LOGIC-PER-TRACK ASSOCIATIVE MEMORY

by

GEOK-SENG TANG
B.Sc., Nanyang University, 1967
M.Sc., University of Ottawa, 1969

A THESIS SUBMITTED IN PARTIAL FULFILMENT OF
THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE

in the Department
of
Computer Science

We accept this thesis as conforming
to the required standard

THE UNIVERSITY OF BRITISH COLUMBIA
April, 1976

In presenting this thesis in partial fulfilment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the Head of my Department or by his representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Geok-Seng Tang

Department of Computer Science

The University of British Columbia
2075 Wesbrook Place
Vancouver, Canada
V6T 1W5

Date April 6, 1976

ABSTRACT

An associative, or content-addressable, memory, one in which data may be retrieved by its value rather than by real address, has always been an attractive idea. Although such a memory has not yet proven practical for files of respectable size, much interesting work has been done on the subject, for example, Minsky (1972), Slotnick (1970) and Parker (1970, 1971). This thesis is concerned with the device proposed by Slotnick and Parker, called 'Logic Per Track Device'. After briefly reviewing the design and capabilities of their device, the thesis proceeds to propose some modifications to the design which not only lead to greatly enhanced performance, but also establish its practical application for files of respectable size. In the device of Slotnick and Parker, there is a fiarly sophisticated logic chip attached directly to each non-movable read-write head. This allows all logic heads to search simultaneously for information matching a given key, so that any desired record could be located within one revolution. However, reading and writing will require a second revolution because part of the record will have passed the head before the match is recognized. Moreover, if more than one record matches the search key, the extra bookkeeping will be needed if matching records on different tracks should partially overlap. These problems have been ignored in the retrieval system developed by Parker (1970, 1971).

The following four additional features of the device have been proposed:

1. Two logic heads on each track has been introduced. The leading head will continue to have the primary responsibility for simultaneous searching. The additional second head, trailing a fixed distance behind will do the actual reading and writing of records.

2. A delay register whose length is the distance between logic heads on the same track, has been added to the read-write head. The function of the delay bit is to tell the read-write head partner where to start reading (or writing) a record whenever a match is recognized so that retrieving (or writing) a single record can always be performed in the same revolution.

3. Another major design change will give the new device the ability to keep track of all records which may be retrieved within a single revolution by parallel search. To this end, the monitor, which synchronizes the activities of all logic head couples, will be provided with a record counter, and a mark entity will be prefixed to every record on the disk itself.

4. A file identification mechanism has been established for the associative memory. Functions of such a mechanism are (a) to manage file names, and (b) to manipulate data on the storage device.

Next step is to explore the use of such a modified device for file-oriented problems. 'Hierarchical search' for records possessing a specified combination of keys can be performed

directly on the key part of records without the intermediate step of transmitting records into the main computer memory. In an application requiring chain processing, the chain pointer can be a key of the record because each record in the associative memory is accessed by content rather than by real address. The chain key can be generated from the key of the record it points to by a simple and reversible procedure. Such a chain technique has a number of advantages: (a) any chain is in fact a two-way chain, (b) each record in the chain can be retrieved by following the chain key, or directly by the key of the record if it is known, and (c) the tangle of actual physical addresses in the chain processing can be avoided. The storage organization for more complex data structure such as tree structures presents another unique feature of the modified memory. In a tree structure, indexes to the subordinate records may be kept with each parent record, or each subordiante record may store an index to its parent record. Both data structures take the same amount of storage space. Comparison of its performance to the conventional counterpart shows that significant improvements in access times can be achived.

CONTENTS

FIGURES

ACKNOWLEDGEMENTS

Special thanks are due Dr. J.R. H. Dempster for his inspiring supervision and generous assistance throughout the duration of this thesis, expecially during hours of numerous weekends.

Valuable discussions with Dr. Allan Ballard of Computer Centre are acknowledged with gratitude.

My heartfelt thanks go to my wife Karen for her sustaining encouragement and her invaluable assistance in the typing of the complete manuscript.

CHAPTER 1

LOGIC-PER-TRACK ASSOCIATIVE MEMORY

1.1. Review of The Hardware Device and Physical Data Format

The logic-per-track device was proposed by Slotnick (1970) and further developed by Parker (1970). Although such a device has not been built, it is worth thinking about in the presence of less expensive, highly reliable electronic parts, as was pointed out by Slotnick during 1970. The device is designed to achieve high performance on file-oriented problems. The structure and the circuitry of the logic head was designed by Parker (1970); a retrieval system for such a device has also been proposed by Parker (1970, 1971). Since then, there has been no further development of retrieval systems for such a device. In this thesis, Chapters 1 and 2 contain a review of the work of Slotnick and Parker; while improvements for such a device and the retrieval system comprise Chapter 3. Finally, Chapter 4 demonstrates some applications of such an improved device and retrieval system. Moreover, comparisons between the proposed device and conventional ones are drawn in order to evaluate the performance of the device.

A logic-per-track disk is a fixed head disk with a

logic chip attached directly to each read-write head. The logic chip allows each head to operate as an independent device. Each head reads and writes, and searches for space to write new records on its own track. The device is intended to have a thousand heads with approximately one million bits for each track. This gives a total bit capacity of 10^9 bits (Parker, 1970). All of the individual logic heads are connected together and communicate with a central source. The disk device is designed to operate as an essentially independent entity, which records and retrieves information on the basis of a set of keys associated with each group of information items. The purpose of the design is to combine a new hardware device with a software system to provide a total information retrieval system. Moreover, it is desirable to have a system such that random access by means of any key in the system to any data item would be extremely rapid, on the order of $\frac{1}{40}$th of a second or less, where $\frac{1}{40}$ second is the revolution time of the disk.

Each track is broken up into four equal segments or disk quadrants. There is a single pair of clock tracks which governs all the heads. One clock track indicates the bit time; the other clock track indicates the beginning of a quadrant, or a quarter of a disk. The disk device treats data and keys as serial bit streams. Each head looks at its own stream of bits as information: keys, data, and holes (spaces

with no data). There is no direct communication in this design between adjacent heads. Everything in the system is serial by bit. In order for each head to know what it is reading at a given time, these information entities are coded in the stream with the following format:

Each information item in the bit stream contains two fixed length fields and one variable length field. The first field is a two-bit indicator of the type of primitive information item that exists in the third field. The second field specifies the length of the variable length third field, which contains an arbitrary bit pattern. Thus each primitive information item can be depicted by Figure 1.1(a) in which,

field 1: two indicator bits specifying the three types of primitive information items,

field 2: binary integer indicating number of bits in field 3,

field 3: arbitrary bit pattern representing logical data entities of a record.

The following describes the basic primitive information items.

1. Hole item:

The bit pattern contains no information in field 3 and is therefore called garbage. It indicates that this slot on the track is available for key or data items. This type of

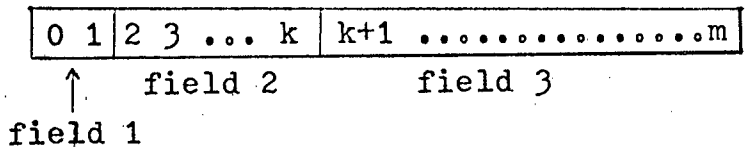

Figure 1.1(a) Primitive information item in bit stream.

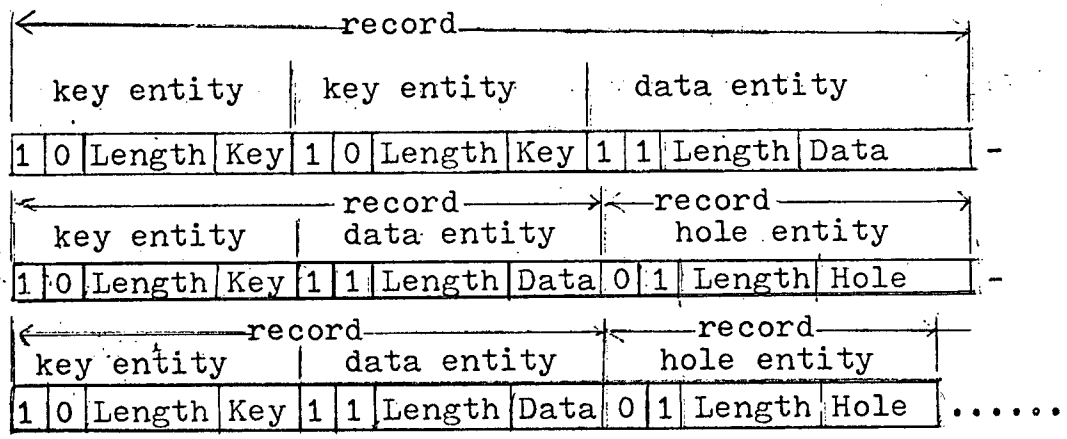

Figure 1.1(b) Sample Data Track in the Logic per Track Disk.

primitive information item spans over unused space.

2. Key item:

This type of primitive information item represents a key and its value, given by the bit pattern in field 3. A key is any logical item used to identify a class of related primitive data items.

3. Data item:

This primitive information item represents a record which is a collection of structured data items.

The following table gives the distribution of two indicator bits and their associated primitive items:

Indicator bits	Primitive information items
00	unused
01	hole
10	key
11	data

The following patterns for the entities have been chosen in (Parker, 1970) as a basic storage organization:

- a quadrant contains a number of records,
- a record is either
  (a) a hole, or
  (b) one key to several keys followed by a data item.

See Figure 1.1(b) for a sample data track. Since a record element is composed of an arbitrary number of key items, there exists a multiple keying potential for retrieving records residing on the disk.

It is described in (Parker, 1970) that each logic head contains four registers needed to process records residing on disk tracks. These four registers are the length register, bit register, auxiliary register and operation register.

1. Length register:

This is used to hold the length (given by field 2) of the current primitive information item being searched for in the bit stream.

2. Bit register:

This contains the current bit address on the track. It is used to find a record by location and for reporting back record locations to the controller.

3. Auxiliary register:

This holds temporarily a bit address while the head skips through an entity of no current value.

4. Operation register:

This register contains the operation code passed down by the controller. It also counts down the length of the current primitive item to find the beginning of the next item. This register has both shift and add capabilities so that it can compare the length of a desired item with the length of

the current item.

The following are some built in primitive instructions on each head:

load the operation register,

transfer one register to another in the same logic head,

search for a hole of specific length,

search for a key of specific bit pattern,

read from a specific location,

write to a specific location.

The important characteristic of the logic head is that it has the capability to identify every primitive entity in the bit stream; moreover, since each physical record in the bit stream ends either with the data entity or with a hole entity, the logic head is also sensitive to the begin/end location of a physical record in the bit stream. In other words, the logic head is designed in such a way that it is sensitive to all primitive entities, and physical records containing the specific patterns of the entities described previously. The patterns of the primitive entities in the bit stream will be generalised in Chapter 3.

1.2. Logic-Per-Track Retrieval System

A record residing on any track is retrieved by searching for one of its key entities. During a search, the key is broadcast from the central source. Since all entities

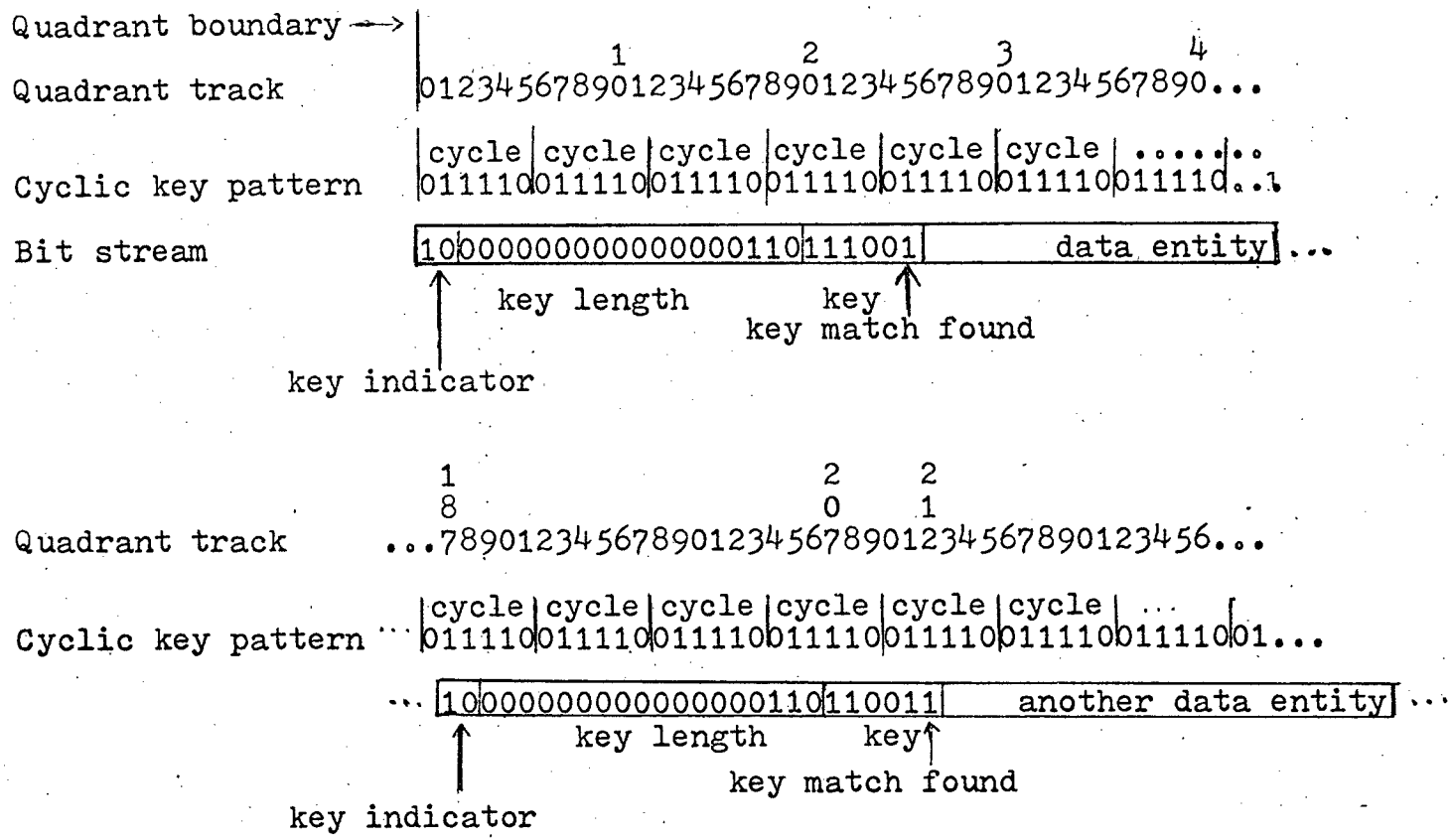

Figure 1.2. Cyclic key searching mechanism

are of variable length, the keys may not start at the same bit numbers. Therefore the key is broadcast to each logic head over and over again cyclically. Please see Figure 1.2 for the cyclic bit pattern of the key '011110' being broadcast from the central source. Each head compares whatever bit it reads from the key with the bit being broadcast. This implies that sometimes the last bits are compared first. On an equal-unequal compare this does not matter. As long as the keys can be written offset by the record writing software, the offset is always compensated by the broadcasting. If the key is to be written at a given bit address in the system that bit address is taken modulo the key length. If the bit address modulo the key length is not zero then the first part of the key is broken off. The number of bits corresponds to the bit address modulo the key length. This is shifted to the other end of the key and is broadcast last. It follows that a key written at different locations on a given track may show different information bit patterns in the third field of the key item. The following example demonstrates the cyclic key pattern.

Example:

Assume that the key value (field 3) is given by the 6-bit pattern 011110, that is,

bit count: 012345

key pattern: 011110

key indicator: 10

key length: 00...0110 in 18-bit format.

Thus, for k=20 two different cyclic key patterns written on two different locations on a track appear as follows:

Cyclic pattern 1:

```
Quadrant bit address:          1111111111 222222
                       01 2345678901234567 89 012345 .....

Bit stream:            10 000000000000000110 111001 .....
```

In this example, the first two bits are the indicator bits for key entity, and the next 18 bits is the binary number specifying the length of the key residing in the third field of this key entity. The bit address for the third field is 20, and this bit address modulo the key length is 2, and hence the key is broken in two parts depicted by the following figure:

first part	last part
01	1110

The last part is then shifted to the other end of the key, so that the physical pattern of the key at the bit location 20 appears as follows:

last part	first part
1110	01

This gives the cyclic bit pattern '111001' to be written in the third field of the key entity. How this physical key pattern is matched to the given key broadcast cyclicly from the central source is depicted in Figure 1.2.

Cyclic pattern 2:

The following is another cyclic key pattern for the same key but written at bit location 207. Please see Figure 1.2 for how the cyclic key pattern is matched to the given key.

```
                                  11 111111111112222222 222222
                                  88 899999999990000000 000111
Quadrant bit address:        ...78 901234567890123456 789012...
                                  10 000000000000000110 110011
```

The above example gives a fairly clear picture of the simultaneous cyclic "search for a fixed key" instruction. In the "search for a hole" instruction, the length of the hole to be searched for is loaded into the length register which will be used to compare to the length of every hole entity encountered by the logic head. Thus each logic head looks for a proper hole, large enough to contain the record that is

to be inserted. If the length of the hole is not exactly equal to the length of the record, it must be at least k bits longer, in order that the unused part can still be recorded as a hole.

On the search operations all heads are searching simultaneously for the given key. Therefore the entire disk can be scanned for a match on a single revolution. This takes about 25 milliseconds with most current disks. As soon as the location of the record is found, the record may be read from or written to the specified location. However, with a single logic head, this has to be done in the next revolution because the logic head can perform only one primitive instruction at a time. We shall return to this point in Chapter 3. The following common operations may be built up from the individual operations of the logic heads:

search for a fixed key,
insert or delete record or a new key to the record,
frequency counting based on a fixed key,
file recopy and restructuring, and
garbage collection--collecting all hole entities.

Simultaneous search for a fixed key comprises the basic idea of the logic per track retrieval system. Application of such a retrieval system to single key records has been described in detail by Parker (1970). However, there is no comprehensive description how such records can be

manipulated within a file. To solve this problem, a file identification mechanism for such a retrieval system is introduced in Chapter 2, and some accessing problems are also discussed.

CHAPTER 2

SOME ACCESSING PROBLEMS

2.1. File Identification Mechanism

Records are grouped together to form a file or files. Each record in a file consists of the same type of information held in an identical format as every other record in the file. An example is a personnel file that consists of employee records. There is one record for each employee and all records hold the following data: employee number, family name and initials, social insurance number, skill code(s), starting employment date, address etc. These data elements are present in all employee records. However, to retrieve a record, we must know before hand the name of the file to which it belongs, and in particular, the physical location of the file in the memory storage. It follows that a storage area identification mechanism for the device should be established. Functions of such a mechanism are (1) to manage file names, and (2) to manipulate data on the storage device.

The mechanism would establish a file name within a context by constructing a physical file identifier. A user refers to a specific file by a symbolic name. For each symbolic name in a given context there is a uniquely defined

internal physical representation in the form of a bit string. This bit string is the unique identifier associated with the symbolic name given by and referenced by the user. Moreover, the physical file identifier will be the key item written on the very beginning of the quadrant boundary of each track where the file resides. Supposing that no two files will share the same track quadrant, then the physical file identifier will identify the actual location of the file in the memory. Those unused quadrants will be identified by a uniquely designed key which is also written on the very beginning of the quadrant boundaries.

In this manner, all quadrants are named, and the very first key in each quadrant is the physical file identifier. To manipulate data on the file, all logic heads will be first positioned on a quadrant boundary, then the physical file identifier associated with the symbolic file name will be broadcast from the central source to all logic heads so that the key for the file name is searched throughout the entire memory. If the quadrant identifier matches the file name, then the corresponding logic head remains active, otherwise it idles. The process of searching for the file name repeated for every quadrant. This scheme obviously makes use of the facility for simultaneous search. Moreover, it will simplify the implementation of the I/O system, since most of the detail in locating physical records is performed

by hardware.

2.2. Some Accessing Problems

Consider the problem of processing information on the disk. Suppose that a set of records must be retrieved from the disk storage. These records may or may not vary in length but are arbitrarily distributed. The bit stream which contains holes, keys and data items must be originated from a fixed location on the track, as otherwise we cannot tell what each bit represents. Thus, there would be an average latency of $\frac{1}{2}$ revolution to get to the starting point. The latency is reduced by using four starting points, that is quadrant boundaries, which divide each disk track into four equal segments. With quadrant segments, the average latency is reduced to $\frac{1}{8}$ revolution; after one has reached the quadrant boundary there is the usual average rotational delay of half a revolution to find the record.

In what follows are brief descriptions of the three basic operations (Parker, 1972): search, insertion and deletion, and their execution time in term of disk revolutions.

1. Search operation:

steps	descriptions
1	Position logic heads to the next quadrant boundary and search for the file name ($\frac{1}{8}$ revolution on average).

2 Search for the fixed key (less than or equal to one revolution to find the matching key plus one revolution back to the record position again).

3 Read the matching record from the disk (fraction T of a revolution identical to the length of the record). If there is only one record to be searched, then the operation is completed. Otherwise, two extra steps (4, 5) are needed for more than one matching record.

4 Return to the position of the last matching point (1-T revolution).

5 Repeat from step 2 to step 4 until the search (step 2) has returned to the quadrant boundary at which it began at step 1.

2. Insert operation:

steps	descriptions
1	Position logic heads to the next quadrant boundary and search for the file name ($\frac{1}{8}$ revolution on average).
2	Search for a proper hole, large enough to contain the record that is to be inserted (less than or equal to one revolution to find the location of a proper hole entity plus one revolution back to the beginning of the hole entity again).
3	Write the new record and a new smaller hole entity - if any is left - into the place of the original

hole entity (a fraction T of a revolution which is identical to the length of the original hole in disk revolutions).

3. Delete operation:

steps	descriptions
1	Position logic heads to the next quadrant boundary and search for the file name ($\frac{1}{8}$ revolution on average).
2	Search for the first record (less than or equal to one revolution to find the first matching key plus one revolution back to the record position again).
3	Write a hole into the place of the record (a fraction T of a revolution identical to the length of the record). If more than one record is to be deleted, then two more extra steps (4 and 5) are needed).
4	Return to the position of the last matching point (1 - T revolution).
5	Repeat from step 2 until the search (step 2) has returned to the quadrant boundary at which it began in step 1.

It follows from the above description that even if it can be guaranteed that there is at most one match to the record key, the basic operations (i.e., search, insertion, deletion) cannot be performed on the same revolution due to the fact

that the same physical head can carry out only one primitive instruction at at time. If these operations are to be performed in less than one revolution time, an additional mechanism is required.

2.3. The Additional Mechanism

It is the purpose of this thesis to develop the additional mechanism (Chapter 3) in order to speed up the basic operations. The following is a brief summary of such additional mechanism:

To speed up the access time, we shall introduce a second logic head on each track. The second logic head, like the first logic head, is also sensitive to each primitive entity, the starting and finishing of each record in the bit stream. Its function is to read a record from the track or to write a record onto the track. Moreover the second logic head, when coupled with the original logic head over each track, will enable the device to perform some basic operations such as search, insertion and deletion within a single revolution. This means that the original logic head will search for a record key or a hole while performing these basic operations, and then the second logic head will carry out the read/write instruction for the operation.

Next, consider the simultaneous search instruction with only one logic head over each track. Suppose that two logic heads find matches at the same time; then these two

matched records must be read simultaneously. However, this may be impossible if a great many simultaneous matches are found. To resolve such a conflict some record(s) will be processed first. For example, simultaneously matched records may be processed according to the sequence number assigned to each track. However, this may require as many extra revolutions as the number of those records that are found. This is of course another inefficient retrieval scheme.

To give a general solution to the simultaneous matching problem, each record will be prefixed by a mark entity, and a record counter will be added to the device. The record counter is a builtin device that saves the number of matched records to be processed. Thus, in the simultaneous match, the number of matching records will be added to the counter. Whenever a matching record is processed, it will be marked, and the record counter decreased by one. A matching record will not be processed twice since it is identified by the marking mechanism, and all matching records will be retrieved as the completion is governed by the content of the record counter.

CHAPTER 3

THE MODIFIED MEMORY DEVICE

3.1. The Modified Memory Device

The main feature of the modified memory device is that there is another logic head associated with each existing logic head on every track. These two logic heads remain a constant distance of d bits apart. One will be called the control head (CH) while the one following will be called the read/write head (RWH).

The functions of all head couples are synchronized by a single monitor which we shall describe in detail later. The structure of the control and read/write heads is given by Figure 3.1. The control head has the same structure as described in (Parker, 1970), and the same functions for its registers: bit, length, auxiliary and operation. The RWH contains a bit register, a length register and an operation register which have the same functions as in the control head. It also contains a circular read/write delay register of d bits in length where d is the distance between logic heads on the same track. The bits of the delay register are scanned cyclically by the RWH at the same rate as bits on the track itself. Consequently, a delay bit set when the CH sees

a certain point on the track will be scanned again when the same point passes the RWH. The monitor also contains a record counter register which is common to all logic heads. Whenever the record key is matched successfully by a control head, the latter notifies its associated read/write head to set the delay and also notifies the monitor unit to add one to the record counter. Figure 3.2 sketches the delay device and the physical locations of its bits. By initialization of the delay device we mean all bits of the delay memory are set to '0'. To set/reset (or turn on/turn off) the delay device means a 1/0 bit is written at the indicated location by the RWH. The bit will also be referred to as the delay bit. We shall return to this topic later.

The following common operations may be built up from the individual operations of control heads:

search for quadrant name,
search for a key of specific pattern,
search for a hole of specific length,
unmark a record,
notify the controller when a match is found,
notify the associated RWH to read, to write or to delete the matching record.

The read/write head which follows the control head over the same track has the following capabilities:

read/write/delete the record from/to the

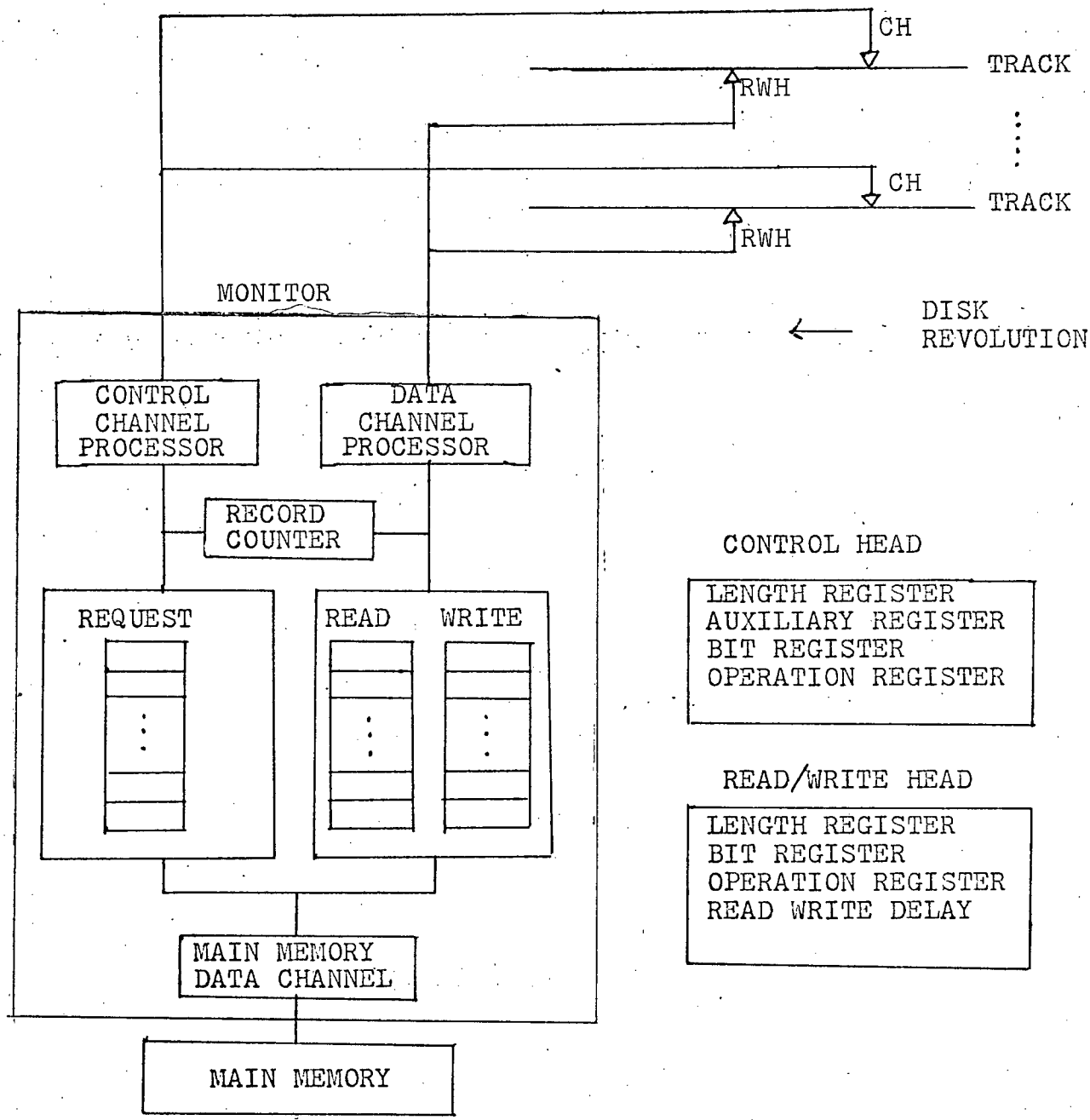

Figure 3.1. System configuration with request queue, read and write queuing buffers

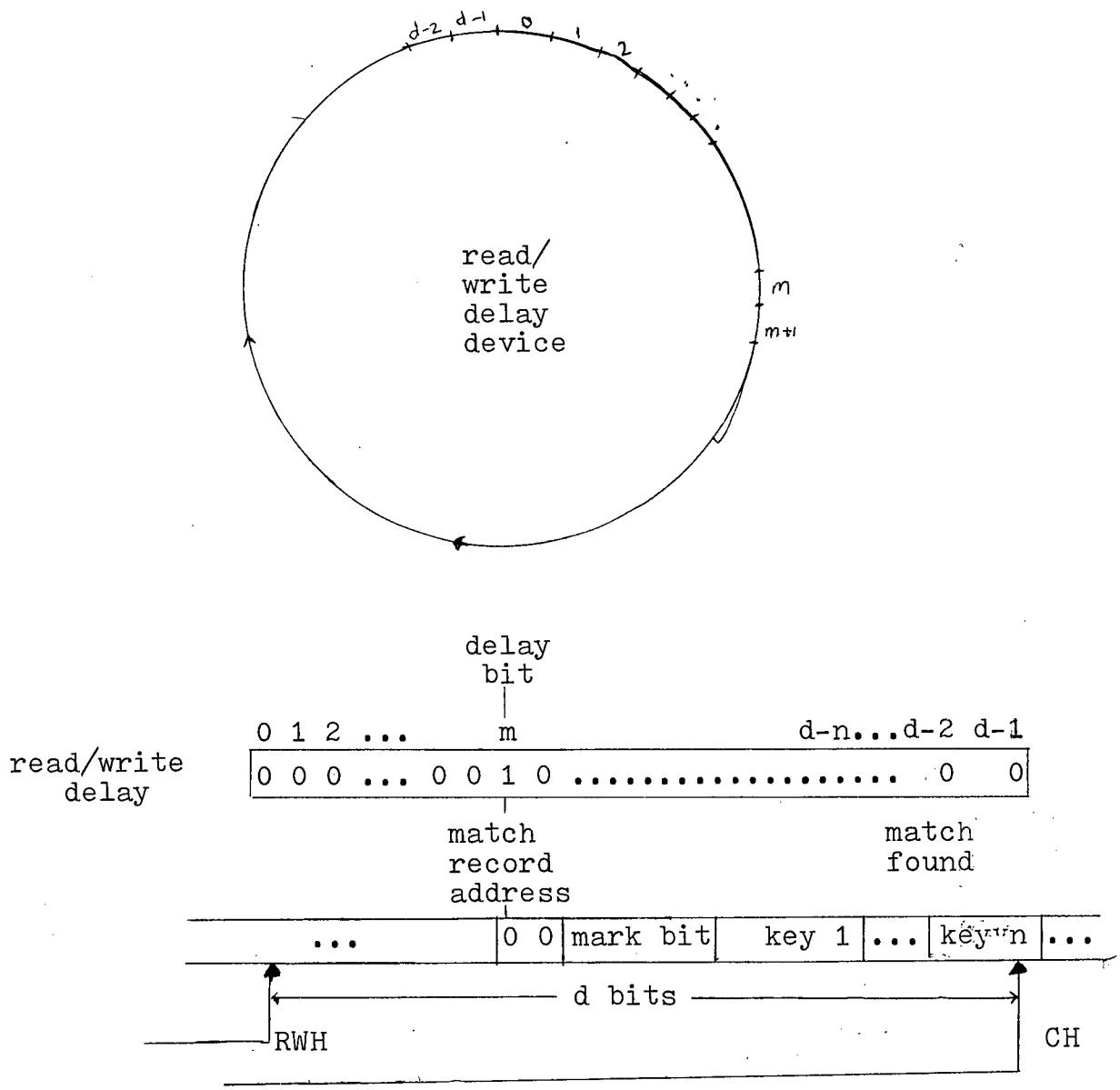

Figure 3.2. The cyclic memory delay device of the associative memory and its linear map to the disk track. The delay bit in the delay device is the bit which corresponds to the address of the matching record.

specific location as instructed by its control head or by the controller,

mark the matching record,

read records into the buffer in the garbage collection instruction.

As illustrated in Figure 3.1, attached to one end of the control channel controller are the control heads. Similarly, attached to the two ends of the data channel controller are RWH's and read/write buffer queues. Communication between the data buffers and the main memory is serviced by the main memory data channel. All outstanding requests are queued up in the request queue. Some possible requests are the following:

insert a record,

delete a record,

retrieve a record or a set of records,

garbage collection,

create a file, delete a file,

retrieve all file names.

We shall call these requests primitive operations with the property that only one at a time can be serviced by the monitor.

The activities of the control heads, read/write heads, control channel, data channel and the main memory data channel are supervised by the monitor. The function of the

control channel is to support the activities of the control heads. In particular, it broadcasts the file name and the record key in the cyclic bit pattern to each active logic control head in performing a search operation. In the meantime, it should be able to notify the monitor when the record is found. Thus it is a fairly sophisticated multiplex channel processor. The function of the data channel is to transmit data between read/write heads and data buffers, and to control the activities of the read/write heads.

3.2. Generalized Physical Data Format

Primitive information entities to be stored on the logic per track disk will be generalized to include the 'mark' entity:

indicator bits	mark bit
00	0 or 1

The mark entity is a one bit entity, and is always the first entity of a record in the bit stream. A 'hole' record will contain two entities: the mark entity followed by the hole entity. A data record will contain the following generalized entity patterns:

(mark, key, ..., key, data).

The following example demonstrates an important concept on the key part of a data record. Consider a simple data record pattern:

(mark, key, data)

The data record may contain personnel information, in which the social insurance number is the key. It may contain payroll information, in which the employee number is the key. These two distinct records share the same record pattern in a single file. Thus, a search for the social insurance number might end up with an employee number. In order to identify the record key properly, the key must be qualified by the object reflected by the key.

The mark entity not only leads a physical record in the bit stream, but also indicates the end of the preceding physical record. With this dual function of the mark entity, the logic heads can identify each variable length record in the bit stream. Because of this capability, the logic-per-track associative memory possesses all features of the conventional disk storage device, and is also an efficient information retrieval oriented device.

3.3. The Dynamic Behaviour of The Memory

The dynamic behaviour of the memory (that is, the complex of main memory, monitor, control channel, data channel, logic heads) is briefly illustrated in Figure 3.3. For simplicity, the memory is represented by five columns: the first column represents the monitor, the second column the control channel processor and the third column the control heads. Column four identifies data channel and read/write head activities, and finally, column five the main memory

MONITOR	CONTROL CHANNEL	CONTROL HEAD	DATA CHANNEL READ/WRITE HEAD	MAIN MEMORY DATA CHANNEL

I

1a: next request arrives

1b: activates control channel

2a: initializes all control heads and position them to quadrant boundaries

ONE REVOLUTION

II.a Phase one: search for file name

1c: initiates search for file name

2b: broadcasts file name to all control heads

3a: all control heads search for the file name

3b: each control head remains active if it finds the file name matched

1d: if necessary, activates main main memory data channel

II.b Phase two: search for the record key

3c: active control heads unmark record during the first revolution

1e: initiates search key instruction

2c: broadcasts key in cyclic pattern to all active control heads

3d: active control heads search for the record key

3e: if a control head finds a match, then it (1) notifies the monitor which in turn adds 1 to record counter during the 1st revolution, (2) marks on the delay device of RWH

1f: activates data channel if necessary

II.c Phase three: read/write operation

1g: if read buffer available and the data channel is free, then broadcast the ready signal to all active RWH's

4a: the RWH being triggered by the ready signal and the delay bit does the following: (1) resets the delay bit, (2) marks and reads the matching record; in the meantime, the monitor substracts 1 from the record counter.

4b: other RWH's being wakened by their delay clocks will just reset the delay bit

.continued ...

MONITOR	CONTROL CHANNEL	CONTROL HEAD	DATA CHANNEL READ/WRITE HEAD	MAIN MEMORY DATA CHANNEL
1i: if necessary, activates main memory data channel				transmits data to main memory
III		TEST FOR COMPLETION		
1j: if record counter≠0 then another revolution is needed and BOX II repeats; otherwise requst is completed.				

Figure 3.3. Control processes for the request:
retrieve a set of records with the given key.

data channel. The activity of various boxes in Figure 3.4 depends upon the specific request being served by the memory. By a request (or a primitive operation) we mean the quadruple: (1) the file name, (2) the record key, (3) the request type, and (4) the data. All the built-in operations of the logic head couple, and all the primitive operations described in the last section form some basic request types for the device. In some requests, such as read a record from or write a record to a specific location, the record key may be a dummy key, which will thus disable the activities of the control heads. With such a dummy record key, the search for the record key will not take place. A request to retrieve a record or a set of records with a given key in a file means all records with the matching key will be retrieved. The service of each request is divided into three major steps:

(1) initialization, box I,

(2) operation, box II, and

(3) test for completion, box III.

We shall describe the activities of the memory using a specific request: retrieve a set of records with key in a file. At the moment, we shall ignore most of the issues involved with the need to synchronize the various parallel activities of the memory.

Suppose that the memory is instructed to retrieve a set of records with the given key. The following steps must

be performed:

a) Position all control heads at the quadrant boundaries.

b) Search for the file name which is the first key in the quadrant.

c) Any control head which finds the file name matched will remain active, others will idle to the next quadrant boundary.

d) All active control heads search for the key.

e) If a control head finds a match, then the record will be read by the associated RWH, but meanwhile, the delay bit is set.

f) The outstanding matched record count will be kept in the record counter register.

g) Continue step d to step e thoughout the quadrant.

h) At the end of one revolution, step b to step g will be repeated if there is an outstanding matched record.

The execution of this request begins by the monitor instructing the control channel to initialize all control heads and position all control heads to the next quadrant boundary. These steps are represented by box I, and are executed once for each request. The next step is to find the quadrants where the file resides, that is, to execute box II.a. Note that we identified the file name by the quadrant name, which is the very first key on each quadrant. Accordingly, using

the simultanuous search feature of the device, the file name is broadcast from the control channel to all logic control heads, searching for the file identifier key. First of all, the length of the file name will be loaded into the length register, which is used to compare with the length of quadrant name. If the lengths are not equal, then the search is unsuccessful and the corresponding control head becomes idle. Consequently, it remains to compare those quadrant names with the same length. As soon as the last bit of the file name is broadcast, all matches will be found. The successful heads will remain active, whereas others will become inactive. Accordingly, the file location in the parallel quadrants is well defined so that records will be retrieved from the proper quadrants. This step is represented by box II.a and will be executed once for every quadrant.

The next and principal step (box II.b) is to search for the key and read every record (box II.c) with a matching key. One of the primary functions of each active control head is to unmark each record whenever the record comes under the control head in the first revolution. The key to be searched will be continuously and cyclically broadcast to all active control heads by the control channel controller throughout the parallel quadrants. If a control head finds a match to the record key, then the monitor will be notified via the channel controller. The channel controller adds one to the record counter register.

In the meantime, the control head will notify its associated head to mark the delay device. The record counter is used to ensure that all matching records are processed for the current request. If a read buffer is available and the data channel is free, the monitor initiates a control signal, called the ready signal, to all active RWH's. The signal will be terminated if either a buffer is not available or the data channel is not free. The RWH, when triggered by its delay clock, checks for the ready signal. If there is no ready signal, then it just resets the delay clock. Otherwise, it does the following:

(1) resets the corresponding bit in the delay device,

(2) reads the matching record into the specific location in the read buffer queue,

(3) notifies the channel controller to subtract one from the record counter, and

(4) marks the matching record.

An acknowledge signal is generated when the reading is completed. The monitor, once it receives the acknowledge signal, activates the main memory data channel and the record is transmitted to the main memory.

The above occurences will be repeated during one complete revolution. At the end of the revolution, the monitor checks the record counter register. If the record counter is greater than zero then another revolution is required and

box II will be repeated. All active control heads will find the same matches as in the previous revolution. However, in the second and subsequent revolutions, the matching control head will notify its associated RWH head to mark the delay device only if the matching record has not already been marked by the RWH. Thus the marking mechanism assures that each matching record will be processed once only.

Some remarks follow:

1. From the above description, at least one revolution is necessary for retrieving a set of records of the given key. If we know that there is at most one match for the given key, then the request is completed as soon as the matching record is read by the RWH, regardless of the rest of the revolution. This will save the unnecessary latency time for the next request.
2. The record counter has its contents changed if (a) the associated control head finds a match, or (b) the RWH begins to read the matching record. In case (a), the record counter will be increased by 1, and the corresponding bit of the delay device will be marked. In case (b), the record counter will be decreased by 1, and the corresponding bit of the delay device will be turned off. These two events may, however, occur simultaneously. To resolve such a conflict, the concurrent processes must be interlocked in order to keep the outstanding matching count up to date.

We should keep in mind that the above description is

only an example. Actually the memory must be able to carry out a variety of primitive instructions, and the actual activity of most boxes in Figure 3.3 is therefore a function of the instruction being served. There should be a way to select the desirable algorithm each time.

3.4. Synchronization of The Parallel Activities of The Memory

In what follows, we shall discuss the need to synchronize the various activities of the controller with each other and with the rotation of the disk. Such synchronization imposes constraints on the sequential processes of the memory that may run concurrently.

1. The controller cycle (Box II) is the total time that the monitor takes to execute boxes 1c to 1f. Box 1c must be executed in a time very significantly less than the time for the disk to move k bits, where k is the length of the first two fields in a primitive information item.

2. If more than one active RWH head is triggered by its delay clock simultaneously, then these heads check for the ready signal. If there is a ready signal, then the channel controller, which controls the activities of logic heads, will enable only one of these RWH heads.

3.5. Performance of The Memory

We shall now try to justify the revised memory device by describing its performance in carrying out the operations

which were defined as primitive instructions (that is, requests to the memory). Basic to every request is to position the control heads to a quadrant boundary, which take $\frac{1}{8}$ revolution on the average.

1. Retrieval of a single record

This is, of course, based on the assumption that there is at most one match for the given key. The average time is $\frac{1}{8} + \frac{1}{2} = \frac{5}{8}$ revolution. In the conventional fixed head file, it takes $\frac{1}{2}$ revolution to position the head to the starting point and since every track must be searched sequentially, this takes another $\frac{n}{2}$ revolution(s) to retrieve the record, where n is the number of tracks allocated to the file, so that the total time is $\frac{1}{2}(n+1)$ revolutions. However, some fixed head disks can detect segment boundaries (for example, IBM 2305 disk); then the average retrieval time is reduced to only $\frac{n}{2}$ revolution(s). Thus, the conventional device may be more efficient if the file has only one track. In what follows, we shall assume that the conventional device is a fixed head disk whose heads are sensitive to segment boundaries so that the time to position the heads to the next starting point is negligible.

2. Insert a record

In the conventional fixed head disk, the insert operation has two steps:

(1) getting to the point where the item has to be written: $\frac{1}{2}$ revolution.

(2) the actual writing process.

Since, in general, the time required to perform step 2 is significantly less than step 1, the service time is $\frac{1}{2}$ revolution. To insert a record in our memory has three steps:

(1) position control head to the quadrant boundary ($\frac{1}{8}$ revolution),

(2) search for an empty slot (hole) of appropriate size ($\frac{1}{8}$ to 1 revolution),

(3) write the record on the slot found, and

(4) write the hole entity on the tail of the slot.

Steps 3 and 4 need approximately s_T revolution time, the same as the conventional device, where s_T is the revolution time to write s bits of information on the device. Again, since this is significantly less than the time to perform steps 1 and 2, it is neglected in the following discussion. Step 2 needs up to one revolution if there is only one quadrant allocated to the file, or all quadrants allocated happen to be in the same disk quadrant. Since this will be the worst case, it should be avoided if possible. It follows that a file should occupy at least four consecutive non-parallel quadrants, since if the empty slots are uniformly distributed over the quadrants, then we can write on the first appropriate empty slot of the first quadrant encountered. In this case, step 2 requires on the average $\frac{1}{8}$ revolution, which gives the total service time of $\frac{1}{4}$ revolution. Thus, if the file is not

heavily loaded, we have a very good chance of improving the total service time.

3. Insert a set of records

In the conventional device, this takes $(\frac{1}{2} + s_T)$ revolution. Since s_T is the same in both devices, we shall drop it in the following comparison. In our memory, this depends heavily on the distribution of empty slots on the file. The worst case will be that these records have to be inserted into four nonparallel quadrants. This will need in general one revolution time. However, if we can insert all records in the first quadrant encountered, then the total service time is reduced to $\frac{1}{4}$ revolution (excluding s_T). Here is the average time distribution (excluding s_T):

Number of nonparallel quadrants encountered:	1	2	3	4
Service time, in revolutions:	$\frac{1}{4}$	$\frac{1}{2}$	$\frac{3}{4}$	1

4. Deletion

This instruction has 3 steps:

(1) position control heads to the quadrant boundary,

(2) search for the key ($\frac{1}{8}$ to 1 revolution),

(3) write a hole entity on the matching record.

If there is more than one match on the file, then the service time is $\frac{9}{8}$ revolutions. However, if there is a single match, then it is about $\frac{5}{8}$ revolution. In the conventional device, since every track should be searched sequentially, it takes

$\frac{n}{2}$ revolution to search for the key, and then erase the record, where n is the number of tracks allocated to the file.

5. Create or delete a file

This takes at most one revolution to write/delete the file name, i.e. change the quadrant name(s). However, in the conventional device, it is necessary only to create/delete a file directory entry which, of course, requires $\frac{1}{2}$ revolution on the average.

6. Garbage collection

Garbage collection is the most inefficient service. It will be called when:

(1) the file is highly loaded,

(2) a large number of small hole entities have been created as a result of insert/delete instructions, or

(3) no empty slot of proper size is found in performing an insert instruction.

Garbage collection in the first two cases is initiated by request, whereas in the third case it is initiated automatically by the monitor. Garbage collection is essentially equivalent to the act of file reorganization which results in improving the service time of other requests. However, the service time, as we have pointed out previously, depends on the loading factor and the distribution of hole entities. Hence, additional quadrants may be required if after garbage collection, each quadrant is still highly loaded.

To improve the efficiency of garbage collection, we shall assume that a free parallel quadrant and free buffer of one quadrant in length are always available. The function of the active control head of the file will be read only, that is, transmitting non-hole entities, one bit at a time to the free buffer via the control channel. The data channel will then transmit the corresponding data to the RWH of the free quadrant. If after the garbage collection, the quadrant is still heavily loaded, then an extra quadrant is required. The first half of the records will be written on the immediate next free quadrant, and the other half will be written on the subsequent quadrant. New quadrants allocated to the file will be marked so that garbage collection will not be applied to them. After the garbage collection, it requires an extra revolution to unmark all quadrants of the file.

3.6. Alternate to the Modified Memory Device

As we have seen so far, the fundamental mechanisms that makes the logic head couple work are the delay clock device and the record counter register. It may be expensive to implement the delay clock mechanism for each RWH. In fact, with a matching record queuing buffer, these can be replaced

by a single record counter register in the channel controller (figure 3.4). Each entity in the matching record queue contains a record address. Attached to one end of the control channel are the request queue and the matching record queue. When the control head finds a match to the record key, it performs the following:

(1) adds one to the record counter register during the first revolution,

(2) notifies the monitor of the matching record.

Whenever the monitor receives the matching signal, it puts the address of the matching record in the matching record queue if there is a free entry in the queuing buffer. In case there is no free entry, then the matching record will be picked up in the next or subsequent revolution. If the read buffer is available and the data channel is free, the monitor, which constantly keeps track of the read/write head location, selects from the matching record queue the record which comes first under a RWH and initiates the ready signal to the corresponding RWH. The RWH then marks and reads the matching record. Whenever a read instruction is completed, the counter will be reduced by 1 and the RWH will generate an acknowledge signal to the monitor so that the next head can be selected. At the end of the revolution, the contents of the counter will be checked. If the contents of the counter is 0, then all

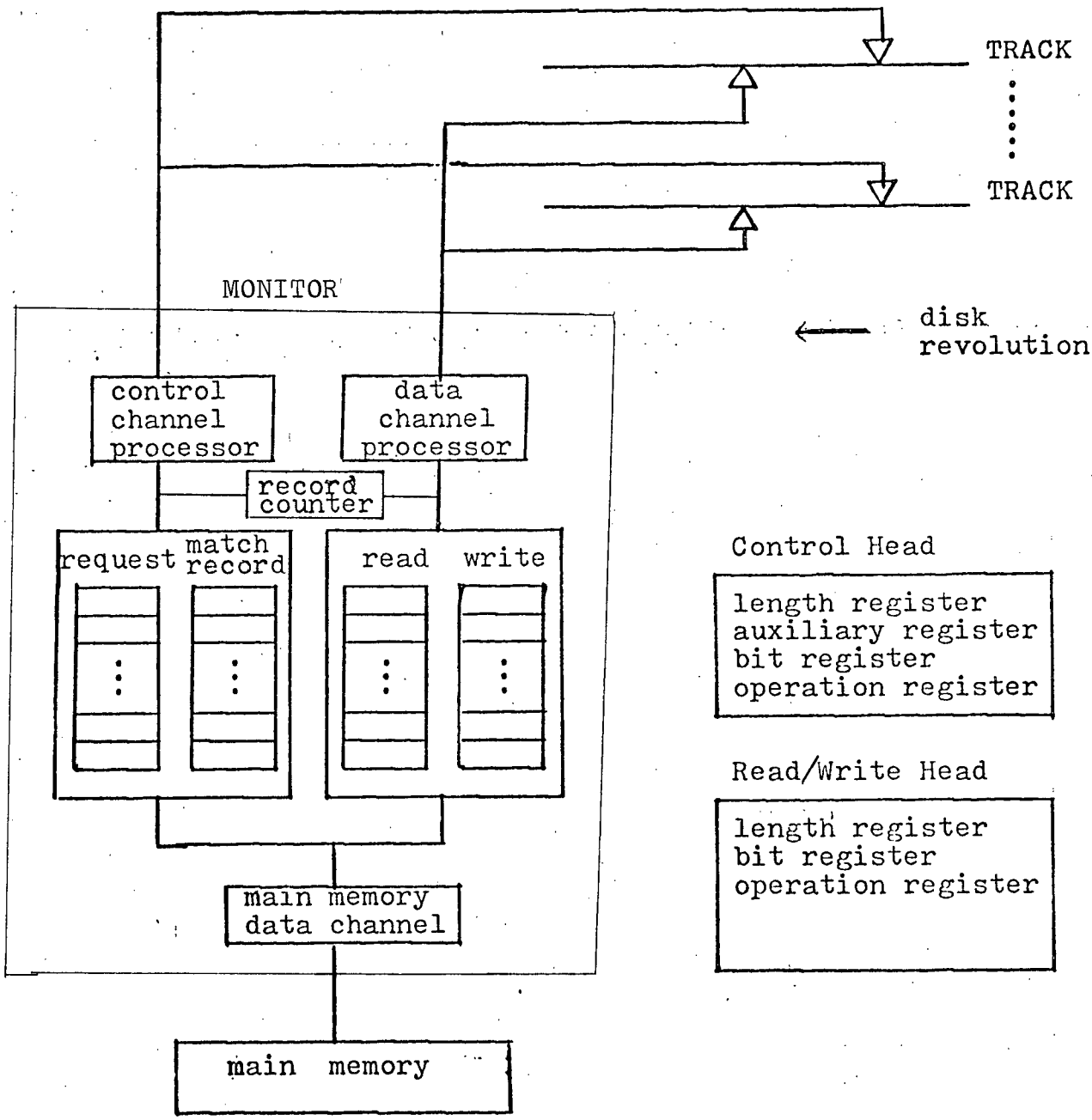

Figure 3.4. Alternate system configuration

matching records have been retrieved; otherwise another revolution is needed. In the next revolution, all control heads search for unmarked records, but do not add 1 to the record counter for matching records. The address of the matching record will be added to the matching record queue and the process is repeated as in the previous revolution. In this alternate scheme, there is one cycle of the selection-read-acknowledge process for each matching record, and each matching record must be in the queue prior to being accessed.

In order to read adjacent matching records of the same track or of parallel tracks sequentially, the time to generate the acknowledge signal and to execute the selection of the matched record should be less than two bits time, which is the time for the logic head to read the indicator bits of the mark entity. The RWH must receive the ready signal prior to encountering the mark bit of the mark entity. If more than one match to the key occurs concurrently, then the matching signals will be intercepted by the control channel processor. The control channel processor will then instruct the record counter to increase the content by the number of matching records; however, it will place only one of these matching records in the matching record queue. Since the monitor constantly keeps track of all RWH locations, it can always have the next selected matching record available, and in the meantime free those matching records whose addresses have

passed the RWH's. This implies that (1) adjacent matching records can be processed sequentially since the ready signal can be generated within two bits time, and (2) a free entry in the queue is available for other matching records. We shall see later that this alternate device limits hierarchical search in the simultaneous keying mechanism.

3.7. Hierarchical Search Mechanism

Consider the modified memory device described in section 3.1; that is, each RWH logic head is equipped with a delay device, but without the matching record queue for the system (Figure 3.1). As we have seen so far, the operational characteristic of the logic per track disk memory device is the simultaneous keying mechaism. This simultaneous keying mechanism enables us to establish another mechanism for the device, that is, an hierarchical search mechanism on the key part of the record. This, in fact, is not yet available in conventional disk devices.

Let k, k_1, k_2, ... k_n be keys of a record. A finite set of primitive predicates may be defined over the key-part of the records. These predicates, in turn define the hierarchical search. The following are examples of predicates which are likely to be included in the primitive predicates:

1. For a given key k, retrieve all records whose key-part has a key matching k, or has no key matching k. These predicates will denoted symbolically by k and -k respectively. In the simultaneous search, the given key is broadcast over and over again cyclically, and each head compares whatever bit is read from the key of the record with bit being broadcast, This implies that sometimes the last bits are compared first. As we have demonstrated previously, this does not matter on an equal-unequal comparision. However, the equal-unequal comparision mechanism enables us to implement the above predicate.

2. The next predicate is the limited combination of 'and' and 'or' logical operations on the key-part of records. Given a set of keys, k_1, k_2, ... k_n say, it is possible to retrieve records whose key-part satisfies the following predicates:

k_1 and k_2 and ... and k_m ,

k_1 or k_2 or ... or k_m ,

$-k_1$ and k_2 or k_3 or k_4 ,

k_1 and k_2 and ... and k_i $\underline{\text{or}}$ k_{i+1} $\underline{\text{or}}$ k_{i+2} $\underline{\text{or}}$

... $\underline{\text{or}}$ k_m

etc.

The introducation of the marking mechanism enables us to implement the above primitive predicates, and the introduction of the logic head couple on each track will speed up the sevice

time.

The following example demonstrates a predicate search on the logic-per-track associative memory. Suppose that we want to access a record whose keys satisfy the following predicate:

$$k_1 \text{ and } k_2.$$

Then in the first revolution, all active control heads will search for key k_1. If a key is found, then the control head will notify the associated RWH to mark the delay device, as described previously (Note that the alternate associative memory described in Section 3.6 contains a matching record queue instead of the delay device. A problem occurs if a match is found while the matching record queue is full; this would explain why it limits hierarchical search). This in fact, memorizes all matched records within d bits distance of each RWH. Then the RWH will in turn mark the matched record when the matched record passes under the RWH. In the next revolution, all active control heads will search the marked records for key k_2. If key k_2 is matched to a key of a marked records then the control head will again notify the associate RWH to turn on the delay bit. In this manner, when the RWH encounters the delay bit, it will leave the record as it is marked, otherwise, the RWH will just unmark the record. It follows that all marked records satisfy the given predicate,

and all desired records are ready for retrieving.

3.8. Evalustion of Hierarchical Search

We shall consider two types of hierarchical search: OR search and AND search.

a. 'OR' search:

An 'OR' search is a disjuction of equal-unequal searchs, k_1 or k_2 or ... or k_m. For example, k_1 is the key reflecting age in the personal record, and is qualified by the prefix: AGE. Thus a key of age 20 would be represented by 'AGE20". The procedure to retrieve records satisfying an 'OR' condition is:

STEP 1: Position all control heads to the quadrant boundary, $\frac{1}{8}$ revolution on average.

STEP 2: In the first revolution, the control heads unmark all records and search for key k_1, If a match is found by a control head, it will be marked by the associated read write head, and the record counter will be increased by one. At the end of the first revolution, all matching records are marked and the number of matching records is contained in the record counter.

STEP 3: In the next revolution, the control heads search for key k_2 on the unmarked records.

If a match is found, then the record will be marked by the RWH and the record counter will be increased by one.

STEP 4: Repeat STEP 3 for keys k_3, ... k_m.

STEP 5: The final step is to read all marked records. This requires at least one revolution, at most n revolutions, where n is the number of parallel quadrants allocated to the file. Therefore, the total access time is at least $m + \frac{9}{8}$ revolutions and at most $m+n+\frac{1}{8}$ revolutions.

b. 'AND' search

An 'AND' search is a conjunction of equal-unequal condition,

$$k_1 \text{ and } k_2 \text{ and } \ldots \text{ and } k_m$$

such that each k_i reflects a distinct object, for example:

'AGE20' and 'SEXFEMALE'.

The retrieval procedure is as follows:

STEP 1: Position control heads to next quadrant boundary, $\frac{1}{8}$ revolution.

STEP 2: In the first revolution, the control heads unmark and search for key k_1. If a match is found, then the record counter is increased by one, and the associated RWH marks the matched record. Thus at the end of the first revolution, the number of matched records is

contained in the record counter, and all matched records are marked. If the record counter is 0 then the request is completed, otherwise proceed to STEP 3. This takes one revolution.

STEP 3: In the next revolution, the control heads search on these marked records only. If a marked record does not match key k_2, then the record will be unmarked by the associated RWH and the record counter decreased by one. At the end of the revolution, if the record counter is 0 then the request is completed, otherwise repeat STEP 3 for keys k_3, ... k_m. This step takes at least one revolution and at most m-1 revolutions.

STEP 4: At this point the number of records satisfying the 'AND' condition is held in the record counter and all such records are marked. If the counter is 0 then the request is completed, otherwise read all marked records which takes at least one revolution, at most n revolutions, where n is the number of parallel quadrants allocated to the file. Therefore, the total access time is at most $(m + n + \frac{1}{8})$ revolutions, assuming that there is at least one record satisfying the AND condition and at least $\frac{9}{8}$ revolutions.

CHAPTER 4

STORAGE ORGANIZATION AND ACCESS METHODS

4.1. Chain Storage Organization In Logic-Per-Track Environment

The basic storage organization and access methods described in the previous chapter facilitate random file organization with direct access. Moreover, the marking mechanism provides an hierarchical access file organization. It is the purpose of this chapter to develop a more general storage organization, that is a chained storage organization, and to demonstrate some applications of such a file organization in a simultaneous search environment as the counterpart of the file organizations and access methods in conventional disk storage devices.

The basic method for structuring and processing records in a file in the logic-per-track environment is that records are stored, updated and retrieved by key or keys. The characteristic of the basic storage organization is that each record in a file consists of the same type of information held in an identical format as other records in a file; this form of record structure will be referred to as multiple-record structure. An example is a student file that consists of personal information, semester information and course information. A typical record in a student file may hold data

given by Table 4.1. With the logical components of information specified in this table, the student record may be stored as a whole as one physical record (the multiple record approach), or may be segmented into three physical records -- personal, semesters and courses. In either case a record key must be chosen, say the student number. Moreover, an additional key is needed in order to differentiate the personal records, semester records and course records in the file (Figure 4.1); the student record can be accessed by the student number, or by both the student number and the student record type. Consequently, records should be stored and retrieved in some way so that each record can be properly identified. One method for such record identification is given by Figure 4.1: each record is identified by the student number. All these records can be retrieved in random ordering by the student number with a single request to the associative memory. If the records must be retrieved in a fixed order, then they should be chained together.

A unary chain is the simplest technique of linking logically related records; related records are linked by a pointer that contains a reference to the next related record in logical sequence so that all related records are chained together by successive pointers. Implementation of a chain in the logic-per-track environment is described below.

1.	Personal Information:	student number
		name
		social insurance number
		sex
		birth date
		place of birth
		institution last attended
		previous degree held
		record to-date
		year of first admission
		address
2.	Semester Information:	faculty, dept. and year attended
		Academic session
		tuition fee
		standing
		etc.
3.	Course Information:	grades
		course number
		course name
		meetings
		description

Table 4.1. Logical components in a typical student record

MARK	KEY	DATA		
	student no.	personal	semester	courses

↑Student information is stored as a whole by one record: the key entity holds the student number, and the data entity holds personal , semester and course information as specified in the logical structure of the student record.

MARK	KEY	KEY	DATA
	student no.	record type	personal record

MARK	KEY	KEY	DATA
	student no.	record type	semester record

MARK	KEY	KEY	DATA
	student no.	record type	course record

↑Student information is stored as three records in the file: personal semester and courses. Moreover, each record is identified by the student number (KEY) and the record type.

Figure 4.1. Examples of the student record stored in the student file.

Depending on the software approach being used, the contents of the pointer can be either of the following:

1. Actual address: The pointer is a key entity which holds an actual physical address. The track address of the logic head, and its location, are controlled by the channel controller. The chain may be created as follows: Each record of the chain is allocated a key entity, which will hold the address of the preceding record or the succeeding record. If the pointer holds the address of the preceding record, then we have a backwards chain; when the current record is created, its address is saved to be used as a key of the next record in the chain. On the other hand, if the pointer holds the address of its succeeding record, each record in the chain will be created in two steps: (1) write the current record and save its address, (2) insert the address into the chain pointer of the preceding record. Because each record in the chain depends on the physical position of the next record, all pointers may have to be revised after each file reorganization or garbage collection.

The revised physical record formats for the associative memory device introduced in Section 3.2 can avoid the tangle of actual physical addresses in the chain structure. A chain organization which utilizes such a revised physical record format is described below. In what follows, the chain

organization (chain structure) simply refers to the chain utilizing such a revised physical record format for the associative memory. In other words, records are not chained by actual physical addresses.

2. Key pointer: The pointer is a key entity of each record in the chain. An element of the chain is a structure and at least one field in each record is a pointer value that is used to point to another record. Obviously, the purpose of the chained organization is to tie together the elements of a data structure and the chain key is properly chosen by the application program. In the logic-per-track environment, each record in a chain needs two key fields: one for the key of the record, and the other for the pointer field (Figure 4.2). The pointer field contains a chain key which is generated from the key field of the record it points to by a simple (and reversible) process such as appending a single special character. For example, in a student record structure, the personal record has a record key '7500001', and a pointer field which contains a chain pointer '$7500002' pointing to a succeeding personal record in the chain. The chain pointer '$7500002' is generated from the succeeding personal record by appending a single special character, '$' say, to the key of the succeeding record of the student number 7500002. By continuing this process, the student personal records are chained together. This simple example is adequate to

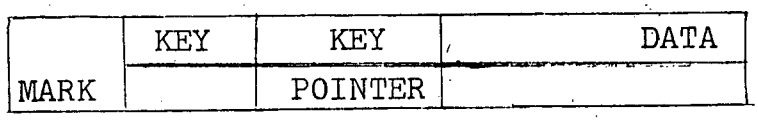

Figure 4.2. Basically, two key fields are needed for each record in a chain: one for the key of the record and the other the chain pointer.

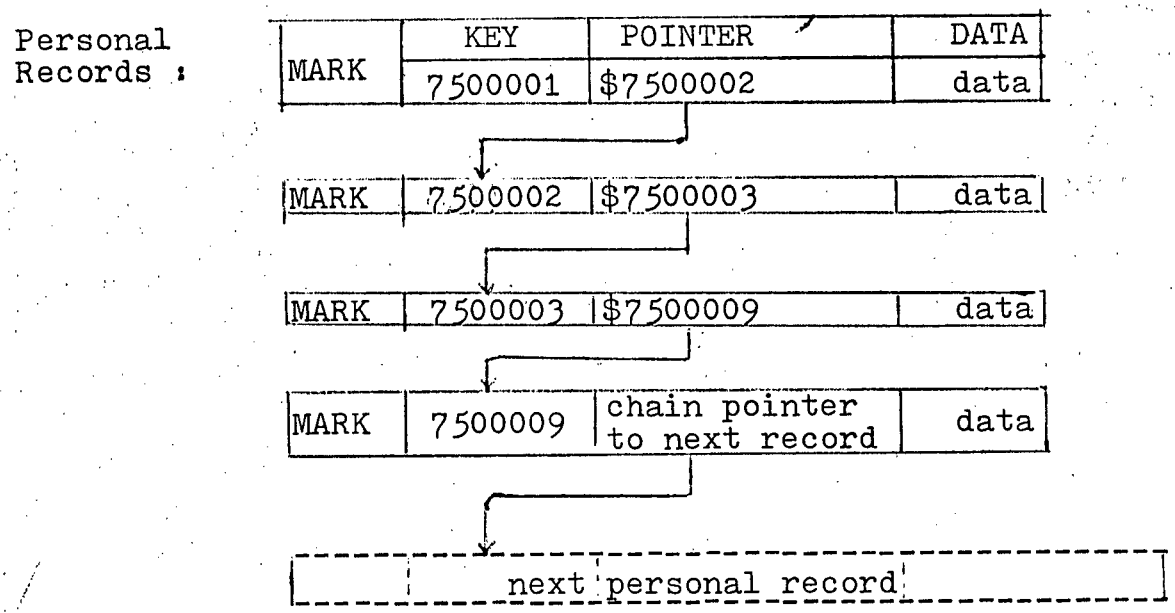

Figure 4.3. Sample chain of student personal records. Each record in the chain has the format as shown in Figure 4.2.

demonstrate the following important features of the logic-per-track retrieval system:

a. The chain key of each record is reversible. For example, with the chain key of any record, say '$7500003', by applying the reverse procedure, we can obtain the key of the succeeding record: 7500003. Thus the chain is in fact a binary chain with only one pointer field in the chain, which is impossible in conventional computer storage.

b. Each record in the chain can be retrieved by following the chain key, or directly by the key of the record if it is known.

In general, a record in the chain can include any number of pointer fields. In the simultaneous search environment, each record in the storage is uniquely identified by one of its key(s). If more than one record possesses the same key, then a simultaneous search for the key would encounter all these records regardless of their physical positions in the chain. If the order of records in the chain is important, then elements in the chain must be linked in the desired order. We shall call such a chain a linked list, analogous to list structures in computer storage. From the remark above, no two distinct records may contain identical keys which serve as pointers in a linked list. The key in the pointer field of a chain may be generated by a simple and reversible procedure.

The chain may be ended with an end-of-chain marker, or by discontinuing the pointing key. The end-of-chain marker is a specified key value. Each chain record once accessed will be checked against this key by software, in order to test for the end-of-chain. If the chain is terminated by discontinuing the chain key, then an attempt to access the record with the chain key of the record will fail to match any record in the file, and accordingly, will signal the end of the chain. The first record of the chain may be accessed by one of its keys. This key may be (1) the chain key of some other record, (2) obtained from some other file, or (3) supplied by the user, depending on the data structure of the file. The following is the procedure to retrieve the entire forward chain:

(1) Retrieve the first record of the chain ($\frac{5}{8}$ revolution on average).

(2) Generate the key of the succeeding record from the chain key of the current record; retrieve the succeeding record by this key ($\frac{5}{8}$ revolution on average).

(3) Repeat step (2) until the end of the chain is encountered.

Thus, with forward linking, we must find and read the record with key K before we can locate its successor. Step (2) reads, with backward linking: construct a chain key for the predecessor from the key K of the current record, then retrieve the preceding record by its chain key.

4.2. Some Aspects of Chain Structures

Some of the most important aspects of processing a chained file are the following:

1. File access -- sequential method,
2. File maintenance -- making insertions and deletions,
3. Describing and building more complex data structures.

1. Sequential File Access

Sequential access means that the data elements (that is, logical records) are referenced in a manner dependent upon the sequence in which data elements are stored. The sequence of records is linked by a chain, and the file records can be accessed sequentially by the procedure described in Section 4.2.

2. Insertions and Deletions

If the logical sequence of records in a chain is of no significance, insertions are simply arranged. If a record is to be inserted in the chain, a pointer is created in the new record to reference the first record in the chain. The key of the first record in the chain is generally held in the master record (for example, a record with a dummy key would work well). The pointer held in the master record is then altered to reference the newly inserted record. Where the logical sequence of records within a chain is important, then with the conventional device, the chain must be followed from the beginning until the insertion point is reached. However,

this may not be necessary in the logic-per-track environment. Insertion need not require that the chain be followed from the beginning if the key of the record at the point of insertion is known. In this case, the record can be retrieved directly by simultaneous search. The pointer of the preceding (or succeeding) record is then altered to reference the new record. The original pointing key in the preceding (or succeeding) record is inserted in the new record, which then re-establishes the link to the next record in the chain. The same is true of the conventional device, if the address of the record at the point of insertion is known. However, it requires that the address of the record in a chain be stored somehow separately whether in the core memory or in another disk file.

The deletion of records is an equally simple matter. Regardless of the logical sequence of the records in the chain, the record to be deleted can be located and can be retrieved directly by simultaneous search. Moreover, the successor (or predecessor) of the deleted record can always be found directly because the chain key of the successor (or predecessor) is constructed by the simple and reversible procedure from the key of the deleted record. This record will be deleted once it is accessed. The chain key of the deleted record will be stored as the pointer key field of the succeeding or preceding record, and the deletion is completed.

4.3. Tree Structure Representations

One of the data structures that model many real-life situations is the tree structure. For example, tree structures can be used to describe: the organization structure of a business; a student registration reporting system organized by department, courses, etc. A tree is defined recursively as a finite set T of nodes structured as follows:

(1) one node is designated as the root of the tree (T); and

(2) the remaining nodes are partitioned into disjoint sets T_1, T_2, ... T_n, each of which is a tree.

The trees T_1, T_2, ... T_n, are known as subtrees of the tree T. A terminal node is a node which has no subtrees and a nonterminal node is referred to as a branch node. For example, in the tree structure depicted in Figure 4.5, node A_1 is the root of the tree, node B_1 and B_2 are branch nodes, and nodes B_3, C_1, C_2, C_3, C_4, C_5, C_6 are terminal nodes. Figure 4.6 gives an example of the student, semester, and course data represented as a tree structure.

There is more than one way to implement tree structures in the logic-per-track environment. For example, in Figure 4.7 indexes to the subordinate records are kept with each parent record, I(B) denotes the index (that is, the chain key) to the record with key B. From a record with key A, we can compute the keys B_1, B_2, B_3 from the index fields and then

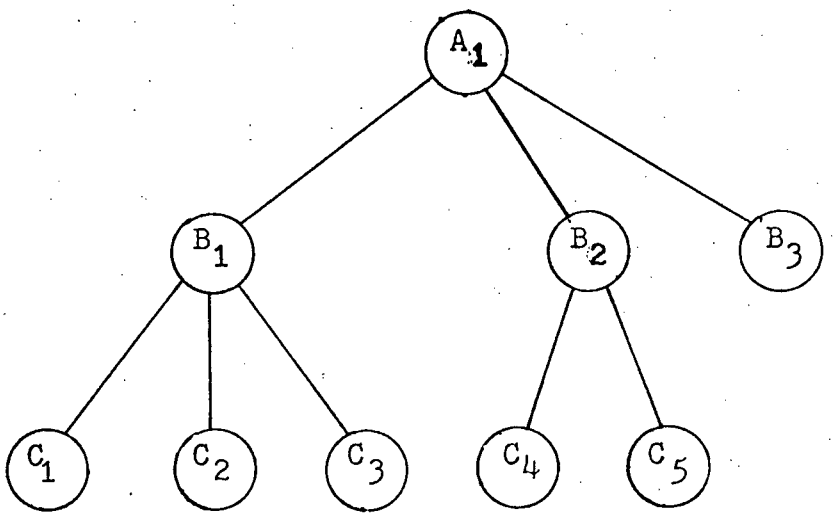

Figure 4.5 Conceptual view of tree structure.

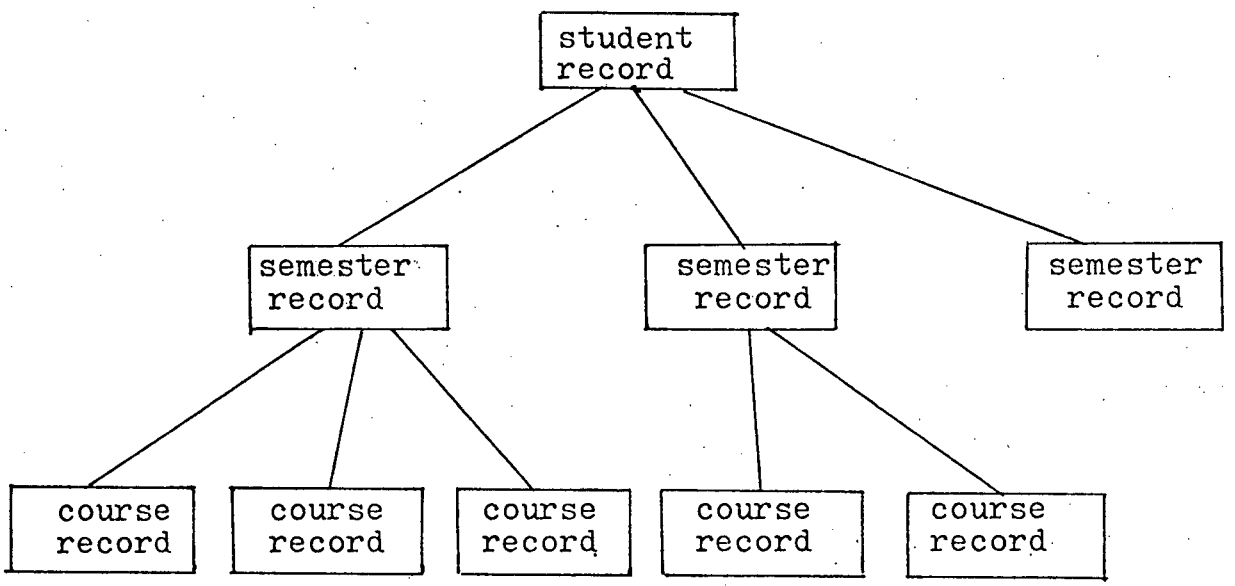

Figure 4.6 Example of student record represented by a tree structure.

A_1	$I(B_1)$	$I(B_2)$	$I(B_3)$	data
B_1	$I(C_1)$	$I(C_2)$	$I(C_3)$	data
B_2	$I(C_4)$	$I(C_5)$	data	
B_3	data			
C_1	data			
C_2	data			
C_3	data			
C_4	data			
C_5	data			

Figure 4.7. A data organization representing Figure 4.5.

A_1	data		
B_1	$I(A_1)$	data	
B_2	$I(A_1)$	data	
B_3	$I(A_1)$	data	
C_1	$I(B_1)$	data	
C_2	$I(B_1)$	data	
C_3	$I(B_1)$	data	
C_4	$I(B_2)$	data	
C_5	$I(B_2)$	data	

Figure 4.8. Another data organization representing Figure 4.5. Note that both representations have the same amount of storage space.

access directly the subordinate record with each of these keys. Also, given record B_1 say, we can access the parent record A by computing the index $I(B_1)$ and using it as a search key. A different data organization is shown in Figure 4.8. Here, each subordinate record stores an index to its parent record. Given any record with key B_2 say, we can retrieve its predecessor which is the record with key A_1, or all its immediate successors which are the records with key $I(B_2)$ because both keys $I(A_1)$ and $I(B_2)$ are formed by a simple and reversible procedure from the keys A_1 and B_2, respectively. Both data organizations take the same total amount of storage space.

4.4. Evaluation of Chain Structure Search

In the logic-per-track environment, the chain structure is the major storage organization besides the basic storage organization described in Chapter 3. Evaluation of the performance of such a storage organization is necessary. Assume that chain records are distributed uniformly around the disk track, and the chain has n records.

a. Retrieval Time for a Chain

A record in the chain may be retrieved in two ways: (1) by directly accessing it if a key of the record is known, and (2) by following the chain otherwise. If the key of the record is known, than the retrieval time is $\frac{5}{8}$ revolution on the average (Section 3.5). Suppose on the other hand that the key of the record to be retrieved is not known; then the chain must

be followed. This will take $\frac{5n}{8}$ revolution.

b. Insert Time

Inserting a new record R in a forward chain can also be done in two ways: If the key of the record P at the point of insertion is known, then the procedure is simply:

1. Retrieve the record P directly ($\frac{5}{8}$ revolution on average). The record P will be deleted in the next revolution (one revolution).
2. Move the chain key of the retrieved record to the chain key field of the record R to be inserted. Replace the chain key of the retrieved record P by the chain key derived from the key of the record R to be inserted.
3. Write record R and the record P ($\frac{10}{8}$ revolution). Therefore, the total time is $\frac{23}{8}$ revolutions on average.

Suppose on the other hand that the key at the point of insertion is not known, then the chain must be followed. This would take at most $\frac{5n}{8}$ extra revolutions, where n is the length of the chain.

c. Deletion Time

Deletion of a record is much simpler because the key of the record to be deleted is known; the record can be deleted directly, which takes $1\frac{5}{8}$ revolutions on average.

4.5. An Inverted Data Base Model for Conventional Disk Device

To draw a comparison with conventional devices, an inverted data base structure (Cardenas, 1975) will be used.

In this model of an inverted data base, each record is accessable via a hierarchical structure of three directory files: a track index file, a key-value index file and an accession index file (Figure 4.9).

The following is the inverted file organization description (Cardenas, 1975, page 255). The inverted file organization consists of four types of blocks, where the block or page is the unit of data physically transferred between secondary storage and main memory as a result of a single I/O operation:

a. Data blocks, which contain the records, that is the data base. The complete record is stored, including the inverted key-values.

b. Key-value index blocks, which contain the access keys. An access key consists of the pair key-name/key-value; a short representation of the inverted key-name is stored addition to the key-value. These are noted I and VALUE, respectively, in the key-value format shown in Figure 4.9. The VALUE could be fixed length with blank fill, but will be considered variable length and thus will be preceded by a key-value length indicator, kl. Associated with each access key is a pointer, PTR (the actual physical address), to the beginning of a list of record addresses (often called the accession list) and the length, LGTH, of this list. The access

keys are stored sequentially in key-value blocks and ordered by key name I, then by VALUE.

c. Record address blocks or accession blocks which contain the lists of "accession numbers", which are pointers to records in the data base. Each list is a series of consecutive words. No information is stored within the accession block as to where a list begins or ends, since this information is provided in the key-value blocks. Each list is ordered according to the value of the pointer.

d. A track index block is provided for fast access to key-value blocks. The format of each track index block entry is identical to the key-value format. The track index block contains each key-value appearing at the beginning of a key-value block and the pointers to these blocks (as diagrammed in Figure 4.9). The use of one track index block probably suffices for most applications.

In order to compute average access time, estimates of three basic units of time are used:

1. T_T, the average time to access a block or page (that is, a track).
2. T_C, the average time to compare an access key or key-value.
3. T_I, the average time to intersect or to merge two blocks of accession numbers.

These estimates vary depending on device environments.

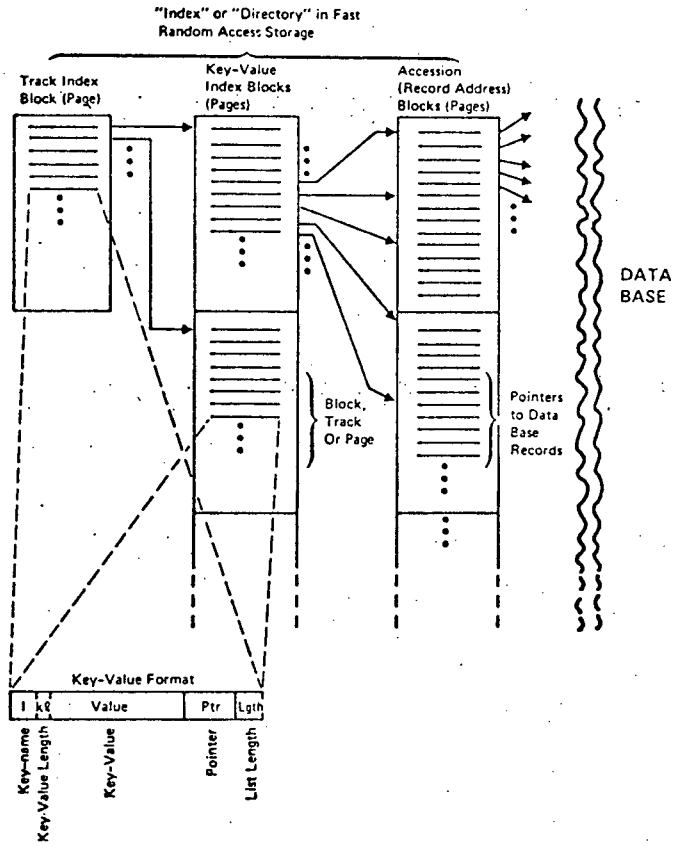

Figure 4.9 Layout for the inverted file organization for the conventional disk device discussed by Cardenas, 1975. (Figure 2, page 254, CACM, May 1975).

In what follows, we describe briefly the average access times for Equal-Unequal search, 'OR' search and 'AND' search in the conventional device; details can be found in Cardenas (1975). The average access time for each search is derived step by step in terms of the basic units of time as if one were taking each step through the directory and into the data base. The track index block, the key-value blocks and the accession blocks are assumed to be in ordered form and thus binary search is used where possible. Moreover, a large block size is used coinciding with track size ... that is, the pages accessed are whole tracks. If N denotes the number of access keys that are stored sequentially in each block, then the average search time for a key is $T_C * \log_2 N$, which is small comparing to T_T and T_I, and therefore, will be ignored in the following discussion.

a. Equal-Unequal Search

An equal-unequal search, 'greater than' and 'less than' searches have the same access time according to the analysis of Cardenas (1975). The following are the procedures and access times for these searches in the conventional disk device:

Step 1: Read track-index: T_T

Step 2: Search track-index

Step 3: Read key-value block: T_T

Step 4: Search key-value block.

Step 5: Read accession pointer list: $T_T * X_a$, where X_a is the average number of accession pointer

blocks.

Step 6: Read data blocks: $T_T * X_D$, where X_D is the average number of data blocks that will be accessed per query. According to the analysis of Cardenas, for the storage organization in the conventional device, the average number of blocks that must be read is: $X_D=D(1-(1-1/D)^K)$ data blocks, where D is the number of data blocks in the data base, and K is the average number of records satisfying the query.

In the above procedure, Step 1 to 5 search for all locations of records residing in the data base satisfying the query, and the total average access time is $T_T*(2+X_a)$. Because T_T is the average time to access a track, and $X_a > 1$, the time to searching the directory will be at least 3 revolutions ($3T_T$) on average comparing to only $\frac{9}{8}$ revolutions in logic-per-track environment. In step 6, $X_D=D(1-(1-1/D)^K)$ is the average number of data blocks containing K records satisfying the query. If $K \ll D$, then $X_D \approx K$. In this case, both devices would take about the same number of revolutions to read K records. If K is large, then $X_D < K$, the logic-per-track device may be slower, but this always can be compensated by the search time.

b. 'OR' and 'AND' searches

'OR' search: k_1 or k_2 or ... or k_m

'AND' search: k_1 and k_2 and ... and k_m.

Step 1: Read track index: T_T

Step 2: Search track index

Step 3: Read key-value block: T_T

Step 4: Search key-value block

Step 5: Repeat Steps 3 to 4 for each of the equal-unequal searches: mT_T

Step 6: 'OR' search: read and merge accession lists.

'AND' search: read, merge and intersect accession lists.

In both cases, the time is $(T_T+T_I)*m*X_a$

Step 7: Read data blocks: T_T*X_D.

Therefore, in either 'OR' or 'AND' search, the directory blocks access time is

$$T_T+m(T_T+T_I)X_a+mT_T$$

$$= (1+m+mX_a)T_T + mX_aT_T$$

According to the analysis in Section 3.8, it takes an average of $m+\frac{1}{8}$ revolutions for the logic-per-track device to search and mark each matching record, compared with $1+m+mX_a$ revolutions (ignoring mX_aT_I) for the conventional device.

It is now clear that significant improvements in access

times can be achieved by using the logic-per-track environment. Moreover, this brief comparison drawn between the logic-per-track associative memory and the conventional one has demonstracted the potential advantage of this new approach in information systems.

4.6. Conclusion

The basic method for structuring and processing records in a file has been discussed: records are stored, updated and retrieved by key or keys. It is important to note that our interest in information systems in the logic-per-track environment is considerably more than an intellectual excursion into new methods of organizing and accessing data. The logic-per-track information system provides immediate and efficient real time query access to integrated informational resources. The characteristic demand in such an informational resource is that inputs to the system are heterogeneous and occur at random intervals. Requests for information from the system are dynamic and time-dependent, so that quries and responses cannot be prepared before hand. Files are large; thus, redundant data must be factored out to reduce the waste of storage space. Moreover, information must be current and up to date to meet the needs of a modern society. Therefore, an informational change must be reflected immediately in all associated files in the informational resource. With the logic-per-track associative memory a retrieval system is able to meet the requirement for

fast access to large, integrated files that are designed to be updated dynamically in a real time operational environment. In an informational query system, the number of records satisfying the query is usually small, and therefore they can be retrieved within one and one-eighth revolutions time which is almost impossible for a large file in a conventional device.

The developement of the rotational associative memory for such an efficient informational query system should be credited to Slotnick (9170) and Parker (1970) for their earlier work. In the device of Slotnick and Parker, there is a fairly sophisticated logic chip attached directly to each read-write head. All individual logic heads are connected together and communicate with a central source. The logic chip allows each head to operate as an independent device, able to read and write its own track, to search <u>simultaneously</u> for information matching a given key, or to search <u>simultaneously</u> for space to write new records. The simple but unique cyclic key pattern (Figure 1.2) introduced by Parker (1970) makes the simultaneous search mechanism possible (Section 1.2). Because all heads could perform searching simultaneously, a desired key, or desired space could be located in no more than a single revolution of the device. However, reading and writing cannot take place on the same revolution once the position of the record has been discovered because the search key may not be first in the record so part of record will have passed the head before the match is

recognized (Section 2.2). Moreover, in the simultaneous key-matching operation, if the search key is not a unique record identifier, but rather appears as a key in a number of different records in the file, then simultaneously matched records (if any) must be retrieved, in an unspecified order. In the device of Slotnick and Parker, one read-write head at a time can actually be reading a record for transmission to the main computer. Extra revolutions, and extra bookkeeping, will certainly be needed if matching records on different tracks should partially overlap. These problems have been ignored in the retrieval system developed by Parker (1970, 1972). It has been the purpose of this thesis to modify the design of Slotnick and Parker in order to provide a means for reading and writing a record in the same revolution after its position has been discovered, and to provide necessary bookkeeping for retrieving simultaneously matching overlapping records.

The following are four additional features of the device:

1. Two logic head on each track, having similar but distinct logical capabilities, remain a constant d bits apart (Figure 3.1). The leading head, called the Control Head (CH) will continue to have the primary responsibility for simultaneous searching. The additional second head, called the Read-Write Head (RWH) trailing a fixed distance behind the control head as the track is scanned, will do the actual reading and writing of records (Section 3.1).

2. A delay register of d bits in length, where d is the constant distance between logic heads on the same track, has been added to the Read-Write Head. In the simultaneous key search and read (or hole search and write) operation, the control head, when it recognizes a successful match, will signal its RWH partner to begin reading (or writing) at the appropriate position on the track. Such an appropriate position should always lie somewhere between the two logic heads as they scan along their own disk track. The bit in the delay register corresponding to such an appropriate position is called a delay bit (Section 3.1), and will be marked by the RWH whenever a match is recognized. The delay time is the number of bits counting from the position of the RWH when a match is reported, to the location of the matching record. After such a delay, the RWH upon encountering the delay bit just set and the desired record position simultaneously (or almost simultaneously), will always reset the delay bit. Thus, the function of the delay bit is to tell the RWH partner where to start reading (or writing) a record so that retrieving (or writing) a single record can always be performed in the same revolution. The functions of all head couples are synchronized by a single monitor (Section 3.3).

3. Another major design change will give the new device the ability to keep track of all records which may be retrieved simultaneously (or almost simultaneously) by a parallel search. To this end, the monitor will be provided with a record counter

(Figure 3.1), and a mark entity (Section 3.2) will be prefixed to every record on the disk itself. The record counter is common to all logic head couples. In a simultaneous search, the number of matching record will be added to the record counter. Whenever a matching record has been processed, it will be marked by the RWH, and the record counter decreased by one. Accordingly, a matching record will not be processed twice since it is identified by the marking mechanism, and all matching records will be read because the completion is governed by the content of the record counter.

4. A file identification mechanism has been established for the associative memory (Section 2.1). Functions of such a mechanism are (a) to manage file names, and (b) to manipulate data on the storage device. In order to ensure that any simultaneous search operation confines its attention to the desired file, the file name will be the key item written on the very beginning of quadrant boundary of each track where the file resides. Supposing that no two files will share the same track quadrant, then the first key of each quadrant will identify the actual location of the file in the memory device. Thus, a file is made up of a number of track quadrants rather than whole track(s).

A schematic representation of the complete associative memory system is given by Figure 3.1: disk, logic head couples, and the monitor. The monitor includes the record counter

register, channel processors for the search and read-write heads, read and write buffers, and a queue for requests received from the main computer memory. The dynamic behaviour of the associative memory system depends upon the specific request being served by the memory. The control process for a request to retrieve a set of records with a given key in a file has been described in detail (Section 3.3 and Figure 3.3). It should be emphasised that the control process for the request described in the thesis is just an example of one of a number of possible control processes. However, this example is sufficient to demonstrate the general activities of the modified associative memory.

In order to explore the capabilities of the modified memory device we have just described in outline, the procedures for its use in a 'hierarchical search' for records possessing a specified combination of keys (Section 3.7), in an application requiring sequential access to all the records of a file (Section 4.1), and in operation on a file with a more complex tree structure (Section 4.3), have been described in detail. With the introduction of the marking mechanism, the search for records satisfying some basic query conditions can be performed directly on the key part of records without the intermediate step of transmitting records into the main computer memory, which is always required with all conventional devices. By comparing its performance to the conventional counterpart (Section 4.5),

significant improvements in access times can be demonstracted.

A chain is a simple but important technique of linking logically related records; related records are linked by a pointer that contains a reference to the next related record in logical sequence so that all related records are chained together by successive pointers. As described in Section 4.1, an element of the chain is a structure and at least one field in each record is a pointer value that is used to point to another record. Because a record in the associative memory is accessed by one of its logical keys rather than by address, the pointer value can be a logical key entity, which is usually generated from the key of the record it points to by a simple and reversible process (for example, see Figure 4.3). Such an organization has a number of advantages: (a) a chain is in fact a two-way chain, (b) each record in the chain can be retrieved by following the chain key, or directly by the key of the record if it is known, and (c) it can avoid the tangle of actual physical addresses in the chain structure.

The storage organization for more complex data structures such as tree structures (Section 4.3) presents another unique feature of the associative memory. For example, Figures 4.7 and 4.8 demonstrate two distinct representations frequently used for tree structure data. In Figure 4.7 indexes to the subordinate records are kept with each parent record, whereas in Figure 4,8, each subordinate record stores an index to its parent record. Both data structures take the same amount of storage space.

An associative memory has always been an attractive idea because (a) data is accessed by content rather than by address, and (b) it has potential application in data bases for information query systems. However, so far only a few designs for an associative memory have been published. Usually, each design is just a simple modification to an existing disk device in order to meet requirements for a specific application (for example, Minsky, 1972). In this thesis, a design for a general purpose associative memory using sophisticated logic chips has been proposed. Readers may question the reality and cost justification of such a design. It has been about 15 years since the electronic industry made its first miniature electronic circuit on a silicon 'chip'. Since then a steady advance in circuit organization and complexity has led to the microprocessor, a device whose logic and memory circuits can be held on one's thumb. According to the survey of current market and electronic industries made by D. J. Theis (1974):

"Microprocessors bring us one step closer to having a whole computer on a single chip of silicon. No larger than ½-inch square, they contain all the essential elements of a central processor, including the control logic, instruction decoding, and arithmatic processing circuitry. To be useful, the microprocessor chip or chips are combined with memory and I/O integrated circuit chips to form a 'microcomputer', a machine almost as powerful as a minicomputer which usually

fills no more than a single printed circuit board and sells for less than $1,000." (Theis, page 90).

In conclusion of his survey, he stated further that:

"In the future, one chip will include the APU, memory, and I/O interfaces, so we will truly have a computer on a chip ... The prefix 'micro' denotes small size and connotes small cost; however, it certainly does not imply small capabilities", (Theis, page 100).

Therefore, with the modern computer hardware technology, the associative memory that is described throughout this thesis not only will become realistic (although at some cost), but also will have its impact on the development of another computer generation.

REFERENCES

Cardenas, A.F. (1975), Analysis and Performance of Inverted Data Base Structures, Communications of the ACM, May 1975, Volumn 18, Number 5, 253-263.

Minsky, N. (1972), Rotating Storage Devices as Partially Associative Memories, Proc. AFIPS 1972 Fall Joint Computer Conference, Volumn 41, Part I, AFIPS Press, Montvale, N.J., 587-592.

Parker, J.L. (1970), A Logic Per Track Information Retrieval System, Ph.D. Thesis, University of Illinois, Department of Computer Science.

Parker, J.L. (1971), A Logic Per Track Retrieval System, Information Processing 71, North-Holland Publishing Company, 711-716.

Slotnick, D.L. (1970), Logic Per Track Devices, Advances in Computers, Academic Press, 291-296.

Theis, D.J. (1974), Microprocessor and Microcomputer Survey, Datamation, December 74, 90-101.