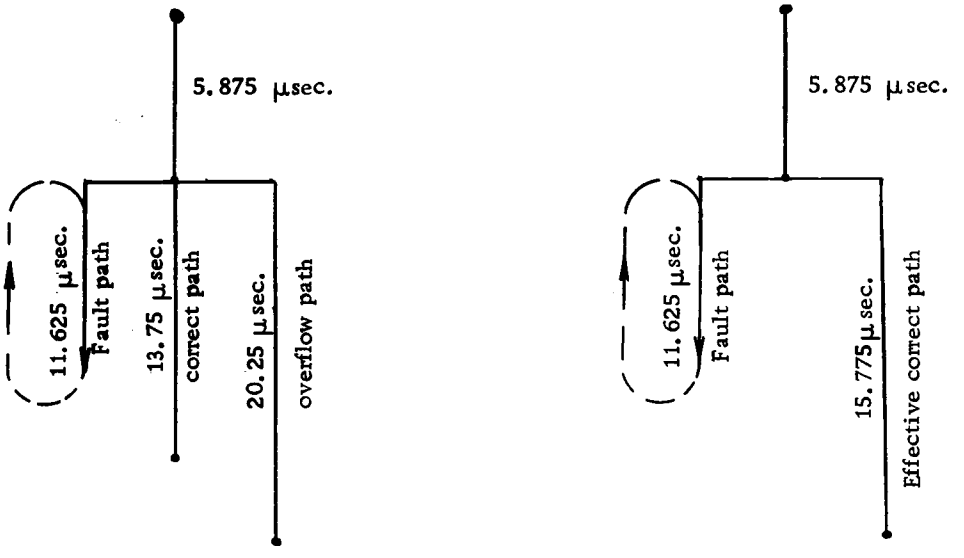


Figure B.9. Actual time path search and effective path search.

If T_{STIENT} = Time required for searching Track Index entries

= Average search time of fault loop + search time of effective correct loop

$$= 11.625 \frac{(P_{AT} + 1 - 1)}{2} + 15.775 \times 1$$

$$= 5.8125 (P_{AT}) + 20.3375 \ \mu sec.$$

$$= 5.8125 (P_{AT} - 1, \text{ dummy}) + 20.3375 \ \mu sec \qquad (9.30)$$

There is no search for a dummy, buffer record.

For illustration, N = 1024 P_{AT} = 10 pairs,

by Equation (9.30)

$$T_{STIENT} = 5.8125 (10 - 1) + 20.3375$$

$$= 72.65 \ \mu sec = 0.073 \ msec.$$

e) For computation of the average search time per record retrieval of cylinder Index and Track Index, the following formula can be set up:

The average search time per record retrieval of Cylinder Index and Track Index, $T_{ASTCYLTI}$.

$$T_{ASTCYLTI} = T_{LSCYLI} + T_{AR/WHPT} + T_{ATW} + T_{RTI} + T_{STIE}^{a/} = T_{LSCYLI} + T_{ATIDS} \quad (9.31)$$

For illustration N = 1024

$$T_{ASCYLTI} = 0.022 + 95 + 12.5 + 25$$

$$= 132.522 \text{ ms.}$$

4. For computation of the average throughput time of Indexed Sequential disk file, the following parameters have to be introduced and computed:

   a) The average search time per record in desired track in the main file of Indexed sequential disk file = $T_{AVADFIDSQF}$.

      The concept of accessing a random record from the Indexed sequential disk file is shown in Figure B. 10, page 259, with only 10% of the search times in the prime track made on the cylinder overflow track. This concept is the same as that for the Strickly sequential

---

a/ Since it is assumed that there is no dropping of the Busy Status during any random record retrieval operation, then T_{STIE} is overlapped with the waiting and read-in time (50 ms) for the desired track. It can be said that $T_{ASCLYTI}$ is a portion of the average throughput time per record retrieval. It is present here as one step of the average throughput time computation. See the results of computation of $T_{LSCYLTI}$ in Table B.9, page 261, column 5.

disk file. Then the following equation can be set up.

$$T_{AVACFIDSQF} = T_{ASTCYLTI} + T_{TAWT} + T_{RINDT} + 10\% (T_{TASOFT}) \quad (9.32)$$

where T_{RINDT} = Desired track read-in time, 25 ms.

T_{TASOFT} = Total cylinder overflow track search time

$= (25 + 25) + T_{ASOFT}$

$= 50.00 + 0.392$ msec.

For illustration N = 1024

$$T_{AVADFIDSQF} = 132.522 + 25 + 25 + \frac{1}{10} [50.392]$$

$= 187.561$ msec.

See the results of computation in Table B. 10, page 262, column 13 from L. H. S.

b) The average throughput time per record retrieval of the Indexed sequential disk file, $T_{ATHRPISQF}$

From the time diagram Figure B. 11, page 260, the following equation can be set up:

$$T_{ATMRPISQF} = T_{CBF} + T_{CFNTFK} + T_{RCLU} + T_{AVACFIDSQF} + T_{RC\,I/O} + T_{RWOUT} \quad (9.33)$$

For illustration N = 1024

$$T_{ATHRPISQF} = 0.3218 + 8.515 + 0.008 + 187.561 + 0.008 + 36.00$$

$$= 232.414 \text{ msec.}$$

See the results of computation of $T_{ATHPPISQF}$ as the faction of N in Table B.11, page 260, column 11 from L.H.S.

e) The approximate formula of $T_{ATHRPISQF} = T_3$

Only the major components of search time (msec) are considered; the minor components of search time (μsec) are omitted.

Then $T_{AVACFIDSQF} \simeq T_{AR/WPT} + T_{ATW} + T_{RTI} + T_{ATW} + T_{DTRW} + 10\% (T_{TASQFT})$

$$\simeq 107.5 + 25 + 25 + 25 + \frac{1}{10} [50] =$$

187.5 msec.

Since 44.8878 (msec) $\simeq T_{CBF} + T_{CFNTFR} + T_{RCLU} + T_{RCI/O} + T_{RWOUT}$

Then from Equation (9.33)

$$T_3 \simeq 1875 \text{ ms} + 44.8878$$

$$\simeq 232.3878 \text{ msec.} \qquad (9.34)$$

5. For computation of the average CPU busy time per record retrieval for the Indexed sequential disk file, both the unique fixed-length key and the full name of the record are

used in accessing. See Figure B. 11, page 260, in which only CPU busy time components are considered.

a) Total CPU time per record accessed using unique fixed-length key

$$t_{TCPUTISQF} = T_{LSCYLI} + T_{RCUL} + T_{STIE} + T_{ALSM} + \frac{1}{10}\,(T_{ALSOFT}) + T_{RCI/O} + T_{RWOUT} \qquad (9.35)$$

$$= T_{RCI/O} + T_{RWOUT}$$

$$= 0.022 + 0.008 + 0.073 + 0.388 + 0.039 + 0.008 + 36.00$$

$$= 36.538 \text{ msec for N} = 1024.$$

b) Total CPU time per record accessed using full name of record, $t_{TCPUTISQFUFN}$. This type of accessing requires more extra time than that of a) with the full name of the record to fixed-length key conversion time, t_{CFNTFK}

Then $t_{TCPUTISQFUFN} = t_{CFNTFK} = t_{TCPUTISQF}$

$$= 8.515 + 36.538 = 45.053 \text{ msec.}$$

See the results of computation in Table B. 11, page 263.

6. For the computation of core storage space and disk storage space required, use the concept and formula that have

been mentioned in Appendix B, page 208. Results of computation are shown in Table B. 12, page 264.

7. The computation of achievable throughput rate capability of the Indexed Sequential disk file, $C_{(3)}$, and Customer operating cost per call, $U_{C(3)}$, is the same as that shown before in Example 1a. See illustrations on page 227.

The computed results of $C_{(3)}$ and $U_{C(3)}$ are tabulated in Tables 7. 3 and 7. 6, pages 152 and 173, and plotted in Figures 7. 7 - 7. 10; Figures 7. 12 - 7. 15 respectively, for making a comparison among the results obtained from the various types of file.

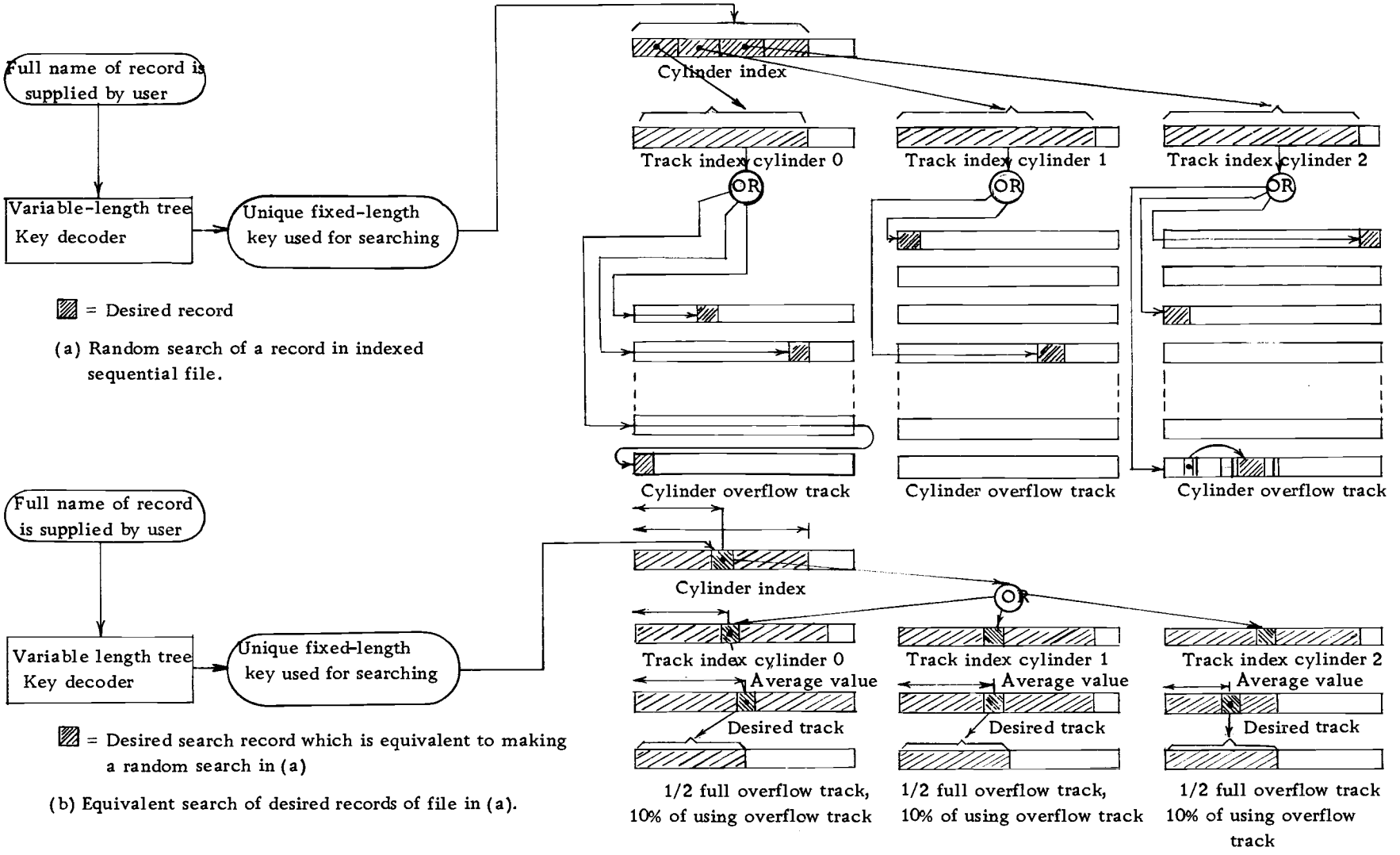


Figure B. 10. Random access of a record from indexed sequential disk file in (a) is equivalent to accessing an average of a record in (b); Average of records, in cylinder index, in track index, and in desired track are to be considered in computation.

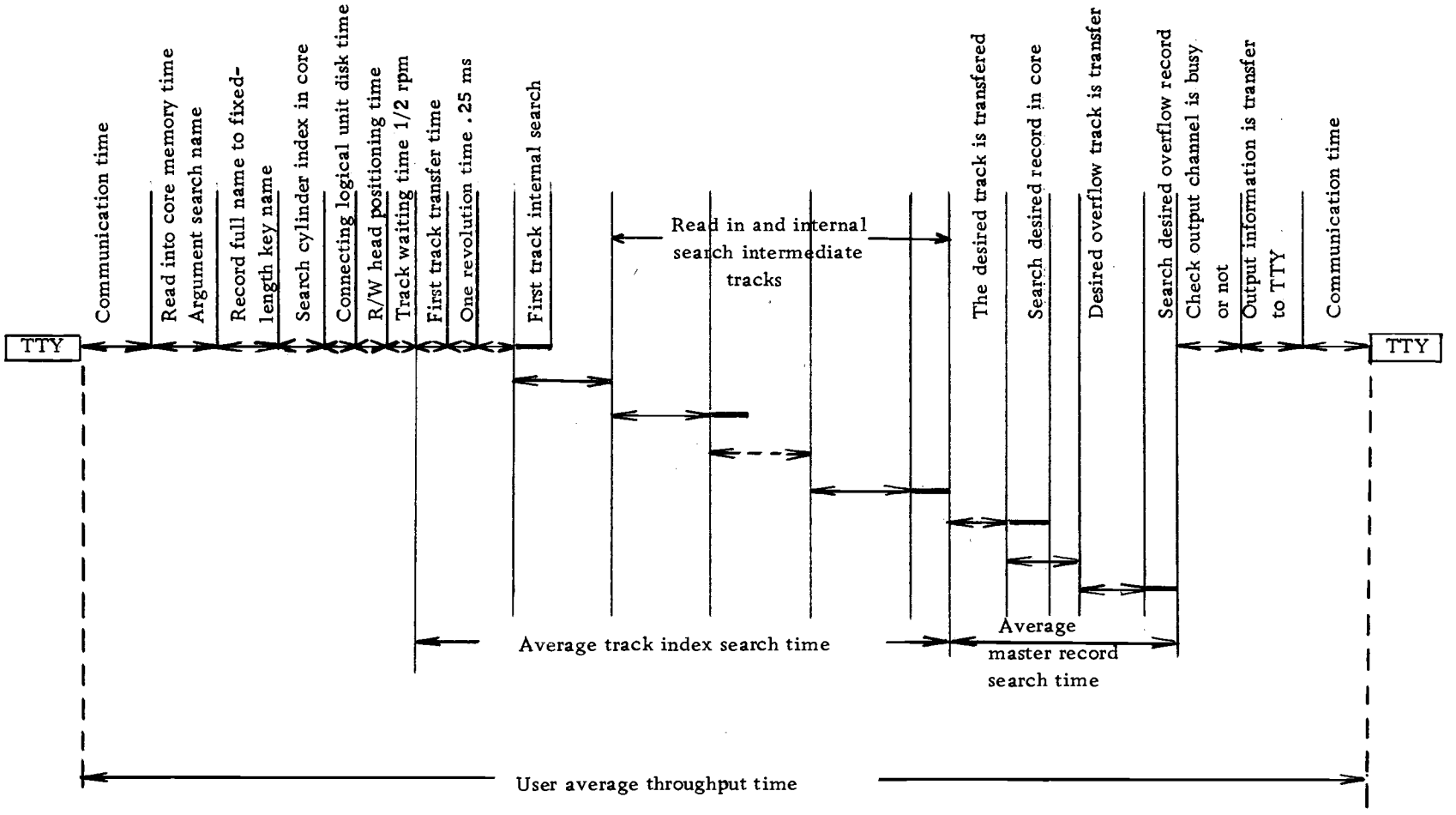


Figure B. 11. Time diagram for random accessing the record from indexed sequential file on disk.

Table B.9. Result of computation of average search time per random access of the entry from cylinder index and track index of indexed sequential disk file.

File size in record	Number of required cylinder	Cylinder index table size		Average search cylinder index per record	Track index table size of file system		Average number pairs per cylinder	R/W head positioning time	Track index average waiting time	Read in desired track index	Desired track index search	Average search time per record accessing in cylinder index and track index
unit(N)	cylinders	entries	words	ms	pairs	words	pairs	ms	ms	ms	ms	ms
128	1	1	2	0.0183	4	8	4	95	12.5	25	.037	132.518
512	2	2	4	0.022	12	24	6	95	12.5	25	.049	132.522
1024	2	2	4	0.022	20	40	10	95	12.5	25	.073	132.522
4096	8	8	16	0.046	75	150	10	95	12.5	25	.073	132.546
8192	15	15	30	0.073	148	296	10	95	12.5	25	.073	132.573
12288	23	23	46	0.105	222	444	10	95	12.5	25	.073	132.605
16384	30	30	60	0.132	304	608	10	95	12.5	25	.073	132.632

Note: Column 12 overlaps with waiting time for read-in the desired track. It is not effect to average throughput time.

Table B. 10. Data results of computation of average search time per random record retrieval with 10% using cylinder overflow track and average disk access time per record retrieval from the indexed sequential disk file.

File size in record	Number of required cylinder	Number of tracks in first cylinder	Number of track in intermediate full cylinder	Number of track in last cylinder	Total required tracks	Average records per track	Per record cylinder index track index searching	1 R. P. M. waiting time	Read-in desired track	Desired track internal search *	10% Overflow search *	Average disk access time per record
unit(N)	cylinder		tracks	tracks	tracks	record	ms	ms		ms	ms	ms
128	1	3	-	-	3	43	132.518	25	25	.332	5.039	187.557
512	2	9	-	1	10	51	132.522	25	25	.364	5.039	187.561
1024	2	9	-	9	18	57	132.522	25	25	.388	5.039	187.561
4096	8	9	54	4	67	61	132.546	25	25	.403	5.039	187.585
8192	15	9	117	7	133	62	132.573	25	25	.407	5.039	187.612
12288	23	9	189	1	199	62	132.605	25	25	.407	5.039	187.644
16384	30	9	252	4	265	62	132.632	25	25	.407	5.039	187.671

Note: Column 11 is overlapped with Column 12, then not considered in average throughput time. It is considered in billing time.

Table B. 11. Data results of computation of average throughput time per random record retrieval and CPU busy time per record access of indexed sequential file using both full name of records and unique fixed-length key in accessing.

Size of full file in records	File loading factor	Communication time back and forth	Full name of record to fixed-length key conversion time	Connecting logical unit disk time	Average disk access time per record	Time required to check I/O	Time required to write out	CPU busy time per record using fixed-length key	CPU busy time per record using full name of record	Average throughput per random access from ISQ disk file with full name
N		T_{CBF}	T_{CFNTFR}	T_{RCLU}		$T_{RCI/O}$	$t_{TCPUISQF}$	$t_{CPUISQRFUL}$		
unit		ms	ms	ms	ms	ms	ms	ms	ms	ms
128	0.0078	.3218	8.395	.008	187.557	.008	36	36.442	44.837	232.286
512	0.0312	.3218	8.475	.008	187.561	.008	36	36.499	44.974	232.374
1024	0.0625	.3218	8.515	.008	187.561	.008	36	36.538	45.053	232.414
4096	0.2500	.3218	8.595	.008	187.565	.008	36	36.578	45.173	232.498
8192	0.5000	.3218	8.601	.008	187.612	.008	36	36.608	45.209	232.551
12288	0.7500	.3218	8.626	.008	187.644	.008	36	36.640	45.266	232.608
16384	1.0000	.3218	8.641	.008	187.677	.008	36	36.667	45.308	232.656

Table B. 12. Results of computation of storage space required as the function of file size of indexed sequential disk file, using both full name of record and unique fixed-length key in accessing.

Size of full file in record	Full name to fixed-length key S_{CFNTFK}		Cylinder index search S_{CYIDS}		Track Index search S_{TRIOS}		Prime track search S_{PRTRS}		Overflows track search S_{OVFLS}	Total core space required with fixed-length key	Total core space required with full name key	Disk space supporting file system
unit	words		words		words		words		words	words	words	track
N	a	b	a	b	a	b	a	b				
128	200	1088	53	2	60	-	62	1024	77	1278	2566	3. 1587
512	200	4352	53	4	60	-	62	1024	77	1280	5832	11. 2539
1024	200	8704	53	4	60	-	62	1024	77	1280	19184	19. 5079
4096	200	34816	53	16	60	-	62	1024	77	1292	36308	75. 0317
8192	200	69632	53	30	60	-	62	1024	77	1306	71139	147. 9365
12288	200	104448	53	46	60	-	62	1024	77	1322	105970	221. 8413
16384	200	139264	53	60	60	-	62	1024	77	1336	140800	294. 873

Note: Column a - core storage space required for processing program.
Column b - core storage space required for reserving area.

Table B. 13. Results of computation of customer operating cost per call (unit cost) as the function of file loading factor of the indexed sequential file, using the full name of record in accessing.

File loading factor	Disk space for data file	Disk space for accessing program	Disk space charge per month	Terminal devices cost per month	Rule of use data file	CPU-time charge per month	CPU-time and disk charge per month	CPU-time and disk charge per month per call
unit	tracks	tracks	tracks	$	calls per month	$	$	$
.0078 (128 records)	3.16	1.506	1.40	760	52500	196.16	957.56	1.82392
				2675	210000	784.65	3461.05	1.64811
				4075	420000	1569.30	5645.70	1.34421
.0312 (512 records)	11.25	4.695	4.78	760	52500	196.76	961.54	1.83150
				2675	210000	787.03	3466.81	1.65086
				4075	420000	1574.06	5653.84	1.34615
.625 (1024 records)	19.51	8.945	8.54	760	52500	197.11	965.65	1.83933
				2675	210000	788.43	3471.97	1.65331
				4075	420000	1576.86	5660.40	1.34771
.2500 (4096 records)	75.03	34.457	32.85	760	52500	197.63	990.48	1.88662
				2675	210000	790.53	3498.38	1.66589
				4075	420000	1581.06	5688.90	1.35450
.5000 (8192 records)	147.94	68.472	64.92	760	52500	197.79	1022.71	1.94801
				2675	210000	791.16	3531.08	1.68146
				4075	420000	1582.32	5722.24	1.36243
.7500 (12288 records)	221.84	102.486	97.30	760	52500	198.04	1055.34	2.01017
				2675	210000	792.16	3564.46	1.69736
				4075	420000	1584.31	5756.61	1.37062
1.0000 (16384 records)	294.87	136.500	129.41	760	52500	198.22	1087.63	2.07167
				2675	210000	792.89	3597.30	1.71300
				4075	420000	1585.78	5790.19	1.37861

Concept of Average Number of Records per Track and Average Number of Searches per Track

The following concepts are introduced to compute the parameters and characteristics of indexed sequential disk file, Example 2 and partitioned disk file, Example 4:

Average Number of Records Per Track (N_{APT})

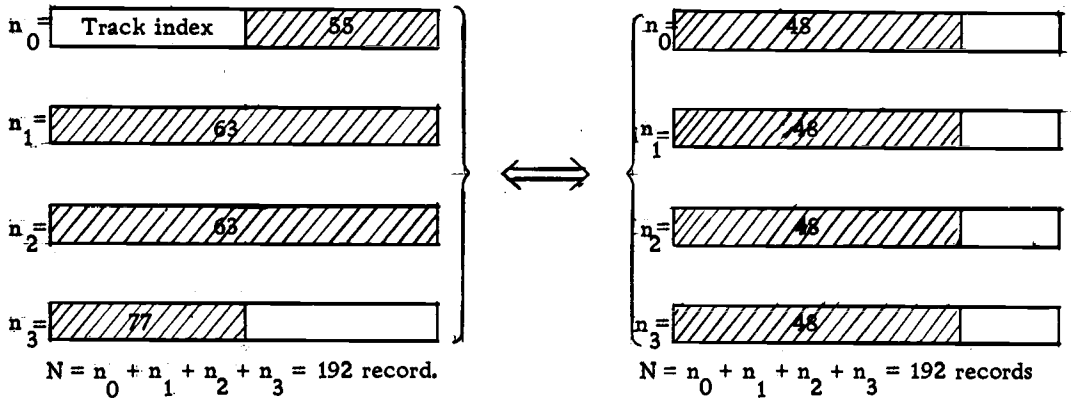


Figure B. 12. Interaction of actual disk file and imaginary disk file.

In general, if n_i = number of records in i^{th} track

where i = 0, 1, 2, 3...

N_{UT} = number of tracks used.

$$N_{APT} = N_{UT} \sum_{k=o}^{k=i} n_k ; \quad k = 0, 1, 2, \ldots, i \tag{9.36}$$

Average Number of Searches per Track (N_{ALS})

In an indexed sequential disk file or partitioned disk file when the desired cylinder and the desired track are located, the desired track is read only into core memory and internal linear search is performed. The average number of searches in the desired track is to be considered.

In Example 3 and 4,

$$N_{ALS} = \frac{N_{APT} + 1}{2} \qquad (9.37)$$

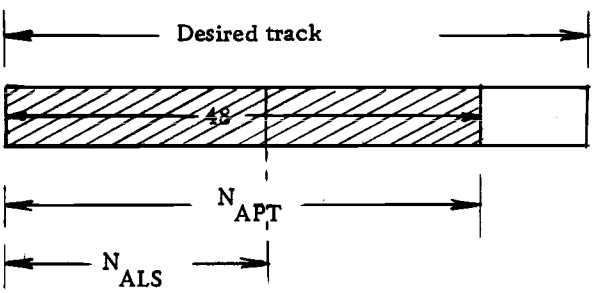


Figure B. 13. Relationship of N_{APT} and N_{ALS}.

In Figure B. 13, N_{APT} = 48 words

then $$N_{ALS} = \frac{(48 + 1)}{2} = 24.5$$

$$= 25 \text{ (This value is used for computation).}$$

Example 3a. The Partitioned File with Single-Level Directory

For accessing a random record from a partitioned disk file with single level directory, on CDC Disk 854 unit, the following assumptions are applied:

- There are N, the current records in the file, each record containing 384 unit (32 byte = 16 words, including 2 bytes, 4 character keys), in contiguous area.
- One data track contains 63 records.
- The records in the main file are stored in key sequence.
- The overflow records are held by one track; the cylinder overflow concept as in the indexed sequential disk file is used here.
- For accessing a random record from the file the full name of the record is used. The variable-length tree decoding technique is used to convert it to a unique fixed-length key. See details on page 75.
- Directory Table and file system processing programs are kept in internal core memory only during operation hours. They are kept permanently at some place in disk memory.

1. For computation of the number of tracks and cylinders required, the following parameters are introduced:

   a) The number of used cylinders = Z.

where Z is the smallest integer such that $Z \geq \frac{N}{N_c}$

N_c = number of records in eacy cylinder.

N_c = 504 records in first cylinder.

N_c = 567 records for the latter cylinders.

Z can be rewritten in the following form:

$$Z \geq \frac{504}{504} + \frac{(N - 504)}{567} = 1 + 0.92$$

$Z = 2$ for $N = 1024$

See results of computation of Z in Table B. 14, page , column 2 from L. H. S.

b) The number of tracks used, N_{UT}

$$N_{UT} \geq \frac{N}{63} \quad \text{for } N < 504 \qquad (9.39)$$

where N_{UT} is the smallest cylinder such that

$$N_{UT} \geq \frac{N}{63}$$

$$N_{UT} = 8 + 9 (Z - 2) + N_{TLC} \text{ for } N > 504 \qquad (9.40)$$

where 8 = number of tracks in the first cylinder.

$9(Z-2)$ = number of successive tracks in successive full cylinders.

N_{TLC} = number of tracks used in the last cylinder used.

$$\geqslant \frac{[N - 504 - (Z - 2)\, 567]}{63}$$

where N_{TLC} is the smallest interger which satisfies the above in equality.

For illustration when $N = 4096$

$\Rightarrow Z = 8$ cylinder by Equation (9.38)

$$\Rightarrow N_{TLC} = \frac{[4096 - 540 - (8 - 2)\, 567]}{63} = 3.0159$$

$= 4$ tracks

$N_{UT} = 8 + 9\,(8 - 2) + 4 = 66$ tracks.

See results of computation of N_{UT} in Table B.14, page 279, column 6 from L.H.S.

2. For computation of the number of entries in the directory and the average search time per random entry accessing from directory table, directory search is performed to obtain the address of the corresponding track in core memory.

a) Number of record entries in Directory, I_{REDP}

where $I_{REDP} = N_{UT}$

For $N \rightarrow \alpha \Rightarrow I_{REDP} > 64$.

Then the ginary search is considered.

The average time for directory search of the single

level directory disk file is

$$T_{ASSLDPF} = 15.245 + 35.000 \times N_{ATBS} \qquad (9.41)$$

where $N_{ATBS} = \frac{1}{N} [2^K (K - 1) + 1 + (K + 1) \times (N - 2^K + 1)] \simeq \log_2 N \qquad (9.42)$

= average number of binary search for N records.

For illustration when $N = 1024$

$N_{ATBS} = 3$ Equation (9.42)

then $T_{ASSLDPF} = 15.245 + 35.00 \times 3$ μsec.

= 0.1212 ms.

See results of computation of $T_{ASSLDPF}$ in Table B.12, column 5 from L.H.S.

3. For computation of the average number of records per track (data track used) and the average search time per random record accessing, a record is searched for from the main file, according to one of the following cases:

- the desired record is on the desired track;
- the desired record is on the cylinder overflow track.

a) The average number of records per track = N_{APT}.

To compute the average search time per random record accessing from the disk partitioned file, consider the

average number of logical records per track. The method of computation has been shown in Example 2, page . The formula is

$$N_{APT} = \frac{1}{N_{UT}} \sum_{k=0}^{k=i} n_k, \quad k = 0, 1, 2, 3, \ldots i \qquad (9.36)$$

For illustration when $N = 1024$

$N_{UT} = 17$ tracks

then $N_{APT} = \frac{1}{17} [1024] = 60$ records/track.

If T_{ALSDT} = Average search time per random record accessing in the desired track.

$$T_{ALSDT} = 7.875 \left(\frac{N_{APT} + 1}{2} - 1\right) + 167.000$$

based on test program.

$$T_{ALSDT} = 3.9375 (N_{APT}) + 163.0625 \ \mu sec. \quad (9.43).$$

For illustration when $N = 1024$ $N_{APT} = 60$

$$T_{ALSDT} = 3.9375 (60) + 163.0625 \ \mu sec$$

$$= 0.399 \text{ msec.}$$

See results of computation of N_{APT} and T_{ALSDT} in Table B.14, columns 7 and 12 from L.H.S.

b) The average access time of a random record from the main disk file with 10% of normal track search

used in cylinder overflow track = $T_{ASRDFOF}$.

Since it is assumed that there is no dropping of the busy status during any random retrieval operation, then T_{ALSDT} is overlapped with 10% of the time spent searching for overflow records. T_{ALSDT} is now omitted. From the time diagram in Figure B.8, page 245, the following equation can be set up:

$$T_{ASRDFOF} = T_{AR/WHPT} + T_{ATW} + T_{RIND\,T} + \frac{1}{10}\,[T_{TASOFT}] \qquad (9.44)$$

$$= 95 + 12.5 + 25 + 5.0392 \text{ msec.}$$

$$= 137.539 \text{ msec.}$$

4. The average throughput time per random record retrieval of the partitioned disk file with single level directory, using full name of record in accessing = $T_{ATHRPSLDPF}$.

a) From the time diagram on page the following equation can be set up:

$$T_{ATHRPSLPF} = T_{CBF} + T_{CFNTFK} + T_{ASSLDPF} + T_{RCLU} + T_{ASRPDFOF} + T_{RCI/O} + T_{RWOUT} \qquad (9.45)$$

For illustration when N = 1024

$$T_{ATHRPSLDPF} = 0.3218 + 8.515 + 0.1212 + 0.008 + 137.539 + 0.008 + 36.00$$

$$= 182.513 \text{ msec.}$$

See the results of computation of $T_{ATHRPSLDPF}$ in Table B.15, page 280.

b) The approximate formula of $T_{ATHRPSLDPF} = T_4$.

Only the major components of search time (msec) are considered; the minor components of search time (μsec) are omitted.

$$\text{Then } T_4 \simeq T_{ASSLDPF} + T_{AR/WHPT} + T_{ATW} + T_{RINDT} + \frac{1}{10}\,[T_{TASOFT}] + 44.8878$$

$$\simeq [15.245 + 35.00\,(\log_2 N)] \times 10^{-3} + 137.539 + 44.8878$$

$$T_4 \simeq \underline{35.00\,(\log_2 N) + 182.4268} \quad (9.46)$$

where 44.8878 (msec) $\simeq T_{CBF} + T_{CFNTKF} + T_{RCLU} + T_{RCI/O} + T_{RWOUT}$

5. For computation of the average CPU busy time per record retrieval for the partitioned disk file, both the unique fixed-length key and the full name of the record

are used in accessing. See Figure B. 14, page 227, in which only CPU busy time components are considered.

a) Total CPU time per record accessing using unique fixed-length key = $t_{TCPUTSLDPF}$.

$$t_{TCPUTSLDPF} = T_{ASSLDPF} + T_{RCUL} + T_{ALSDT} + \frac{1}{10}(T_{ALSOFT}) + T_{RCI/O} + T_{RWOUT} \quad (9.47)$$

$$= 0.1212 + 0.008 + 0.399 + 0.0392 + 0.008 + 36.00$$

$$= 36.575 \text{ msec. for N} = 1024.$$

b) Total CPU time per record accessing using full name of the record = $t_{TCPUTSLDPFUFN}$; this type of accessing requires more extra time than that of a) by the full name of record to fixed-length unique key conversion time, T_{CFNTFK}.

then $t_{TCPUTSLDPFUFN} = t_{CFNTFN} + t_{TCPUTSLDPF}$

$$= 8.515 + 36.575$$

$$= 45.090 \text{ msec. for N} = 1024.$$

6. For computation of core storage space and disk storage space required, the concept and formula that have been mentioned in Appendix B, page 208, are used.

Results of the computation are shown in Table B. 16, page 281.

7. For computation of achievable-throughput rate capability of the single-level partitioned disk file, $C_{(4)}$ and the customer operating cost per call, $U_{C(4)}$ the methods are the same as those illustrated in Example 1a, pages . The computed results of $C_{(4)}$ and $U_{C(4)}$ are tabulated in Tables 7. 3 and 7. 6, pages 152 and 173, and plotted in Figures 7. 7 - 7. 10, Figures 7. 13 - 7. 16, pages 169 - 172.

for comparison with the results obtained from other types of file.

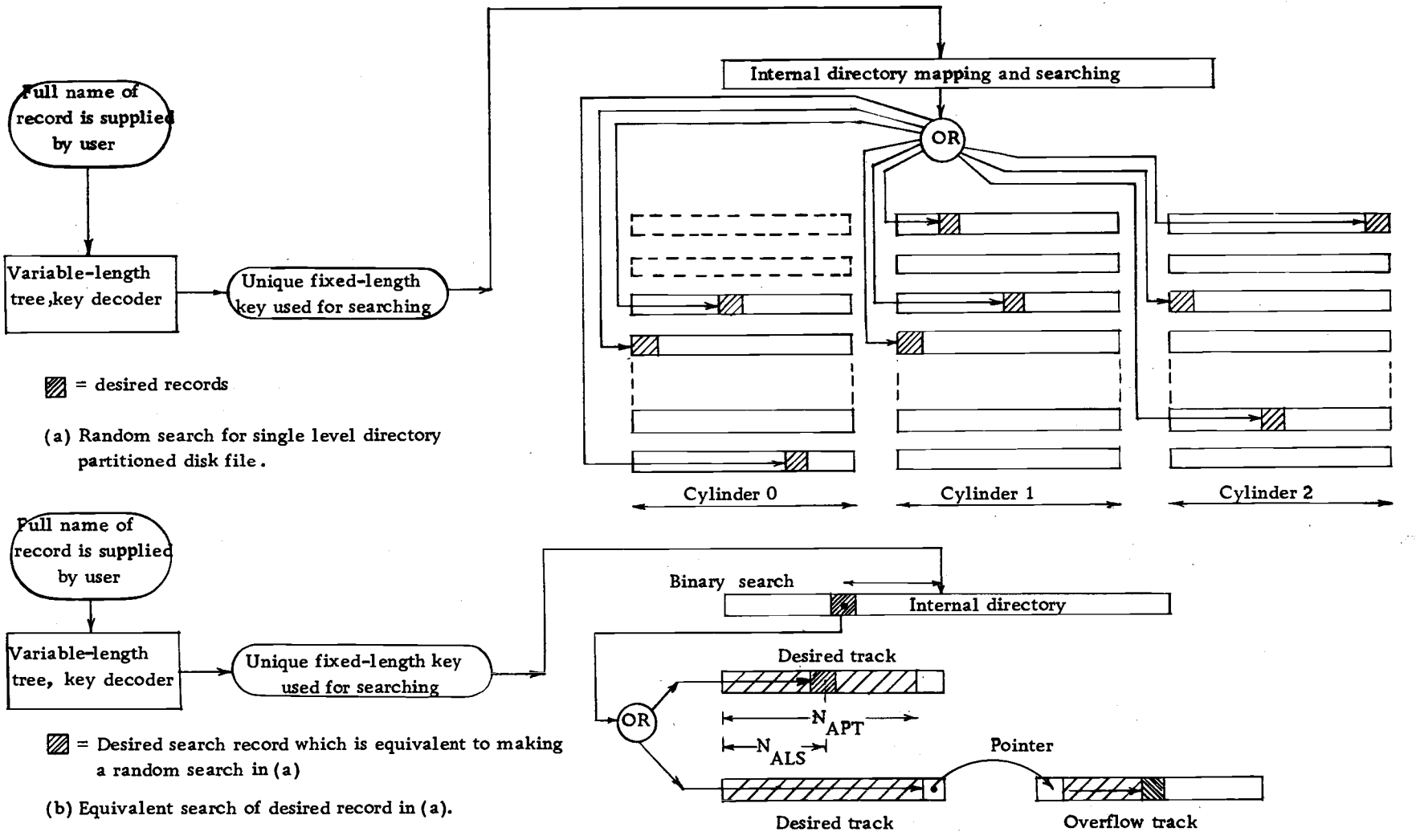


Figure B. 14. Random access of a record from single level directory partitioned disk file in (a) is equivalent to making an average access of a record in (b).

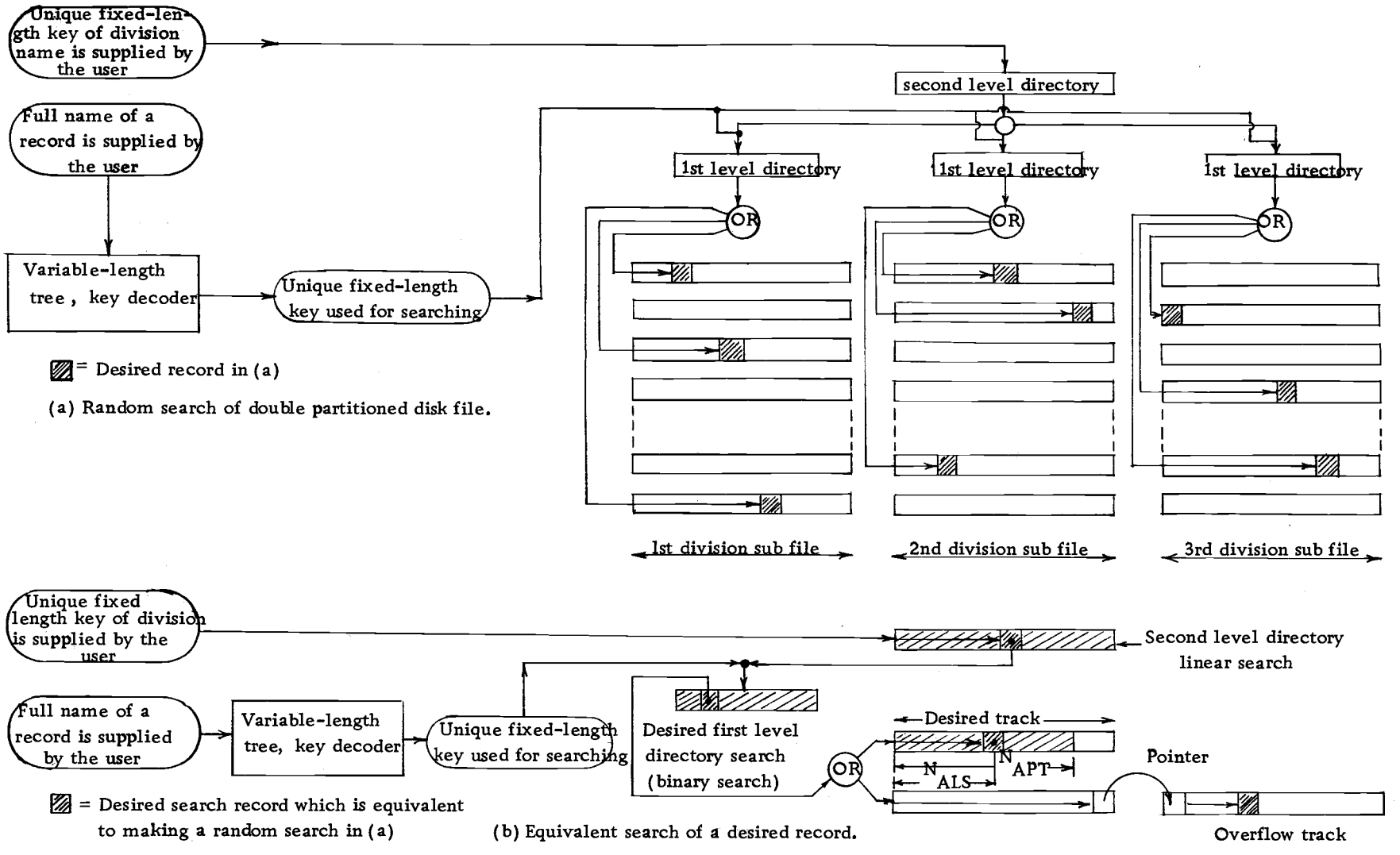


Figure B. 15. Random access of a record from double level partitioned disk file in (a) is equivalent to accessing an average record in the file as in (b).

Table B. 14. Data results of computation of disk average access time per record retrieval as a function of file sizes of a single level directory partitioned disk file.

File size in record	Number of required cylinders	Number of track in first cylinder	Number of track in intermediate full cylinder	Number of tracks in the last cylinder	Total tracks	N_{ATBS}	Average records per track	Directory internal search	Initial positioning and waiting time	Read-in a desired track	Desired track internal search	10% overflow search	Average disk access time per record
unit	cylinders	tracks	tracks	tracks	tracks	records	records	ms	ms	ms	ms	ms	ms
128	1	3	-	-	3	2	43	.0859	107.5	25	.332	5.0392	137.539
512	2	8	-	1	9	3	57	.1212	107.5	25	.388	5.0392	137.539
1024	2	8	-	9	17	3	60	.1212	107.5	25	.399	5.0392	137.539
4096	8	8	6	4	66	5	62	.1918	107.5	25	.407	5.0392	137.539
8192	15	8	117	6	131	6	63	.2271	107.5	25	.411	5.039	137.539
12288	22	8	180	8	196	7	63	.2624	107.5	25	.411	5.039	137.539
16384	30	8	252	1	261	17	63	.2624	107.5	25	.411	5.039	137.539

Table B. 15. Data results of computation of average throughput time per record retrieval as the function of file size or loading factors of a single-level directory partitioned disk file.

File size in record, N	File loading factor, α	Communication time	Full name to fixed-length key conversion	Internal directory search time	Connect logical disk unit	Average access time with 10% overflow	Check I/O logical unit	Write out time	CPU busy time per record access with using fixed-length key	CPU busy time per record access with using full name of record	Average throughput per record with using full name of a record
unit		ms	ms	ms	ms	ms	ms	ms	ms	ms	ms
128	0.0078	.3218	8.395	.0859	.008	137.539	.008	36	36.473	44.868	182.358
512	0.0312	.3218	8.475	.1212	.008	137.539	.008	36	36.564	45.039	182.473
1024	0.0625	.3218	8.515	.1212	.008	137.539	.008	36	36.575	45.090	182.513
4098	0.2500	.3218	8.595	.1918	.008	137.339	.008	36	36.655	45.250	182.663
8192	0.5000	.3218	8.601	.2271	.008	137.539	.008	36	36.697	45.297	182.705
12288	0.7500	.3218	8.626	.2624	.008	137.539	.008	36	36.728	45.354	182.765
16394	1.0000	.3218	8.641	.2624	.008	137.539	.008	36	36.728	45.369	182.780

Table B. 16. Data results of storage space required as the function of file size of a single level directory of partitioned disk file.

File size in record	Full name to fixed-length key conversion required space		Directory search binary search required space		Main file search required space		Overflow search required space	Total required space for file using fixed-length key	Total required space for file using full name of record	Disk required space
unit	words (a)	words (b)	words	words	words	words	words	words	words	tracks
128	200	1088	78	9	62	1024	77	1250	2538	4.0318
512	200	4362	78	27	62	1024	77	1268	5830	11.1270
1024	200	8704	78	51	62	1024	77	1292	10196	19.2540
4096	200	34816	78	198	62	1024	77	1439	36455	74.0159
8192	200	69632	78	393	62	1024	77	1634	71466	145.0317
12288	200	104448	78	588	62	1024	77	1829	106477	218.0476
16384	200	139264	78	783	62	1024	77	2024	141488	291.0635

Table B. 17. Results of computation of customer operating cost per call (unit cost) as the function of file loading factor of the single level directory partition disk file using full name of record in accessing.

File loading factor,	Disk space for data file	Disk space for accessing program	Disk space charge per month	Terminal devices cost per month	Rule of use data file	CPU-time charge per month	CPU-time and disk charge per month	CPU-time and disk charge per month per call
unit	tracks	tracks	$	$	calls per month	$	$	$
.0078 (128 records)	4.032	1.479	1.653	760	52500	196.29	957.94	1.82464
				2675	210000	785.19	2461.84	1.64844
				4075	420000	1570.38	5647.03	1.34453
.0312 (512 records)	11.127	4.693	4.746	760	52500	197.04	961.79	1.83198
				2675	210000	788.18	3467.93	1.65139
				4075	420000	1576.38	5658.84	1.34734
.625 (1024 records)	19.254	8.957	8.463	760	52500	197.04	961.79	1.83198
				2675	210000	789.09	3472.55	1.65359
				4075	420000	1578.15	5661.60	1.34800
.2500 (4096 records)	74.016	34.601	32.585	760	52500	197.97	990.56	1.88677
				2675	210000	791.88	3499.48	1.66642
				4075	420000	1583.76	5691.36	1.35508
.5000 (8192 records)	145.032	68.791	64.147	760	52500	198.18	1022.33	1.94728
				2675	210000	792.68	3531.83	1.68182
				4075	420000	1585.41	5724.56	1.36298
.7500 (12288 records	218.048	102.981	96.309	760	52500	198.42	1054.76	2.00906
				2675	210000	793.71	3565.02	1.69762
				4075	420000	1587.39	5758.70	1.37111
1.0000 (16384 records)	291.064	137.202	128.480	760	52500	198.48	1086.96	2.07040
				2675	210000	793.95	3597.43	1.71306
				4075	420000	1587.92	5791.40	1.37890

Example 3b. The Partitioned Disk File with Double Level Directory

To illustrate the characteristics of the partitioned file with the multi-level directory, the following axxumptions are considered:

- That there are N records (one record for one employee) belonging to one department, Directorate of Air Installation, which is divided into four divisions as follows:

Division Full Name	Unique Fixed-length Key	Records in a Sub-File
Air Field Construction Div.	AIRF	N/4
Building Construction	BUIL	N/4
Technical Division	TECH	N/4
Utilities Division	UTIL	N/4
	Total	N

where $N \rightarrow 0 \leq N < 16384$, when the file is full.

- That this data file system is supported by CDC Disk 854 unit; and the organization of the data is the same as mentioned in Example 3a.

When this file system is organized in multilevel partitioned file, the following features are considered:

1. The fixed-length unique key name of division names is used for the Division Directory Entries (first level directory), and the address of the beginning of each division (sub-sequential file) is used for its corresponding data in

this Directory. To retrieve any record from this partitioned file, the user has to know in which division the desired record belongs. The retrieval task starts from searching this directory first.

2. The user must also supply the name of the desired record. (In some cases the user may supply the fixed-length code number of the desired record. The operating program will directly use this record code number as the key of the record for the sub-directory searching). The full name to unique fixed-length key conversion routine converts the full name of the record into a fixed-length key and uses this key for searching in the sub-directory (second-level directory).
3. If the sub-directory searching is satisfied, the desired record from the desired division main file is retrieved.
4. If the sub-directory searching fails, the operating system calls for the un-found subroutine to notify the user.
5. In the organization of the multilevel directory partitioned file, the higher level directory (division directory) has to be added. More storage space is required in supporting this directory, but it saves time in the sub-directory search.

For example, when N = 16384 records, the file is full.

The required disk tracks = 261 tracks.

Then there are 261 entries in the single-level directory of the partitioned file.

The number of average searches $\simeq \log_2 261 = 7$, for binary search.

Then there $261 \div (4 = \text{number of division}) = \frac{261}{4} \simeq$ 65 entries in the sub-directory.

Then the number of average searches $\simeq \log_2 (\frac{261}{4}) =$ 5 for binary search for the double-level directory partitioned disk file.

If $T_{ASSLDPF}$ = The average binary search time for second level directory per retrieval record of double level directory of partitioned disk file,

Equation (9.41) in Example 3a, has to be modified and used as follows:

$$T_{ASSLDPF} = 15.245 + 35.00 + \frac{N_{ATBS}}{4} \qquad (9.48)$$

6. For computation of the average throughput time per record retrieval of double-level directory partitioned disk file, using full name in accessing, $T_{ATHRPDLDPF}$

Equation (9.45) in Example 3a, has to be modified and used as follows:

a) $$T_{ATHRPHLDPF} = T_{CBF} + T_{ASFLDPF} + T_{CFNTFK} + T_{ASSLDPF} + T_{RCLU} + T_{ASRPDFOF} + T_{RCI/O} + T_{RWOUT} \quad (9.49)$$

where $T_{ASFLDPF}$ = Average linear search time for first level directory per retrieval-record of double level directory directory of partitioned file.

The search program is similar to the internal cylinder index search program of the indexed sequential disk file.

Then $T_{ASFLDPF} = 3.9375\, N_{FLDPF} + 21.5625$

$= 3.9375\,(4) + 21.5625 = 37.3125$ μsec.

$= 0.0373$ msec.

where $N_{FLDPF} = 4$,

according to the statement of Example 3b, there are four divisions, i.e., there are four sub-directory files.

then $T_{ATHRPDLDPF}$ = 0.3218 + 0.0373 + 8515 + 0.859
+ 0.008 + 137.539 + 0.008 +
36.00
= 182.516 msec. for N = 1024.

The results of computation of $T_{ATHRPDLDPF}$ are shown in Table

b) The approximate formula of $T_{ATHRPDLDPF} = T_5$

The method of computation is the same as that of T_4; the difference is as follows:

$$T_4 \simeq 35.00 (\log_2 N) + 182.4268$$

$$T_5 \simeq 35.00 (\log_2 \frac{N}{4}) + 182.4268$$

Since the average number of searches per record retrieval in sub-directory is reduced to $\log_2 \frac{N}{4}$.

c) Total CPU busy time per record accessed using unique fixed-length key = $t_{TCPUTDLDPF}$.

$$t_{TCPUTDLDPF} = T_{ASFLDPF} + T_{ASSLDPF} + T_{RCLU} + T_{ALSDT} + \frac{1}{10} (T_{ALSOFT}) + T_{RCI/O} + T_{RWOUT}$$

= 0.0373 + 0.859 + 0.008 + 0.364 +
0.0392 + 0.008 + 36.00

= 36.543 msec for N = 1024.

d) Total CPU busy time per record accessed using full name of the record = $t_{TCPUTDLDPFUFN}$, as mentioned in Example 3a.

Then $t_{TCPUTDLDPFUFN} = t_{TCPUTDLDPF} + T_{CFNTFK}$

= 36.543 + 8.515 = 45.058 msec for N = 1024.

7. For computation of core storage space and disk storage space required, the method is the same as that in Example 1a. The extra disk storage space has to be considered for supporting the first-level directory. See results of computation in Table B.20, page 291.

8. For computation of the achievable-throughput rate capability of the double-level partitioned disk file, $C_{(5)}$, and the customer operating cost per call, $U_{C(5)}$, the methods are the same as those shown in Example 1a, pages . The computed results of $C_{(5)}$ and $U_{C(5)}$ are shown in Tables 7.5 and 7.6 and plotted in Figures 7.7 - 7.10, Figures 7.12 - 7.15, pages 169 - 172, for comparison with the results obtained from other types of file.

Table B. 18. Data results of computation of disk access time per record retrieval as a function of file sizes of a double-level directory partitioned disk file.

File size in records	Directory entries	Number of records per member	Number of total required tracks per member	Number of tracks in first cylinder per member	Number of full track in intermediate cylinder	Number of tracks in last cylinder	N_{ATES}	Average records per track	Sub-directory internal search	Positioning and waiting time	Read-in desired tracks	Desired track internal search *	10% overflow search *	Average disk access per record
unit				tracks	tracks	tracks	records	records	ms	ms	ms	ms	ms	ms
128	4	32	1	1	-	-	1	32	.0505	107.5	25	0.289	5.039	132.539
512	4	123	3	3	-	-	2	43	.0859	107.5	25	0.332	5.039	137.539
1024	4	255	5	5	-	-	2	51	.0859	107.5	25	0.364	5.039	137.539
4096	4	1024	17	8	-	9	3	60	.1212	107.5	25	0.399	5.039	137.539
8192	4	2048	33	8	18	7	4	62	.1565	107.5	25	0.407	5.039	137.539
12288	4	3072	49	8	36	5	5	63	.1918	107.5	25	0.411	5.039	137.539
16484	4	4096	66	8	54	4	5	62	.1918	107.5	25	0.407	5.039	137.539

Note: * Columns 13 and 14 - the time is overlapped; hence considered only column 14 for average disk accessing time per record retrieval.

Table B. 19. Data results of computation of average throughput time per record retrieval as the function of file size or loading factors of a double-level directory partitioned disk file.

File size in records	File looding factor α	Communication time	Full name to fixed length conversion	Directory internal search	Sub-directory internal search	Connect logical unit disk	Average disk access time per record with 10% overflow	Check I/O logical unit time	Write output time	CPU busy time with using fixed-length key per record	CPU busy time with using full name record	Average throughput per record with using full name of a record
unit		ms	ms	ms	ms	ms	ms	ms	ms	ms	ms	ms
128	.0078	.3218	8.395	0.0373	.0505	.008	137.539	.008	36	36.432	44.828	182.360
512	.0312	.3218	8.475	0.0373	.0859	.008	137.539	.008	36	36.511	44.986	182.476
1024	.0625	.3218	8.515	0.0373	.0859	.008	137.539	.008	36	36.543	45.058	182.516
4096	.2500	.3218	8.595	0.0373	.7212	.008	137.539	.008	36	36.613	45.208	182.631
8192	.5000	.3218	8.601	0.0373	.1565	.008	137.539	.008	36	36.647	45.248	182.672
12288	.7500	.3218	8.626	0.0373	.1918	.008	137.539	.008	36	36.696	45.322	182.732
16384	1.0000	.3218	8.641	0.0373	.1918	.008	137.539	.008	36	36.692	45.333	182.748

Table B.20. Data results of storage space required as the function of file size of a double level directory of partitioned disk file.

File size in record	Full name of fixed-length conversion required space		Directory search Linear search required space		Sub-directory search, Binary search required space		Main file search required space		Overflow search required space	Total required space for using fixed-length key	Total required space for using full name of record	Disk required space
unit	a	b	a	b	a	b	a	b	words	words	words	tracks
128	200	1088	57	12	78	12	62	1024	77	1322	2610	4.0318
512	200	4362	57	12	78	36	62	1024	77	1346	5908	11.1270
1024	200	8704	57	12	78	60	62	1024	77	1370	10274	19.2540
4096	200	34816	57	12	78	204	62	1024	77	1513	36530	74.0159
8192	200	69632	57	12	78	396	62	1024	77	1206	71530	145.0317
12288	200	10448	57	12	78	588	62	1024	77	1898	106546	218.0476
16384	200	139264	57	12	78	792	62	1024	77	2102	141566	291.0635

Table B.21. Results of computation of customer operating cost per call (unit cost) as the function of file loading factor of double level directory partitioned disk file using record's full name in accessing.

File loading factor	Disk space for data file	Disk space for accessing program	Disk space charge per month	Terminal devices cost per month	Rule of use data file	CPU-time charge per month	CPU-time and disk charge per month	CPU-time and disk charge per month per call
unit	tracks	tracks	$	$	calls per month	$	$	$
.0078 (128 records)	4.032	1.549	1.67	760	52500	196.11	957.78	1.82434
				2675	210000	784.50	3461.17	1.64817
				4075	420000	1568.97	5645.64	1.34420
.0312 (512 records)	11.127	4.770	4.77	760	52500	196.83	961.60	1.83160
				2675	210000	787.26	3467.03	1.65096
				4075	420000	1574.52	5654.29	1.34625
.625 (1024 records)	19.254	9.033	8.49	760	52500	197.13	965.62	1.83927
				2675	210000	788.52	3472.01	1.65333
				4075	420000	1577.01	5660.50	1.34173
.2500 (4096 records)	74.016	34.674	32.61	760	52500	197.79	990.40	1.88657
				2675	210000	791.13	3498.74	1.66606
				4075	420000	1582.29	5689.90	1.35473
.5000 (8192 records)	145.032	68.854	64.17	760	52500	197.97	1022.14	1.94633
				2675	210000	791.85	3531.02	1.68143
				4075	420000	1583.67	5722.84	1.36258
.7500 (12288 records)	218.048	103.049	96.33	760	52500	198.30	1054.63	2.00880
				2675	210000	793.14	3564.47	1.69736
				4075	420000	1586.28	5757.61	1.37085
1.0000 (16384 records)	291.064	137.248	128.49	760	52500	198.33	1086.82	2.07013
				2675	210000	793.32	3596.81	1.71276
				4075	420000	1586.64	5790.13	1.37860

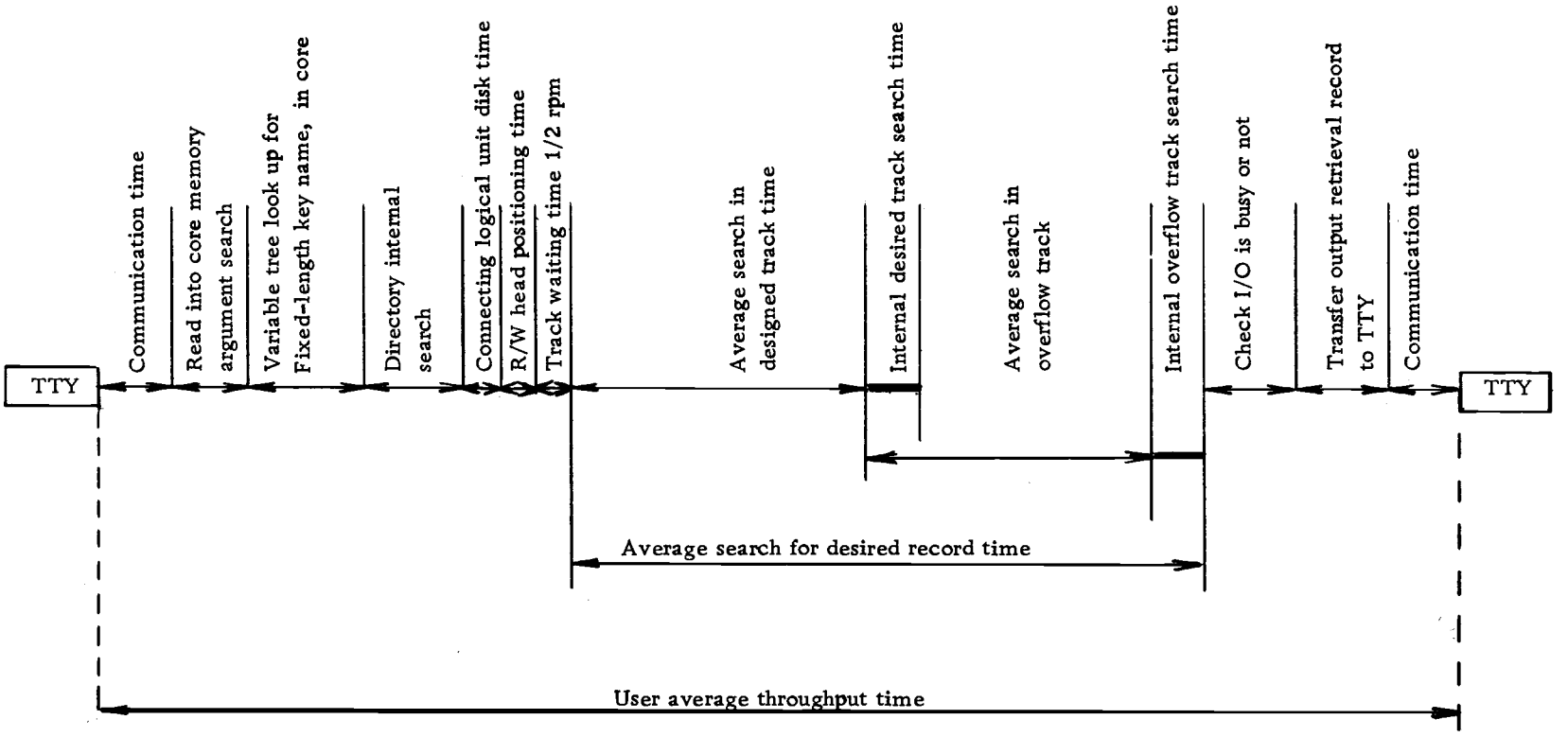


Figure B. 16. Time diagram for random accessing the record from partitioned file, on disk.

Example 4. Direct Disk File

To illustrate the characteristics of Direct or Random Disk File organization, the accessing of a random record from the Direct Disl File, assume that there are N ($0 < N \leqslant 16384$) logical records in the file, each record being the same as mentioned in Examples 2 and 3.

Each record is formatted with a key. The full name of the record is not more than four words, 16 characters. In case the full name of the record is more than words, the rest will be truncated, so that only the first four words are used to convert into a fixed-length bits by hash coding technique. This fixed-length bits (hash address) will represent the cylinder number, track number, to which the logical record belongs, and address of the records when they are mapped in internal core. See Figure B. 19, page 301.

Five hash coding techniques are simulated to evaluate the average length of search per record retrieval with three methods ofhandling the redundant keys. But only the best method of hash coding has been selected for computing the results with those of other methods of the organization. No overflow area will be considered for the direct file.

1. Four Hash Coding Techniques are as follows:

   a) Hash 1. In this method the first four words of the

full name of the record are "exclusive or" together bit by bit. Square the results and take only 10 lower bits of the 12 middle bits as the hash address of each logical record on the disk storage.

Hash address = $(W_1, \text{Ex. OR} W_2 \text{ Ex. OR} W_3 \text{ Ex. or } W_4)^2$ and takes only 10 low bits of the 12 middle bits of the results as Hash address.

Where W_1, W_2, W_3, W_4 are the binary number of the first four successive words respectively.

b) Hash 2. In this method, add the first four successive words together, square the result and take only 10 lower bit of 12 middle bits as Hash address.

Hash address = $(W_1 + W_2 + W_3 + W_4)^2$ and take 10 lower bit of 12 middle bits.

Where W_1, W_2, W_3, W_4 are the binary numbers of the first four words, respectively.

c) Hash 3. In this method the right-most character of each of the first-four words is to be considered as the "Selected Character," and the third bit from the right-most end of each "Selected Character" is kept as the string code in Q register. Shift left A and Q, 10 bits position. Use the results in A register as Hash Address. See Figure B. 17.

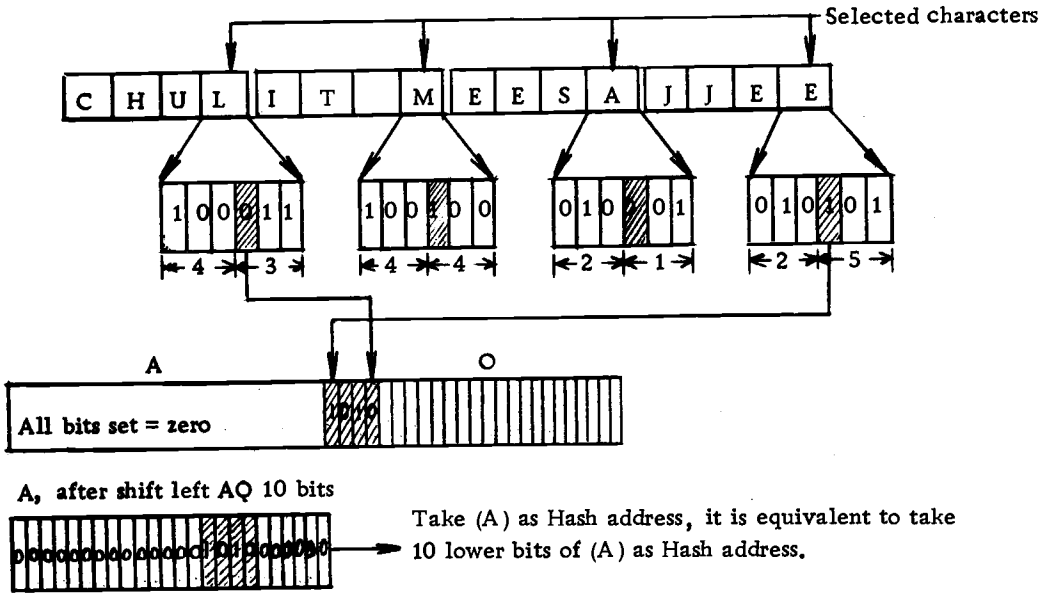


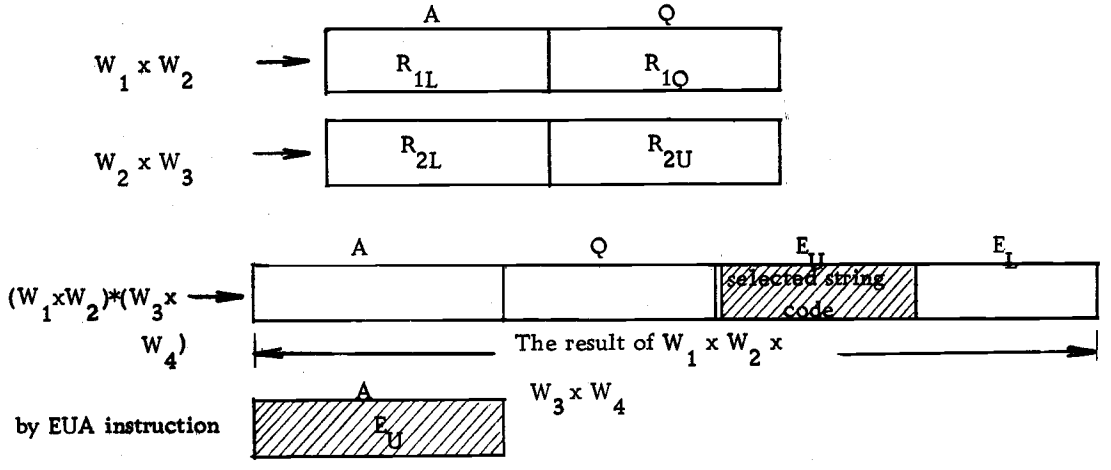
Figure B.17. Hash 3 performed hash address.

d) Hash 4. In this method the first four words of the full name of the record are "Exclusive or" together bit by bit. Power fourth the result and take only 10 lower bits of 12 middle bits of the result as the Hash Address.

Hash address = $(W_1 \text{ Ex. } ORW_2 \text{ Ex. } ORW_3 \text{ Ex. } ORW_4)^4$ and take only 10 lower bits of 12 middle bits of the result as Hash address.

e) Hash 5. In this method multiply the first-four words of the full name of the record and take only 24 bits of the lower middle range bits of the result as a

"Selected String Code". Take 10 lower bits of 12 middle bits of A. The result of the operation is used as the Hash address. See Figure B. 17 fo illustration.



by SHA -12, instruction ⟶ take 10 lower bits of (A) as Hash Address

Where $W_1 W_2 W_3 W_4$ are the same as mentioned in Hash 1, 2, 4

R_{1L} = lower part of result 1

R_{2L} = lower part of result 2

E_L = lower part of E register

R_{1U} = upper part of the result 1

R_{2U} = upper part of the result 2

E_U = upper part of E register

Figure B. 18. Hash 5 performing hash address.

2. There are three methods of handling a redundant key in the Direct File.

   a) Linear probing. In this method the linear search for both the first empty space and the desired record is simulated i = -1, + 3, + 1, and + 1 with records with the same hash address grouped together, and value of i are used from the statistical formula. The results of the simulation are computed and plotted as a function of File Size as shown in Figures B. 20 - B.23, pages 302 - 308 where i is the interval search designator, while the minus sign indicates that the search is backward to the beginning of the table, and the positive sign indicates that the search is forward to the end of the table.

   b) Random probing. In this method its detail the following algorithm of pseudorandom number generator is selected and performed:

      -- Initialize an integer R by setting R equal to 1 every time the random number routine is called.

      -- Then on each successive call for a random number set R = R*5

      -- Use only low-order n + 2 bits of the product and replace the result in R

-- Set $\rho = R/4$ and return to hash coding program.

The results of simulation are computed and plotted as shown in Figures B. 20 - B.23, pages 302 - 308.

c) Direct chain probing. The simulation of direct chaining to handle the redundant key is as follows:

-- From the result of measuring the average number of searches, it is reasonable to use Hash 1 as the hash address generator.

-- The search routine is to find the first empty space by i = +1 interval search simulation.

-- Each logical record needs one extra word for a pointer. See the result of the simulation on Figures B. 20 - B. 23 on pages 302 - 308, showing the average search length as a function of file size.

3. The simulation of four hash coding techniques in the three methods of handling a redundant key are shown in Table B. 22.

The most effective results of simulation in (3) are computed and plotted as in Tables B. 23 - B. 25, Figures B. 20 - B. 23.

Table B. 22. Schedules of simulations of a direct file.

Redundant key handling method	HASH 1	HASH 2	HASH 3	HASH 4	Formulas
Linear i = -1	yes	yes	yes	yes	---
Linear i = -3	yes	---	yes	yes	---
Linear i = +1	yes	---	---	---	yes
Linear i = +1 group	yes	---	---	---	---
Random	yes	---	yes	yes	yes
Direct chain	yes	---	yes	yes	yes

4. Selected hash coding can be used for evaluation of direct disk file organization. According to the results of the simulations in (3), it is evident that Hash 1 gives the best results, average search length per probing.

Fectch	Extution			
1.375 +	0.00	ANA, S	77777 B Address	15 bit Hash
1.325 +	1.375	STA		Address: Hash address
1.375 +	0.687	SHA	-10	5 bits
1.375 +	0.687	SHA	-12	5 bit
1.375 +	1.375	STA	CYL	Cyl: 5 bit
1.375 +	1.375	LDA	Address	Hash address
1.375 +	0.000	ANA, S	277 B	8 bit
1.375 +	1.375	STA	Address	Address: 8 bit
1.375 +	1.375	LDA	CYL	5 bit
1.375 +	1.375	ADA	Address	5 bit, 8 bit
1.375 +	1.375	STA	Disk	Disk: 5 bit, 8 bits
1.375 +	0.000	ANA, S	77 B	8 bit
1.375 +	1.375	STA	LOC	LOC: 6 bit
1.375 +	1.375	LDA	Disk	5 bit, 5 bit
1.375 +	0.000	SHA	-3	5 bit, 5 bit
1.375 +	0.000	SHA	3	5 bit, 5 bit
1.375 +	1.375	STA	Disk	Disk: 5 bit, 5 bit (upper portion, lower portion)
Total ≃ 0.37 ms.				

Figure B. 19. Illustration of conversion of Hash address to Disk address.

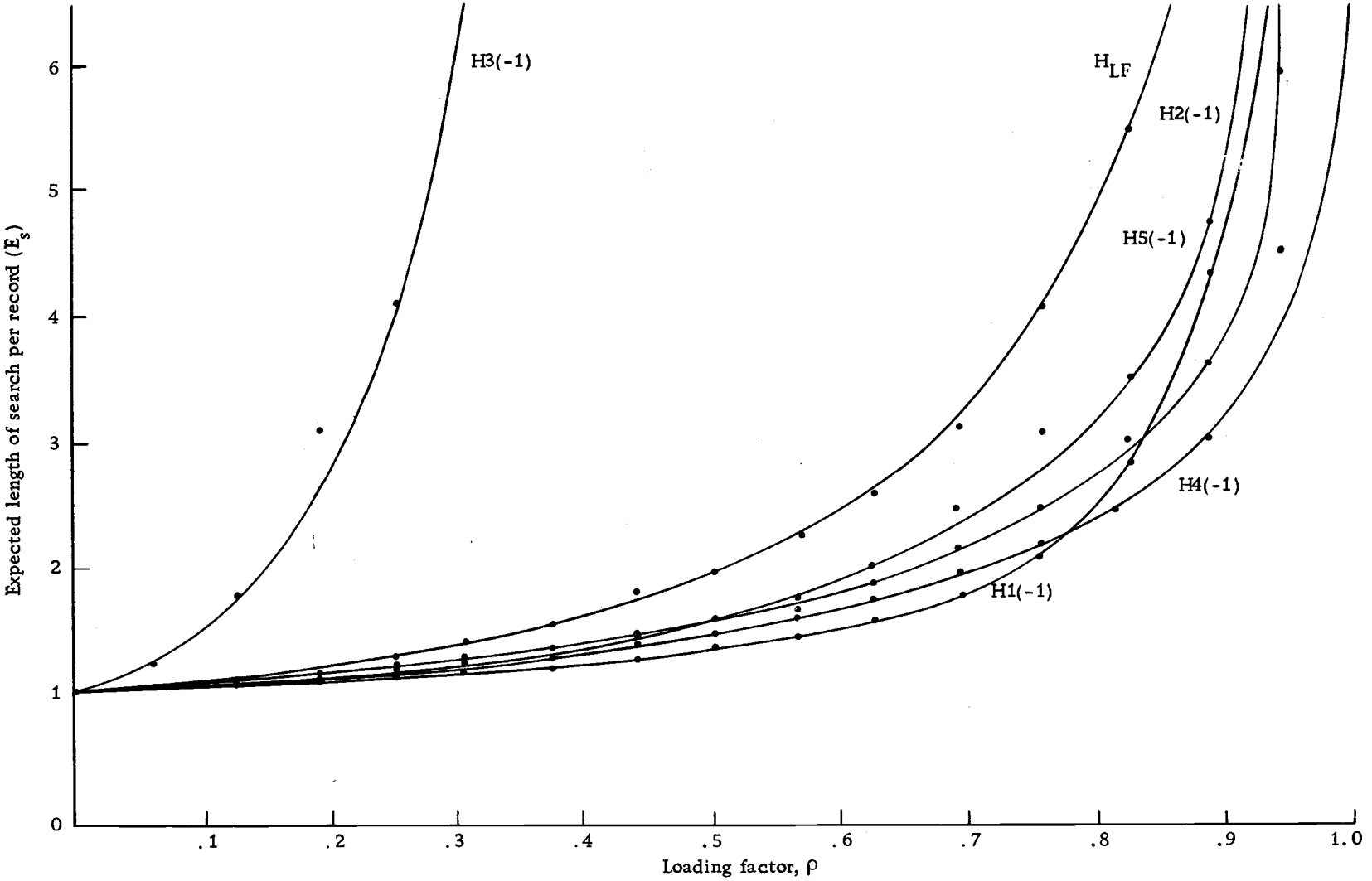


Figure B. 20. Expected length of search per record, as a function of the loading factor for 6 selected Hash functions with -1 displacement of records in the file.

Table B. 23. Results of computation of expected length of search as a function of loading factor for five hash function with linear probing compared with statistical formula.

Items N	File loading factor in %	Hash 3		Hash 1		Hash 2		Hash 5		Hash 4		
		S_T	$E = 1+\frac{S_T}{N}$	S_T	$E = 1+\frac{S_T}{N}$	S_T	$E = 1+\frac{S_T}{N}$	S_T	$E = 1+\frac{S_T}{N}$	S_T	$E = 1+\frac{S_T}{N}$	$E = \frac{1-\rho/2}{1-\rho}$
64	6.12	16	1.250	4	1.063	3	1.047	3	1.047	3	1.047	1.0
128	12.50	111	1.870	12	1.094	12	1.094	15	1.117	8	1.063	1.066
199	18.72	457	3.130	33	1.172	25	1.130	28	1.145	23	1.120	1.115
256	25.00	814	4.180	60	1.234	39	1.152	40	1.156	49	1.192	1.167
320	31.20	1917	6.990	95	1.297	82	1.257	59	1.184	78	1.244	1.227
384	37.50	4165	11.840	157	1.410	131	1.342	90	1.234	124	1.322	1.300
448	43.70	7254	17.190	217	1.485	213	1.476	126	1.281	166	1.437	1.389
512	50.00	16408	28.480	317	1.620	316	1.616	217	1.423	260	1.507	1.500
576	56.40	16580	33.910	456	1.790	463	1.800	335	1.581	397	1.690	1.647
640	62.50	23675	37.890	595	1.931	681	2.060	510	1.796	499	1.780	1.911
704	68.60	31552	45.810	910	2.292	1084	2.540	697	1.990	703	1.999	2.092
768	75.50	42556	56.410	1178	2.530	1659	3.160	1035	2.350	971	2.264	2.541
832	81.25	56866	69.340	1713	3.060	2237	3.680	1594	2.920	1265	2.520	3.167
896	87.50	70060	79.190	2463	3.760	3382	4.780	3023	4.380	1821	3.032	4.500
960	93.80	91673	96.490	4375	5.900	6897	8.160	6257	7.500	3397	4.538	8.565
1024	100.00	132298	130.190	13965	14.600	24974	25.300	20709	27.200	16508	17.121	--

S_T = Total number of searches before hit the desired record.

E = Average number of searches per random record accessing.

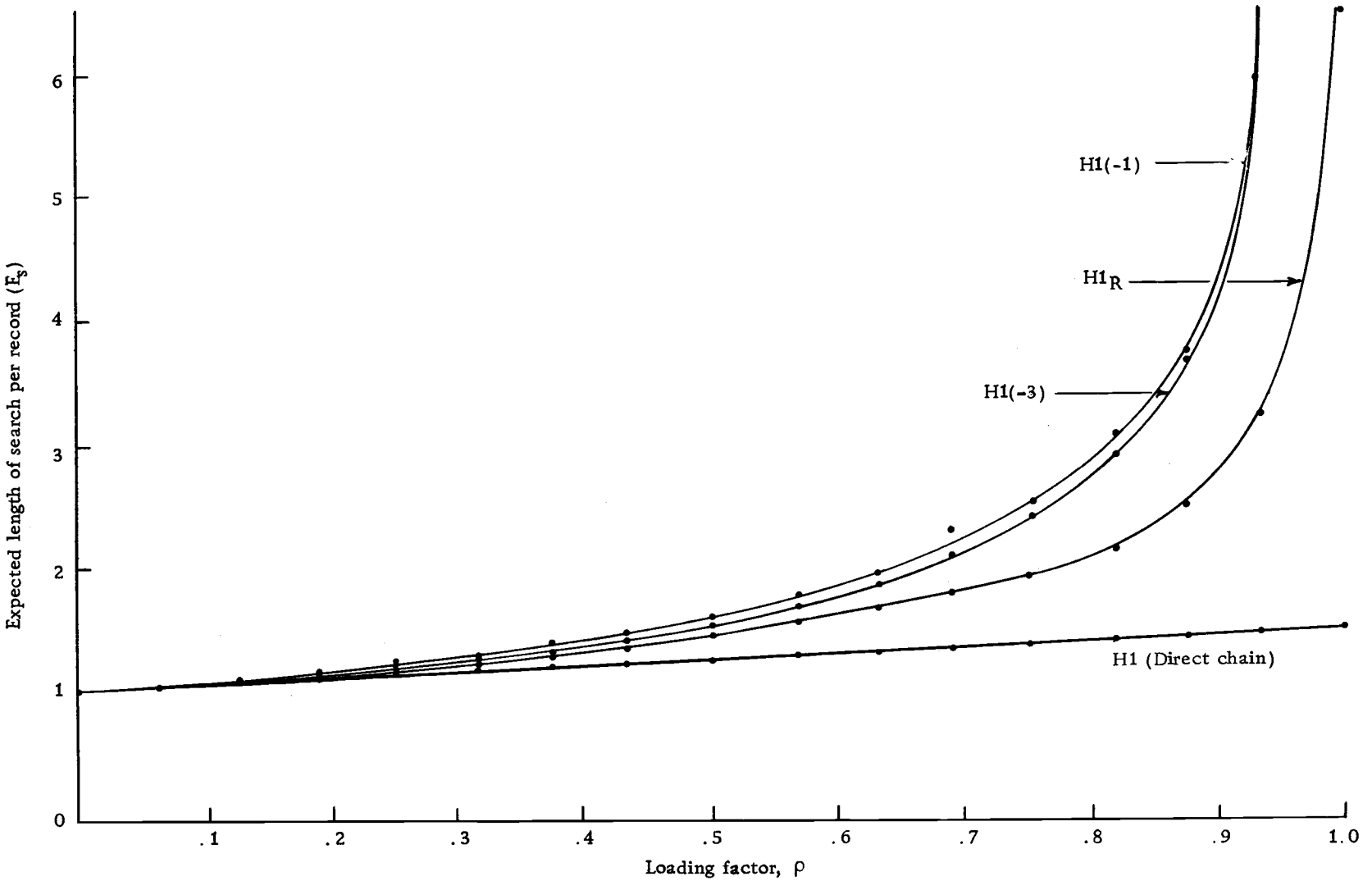


Figure B.21. Expected length of search per record, as a function of the loading factor for Hash 1 (H1) mapping function with 4 methods of handling redundant records in the file.

Table B. 24. Results of computation of expected length of search per record, as a function of loading factor for hash 1, with four methods of handling redundant records in the file.

N = No. items in file n_i	% full capacity	Hash 1, i = -1		Hash 1, i = 3		Hash, random		Hash 1, chain	
		S_T	$E = 1 + \frac{S_T}{N}$	S_T	$E = 1 + \frac{S_T}{N}$	S_T	$E = 1 + \frac{S_T}{N}$	S_T	$E = \frac{S_{c(T)}}{N}$
64	6.12	4	1.063	5	1.023	4	1.063	68	1.063
128	12.50	12	1.094	12	1.094	10	1.015	138	1.080
192	18.72	33	1.172	28	1.146	26	1.135	214	1.130
256	25.00	60	1.234	54	1.215	46	1.180	296	1.150
320	31.20	95	1.297	83	1.260	78	1.246	378	1.180
384	37.50	157	1.410	132	1.344	117	1.302	468	1.220
448	43.70	217	1.485	193	1.430	168	1.375	557	1.240
512	50.00	317	1.620	284	1.550	244	1.476	659	1.290
576	56.40	456	1.790	410	1.710	341	1.590	762	1.320
640	62.50	595	1.931	549	1.860	413	1.640	849	1.330
704	68.60	910	2.292	760	2.080	580	1.820	954	1.360
768	75.50	1178	2.530	1111	2.410	723	1.940	1066	1.390
832	81.25	1713	3.060	1591	2.910	951	2.140	1188	1.420
896	87.50	2463	3.760	2482	3.780	1390	2.500	1300	1.450
960	93.80	4375	5.900	4862	6.070	2139	3.220	1423	1.480
1024	100.00	13965	14.600	17962	18.600	5839	6.100	1556	1.520

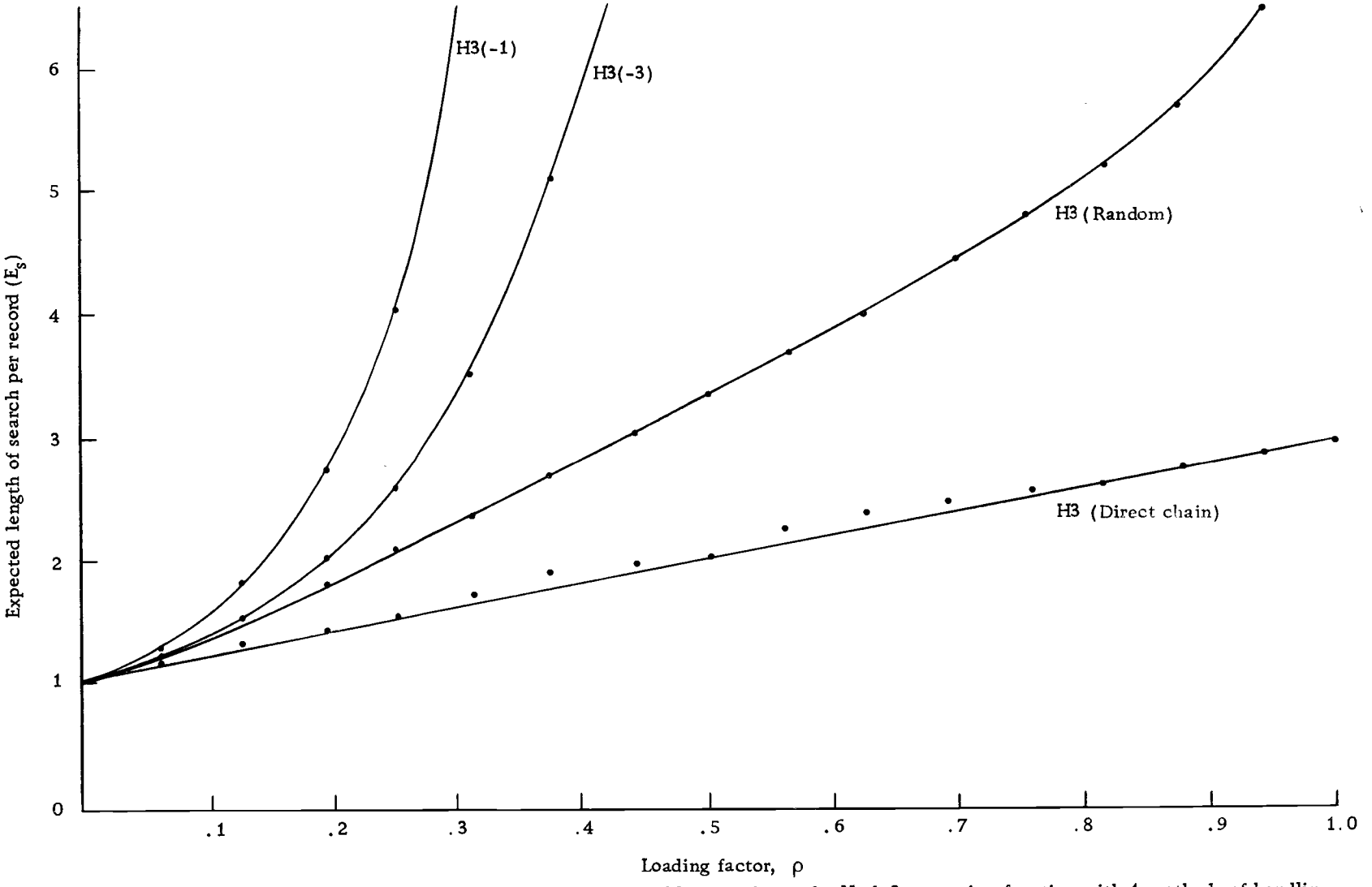


Figure B.22. Expected length of search per record, as a function of loading factor for Hash 3, mapping function with 4 methods of handling redundant records in the file.

Table B.25. Results of expected length of search per record, as a function of loading factor for hash 3 with four methods of handling redundant records in the file.

N = No. items in file n_i	% full capacity	Hash 3, (-1)		Hash 3, (-3)		Hash 3, random		Hash 3, chain	
		S_T	$E = 1 + \frac{S_T}{N}$	S_T	$E = 1 + \frac{S_T}{N}$	S_T	$E = 1 + \frac{S_T}{N}$	S_{cT}	$E = \frac{S_{c(T)}}{N}$
64	6.12	16	1.250	16	1.250	19	1.290	76	1.190
128	12.50	111	1.870	82	1.640	73	1.570	173	1.350
192	18.72	451	3.350	205	2.060	153	1.790	298	1.450
256	25.00	814	4.180	343	2.330	264	2.030	386	1.500
320	31.20	1917	6.990	852	3.660	478	2.500	556	1.730
384	37.50	4165	11.840	1688	5.390	704	2.830	730	1.900
448	43.70	7254	17.190	2715	7.060	898	3.000	884	1.970
512	50.00	16580	33.910	4131	9.060	1156	3.260	1069	2.080
576	56.40	16408	29.480	6144	11.660	1478	3.560	1291	2.240
640	62.50	23615	37.890	8759	14.680	1917	4.000	1547	2.420
704	68.60	31552	45.810	11500	15.330	2376	4.370	1782	2.530
768	75.50	42556	56.410	15398	21.040	2934	4.820	2040	2.650
832	81.25	56866	69.340	20057	25.100	3536	5.250	2274	2.740
896	87.50	70060	79.190	25090	29.000	4165	6.650	2495	2.780
960	93.80	91673	96.490	33558	35.950	5186	7.400	2741	2.850
1024	100.00	132298	130.790	50926	50.730	9576	10.350	3056	2.920

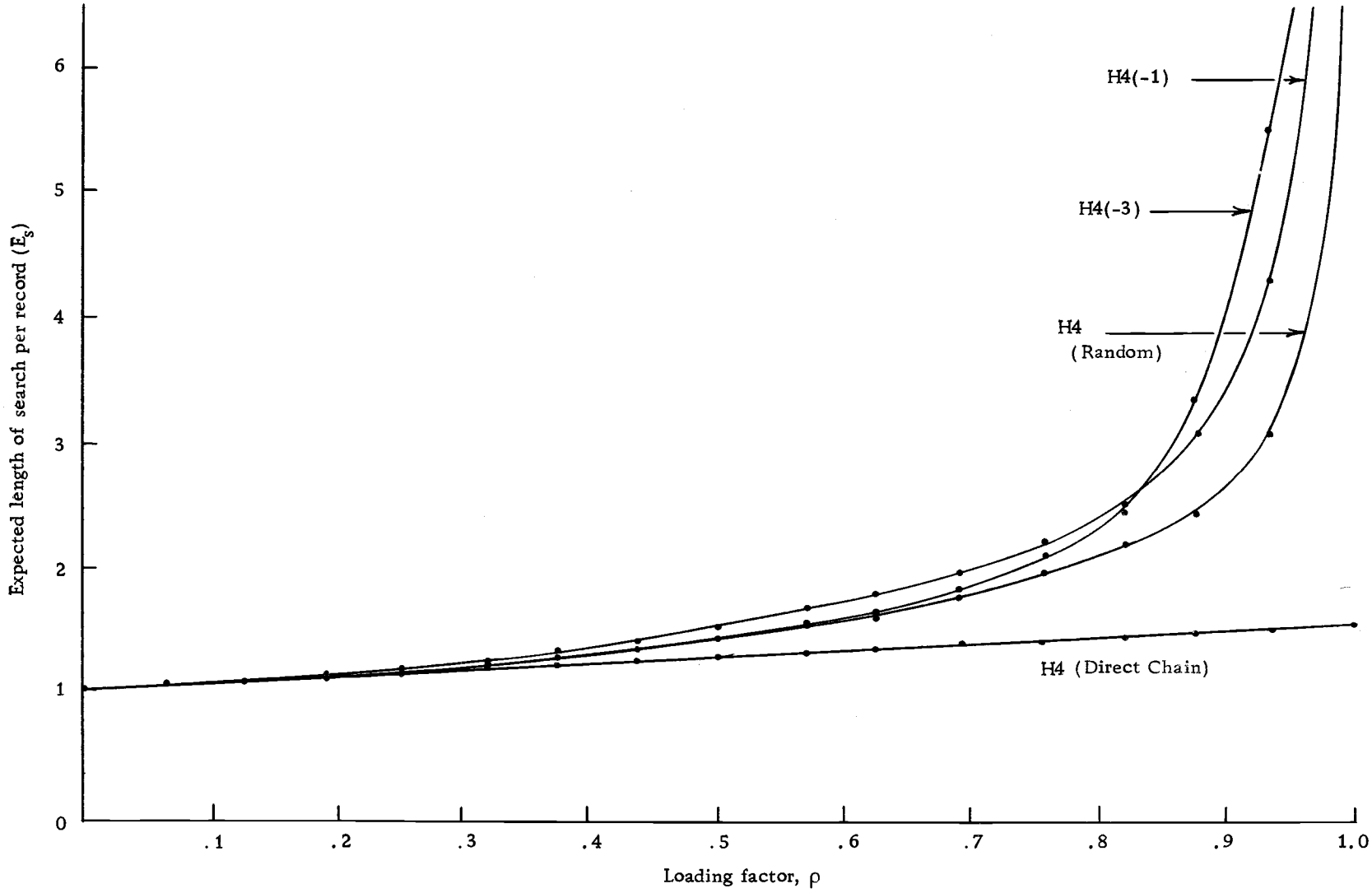


Figure B. 23. Expected length of search per record, as a function of the loading factor for Hash 4, with 4 methods of handling redundant records in the file.

Table B. 26. Results of computation of expected length of search record, as a function of loading factor for hash 4, with four methods of handling redundant records in the file.

N = No. items in file n_i	% full capacity	Hash 4, i = -1		Hash 4, 1 = -3		Hash 4, random		Hash 4, chain	
		S_T	$E = 1 + \frac{S_T}{N}$	S_T	$E = 1 + \frac{S_T}{N}$	S_T	$E = 1 + \frac{S_T}{N}$	S_{cT}	$E = \frac{S_{cT}}{N}$
64	6.12	3	1.047	4	1.063	3	1.047	67	1.047
124	12.50	8	1.063	8	1.063	7	1.056	135	1.055
192	18.72	23	1.120	21	1.110	22	1.115	209	1.089
256	25.00	49	1.191	45	1.170	49	1.191	291	1.137
320	31.20	78	1.244	76	1.238	84	1.263	376	1.175
384	37.50	124	1.323	112	1.292	129	1.336	459	1.195
448	43.70	166	1.370	154	1.343	171	1.382	549	1.225
512	50.00	260	1.510	215	1.420	243	1.475	644	1.258
576	50.40	397	1.690	314	1.545	319	1.554	750	1.302
640	62.50	499	1.780	410	1.640	407	1.636	852	1.331
704	68.60	703	1.998	585	1.830	568	1.807	968	1.375
768	75.50	971	2.200	865	2.130	753	1.986	1093	1.431
832	81.25	1265	2.520	1228	2.470	1011	2.215	1208	1.452
896	87.50	1821	3.040	2085	3.330	1328	2.482	1316	1.468
960	93.80	3397	4.300	4359	5.550	1972	3.054	1451	1.511
1024	100.00	16508	17.100	17108	17.700	6842	7.682	1580	1.543

Figure B. 20 makes a graphical comparison among five related hash functions with linear probing by (-1) and the expected value obtained from the linear probing formula $\frac{1 - \alpha/2}{1 - \alpha}$. This graph shows how the expected search per record retrieval ($E = 1 + \frac{ST}{N}$) increases when the number of records in the Table or the file loading factor increases. Every curve follows the general trend of E.; i. e., when α is a small there are many available spaces, the number of secondary records (redundant records) is less; then the expected number of search records per random accessing is less. But for $\alpha \longrightarrow 1.000$ the table is almost full; it is not easy to find a space available and the expected number of searched records per random accessing is enormously high.

Increase in the value of E, is dependent on the scattering pattern of the calculated hash addresses for each investigated hash function. Data in Table B. 23 indicated that Hash 1 is the best hash function. Hash 3 offered the worst results.

Figure B. 21 shows how the expected searched record per random accessing, E, increases as the values of file loading factor; α, increases in Hash 1, with three methods of handling redundant records: linear probing by (-1) and (-3), random probing and direct chain probing. The rate of increase of E is dependent on the method of handling redundant records from the results of both simulation and computation. Hash 1 with direct chain probing gives the best

results. Hash 1 with random probing gives worse results than Hash 1 with direct chain probing. Hash 1 with linear probing gives worse results than Hash 1 with random probing. Hash 1 with (-1) linear probing is a little worse than Hash 1 with (-3) linear probing for $0 < \alpha < 0.825$; it is better than Hash 1 with (-3) linear probing for $0.825 < \alpha < 1.000$ when the table is almost full.

Figure B. 21, Table B. 25 and Figure B. 22, Table B. 26, make a graphical comparison of Hash 3 and Hash 4, the objective of which is the same as that in Figure B. 19 and Table B. 23. From the comparative results shown in Table B. 23 - B. 26 and Figures B. 19 - B. 22 it is clear that Hash 1 gives the best results and is the reasonable selection for evaluating the direct disk file, for making a comparison with the results of the other types of files as shown in Appendix B, Example 4.

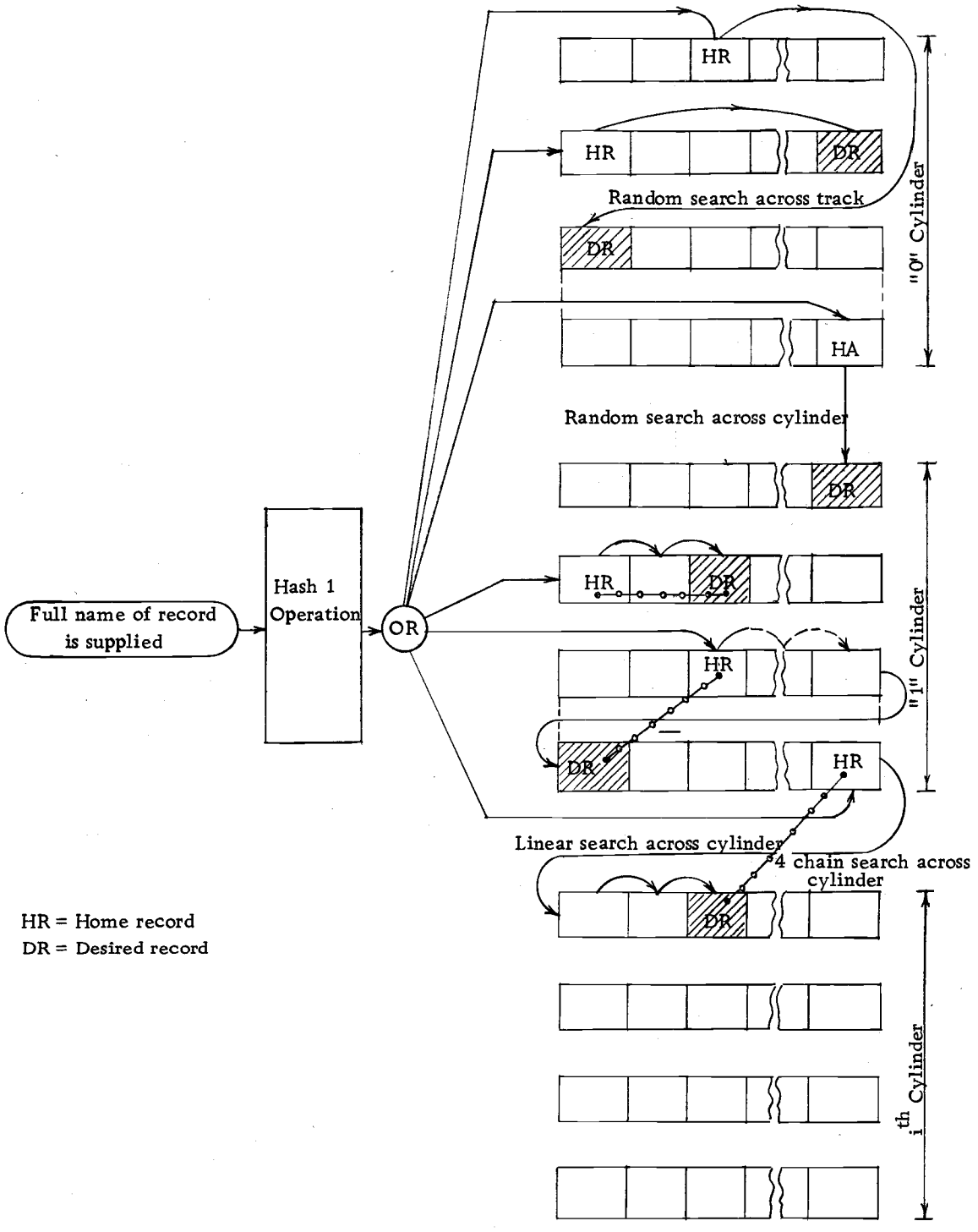


Figure B.24. Illustration of search across the track and search across the cylinder of direct disk file using Hash 1, with linear probing, random probing and direct chain probing.

Table B.27. Percent of search across the track and percent of search across the cylinder of disk direct file using Hash 1 with linear probing, random probing and direct chain probing.

File size in records	Loading factor, α (%)	Linear probing (+1)				Random probing				Direct chain probing			
		number of search across track, %		number of search across cylinder, %		number of search across track, %		number of search across cylinder, %		number of search across track, %		number of search across cylinder, %	
64	6.12	0	0.000	0	0.000	0	0.000	0	0.000	0	0.000	0	0.000
128	12.50	0	0.000	0	0.000	0	0.000	0	0.000	0	0.000	0	0.000
192	18.72	0	0.000	0	0.000	3	1.563	0	0.000	0	0.000	0	0.000
256	25.00	1	0.390	0	0.000	5	1.953	1	0.390	1	0.390	0	0.000
320	31.20	3	0.938	0	0.000	8	2.500	2	0.625	3	0.938	0	0.000
384	37.50	3	0.781	0	0.000	12	3.125	3	0.781	3	0.781	0	0.000
448	43.70	4	0.893	0	0.000	14	3.125	3	0.670	4	0.893	0	0.000
512	50.00	9	1.758	0	0.000	22	4.297	3	0.586	9	7.758	0	0.000
576	50.40	12	2.083	0	0.000	29	5.035	5	0.868	12	2.083	0	0.000
640	62.50	14	2.188	2	0.195	37	5.781	5	0.787	14	2.188	2	0.195
704	68.60	18	2.557	4	0.568	50	7.102	11	1.563	18	2.557	4	0.568
768	75.50	24	3.125	5	0.651	67	8.734	17	2.214	24	3.125	5	0.651
832	81.25	36	4.327	5	0.607	93	11.178	25	3.005	36	4.327	5	0.601
896	87.50	45	5.022	7	0.781	115	72.835	36	4.018	45	5.022	7	0.781
960	93.80	62	6.458	12	1.250	146	15.208	52	5.417	62	6.458	12	1.250
1024	100.00	91	8.887	19	1.855	175	17.090	80	7.813	91	8.887	19	1.855

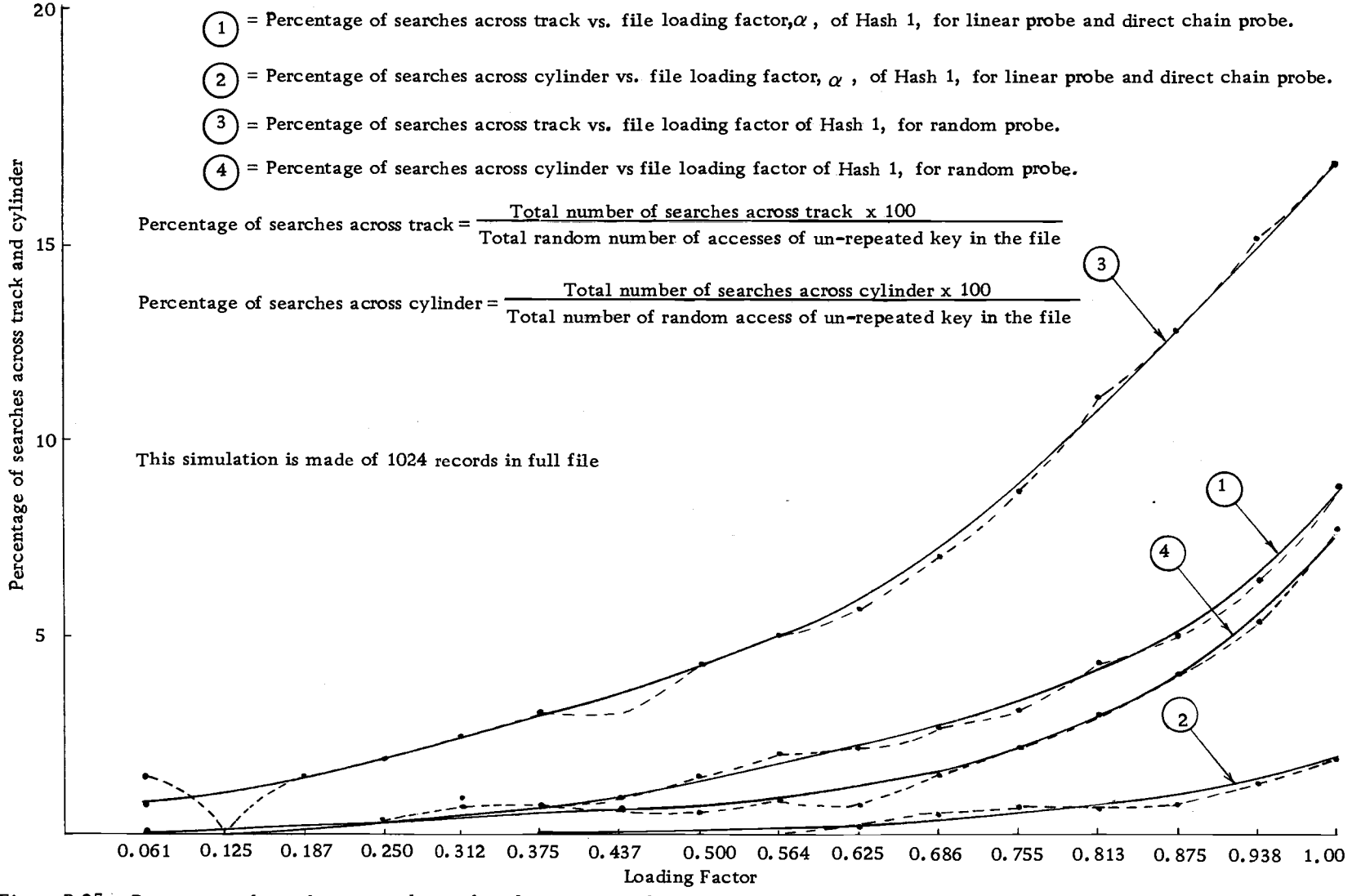


Figure B.25. Percentage of searches across the track and percentage of searches across the cylinder as a function of loading factor, α, for disk direct file using Hash 1, with linear probe, random probe and direct chain probe.

4. Selected hash coding can be used for evaluation of direct disk file organization. According to the results of the simulation in (3), it is evident that Hash 1 gives the best results, average search length per record retrieval, E, and follow its statistical formula. Hence it is reasonable to select Hash 1 for "selected hash coding", for evaluation and comparison with other file organization methods.

5. The evaluation of the direct disk file using Hash 1 with linear probing is as follows:

   a) For the computation of the average throughput time of direct disk file with linear probing,

   If $T_{ATHRPDFLP}$ = Average throughput time per record retrieval from direct disk file with linear probing

   from the diagram Figure B. 24, the following equation can be set up:

$$T_{ATHRPDFLP} = T_{CBF} + T_{CFNTFK} + T_{HASH\ 1} + T_{RCLU} + T_{ASRDFLP} + T_{RCI/O} + T_{RWOUT} \qquad (9.50)$$

   where $T_{HASH\ 1}$ = Hash 1 decoding time per record retrieval

= 0.071 ms (based on the tested program on page 352.

$T_{ASRDFLP}$ = Average access time per record including internal search with linear probing and search across track and cylinder.

$$= T_{AR/WHPT} + T_{AWT} + T_{RINDT} + T_{LS} + T_{LPSACT} + T_{LPSACYL} \quad (4.51)$$

where T_{LPSACT} = Linear probing search across track

$$= \frac{E_1}{100} (T_{AWT} + T_{RINDT})$$

$$= \frac{(25 + 25)}{100} \quad E_1 = 0.5\, E_1$$

E_1 = Total number or searches across the track using linear probing / Total random accesses of un-repeated keys in the file

x 100

E_1, the empirical parameter, is obtained by simulation.

and $T_{LPSACYL}$ = Linear probing search across the cylinder

$$= \frac{E_2}{100} (T_{AR/WHPT} + T_{AWT} + T_{RINDT}) \quad (9.52)$$

$$= \frac{E_2}{100} (30 + 12.5 + 25) = 0.675\ E_2$$

where

$E_2 = \dfrac{\text{Total number of searches across the cylinder using linear probing}}{\text{Total random accesses of un-repeated keys in the file}}$

x 100

E_1 and E_2, the empirical values, are obtained by simulation

Then Equation (9.51) becomes

$$T_{ASRDFLP} = 95 + 12.5 + 25 + T_{LPS} + 0.5\ P_{LPSACT} + 0.675\ P_{LPSACYL}$$

$$= 132.5 + T_{LPST} + 0.5\ E_1 + 0.675\ E_2$$

For illustration, when N = 12288 (α = 0.75) from Figure B.24, page 314, E_1 = 3.125%, E_2 = 0.651 %.

T_{LPS} = Linear search in the track containing the desired record

$$= 7.875\ (E_{LP} - 1) + 165.625$$

where E_{LP} = Expected number of searcher per record retrieval with linear probing

$E_{LP} = \dfrac{(1 - \alpha/2)}{(1 - \alpha)}$ α = file loading factor (9.53)

= 2.5 for N = 12288

Then T_{LPS} = 7.875 (2.5 - 1) + 165.625 =

177 μsec = 0.177 msec.

$T_{ASRDFLP}$ = 132.5 + 0.177 + 0.5x3.125 +

0.675x 0.651

= 134.679 ms

From Equation (9.50)

$T_{ATHRPDFLP}$ = 0.3218 + 8.626 + 0.071 + 0.008 +

134.679 + 0.008 + 36.00

= 179.714 for N = 12288 or α = 0.75

See results of computation in Table B.29, page 330.

b) Approximate formula of $T_{ATHRPDFLP} = T_6$

Only major components of search time (msec) are considered; the minor components of search time (μsec) are omitted.

$T_6 \simeq T_{AR/WPT} + T_{AWT} + T_{RINDT} +$

$T_{LPS} + 44.959$

where 44.969 $\simeq T_{CBF} + T_{CFNTFR} + T_{RCLU} +$

$T_{RCI/O} + T_{RWOUT} + T_{HASH\ 1}$

$T_6 \simeq 95 + 12.5 + 25 + T_{LPS} + 0.5 \times$

$10^{-2} E_1 + 0.675 \times 10^{-2} E_2$

$\simeq 132.5 + (7.875\ E_{LP} + 157.375)$

$10^{-3}\ 0.005\ E_1 + 0.00675\ E_2 + 44.959$

$$T_6 \simeq 177.545 + 0.007875\, E_{LP} + 0.5\, E_1 + 0.675\, E_2 \text{ ms}$$

for $0.25 < \alpha \leqslant 1.0$

$$\simeq 177.545 \text{ ms for } 0.0 < \alpha < 0.25$$

c) Total CPU busy time per record accessed using unique fixed-length key, $t_{TCPUTDFLP}$

$$t_{TCPUTDFLP} = T_{HASH\,1} + T_{RCLU} + T_{LPS} + T_{RCI/O} + T_{RWOUT} \qquad (9.54)$$

$$= 0.071 + 0.008 + 0.177 + 0.008 + 36.00$$

$$= 36.264 \text{ msec.} \quad N = 12288, \quad \alpha = 0.75$$

d) Total CPU busy time per record accessed using full name of the record, $t_{TCPUTDFLPUFN}$

$$t_{TCPUTDFLPUFN} = t_{TCPUTDFLP} + T_{CFNTFN}$$

$$= 36.264 + 8.626 = 44.890 \text{ msec.}$$

for $N = 12288$, $\alpha = 0.75$

6. The evaluation of direct disk file using Hash 1 with random probing is as follows:

a) For computation of the average throughput time of the direct disk file with linear probing

If $T_{ATHRPDFRP}$ = Average throughput time per record retrieval from direct disk file with random probing.

From the time diagram Figure B. 25, page the following equation can be set up:

$$T_{ATHRPDFRP} = T_{CBF} + T_{CFNTFK} + T_{HASH\ 1} + T_{RCLU} + T_{ASRDFRP} + T_{RCI/O} + T_{RWOUT} \quad (9.55)$$

where $T_{ASRDFRP}$ = Average access time per record retrieval including internal search with random probing and search across track and cylinder.

$$= T_{AR/WHPT} + T_{AWT} + T_{RINDT} + T_{LS} + T_{RPSACT} + T_{RPSACYL} \quad (9.56)$$

where T_{RPSACT} = Random probing search across track

$$= \frac{E_3}{100} (T_{AWT} + T_{RINDT}) = 0.5\ E_3$$

$$E_3 = \frac{\text{Total number of searches across the track of random probing}}{\text{Total random accesses of un-repeated keys in the file}} \times 100$$

and $T_{RPSACYL}$ = Random probing search across the cylinder.

$$= \frac{E_4}{100} \; (T_{AR/WHPT} + T_{AWT} + T_{RINDT}) \qquad (9.57)$$

$$= \frac{E_4}{100} \; (30 + 12.5 + 25) = 0.675 \; E_4$$

where

$$E_4 = \frac{\text{Total number of searches across cylinder of random probing}}{\text{Total random accesses of un-repeated keys in the file}} \times 100$$

E_3, E_4 the empirical values, are obtained by simulation, Figure B.24 on page 314.

Then $T_{ASRDFRP} = 95 + 12.5 + 25 + T_{RPS} + 0.5 \; E_3 + 0.675 \; E_4$

$$= 132.5 + T_{RPS} + 0.5 \; E_3 + 0.675 \; E_4$$

For illustration when N = 12288 ($\alpha = 0.75$) from Figure B.24, page 314.

E_3 = 8.734%; E_4 = 2.217 %

T_{RPS} = Random search time in the track containing the desired record

= 34.625 (E_{RP} - 1) + 179.375 (based on test program, page 354.

where E_{RP} = Expected number of searches per record retrieval using random probing.

$E_{RP} = -\frac{1}{\alpha} \log_e (1 - \alpha)$ where α = file loading factor, (9.58)

$= 1.84$ for N = 12288.

then $T_{RPS} = 34.625 (1.84 - 1) + 179.375 =$

$208.46 + 0.209$ ms.

$T_{ASRDFRD} = 132.5 + 0.209 + 0.5 * 8.734 +$

$0.675 * 2.217$

$= 138.571$ msec.

then $T_{ATHRPDFRP} = 0.3218 + 8.626 + 0.071 + 0.008 +$

$138.571 + 0.008 + 3600$

$= 183.606$ msec. for N = 12288

or $\alpha = 0.75$

See the results of computation of $T_{ATHRPDFRP}$ in Table B.33 on page 334.

b) Approximate formula of $T_{ATHRPDFRP} = T_7$

Only major components of search time (msec) are considered; the minor components of search time (μsec) are omitted.

$T_7 \simeq T_{AR/WPT} + T_{AWT} + T_{RINDT} + T_{RPS} + 44.959$

where $44.959 \simeq T_{CBF} + T_{CFNTFR} + T_{RCLU} + T_{RCI/O} + T_{RWOUT} + T_{HASH\ 1}$

$$T_7 \simeq 95 + 12.5 + 25 + (34.625\, E_{RP} + 155.75)\, 10^{-3} + 0.5\, E_3 + 0.675\, E_4 + 44.959$$

$$T_7 \simeq 177.604 + 0.0346\, E_{RP} + 0.5\, E_3 + 0.675\, E_4 \text{ for } 0\ \alpha < \alpha < 1.000$$

$$T_7 \simeq 177.604 \quad \text{for } 0 < \alpha < 0.25$$

c) Total CPU busy time per record accessed using unique fixed-length key = $t_{TCPUTDFRP}$.

$$t_{TCPUTDFRP} = T_{HASH\ 1} + T_{RCLU} + T_{RPS} + T_{RCI/O} + T_{RWOUT}$$

$$= 0.071 + 0.008 + 0.209 + 0.008 + 36.00$$

$$= 36.296 \text{ msec for } N = 12288 \text{ or } \alpha = 0.75$$

d) Total CPU busy time per record accessed using full name of the record = $t_{TCPUTDFRPUFN}$.

$$t_{TCPUTDRDUFN} = t_{TCPUTDFRP} + T_{CFNTFK}$$

$$= 36.296 + 8.626 = 44.922 \text{ msec}$$

for $N = 12288$, $\alpha = 0.75$

7. The evaluation of direct disk file using Hash 1 with direct

chain probing is as follows:

a) For the computation of the average throughput time of direct file with direct chain probing,

If $T_{ATHRPDFDCH}$ = Average throughput time per record retrieval from direct disk file with random probing

From the time diagram Figure B. 25 the following equation can be set up:

$$T_{ATHRPDFDCH} = T_{CBF} + T_{CFNTFK} + T_{HASH\ 1} + T_{RCLU} + T_{ASRDFRP} + T_{RCI/O} + T_{RWOUT} \quad (9.59)$$

where $T_{ASDFDCH}$ = Average access time per record retrieval including internal search with direct chain probing and search across track and clyinder.

$$= T_{AR/WHPT} + T_{AWT} + T_{RINDT} + T_{DCH} + T_{RPSACT} + T_{RPSACYL}$$

where $T_{DCHSACT}$ = Direct chain probing search across track

$$= \frac{E_5}{100} (T_{AWT} + T_{RINDT}) = 0.5\ E_5 \quad (9.60)$$

where E_5 = $\frac{\text{Total number of searches across the track of direct chain}}{\text{Total number of un-repeated records accessed from the file}}$

x 100

$T_{DCHSACYL}$ = Direct chain search time across the cylinder

$$= \frac{E_6}{100} (T_{AR/WHPT} + T_{AWT} + T_{RINDT}) \qquad (9.61)$$

$$= \frac{E_6}{100} (30 + 12.5 + 25) = 0.675 \; E_6 \text{ msec.}$$

where

E_6 = $\frac{\text{Total number of searches across the cylinder of direct chain probing}}{\text{Total number of un-repeated records accessed from file}}$

x 100

E_5 and E_6 are the empirical values obtained by simulation as shown in Figure B.24, page 314. In this investigation the method of finding the available space for successive records in the chain is the same as that of linear probing. Hence the value of $E_1 = E_5$ and $E_2 = E_6$.

Then $T_{ASRDFCHD}$ = $95 + 12.5 + 25 + T_{DCHS} + 0.5\, E_5 + 0.675\, E_6$

$$= 132.5 + T_{DCHS} + 0.5\,E_6 + 0.675\,E_6$$

For illustration when N = 12288 ($\alpha = 0.75$) from Figure B.24, page 314.

$E_5 = E_1 = 3.125\%$; $E_6 = E_2 = 0.651\%$

T_{DCHS} = Direct chain probing in the track containing the desired record

$$= 164.25 + 19.875\,(E_{DCH} - 1)$$

where E_{DCH} = Expected number of searches per record retrieval using direct chain probing

$$E_{DCH} = 1 + \frac{\alpha}{2} \quad \text{where } \alpha = \text{file loading factor} \qquad (9.62)$$

$$= 1.375 \quad \text{for N = 12288 or } \alpha = 0.75$$

Then $T_{DCHS} = 164.25 + 19.875\,(1.375 - 1) =$ 171.703 μsec = 0.172 msec.

$T_{ASRDFDCH} = 132.5 + 0.172 + 0.5 \times 3.125 + 0.675 \times 0.651$

$= 134.673$ msec.

Then $T_{ATHRPDFDCH} = 0.3218 + 8.626 + 0.071 + 0.008 + 36.00$

$= 179.708$ msec.

See the results of computation of $T_{ATHRPDFDCH}$ in Table B.36 on page 337.

b) Approximate formula of $T_{ATHRPDFDCH} = T_8$

Only major components of search time (msec) are considered; the minor components of search time (μsec) are omitted.

$$T_8 \simeq T_{AR/WPT} + T_{AWT} + T_{RINDT} + T_{DCHS} + 44.959$$

where $44.959 \simeq T_{CBF} + T_{CFNTFK} + T_{RCLU} + T_{RCI/O} + T_{RWOUT} + T_{HASH\ 1}$

$$T_8 \simeq 95 + 12.5 + 25 + (19.875\ E_{DCH} + 144.375) \times 10^{-3} + 0.5\ E_5 + 0.675\ E_6 + 44.959$$

$$T_8 \simeq 177.573 + 0.019875\ E_{DCH} + 0.5\ E_8 = 0.675\ E_6 \text{ msec. for } 0.5 < \alpha < 1.00.$$

$$T_8 \simeq 177.573 \text{ msec for } 0 < \alpha < 0.5$$

c) Total CPU busy time per record accessed using unique fixed-length key = $t_{TCPUTDFDCHP}$.

$$t_{TCPUTDFDCHP} = T_{HASH\ 1} + T_{RCLU} + T_{DCHS} + T_{RCI/O} + T_{RWOUT}$$

= 36.259 + 8.626

= 44.885 msec for N = 12288,

α = 0.75

8. For computation of core storage space and disk storage space required for the direct disk file using linear probing, random probing and direct chain probing, the method is the same as that in Example 1a. The extra disk storage space is reserved for supporting the file system processing program. See the results of computation in Tables B.29, B.33 and B.37 on pages 330 - 338.

9. For computation of the achievable-throughput rate capability of the direct disk file $C_{(6)}$, $C_{(7)}$ and $C_{(8)}$ and the customer operating cost per call, unit cost, $U_{C(6)}$, $U_{C(7)}$ and $U_{C(8)}$ the methods are the same as those shown in Example 1a, pages 226 - 227. The computing results of $C_{(6)}$, $C_{(7)}$ and $C_{(8)}$ are shown in Table 7.4, page 155. The computed results of $U_{C(6)}$, $U_{C(7)}$ and $U_{C(8)}$ are shown in Tables B.30, B.34 and B.38, pages 331 - 339 respectively. for comparison with the results obtained from the various other types of file. See results of computation in Table 7.6, page 173.

Table B. 28. Data results of computation of disk average access time per record retrieval as a function of loading factor of direct disk file using Hash 1 with Linear probing (+1)

File size in records	Loading factor α	File supporting space in		Average search per random record access, with Hash I, E	Cylinder position time in	Track waiting time in	Read-in one track track time	Average internal search time/record	Search across track time		Search across cylinder time		Disk search time per record
		cylinder = tracks			ms	ms	ms	ms	%	ms	%	ms	ms
128	0.0078	27	261	1.000	95	12.5	25	0.166	-	-	-	-	132.666
512	0.0312	27	261	1.020	95	12.5	25	0.166	-	-	-	-	132.666
1024	0.0625	27	261	1.030	95	12.5	25	0.166	-	-	-	-	132.666
4096	0.2500	27	261	1.167	95	12.5	25	0.167	-	-	-	-	132.667
8192	0.5000	27	261	1.500	95	12.5	25	0.170	1.785	0.893	0.000	0.000	133.563
12288	0.7500	27	261	2.500	95	12.5	25	0.177	3.125	1.563	0.651	0.439	134.679
16384	1.0000	27	261	14.600*	95	12.5	25	0.273	8.887	4.444	1.855	1.252	138.469

Note: $E = (1 - \alpha/2)/(1 - \alpha)$ for linear probing.

* The value is obtained by simulation.

(Direct file 16 words/record, 64 records per track).

Table B.29. Data results of computation of CPU busy time and average throughput time per record retrieval of direct disk file organization with Linear probing using full name of a record in accessing a random record from file.

File size in records	Communication time back and forth in -	Full name of a record decoding time in -	Hash 1 decoding time in -	Connecting logical limit disk time	Average disk access + internal search time per record	Checking I/O logical unit time	A record write out time	Using fixed-length key CPU busy time/ record	Using full name of a record CPU busy time	Using full name of a record average through-put time
	ms	ms	ms	ms	ms	ms	ms	ms	ms	ms
128	0.3218	8.395	.071	.008	132.666	.008	36	36.253	44.648	177.470
512	0.3218	8.475	.071	.008	132.666	.008	36	36.253	44.728	177.550
1024	0.3218	8.515	.071	.008	132.666	.008	36	36.253	44.768	177.590
4096	0.3218	8.595	.071	.008	132.667	.008	36	36.254	44.849	177.671
8192	0.3218	8.601	.071	.008	133.563	.008	36	36.257	44.858	178.573
12288	0.3218	8.626	.071	.008	134.679	.008	36	36.264	44.890	179.714
16382	0.3218	8.641	.071	.008	138.469	.008	36	36.360	45.001	183.519

Note: ms or msec = millisecond.

T_{HASH1} = Hash 1 decoding time + hash address to disk address conversion time, see Figure B.18, page

= 0.34 ms + 0.37 ms = 0.071 msec.

Table B.30. Data results of computation of storage space required as the function of file size of direct file organization with linear probing using both record's full name and fixed length numerical code for accessing a record from file.

File size	Full name to key S_{CFNFK}		Hash 1 program space		Hash code to disk address conversion	Main file search with linear probing		Total space required without using full name	Total space with using full name of a record
	words (a)	words (b)	words (a)	words (b)		a	b		
128	200	1088	24	2	20	61	1024	1131	2419
512	200	4352	24	2	20	61	1024	1131	5663
1024	200	8704	24	2	20	61	1024	1131	10035
4096	200	34816	24	2	20	61	1024	1131	36147
8192	200	69632	24	2	20	61	1024	131	70963
12288	200	104448	24	2	20	61	1024	1131	105779
16384	200	139264	24	2	20	61	1024	1131	140696

Table B.31. Results of computation of customer operating cost (unit cost) as the function of file loading factor of direct disk file with linear probing (+ 1) using full name of record in accessing.

File loading factor	Disk space for data file	Disk space for accessing program	Disk space charge per month	Terminal devices cost per month	Rule of use data file	CPU-time charge per month	CPU-time and disk charge per month	CPU-time and disk charge per month per call
unit	tracks	tracks	$	$	calls per month	$	$	$
.0078 (128 records)	261	1.362	78.71	760	52500	195.34	1034.05	1.96960
				2675	210000	781.35	3535.06	1.68336
				4075	420000	1562.67	5716.38	1.36104
.0312 (512 records)	261	4.530	79.66	760	52500	195.69	1035.35	1.97209
				2675	210000	782.73	3537.39	1.68447
				4075	420000	1565.49	5720.15	1.36194
.625 (1024 records)	261	8.800	80.94	760	52500	195.87	1036.81	1.97480
				2675	210000	783.45	3539.39	1.68542
				4075	420000	1565.87	5721.81	1.36233
.2500 (4096 records)	261	34.300	88.59	760	52500	196.23	1044.82	1.99013
				2675	210000	784.86	3548.45	1.68973
				4075	420000	1569.72	5733.31	1.36507
.5000 (8192 records)	261	68.300	98.79	760	52500	196.26	1055.05	2.00961
				2675	210000	785.01	3558.98	1.69475
				4075	420000	1570.02	5743.79	1.36757
.7500 (12288 records)	261	102.300	108.99	760	52500	196.41	1065.40	2.02933
				2675	210000	785.58	3569.57	1.69979
				4075	420000	1571.16	5755.15	1.37027
1.0000 (16384 records)	261	136.300	119.19	760	52500	196.89	1076.08	2.04967
				2675	210000	787.54	3581.73	1.70558
				4075	420000	1575.03	5801.92	1.37362

Table B.32. Data results of computation of disk average access time per record retrieval as a function of loading factor of direct disk file using Hash 1 with random probing.

File size in records	Loading factor, α	File supporting space in cylinder = tracks		Average search per random record access	Cylinder positioning time in -	Track waiting time in -	Read-in one track time	Average internal search time per record	Search across track time in -		Search across cylinder time in -		Disk search time per record
unit				with Hash 1, E	ms	ms	ms	ms	%	ms	%	ms	ms
128	0.0078	27	261	1.03	92	12.5	25	0.180	-	-	-	-	132.680
512	0.0312	27	261	1.04	95	12.5	25	0.181	-	-	-	-	132.681
1024	0.0625	27	261	1.04	95	12.5	25	0.181	-	-	-	-	132.681
4096	0.2500	27	261	1.16	95	1.25	25	0.185	1.953	0.977	0.390	0.263	133.925
8192	0.5000	27	261	1.39	95	12.5	25	0.193	4.297	2.149	0.586	0.396	135.238
12288	0.2500	27	261	1.84	95	12.5	25	0.209	8.734	4.363	2.214	1.493	138.571
16384	1.0000	27	261	6.70*	95	12.5	25	0.377	17.090	8.545	7.812	5.274	146.696

Note: $E = -\alpha^{-1} \log_e (1 - \alpha)$ for random probing.

* The value is obtained from simulation.

Direct file random probing uses 16 words per record, 64 record per track.

Table B.33. Data results of computation of CPU busy time and average throughput time per record retrieval of direct disk file organization with random probing using full name of a record in accessing a random record from file.

File size in records	Communication time back and forth	Full name of a record decoding	Hash 1 decoding time	Connecting logical unit disk time	Average disk access internal search time per record	Checking I/O logical unit disk	A record write-out time	Using fixed-length key CPU busy time record	Using full name of a record CPU busy time	Using full name of a record average throughput time
unit	ms	ms	ms	ms	ms	ms	ms	ms	ms	ms
128	0.3218	8.395	.071	.008	132.180	.008	36	36.267	44.662	177.484
512	0.3128	8.475	.071	.008	132.681	.008	36	36.268	44.743	177.515
1024	0.3218	8.515	.071	.008	132.681	.008	36	36.268	44.783	177.605
4096	0.3218	8.595	.071	.008	133.925	.008	36	36.272	44.267	178.929
8192	0.3218	8.601	.071	.008	135.238	.008	36	36.280	44.881	180.248
12288	0.3218	8.626	.071	.008	138.571	.008	36	36.296	44.922	183.606
16384	0.3218	8.641	.071	.008	146.696	.008	36	36.464	45.105	191.746

Table B.34. Data results of computation of storage space required as the function of file size of direct file organization with random probing using both record's full name and fixed-length numerical code in accessing.

File size	Full name to fixed-length key		Hash 1 space required		Main file search random probing		Space required for disk address task	Total space for using fixed-length key	Total core space for using full name of record
unit	words (a)	words (b)	words (a)	words (b)	words (a)	words (b)	words	words	words
128	200	1088	24	2	80	1024	20	1148	2438
512	200	4352	24	2	80	1024	20	1148	5702
1024	200	8704	24	2	80	1024	20	1148	10054
4096	200	34816	24	2	80	1024	20	1148	36166
8196	200	69632	24	2	80	1024	20	1148	70982
12288	200	104448	24	2	80	1024	20	1148	105798
16384	200	139264	24	2	80	1024	20	1148	140614

Table B.35. Result of computation of customer operating cost (unit cost) as the function of file loading factor of direct disk file with random probing using full name of record in accessing.

File loading factor	Disk space for data file	Disk space for accessing program	Disk space charge per month	Terminal devices cost per month	Rule of use data file	CPU-time charge per month	CPU-time and disk charge per month	CPU-time and disk charge per month per call
unit	tracks	tracks	tracks	$	calls per month	$	$	$
.0078 (128 records)	261	1.381	78.71	760	52500	195.39	1034.10	1.96970
				2675	210000	781.59	3535.30	1.68347
				4075	420000	1563.18	5716.89	1.36116
.0312 (512 records)	261	4.568	79.67	760	52500	195.75	1035.42	1.97222
				2675	210000	783.00	3537.67	1.68560
				4075	420000	1566.00	5720.67	1.36206
.625 (1024 records)	261	8.818	80.96	760	52500	195.93	1036.87	1.97499
				2675	210000	783.69	3539.63	1.68553
				4075	420000	1567.41	5723.35	1.36270
.2500 (4096 records)	261	34.318	88.60	760	52500	196.29	1044.89	1.99026
				2675	210000	785.16	3548.76	1.68988
				4075	420000	1570.35	5733.95	1.36522
.5000 (8192 records)	261	68.318	98.80	760	52500	196.35	1055.15	2.00098
				2675	210000	785.43	3559.23	1.69487
				4075	420000	1570.83	5744.63	1.36776
.7500 (12288 records)	261	102.318	109.000	760	52500	196.53	1065.53	2.02958
				2675	210000	786.15	3570.15	1.70070
				4075	420000	1512.27	5756.27	1.37054
1.0000 (16384 records)	261	136.318	119.20	760	52500	197.34	1076.54	2.05055
				2675	210000	789.33	3583.53	1.70644
				4075	420000	1578.69	5772.89	1.37449

Table B.36. Results of computation disk average access time per record retrieval as a function of loading factor of direct disk file using Hash 1 with direct chain probing.

File size in record	Loading factor, α	Disk file supporting space		Average search per random record access with Hash I, E	Cylinder positioning time	Track waiting time in -	Read-in one track time in -	Average internal search time per record	Search access track time in -		Search access cylinder time in -		Disk research time per record
unit		cylinders	tracks	records	ms	ms	ms	ms	% ms		%ms		ms
128	0.078	28	274	1.01	95	12.5	25	0.164	-	-	-	-	132.664
512	0.0312	28	274	1.02	95	12.5	25	0.165	-	-	-	-	132.665
1024	0.0625	28	274	1.03	95	1.25	25	0.165	-	-	-	-	132.665
4096	0.2500	28	274	1.13	95	12.5	25	0.167	0.390	0.09	-	-	132.765
8192	0.5000	28	274	1.25	95	12.5	25	0.169	1.758	0.893	-	-	133.562
12288	0.7500	28	274	1.38	95	12.5	25	0.172	3.215	1.563	0.657	0.439	134.673
16384	1.0000	28	274	1.50	95	12.5	25	0.174	8.887	4.444	1.855	1.252	138.370

Note: $E = 1 + \alpha/2$ for direct chain probing.

Direct file with direct chain probing use 17 words per record; 60 records per track.

Table B.37. Data results of computation of CPU busy time and average throughput time per record retrieval of direct disk file organization with direct chain probing using full name of a record in accessing a random record from disk file.

File size in records	Communication time back and forth	Full name of a record decoding time	Hash 1 decoding time	Connecting logical unit disk time	Average disk access internal search time per record	Checking I/O unit time	A record write out time	Using fixed-length key CPU busy time	Using full name of record CPU busy time	Using full name of average throughput/ record time
unit	ms	ms	ms	ms	ms	ms	ms	ms	ms	ms
128	0.3218	8.395	.071	.008	132.664	.008	36	36.261	44.646	177.468
512	0.3218	8.475	.071	.008	132.665	.008	36	36.252	44.727	177.543
1024	0.3218	8.515	.071	.008	132.665	.008	36	36.252	44.767	177.589
4096	0.3218	8.595	.071	.008	132.765	.008	36	36.254	44.849	177.769
8192	0.3218	8.601	.071	.008	133.562	.008	36	36.256	44.857	178.572
12288	0.3218	8.626	.071	.008	134.673	.008	36	36.259	44.885	179.708
16384	0.3218	8.641	.071	.008	138.370	.008	36	36.261	44.902	183.420

Table B.38. Data results of computation of storage space required as the function of file size of direct file organization with direct chaining using both record's full name and fixed-length numerical code for accessing a record from the file.

File size	Full name to key S_{CFNFK}		Hash 1 program's space required		Main file search direct chaining		Hash 1 code to disk address conversion	Total space with using unique fixed-length key	Total space with using full name of record
unit	words (a)	words (b)	words (a)	words (b)	word (a)	word (b)	words	words	words
128	200	1088	24	2	69	1024	20	1139	2427
512	200	4352	24	2	69	1024	20	1139	5691
1024	200	8704	24	2	69	1024	20	1139	10043
4096	200	34816	24	2	69	1024	20	1139	36155
8192	200	69632	24	2	69	1024	20	1139	70971
12299	200	104448	24	2	69	1024	20	1139	105787
16384	200	139264	24	2	69	1024	20	1139	140603

Table B. 39. Result of computation of customer operating cost (unit cost) as the function of file loading factor of direct disk file with direct chain probing using full name of record in accessing.

File loading factor	Disk space for data file	Disk space for accessing program	Disk space charge per month	Terminal devices cost per month	Rule of use data file	CPU-time charge per month	CPU-time and disk charge per month	CPU-time and disk charge per month per call
unit	tracks	tracks	tracks	$	calls per month	$	$	$
.0078 (128 records)	261	1.370	78.71	760	52500	195.33	1034.04	1.96960
				2675	210000	781.31	3535.02	1.68334
				4075	420000	1562.61	5716.32	1.36102
0.0312 (512 records)	261	4.558	79.68	760	52500	195.68	1035.36	1.97211
				2675	210000	782.73	3537.41	1.68448
				4075	420000	1565.46	5720.14	1.36193
.625 (1024 records)	261	8.808	80.94	760	52500	195.87	1036.81	1.97487
				2675	210000	783.42	3539.36	1.68540
				4075	420000	1566.84	5722.78	1.36256
.2500 (4096 records)	261	34.308	88.59	760	52500	196.20	1044.79	1.99007
				2675	210000	784.86	3548.45	1.68973
				4075	420000	1569.72	5733.31	1.36507
.5000 (8192 records)	261	68.308	98.79	760	52500	196.26	1055.05	2.00961
				2675	210000	785.00	3558.79	1.69466
				4075	420000	1569.99	5743.78	1.36756
.7500 (12288 records)	261	102.308	108.99	760	52500	196.38	1065.37	2.02927
				2675	210000	785.49	3569.48	1.69975
				4075	420000	1570.98	5754.97	1.37023
1.0000 (16384 records)	261	136.308	119.20	760	52500	196.44	1075.64	2.04883
				2675	210000	785.79	3579.99	1.70475
				4075	420000	1571.58	5765.78	1.37280

APPENDIX C

Flow Charts, Tested Programs and Time Analysis

Symbolic Program for Search in Sequential Cylinder Index Track

LOCATION	OPERATION CODE	ADDRESS	COMMENT
IDENT	TEST 1C		
ENTRY	START		
START	ENI	0, 1	
	ENI	11, 3	
LOOP	ENQ	1	(read in data, one word at a time)
	ENA	A	
	AIA	1	
	INI	1, 1	
	IJD	LOOP, 3	
	ENQ	1	
	ENA	KEY	
	READ	60	
	ENI	0, 1	
	ENI	5, 2	
	LDQ	KEY	
LOOP 1	LDA	A, 1	
	AQJ, LT	NEXT	
	INI	1, 1	
	LDA	A, 1	
	STA	KEY1	
	ENQ	11	
	ENA	BLANKS	
	WRITE	61	
	UJP	END	
NEXT	INI	2, 1	
	IJD	LOOP 1, 2	
	STQ	LOSS	
	ENQ	9	
	ENA	BLANK 2 (	(In case search is lost)
	WRITE	61	
END	SBJP		
BLANKS	BCD	1,	
KEY	BSS	1	
MESSAGE 1	BCD	8, HASH ADDRESS IN THE FILE IS	
KEY 1	BSS	1	
BLANK 2	BCD	1,	
LOSS	BSS	1	
MESSAGE	BCD	7, NO SUCH KEY IN DIRECTORY	
A	BSS	12	
	END		

Average linear search time for cylinder index search = T_{ALSCYL}

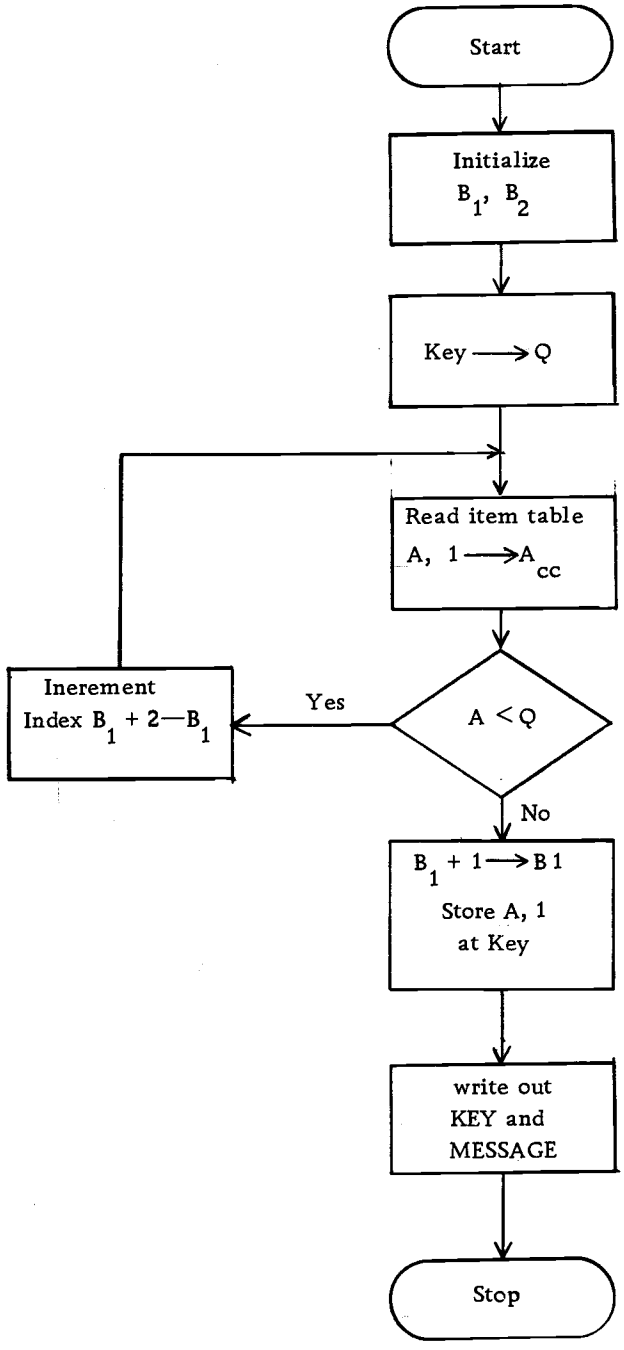
$$T_{ALSCYL} = \text{Fault loop search time } \left(\frac{N_{C1}+1}{2} - 1\right) + (1) \text{ corect loop search time}$$

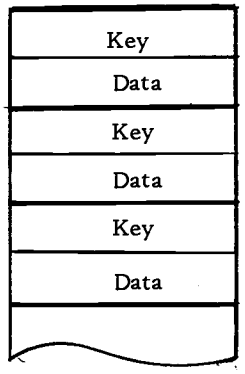
$$= 7.875 \left(\frac{N_{C1}-1}{2}\right) + 18.25$$

$$T_{ALSCYL} = 3.9375\, N_{C1} + 14.3125\ \mu\text{sec.}$$

where N_{C1} = number of entries in cylinder index.

Flow Chart of Sequential Cylinder Index Search





Cylinder table mapping in core

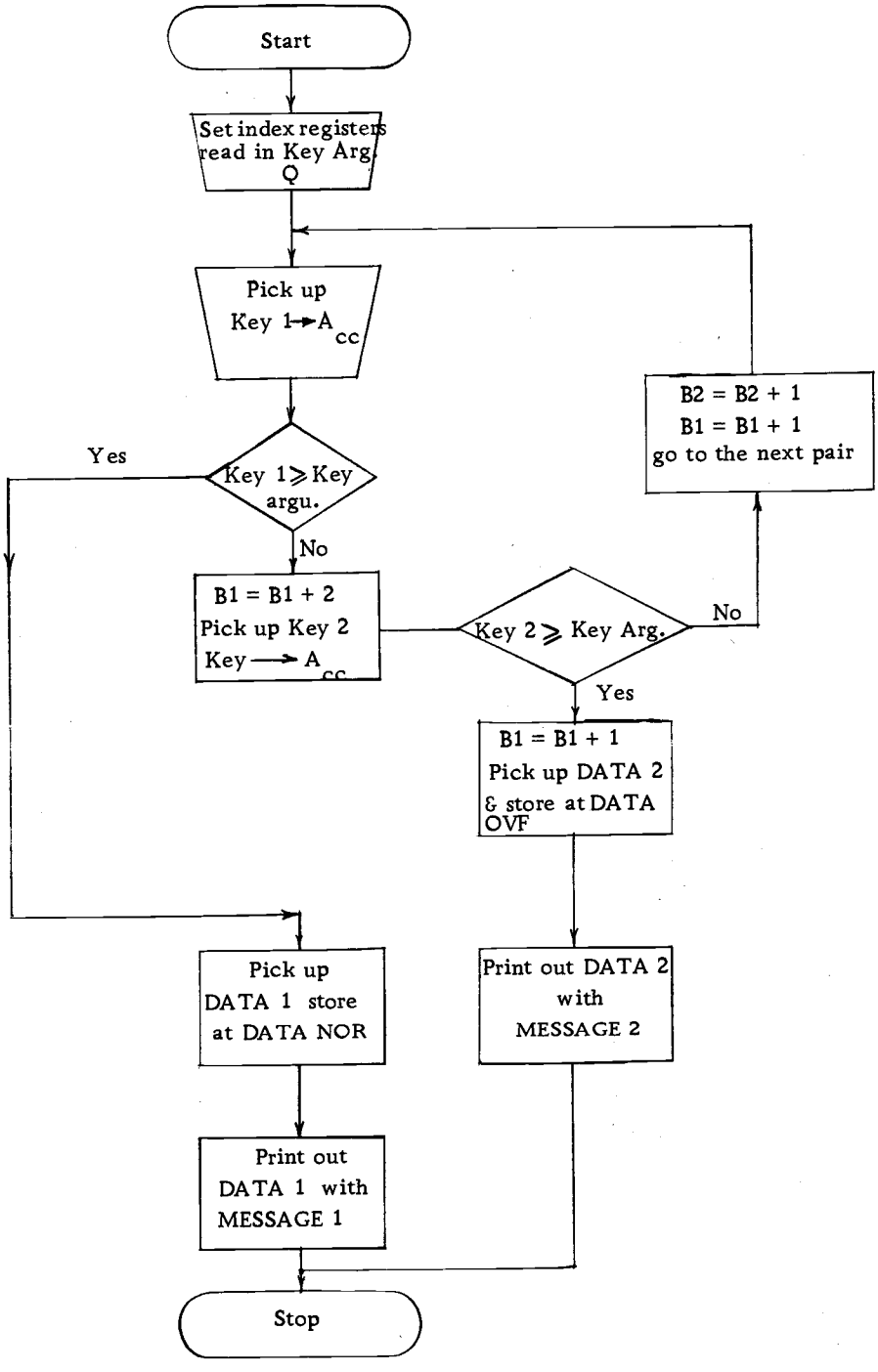
Note: Flow chart of search for a desired record in the main file is the same as above.

Symbolic Program for Search in Track Index of Indexed Sequential File

LOCATION	OPERATION CODE	ADDRESS	COMMENT
IDENT	TEST 1 D		
ENTRY	START		
START	ENI	0, 1	
	ENI	23, 3	(initializes read-in operation)
LOOP	ENQ	1	
	ENA	A	(read-in data one word at a time,
	AIA	1	and stores at location A, A+1, A+2,
	READ 60		..., A+23)
	INI	1, 1	
	IJD	LOOP, 3	(read-in "key" and stores at KEY)
	ENQ	1	
	ENA	KEY	
	READ	60	
	ENI	0, 1	
	ENI	5, 2	
	LDQ	KEY	
LOOP 1	LDA	A, 1	
	AQJ, LT	OVERFLOW	
	INI	1, 1	(if key is in prime track, stores
	LDA	A, 1	data of normal entry at DATANOR,
	STA	DATANOR	and notify the user)
	ENQ	9	
	ENA	BLANKS	
	WRITE	61	
	UJP	END	
OVERFLOW	INI	2, 1	
	LDA	A, 1	(if key is in overflow track, stores
	AQJLT	NEXTPA IR	data of overflow entry at DATAOFLO,
	INI	1, 1	and notify the user)
	LDA	A, 1	
	STA	DATAOFLO	
	ENQ	9	
	ENA	BLANK 1	
	WRITE	61	
	UJP	END	
NEXT PAIR	INI	1, 1	(if the current pair is not satisfied,
	IJD	LOOP 1, 2	goes to next pair, performs loop - operation)
END	SBJP		
BLANKS	BCD	1,	
DATANOR	BSS	1	
MESSAGE 1	BCD	7,	ADDRESS OF DESIRED TRACK
BIANK 1	BCD	1	
DATA O FIOW	BSS	1	
MESSAGE 2	BCD	7	ADDRESS OF OVERFLOW TRACK
KEY	BSS	1	

LOCATION	OPERATION CODE	ADDRESS	COMMENT
A	BSS	24	
	END	START	
	FINIS		
LOGOFF			

Flow Chart of Search for Desired Track Address in Track Index



Note(1): Key Argu = Key argument
Key 1 = Key of normal record
Key 2 = Key of overflow record
Data 1 = Data of normal record
Data 2 = Date of overflow record

(2) Program time analysis has been shown on pages 252.

Symbolic Program for Sequential Search for a Record in Main File

LOCATION	OPERATION CODE	ADDRESS	COMMENT
IDENT	TEST 1A	(Test 1A is the name of the program)	
ENTRY	START	(Initial statement to start program)	
START	ENI	0, 1	(initializes read-in operation)
	ENI	47, 3	
LOOP	ENQ, S	1	
	ENA	A	(read-in data one word at a time,
	AIA	1	and stores at locations:
	READ	0	A, A+1, A+2, . . . , A+1007)
	INI	1, 1	
	IJD	LOOP, 3	
	ENQ, S	1	(read-in argument key and
	ENA	KEY	stores at KEY)
	READ	60	
	ENI	0, 1	
			Use x_3 as counter set $x_3 = 1 = 62$
	ENI	62, 3	(search the table, the desired
	LDQ	KEY	track, if the desired record is found,
LOOP 1	LDA	A, 1	moves the desired record and stores
	LDA	A, 1	at OUTPUT, OUTPUT+1, . . . ,
	AQJ, LT	NEXT	OUTPUT+15; present it to the user)
	STA	OUTPUT	
	ENI	-14, 2	
LOOP 2	INI	1, 1	
	LDA	A, 1	
	STA	OUTPUT + 15, 2	
	IJI	LOOP 2, 2	
	ENQ	17,	
	ENA	BLANKS	
	WRITE	61	
	UJP	END	
NEXT	INI	16, 1	(if the current record is not satisfied,
	IJD	LOOP 1, 3	goes to the next record)
	ENQ	8	
	ENA	BLANK 2	(in case search is lost, it will notify
	WRITE	61	the user)
END	SBJP		
BLANKS	BCD	1,	
OUTPUT	BSS	16	
BLANK 2	BCD	1,	
KEY	BSS	1	
MESSAGE	BCD	6, NO SUCH KEY IN FILE	
A	BSS	48	
	END		
	FINIS		

Note: 1) The search flow chart is almost the same as in sequential cylinder index search, page

2) There are 63 logical record per one prime track.

T_{ALS} = Average search time per random record retrieval in the main file

T_{LSCL} = Linear search for correct loop time

= 18.75 + 2.75 + 1.375 + 16*9

= 167.00 μsec.

T_{LSFL} = Linear search for fault loop time

$$= 7.875 \left(\frac{N_{APT}+1}{2} - 1\right)$$

$T_{ALS} = T_{LSCL} + T_{LSFL}$

$= 3.937\,5N_{APT} + 163.0625$ μsec.

Where N_{APT} = Average number of record per track.

Symbolic Program Sequential Search for a Record in Overflow Area

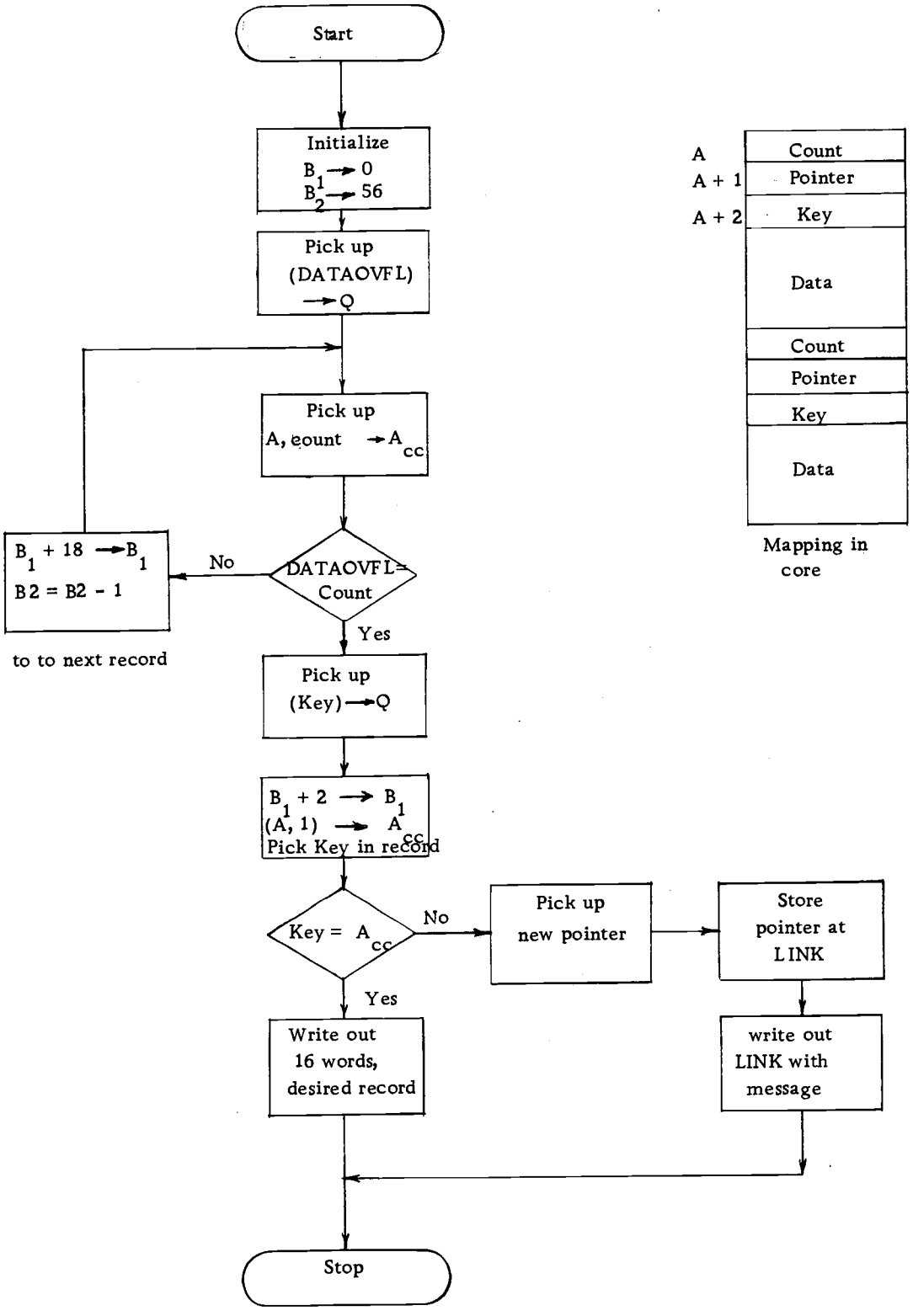
LOCATION	OPERATION CODE	ADDRESS	COMMENT
IDENT	TEST 1B		
ENTRY	START		
START	ENI	0, 1	(Initializes read-in operation)
	ENI	53, 3	
LOOP	ENQ	1	
	ENA	A	(read-in data one word at a time,
	AIA	1	and stores at locations: A, A+1,
	READ	60	A+2, . . . , A+1007)
	INI	1, 1	
	IJD	LOOP, 3	
	ENQ	1	
	ENA	KEY	(read-in argument key and stores
	READ	60	at KEY)
	ENQ	1	
	ENA	DATAOFLO	(read-in overflow pointer and stores
	READ	60	at DATAOFLO)
	ENI	0, 1	(initializes for searching in
	ENI	3, 2	overflow track)
	LDQ	DATAOFLO	
BEGIN	LDA	A, 1	
	AQJ, NE	NEXT	(search the current overflow record
	LDA	A+2, 1	by checking the pointer first, and
	LDQ	KEY	checking the key, secondly. If the
	AQJ, LT	CHEK LINK	search is satisfied, move the desired
	INI	2, 1	record and store at OUTPUT, . . . ,
	ENI	-15, 3	OUTPUT + 16, present it to the
LOOP 1	LDA	A, 1	user)
	STA	OUTPUT+15, 3	
	INI	1, 1	
	IJI	LOOP 1, 3	
	ENQ	17	
	ENA	BLANKS	
	WRITE	61	
	UJP	END	(if the current record is not the
CHEK LINK	INI	1, 2	desired record, picks up the link-
	LDA	A, 1	pointer and goes to the next record
	STA	LINK	in the chain)
	UJP	NEX RECOD	
NEXT	ENI	18, 1	
	IJD	BEGIN, 2	
NEXRECOD	INI	17, 1	
	IJD	NEWREC, 2	(notify the user in case the search
	ENQ	11	needs to be performed in the next
	ENA	BLANK 2	record in the same chain)
	WRITE	61	

LOCATION	OPERATION CODE	ADDRESS	COMMENT
END	SBJP		
BLANKS	BCD	1,	
OUTPUT	BSS	16	
BLANK 2	BCD	1	
LINK	BSS	1	
MESSAGE	BCD	7,	GO TO NEXT RECORD ROUTINE
KEY	BSS	1	
A	BSS	54	
DATAOFLO	BSS	1	
	END	START	
	FINIS		

LOGOFF

Note: There are 56 logical records in overflow track. See time analysis on page 236.

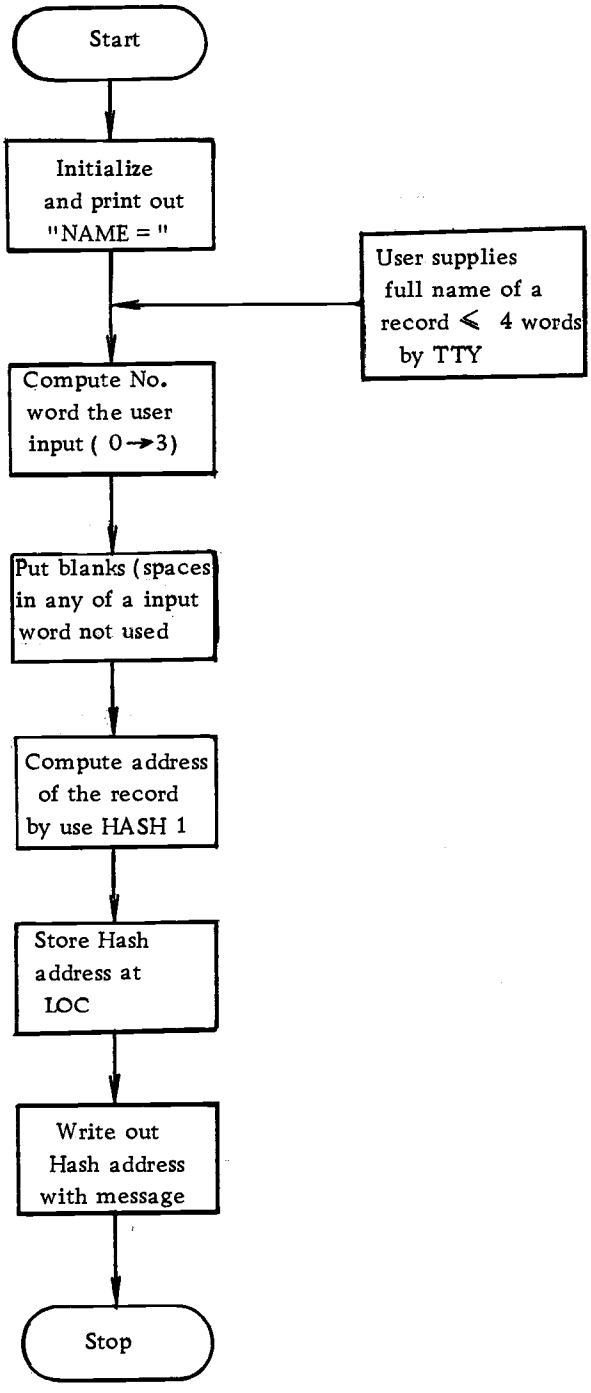
Flow Chart of Search in Overflow Track



Symbolic Program of Hash 1 to Perform Hash Address

```
LOCATION        OPERATION CODE      ADDRESS          COMMENT

IDENT           TEST 9A
ENTRY           START
START           ENA                 MESS 1           (prints "NAME =" )
                ENQ                 2
                WRITE               21
                ENA                 NAME             (reads in the NAME up to
                ENQ                 4                16 characters)
                READ                20
                QSG, S              0
                ENQ                 0
                XOQ, S              -0               (compute number of words input)
                INQ                 3
                SHAQ                24
                TAI                 1
                LAD                 BLANK 1
                UJP                 * +2             (stores blanks spaces which is not
                STA                 NAME, 1          used)
                ISI                 3, 1
                UJP                 * -2
                LDA                 NAME             (checks to see if KEY = $ END)
                LDQ                 DONE
                AQJ, EQ             DUMP
                LDA                 NAME
                SCA                 NAME+1           (perform Hash 1 by "exclusive or"
                SCA                 NAME+2           all 4 words together; square the
                SCA                 NAME+3           result; take 24 middle bits and
                STA                 MULT             consider only 10 lower bits as hash
                MUA                 MULT             address)
                SHA                 -12
                ANA, S              1777B
                TAI                 1
                STA                 LOC              (stores hash address at LOC)
DUMP            ENQ                 10
                ENA                 BLANKS 2
                WRITE               21               (presents hash address to the user)
END             SBJP
BLANK 2         BCD                 1
NAME            BSS                 4
MESSAGE         BCD                 4,    HASH ADDRESS=
LOC             BSS                 1
DONE            BCD                 1,    $ END
BLANK 1         BCD                 1,
MESSAGE 1       BCD                 2,    NAME =
                END
                FINIS
```

Flow Chart of Hash 1 to Perform Hash Address



Note: See time analysis on page 316.

Symbolic Program of Random Search in the Main File

LOCATION	OPERATION CODE	ADDRESS	COMMENT
IDENT	TEST 9C		
ENTRY	START		
START	ENI	0, 1	
	ENI	24, 3	(Initial setting)
REDO	ENQ, S	1	
	ENA	A	
	AIA	1	
	READ	60	(Read in data)
	INI	20, 1	
	IJD	REDO, 3	
	ENQ, S	1	
	ENA	KEY	(Read in key)
	READ	60	
	ENA	499	(max. beginning address of record
	STA	FULL	in track)
	ENI	0, 2	
	ENA	1	(initializes R = 1 every time
	STA	RAND	random generator is called)
SEARCH	LDA	RAND	
	SHA	-2	divide (A_{cc}) by 4
	ADA	LOC	
	ANA, S	1777B	(keeps 10 lower bit of A_{cc} the less
	SHA	4	of them set = zeros)
	TAI	1	
	LDA	A, 1	
	LDQ	KEY	
	AQJ, EQ	FOUND	check table at location (A + 1 ρ) its content is = KEY or not
	INI	1, 2	
	LDA	RAND	
	SHA	2	R * 5 → A_{cc}
	ADA	RAND	
	ANA, S	7777B	keep 12 lower bit of A_{cc}
	STA	RAND	the less of them set = zero
	LDQ	FULL	
	AQJ, LT	SEARCH	jump to SEARCH if (A) < Q
	ENQ	10	
	ENA	BLANKS	
	WRITE	61	
	UJP	END	
FOUND	STA	OUTPUT	
	ENI	-14, 2	
LOOP 1	INI	1, 1	
	LDA	A, 1	
	STA	OUTPUT +15	
	IJI	LOOP 1, 2	
	ENQ	17	
	ENA	BLANKS	
	WRITE	61	

LOCATION	OPERATION CODE	ADDRESS	COMMENT
END	SBJP		
BLANKS	BCD	1,	
OUTPUT	BSS	16	
BLANK	BCD	1,	
KEY	BSS	1	
MESSAGE	BCD	8,	NO SUCH KEY IN THIS FILE
FUIL	BSS	1	
RAND	BSS	1	
A	BSS	500	
LOC	BCD	1, 000	
	END	START	
	FINIS		

Note: See time analysis on pages 319 - 320.

If T_{RSDF} = Average random search time per record retrieval of direct disk file, for internal search.

T_{CLRSDF} = Average correct loop random search time/record retrieval

= 35.375 + 9x (16) = 179.375 μsec.

T_{FLRSDF} = Average fault loop random search time/record retrieval

= 34.625 μsec.

Then $T_{RSDF} = T_{CLRSDF} + T_{FLRSDF} \times (E - 1)$

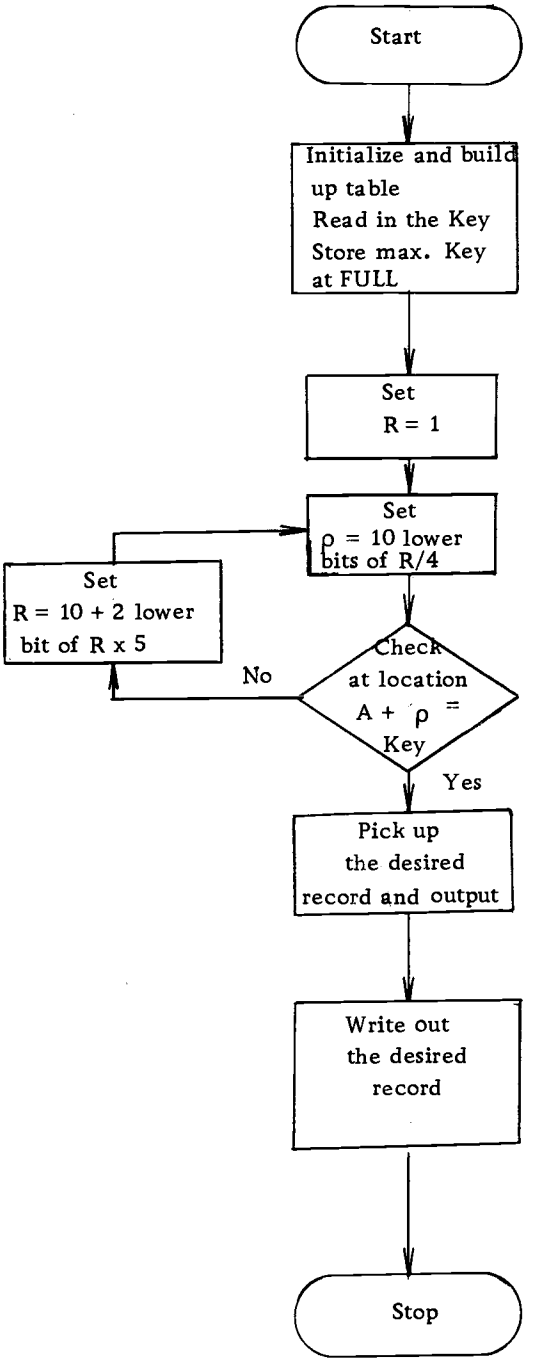
= 179.375 + 34.625 (E-1) μsec.

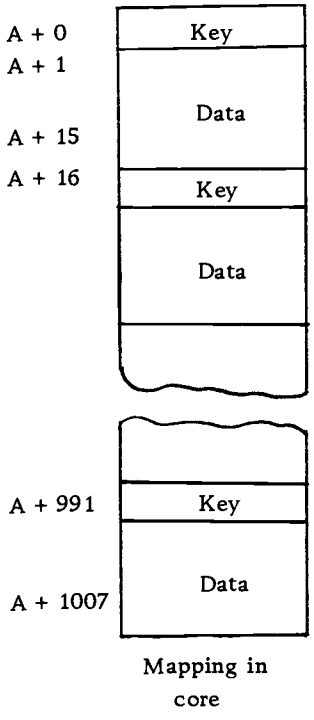
Where E = Average search length of random probing

$= -(1/\alpha) \log (1-\alpha)$

Where α = Loading factor of table.

Flow Chart of Random Search in the Direct Disk File





Mapping in core

Note: In this experiment when $\rho = 169$ random search is satisfied.

Symbolic Program for Search with Chain in Direct Disk File

LOCATION	OPERATION CODE	ADDRESS	COMMENT
IDENT	TEST 9 B		
ENTRY	START		
START	ENI	0, 1	
	ENI	67, 3	(1020 word track)
REDO	ENQ, S	1	
	ENA	A	
	AIA	1	(reads in data)
	READ	60	
	INI	1, 1	
	IJD	REDO, 3	
	ENQ, S	1	
	ENA	KEY	(read in Key→KEY)
	READ	60	
	ENI	0, 1	
LOOP 1	LDQ	KEY	(set B1=0, KEY→Q)
	LDA	A, 1	(Current record = desired record
	AQJ, EQ	FOUND	of not)
	INI	-1, 1	
	LDA	A, 1	
	LDQ	ENDCHAIN	(pointer of current record = ***
	AQJ, EQ	NONE	or not)
	TAI	1	
			(LOC)→B_1,
	UJP	LOOP 1	B_1 point to next record in chain
FOUND	STA	OUTPUT	
	ENI	-41, 2	
LOOP 2	INI	1, 1	(If the current record is desired
	LDA	A, 1	record print out the data)
	STA	OUTPUT+16, 2	
	IJI	LOOP 2, 2	
	ENQ	18	
	ENA	BLANKS	
	WRITE	61	
	UJP	END	
NONE	ENQ	10	
	END	BLANK 1	(If the current record is not the
	WRITE	61	desired record notify the user)
END	SBJP		
BLANKS	BCD	1	
OUTPUT	BSS	16	
BLANK 1	BCD	1,	
KEY	BSS	1	
MESSAGE 1	BCD	8,	NO SUCH KEY IN DESIRED TRACK
A	BSS	68	
END CHAIN	BCD,	1, ***	
LOC	BSS	1	
	END		
	FINIS		

T_{CLSCH} = Correct loop search time for chain

= 18.875 + 9* (17, no. words per record) +1.375

= 164.250 μsec. (based on test program

T_{FLSCH} = Fault loop search time for chain.

= 19.875 μsec. (based on test program)

The value of expected search per record retrieval, E, is obtained by simulation, as shown in Figures B.20 and Table B.23, page 303. The statistical value of e can be computed directly from the equation.

$$E = 1 + \frac{\alpha}{2}$$

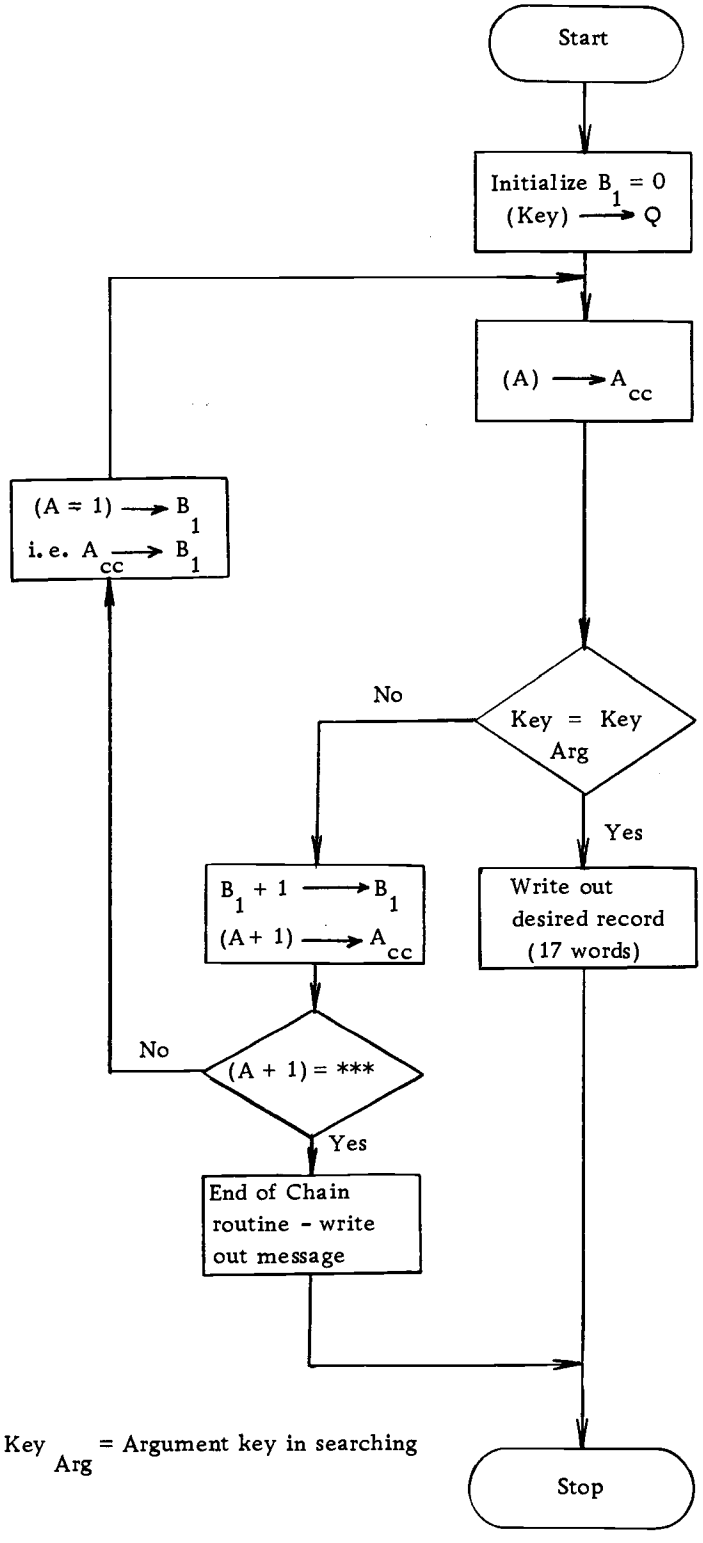
Then T_{ASCH} = Average chain search

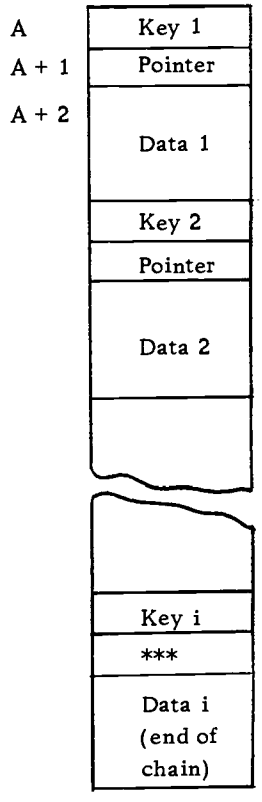
$$= T_{CLSCH}(1) + T_{FLSCH}(1 + \frac{\alpha}{2} 1)$$

$$= 164.250 + 19.875 (\alpha/2)$$

Where α = File loading factor.

Flow Chart of Chain Search in Direct Disk File





Mapping in core

Symbolic Program of Variable Tree Decoding of Full Name of Record to Fixed-Length Key Name

LOCATION	OPERATION CODE	ADDRESS	COMMENT
START	IDENT	TEST 8	
	ENTRY	START	
START	LDAQ	DNAME	
	ENI	2, 1	
	X REQ	40	
	ENI	3, 1	
	X REQ	40	
	ENQ	4	
	CNTL	40	
	ENI	0, 1	
	ENQ	1	
	ENA	A	
	AIA	1	
	READ	40	
	SHA	3	
	AZJ, LT	* + 23	
	LDA	A, 1	
	SCA	MASK	
	STA	A, 1	
	ENQ	1	(Read-in the random names which
	ENA	A+1	have been stored in file "DATA 8")
	AIA	1	
	READ	40	
	ENQ	1	
	ENA	A+2	
	AIA	1	
	READ	40	
	ENQ	1	
	ENA	A+3	
	AIA	1	
	READ	40	
	ENQ	1	
	ENA	A+4	
	AIA	1	
	READ	40	
	INI	5, 1	
	ISE	400, 1	
	UJP	* -27	
REDO	ENQ	2	
	ENA	MESS 1	
	WRITE	61	(Prints "NAME=")
	ENQ	20	
	ENA	NAME	
	READ	60	(Reads in the NAME (up to 80
	XOQ, S	-0	characters))
	INQ	19	(Computes # of words input)
	SHAQ	24	

LOCATION	OPERATION CODE	ADDRESS	COMMENT
	SWA	FOUND	
	SWA	LOOP	
	SWA	FOUND+5	(Use that # of words to control
	SWA	*+10	
	INA	2	
	SWA	FOUND+11	
	ENI	0, 2	
	LDA	NAME	
	LDQ	DONE	(Check to see if NAME = $ END)
	AQJ, EQ	DUMP	
	LDA	NAME, 2	
	SCA	MASK	(Compliment first bit of each
	STA	NAME, 2	
	ISI	**, 2	
	UJP	*-4	
	ENI	0, 1	
	ENI	0, 2	
	LDA	A, 1	(only if the tree is empty)
	AZJ, EQ	EMPTY	
	LDQ	NAME, 2	
	AQJ, EQ	FOUND	(Test for right, left or successor)
	AQJ, LT	LESS	
	LDA	A+3, 1	
	AZJ, LT	*+3	
	TAI	1	Move left
	UJP	OVER	
	RTJ	FREE	
	LDA	LAS	
	STA	A+3, 1	(No left pointer, add newnode)
	UJP	ADDITEM	
LESS	IDA	A+4, 1	
	AZJ, LT	*+3	
	TAI	'	Move right
	UJP	OVER	
	RTJ	FREE	
	LDA	LAS	
	STA	A+4, 1	(No right pointer, add new node)
	UJP	ADDITEM	
FREE	UJP	**	
	STI	SAVE 1, 1	
	ENI	0, 1	
	LDA	A, 1	

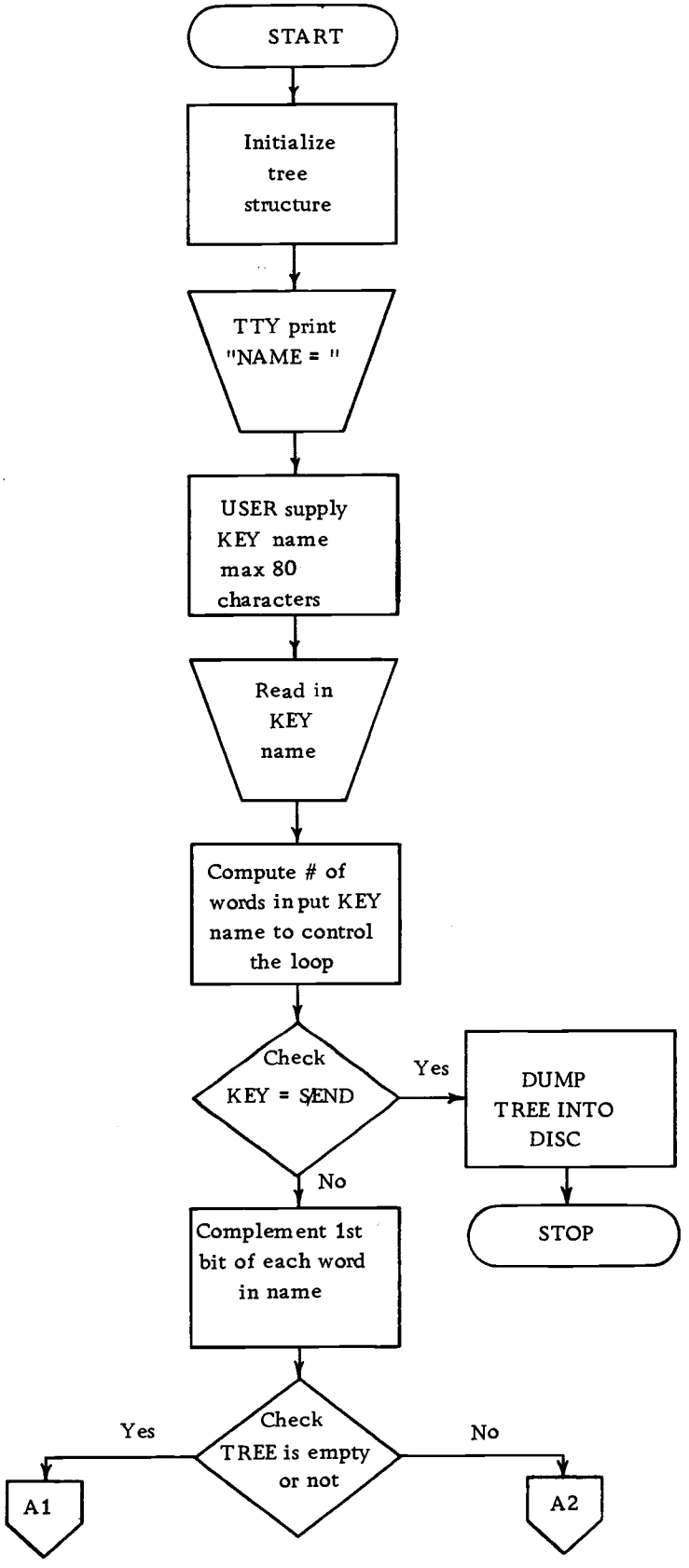
```
LOCATION        OPERATION CODE      ADDRESS         COMMENT

                AZJ, EQ             *+8
                INI                 5, 1
                ISE                 400, 1          (Subroutine to find next 5 words
                UJP                 *-4             of available space)
                ENQ                 7
                ENA                 MESS 3
                WRITE               61
                UJP                 DUMP
                TIA                 1
                STA                 LAS
                LDI                 SAVE 1, 1
                UJP                 FREE
INDATA          UJP                 **
                ENQ                 2
                ENA                 MESS 2
                WRITE               61              (Subroutine to pick up data for name)
                ENQ                 1
                ENA                 A+1
                AIA                 1
                READ                60
                UJP                 INDATA
EMPTY           ENA                 0               (only for completely empty)
                UJP                 ADDITEM
FOUND           ISE                 **, 2
                UJP                 MORE
                LDA                 A+1, 1
                AZJ, LT             NEW
                STA                 DATA            (See if data is there go to new
                ENI                 **, 2           if no data)
                LDA                 NAME, 2
                SCA                 MASK            (If the looking variable length key
                STA                 NAME, 2         name produces matching and also
                IJD                 *-3, 2          has data i.e., complete matching;
                ENA                 B1              so print out the information)
                ENQ                 **
                WRITE               61
                ENA                 B2
                ENQ                 2
                WRITE               61
                UJP                 REDO
                RTJ                 INDATA          (Incompletes matching and no data)
                UJP                 REDO
                LDA                 A+2, 1
                AZJ, LT             LOOP
                TAI                 1               Matching not yet complete, try
                INI                 1, 2            next word from .....
                UJP                 OVER
```

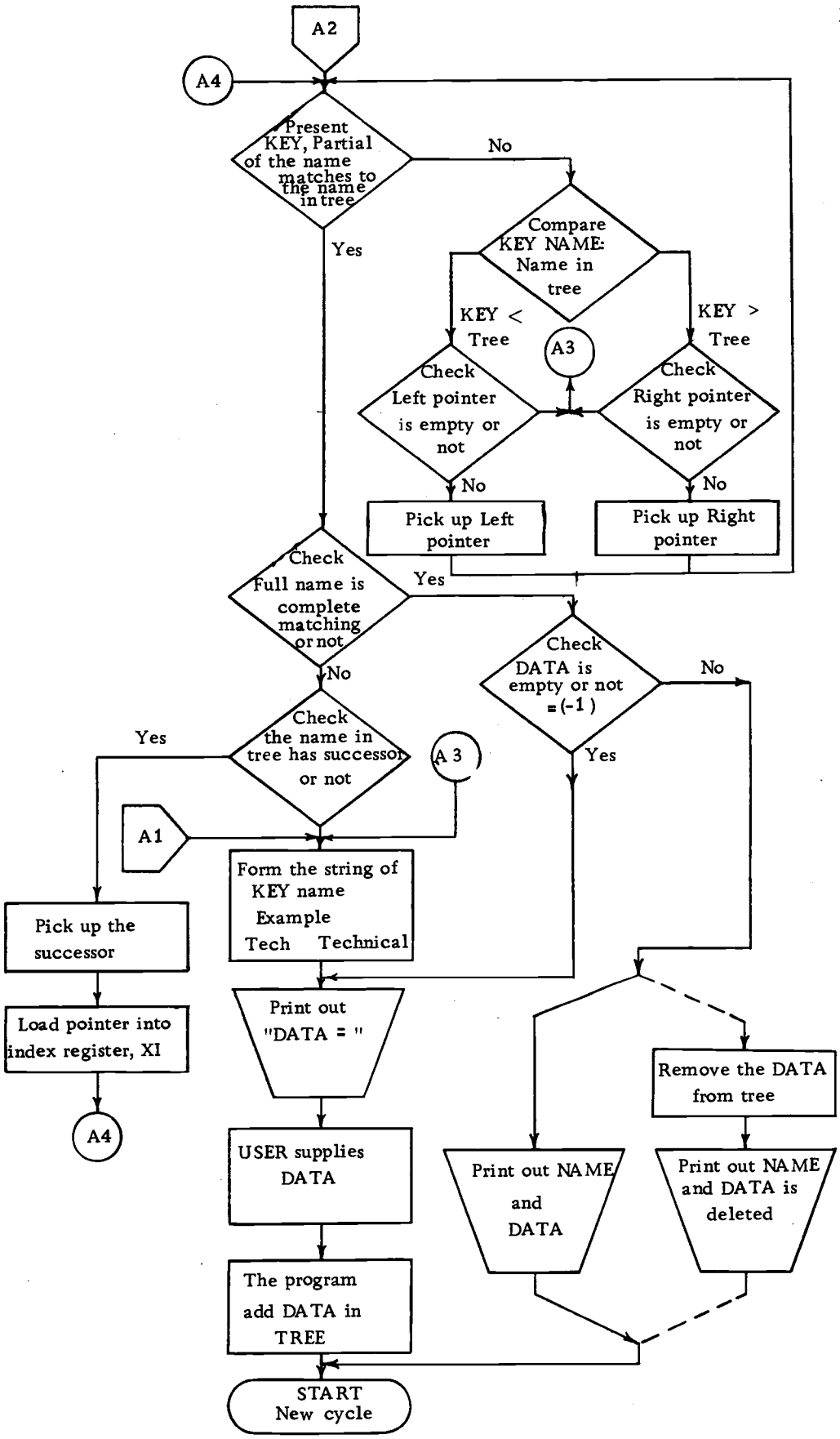
LOCATION	OPERATION CODE	ADDRESS	COMMENT
ADDITEM	TAI	1	
	LDA	NAME, 2	
	STA	A, 1	
	ENA, S	-1	
	ENQ, S	-1	
	STAQ	A+1, 1	
	STAQ	A+3, 1	
LOOP	ISI	**, 2	(Add the rest of the words in
	UJP	*+2	name to the tree structure)
	UJP	*+5	
	RTJ	FREE	
	LDA	LAS	
	STA	A+2, 1	
	UJP	ADDITEM	
	RTJ	INDATA	
	UJP	REDO	
DUMP	ENQ	4	
	CNTL	40	
	ENI	0, 1	
	LDA	A, 1	
	SCA	MASK	
	STA	A, 1	
	ENQ	1	
	ENA	A	
	AIA	1	
	WRITE	40	
	ENQ	1	
	ENA	A+1	
	AIA	1	
	WRITE	40	(Returns the variable length
	ENQ	1	tree structure in to DATA 8)
	ENA	A+2	
	AIA	1	
	WRITE	40	
	ENQ	1	
	ENA	A+3	
	AIA	1	
	WRITE	40	
	ENQ	1	
	ENA	A+4	
	AIA	1	
	WRITE	40	
	INI	5, 1	
	ISE	400, 1	
	UJP	+-25	
	ENQ	2	
	CNTL	40	
	ENI	2, 1	
	XREQ	40	

LOCATION	OPERATION CODE	ADDRESS	COMMENT
END	SBJP		
DNAME	BCD	2, DATA 8	
MESS 1	BCD	2, NAME =	
MESS 2	BCD	2, DATA =	
MESS 3	BCD	7, NO MORE SPACE AVAILABLE	
MASK	OCT	40000000	
LAS	BSS	1	
SAVE 1	BSS	1	
B1	BCD	1,	
NAME	BSS	20	
B2	BCD	1,	
DATA	BSS	1	
DONE	BCD	1, $ END	
A	BSS	400	
	END	START	
	FINIS		

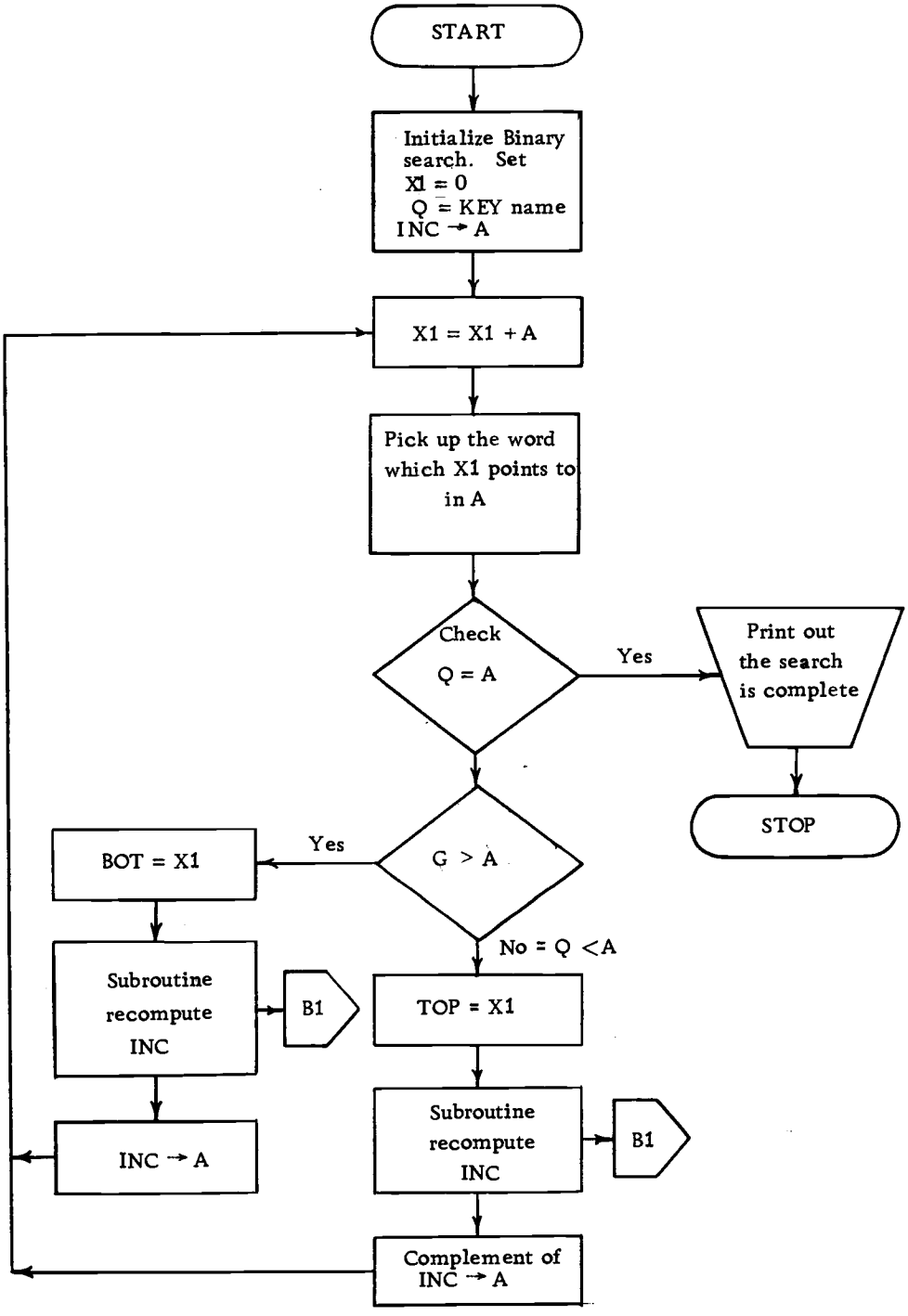
Note: Time analysis for search path is shown on page 200.

FLOW CHART OF THE SIMULATION OF ADDING AND DELETING KEY NAME WITH VARIABLE LENGTH USING TREE




A2
A4
Present KEY, Partial of the name matches to the name in tree
No
Yes
Compare KEY NAME: Name in tree
KEY < Tree
KEY > Tree
A3
Check Left pointer is empty or not
Check Right pointer is empty or not
No
No
Pick up Left pointer
Pick up Right pointer
Check Full name is complete matching or not
Yes
Check DATA is empty or not = (-1)
No
No
Yes
Check the name in tree has successor or not
Yes
A 3
A1
Form the string of KEY name Example Tech Technical
Pick up the successor
Load pointer into index register, XI
A4
Print out "DATA = "
USER supplies DATA
The program add DATA in TREE
START New cycle
Remove the DATA from tree
Print out NAME and DATA
Print out NAME and DATA is deleted

FLOW CHART OF THE SIMULATION OF BINARY SEARCH USE ADDING AND RETRIEVAL ITEMS FROM FILE



APPENDIX D

List of Symbols

Symbol	
a	Cost/bit of core memory, in this case a = \$0.12
b	Cost/bit of disk memory, in this case b = \$0.00025
C_i	Achievable throughput-rate capability of a specific file i
CR	Cathad Ray display
COCR	Cylinder overflow control record
C_{cpu}/M	Amount of charge per month for CPU busy time
DASD	Direct access storage devices
D_s	Disk storage space required in bits
E	Expectation or average number of search per record retrieval from a type of file
E_1, E_2, E_3, E_4, E_5, E_6	See detail on page 161
E_{LP}	Expected number of searches per record retrieval with linear probing of direct file
E_{RP}	Expected number of searches per record retrieval with random probing of direct file
E_{DCH}	Expected number of searches per record retrieval with direct chain of direct file
$F/_{hr}$	Number of calls/hour

Symbol	
$F/_M$	Number of calls/month
H_1	Hash 1
H_2	Hash 2
H_3	Hash 3
H_4	Hash 4
H_5	Hash 5
I_{REDP}	Number of record entries in directory
I_T	Number of required track index
l	Random address location in Hash Table
M	File project active life in terms of months
N,n	Number of logical records in the file, file size
N_{ALS}	Average number of searches per track
N_{APT}	Average number of records per track
N_{ATBS}	Average number of looking up item with binary search
N_{CI}	Number of cylinder index entries
N_{FLDPF}	Number of entries in first level directory of partitioned file
N_{OT}	Number of records in overflow track
N_T	Number of tracks on disk required to support the file
$N_{T(i)}$	Number of tracks on disk required for specific file i
N_{TLC}	Number of tracks used in the last cylinder

Symbol	
N_{UT}	Number of tracks used
P_T	Number of pairs, normal entry, overflow entry in track index
P_{AT}	Average number of pairs in each track index
R	Random number
R_{CM}	Rate of use, calls per month
R_D/M	Amount of charge per month for disk file
S_{AVLTR}	Average core storage space required for variable length tree
S_P	Core storage space used for supporting computer program in bits
S_{SD}	Core storage space used for supporting the data file which is transfered from disk into core
S_T	Number of search length for each file size
S_{cT}	Expectation of searches per record retrieval when direct file uses direct chain probing
T_{ALSM}	Average linear search a random record from the main file
T_{ALSOFT}	Average time to search a record in clyinder overflow track
T_{ALT}	Transfer and average search time in the last track
$T_{AR/WHP}$	Average read/write head positioning time
$T_{ASDFDCH}$	Average access time per record retrieval including internal search with direct chain probing and search across track and cylinder

Symbol	
$T_{ASDLDPF}$	Average search time for first level of directory of double partitioned file
T_{ASFSQF}	Average search time per record retrieval, in the main file for disk sequential file
$T_{ASRDFOF}$	Average access time of a random record from in the main disk file with 10% of normal track search used in cylinder overflow track including disk times
$T_{ASRDFCH}$	Average access time per record of direct file with direct chain
$T_{ASRDFLP}$	Average access time per record from disk direct file with linear probing
$T_{ARSDFRP}$	Average access time per record of direct file random probing
$T_{ASRFTIDFOF}$	Average time required to read in and perform internal search per record retrieval from disk file, with 9% cylinder overflow search, including track index search
$T_{ASSLDPF}$	Average time search for directory search of partitioned file with single level directory
$T_{ATHRPDFDCH}$	Average throughput per record retrieval direct file with direct chain
$T_{ATHRPDFLP}$	Average throughput per records retrieval of direct file with linear probing
$T_{ATHRPDFRP}$	Average throughput per record retrieval with direct file for random probing
$T_{ATHRPISQ}$	Average throughput time of accessing a random record from indexed sequential disk file
$T_{ATHRPSLDPF}$	Average throughput time per record retrieval of single level directory partitioned disk file

Symbol	
T_{ATW}	Average track-waiting time
T_{CBF}	Communication back and forth time
$T_{CPU/R}$	CPU busy time per record retrieval in seconds
T_{CFNTFK}	Time required to convert record's full name to fixed-length key name
T_{DCHS}	Direct chain probing in the track containing the desired record
$T_{DCHSACT}$	Direct chain probing search across track
$T_{DCHSACYL}$	Direct chain search time across the cylinder
T_{DRPM}	Disk revolution time
T_{FLSCH}	Fault loop search time for chain
T_i	Average throughput time of specific file, i
$T_{(i)}$	CPU busy time per call in hour of the specific file, i
$T_{HASH\ 1}$	Hash 1 decoding time per record retrieval
T_{CLSCH}	Correct loop search time for chain
t_{LLAC}	Number of looking up track in last cylinder
T_{LPS}	Linear search in the track containing the desired record
T_{LPSACT}	Linear probing search across track
$T_{LPSACYL}$	Linear probing search across the cylinder
T_{LS}	Time required for sequential search a record in the main file
T_{LSCYLI}	Time required for cylinder index search/ record

Symbol	
$T_{RCI/O}$	Time required to check and connect I/O logical unit
T_{RCLU}	Time required to request and connect the logical disk unit
T_{RPS}	Random search time in the track containing the desired record
T_{RPSACT}	Random probing search across track
$T_{RPSACYL}$	Random probing search across cylinder
$T_{RSDFDCH}$	Average time required for an internal random search for a record from direct file with direct chain probing
T_{RSDFRP}	Average time required for an internal random search of desired record from direct file with random probing
T_{RTRI}	Read track index time
T_{RWOUT}	Time required to write out the desired record
T_{SFLP}	Search time spent in fault loop path
T_{SOVFP}	Search time spent in overflow path
T_{SSFP}	Search time spent in satisfied path
T_{STIENT}	Time required for search track index entries
$t_{TCPUTDFDCHP}$	Total CPU busy time per record accessed using unique fixed-length key for direct file with direct chain probing
$t_{TCPUTDFDCHUFN}$	Total CPU busy time per record accessed using full name of the record for direct file with direct chain probing
$t_{TCPUTDFLP}$	Total CPU busy time per record accessed using unique fixed-length key

Symbol	
$t_{TCPUTDFRP}$	Total CPU busy time per record accessed using unique fixed-length key for direct disk file with random probing
$t_{TCPUTDLDPF}$	Total CPU busy time per record accessed using unique fixed-length key for double directory partitioned file
$t_{TCPUTDLDPFUFN}$	Total CPU busy time per record accessed using full name of the record for double directory partitioned file
$t_{TCPUTDRPUFN}$	Total CPU busy time per record accessed using full name of the record for direct disk file with random probing
$t_{TCPUTISQF}$	Total CPU time per record accessed using unique fixed-length key
$U_C(i)$	User or customer operating cost per call (unit cost) of the specific file, i
Z	Number of required cylinders to support the file
Z_a	Average number of required cylinders