# The File Searching, Record Validating and Record Formatting Functions of the Supervisor for an Extended Data Management Facility

Agu R. Ets
*University of Pennsylvania*

# The File Searching, Record Validating and Record Formatting Functions of the Supervisor for an Extended Data Management Facility

## Abstract

The purpose of the Supervisor in an Extended Data Management Facility (EDMF) is to direct the Facility's handling of a user's request for service. The Supervisor employs the five main functions of Access Controlling, Retrieval Optimizing, File Searching, Record Validating and Record Formatting in order to accomplish its task. This report is concerned mainly with the design and implementation of the File Searching and Record Validating Functions, although it also covers the Record Formatting Function. The File Searching and Record Validating Functions form that part of the Supervisor which actually controls the retrieval of records from the files of the EDMF. The major part of the report is concerned with discussing the File Searching Function because of the novel feature which has been implemented. This feature is the parallel processing of record lists in a generalized file structure, which eliminates redundant retrievals while at the same time reducing the access time of the device on which the records are stored. The Record Validating Function checks the record for compliance with the user's request and verifies the user's authority to access the record. A validated record is then subject to the Record Formatting Function which outputs it to the user.

## Comments

University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-71-04.

University of Pennsylvania
THE MOORE SCHOOL OF ELECTRICAL ENGINEERING
Philadelphia, Pennsylvania 19104

TECHNICAL REPORT

# THE FILE SEARCHING, RECORD VALIDATING AND RECORD FORMATTING FUNCTIONS OF THE SUPERVISOR FOR AN EXTENDED DATA MANAGEMENT FACILITY

by

Agu Raymond Ets

August 1970

Submitted to the
Office of Naval Research
Information Systems Branch
Arlington, Virginia

under
Contract N00014-67-A-0216-0014
Research Project NR 049-153

Moore School Report No. 71-04

# THE FILE SEARCHING, RECORD VALIDATING AND RECORD FORMATTING FUNCTIONS OF THE SUPERVISOR FOR AN EXTENDED DATA MANAGEMENT FACILITY

## Abstract

The purpose of the Supervisor in an Extended Data Management Facility (EDMF) is to direct the Facility's handling of a user's request for service. The Supervisor employs the five main functions of Access Controlling, Retrieval Optimizing, File Searching, Record Validating and Record Formatting in order to accomplish its task. This report is concerned mainly with the design and implementation of the File Searching and Record Validating Functions, although it also covers the Record Formatting Function. The File Searching and Record Validating Functions form that part of the Supervisor which actually controls the retrieval of records from the files of the EDMF. The major part of the report is concerned with discussing the File Searching Function because of the novel feature which has been implemented. This feature is the parallel processing of record lists in a generalized file structure, which eliminates redundant retrievals while at the same time reducing the access time of the device on which the records are stored. The Record Validating Function checks the record for compliance with the user's request and verifies the user's authority to access the record. A validated record is then subject to the Record Formatting Function which outputs it to the user.

# TABLE OF CONTENTS

TABLE OF CONTENTS (continued)

# CHAPTER 1

# INTRODUCTION

Today, there is a rapid and ever increasing growth in the total volume of information. This huge volume threatens to make the information useless unless ways can be found to manage it. The purpose of the Extended Data Management Facility (EDMF) is to provide a flexible, general purpose, time-shared file management system for the orderly accumulation and dissemination of information [9].

## 1.1 The Extended Data Management Facility

In order to achieve its purpose, the EDMF must be relatively simple to use, so as to encourage its use by individuals for data management. To facilitate this data management, information is organized as records. A user wishing to access records in the EDMF has merely to express, as a logical expression, those contents which characterize the required records. This eliminates considerable work on the part of the user to determine these required records, and to get their actual addresses. In addition to the retrieval of data, the EDMF also makes efficient use of storage space for the user. The heart of the Facility is the implementation of the generalized file structure and its general retrieval algorithm as suggested by Hsiao and Harary in [8]. For an overall description of the EDMF, the reader is referred to [9].

## 1.2 The Supervisor of the EDMF

The purpose of the Supervisor in the EDMF is to direct the Facility's handling of a user's request for service. In this capacity, the Supervisor assumes the roles of 'doorman', 'foreman', 'administrator', and 'dispatcher'. It is at first as a 'doorman' who accepts the service requests and initiates their request handling routines. Then as a 'foreman', the Supervisor regulates the use of the primitive storage and retrieval routines [6] and system subroutines, and also optimizes the storage and retrieval strategy for a time-sharing environment. In its role as an 'administrator', the Supervisor controls the user's access to files and validates the systems output of records to the user. It is also a 'dispatcher' who returns the results of the service to the user.

The Supervisor employs the five main functions of Access Controlling, Retrieval Optimizing, File Searching, Record Validating, and Record Formatting in order to accomplish its task. These functions in combination with each other satisfy the above roles which the Supervisor must assume.

## 1.3 The Scope of the Thesis

This **report** is concerned mainly with the design and implementation of the File Searching and Record Validating Functions of the Supervisor. These functions partially fulfill those aspects of the 'foreman' and 'administrator' which are concerned with the retrieval and validation of individual records. The thesis also covers briefly the Record Formatting Function which fulfills that aspect of the 'dispatcher' which is concerned with outputting the individual records to the user.

The File Searching and Record Validating Functions form that part of the Supervisor which actually controls the retrieval of records from the files of the Facility. The major part of the **report** is concerned with discussing the File Searching Function because of the novel feature which has been implemented as part of this function. This feature is the parallel processing of record lists in a generalized file structure. The parallel processing retrieval algorithm ensures that the records are retrieved in a manner which reduces the access time of the peripheral device on which the records are stored on the one hand, and eliminates redundant record retrievals (i.e., meaning high precision) on the other hand. Once it is retrieved by primitive storage and retrieval routines, the record is subject to the Record Validating Function. This function checks the record for compliance with the user's request and verifies the user's authority to access the record. When a record has been validated it is then subject to Record Formatting Function which prepares the record for output to the user. Since the Supervisor is a set of system programs operating in a time sharing environment, the implementation employs re-entrant coding and shared code conventions which allow the use of the same Supervisor for the processing of multiple requests.

# CHAPTER 2

# THE FILE SEARCH FUNCTION

## 2.1 Definitions

Before a discussion of the File Searching Function can be undertaken, the terms and concepts which are basic to the EDMF must be given precise definitions. The definitions used in this report are consistent with those in [8]. However, they will be found to be less formal and more descriptive.

### 2.1.1 Attribute-Value Pair

The most basic concept which must be defined is that of the attribute-value pair. Let there be two sets, A and V. The elements of A are those terms which are considered as "attributes", and the element of V are those terms which are considered as "values". Let a third set D be the subset of the Cartesian product A x V, whose elements are the ordered pairs of the elements of A and V. A single element of D is called an attribute-value pair, and intuitively it constitutes the basic element of information. Some examples of attributes, values, and attribute-value pairs are shown in Example 1.

### 2.1.2 Record

A record R is a set of attribute-value pairs which collectively convey some meaningful information. An example of R, a subset of the set of all attribute-value pairs, is shown in Example 2. The attribute-value pairs in this record convey to the reader information about a paper which appeared in the Communications of the ACM.

1a: A set A of attributes.

A = {author, year, topic, abstract, text}

1b: A set V of values.

V = {Hsiao, 1970, information retrieval, [the complete abstract of a paper], [the complete text of a paper]}

1c: A set D of ordered pairs which are attribute-value pairs.

$A \times V \supset D$ = {(author, Hsiao), (year, 1970), (topic, information retrieval), (abstract, [the complete abstract of a paper]), (text, [the complete text of a paper])}

Example 1: Examples of attributes and values and attribute-value pairs

R = {(record number, H001),

(author, D. K. Hsiao),

(author, F. Harary),

(title, A Formal System for Information Retrieval from Files),

(topic, information retrieval),

(publisher, Association for Computing Machinery),

(year, 1970),

(month, February),

(abstract, [the complete abstract of the paper]),

(text, [the complete text of the paper])}

Example 2: Record of a paper published in the Communications of the ACM

### 2.1.3 Keywords

A record can be characterized by any combination of the attribute-value pairs which are in the record. Due to pragmatic considerations, it would be desirable to have those attribute-value pairs which are short and can be simply expressed, to characterize the record. These short attribute-value pairs are called <u>keywords</u>, and will henceforth be denoted symbolically by $K_i$, $i = 1,2,\ldots n$. Thus we can refer to a record R by referring only to the keywords in R. The record in Example 2 can be characterized by the set of keywords shown in Example 3. In general, the set of keywords of a record R is called an <u>index</u> of the record R and it is usually a proper subset of R.

The index of R = {(record number, H001),
(author, D. K. Hsiao),
(author, F. Harary),
(topic, information retrieval),
(year, 1970),
(month, February)}

Example 3: The keywords characterizing the record in Example 2

At this point we would like to introduce a notational change for the attribute-value pair. Hereafter an attribute-value pair will be written in the following manner:

Attribute = Value

This is the actual notation used in the EDMF for specifying attribute-value pairs.

2.1.4 Keyword Lists

Each record is also characterized by another parameter which is not part of the actual information contained in the record. This unique number is the address of a record, which indicates the whereabouts of the record in the computer storage.

Each keyword $K_i$ in R may have associated with it the address of another record R' which also contains the keyword $K_i$. Effectively this address in R "points" to R' and for this reason it is called the pointer of R with respect to $K_i$ or the $\underline{K_i\text{-pointer}}$ of R. If the record R' is non-existent then the $K_i$ pointer of R is known as the <u>null</u> <u>pointer</u>. It will be assumed hereafter that every keyword has a pointer associated with it. Thus we see that records containing a common keyword $K_i$ can be linked by these pointers into a chain which is called a $\underline{K_i\text{-list}}$. Putting it more precisely, a $K_i$-list is a chain of records, each record containing the keyword $K_i$, satisfying the following five conditions:

1) Each of the pointers in the $K_i$-list is distinct.
2) Each non-null pointer is the address of a record in the $K_i$-list only.
3) There is <u>one</u> record not pointed to by any other record in the $K_i$-list. This is the <u>beginning</u> of the $K_i$-list.
4) There is <u>one</u> record which has the null pointer; this is the <u>end</u> of the $K_i$-list.
5) For every record in the $K_i$-list at the address $a_n$ $(n > 1)$, there is a sequence of $K_i$-pointers

$$(a_1, a_2, \ldots, a_n)$$

such that:

i) $a_1$ is the address of the beginning of the $K_i$-list.

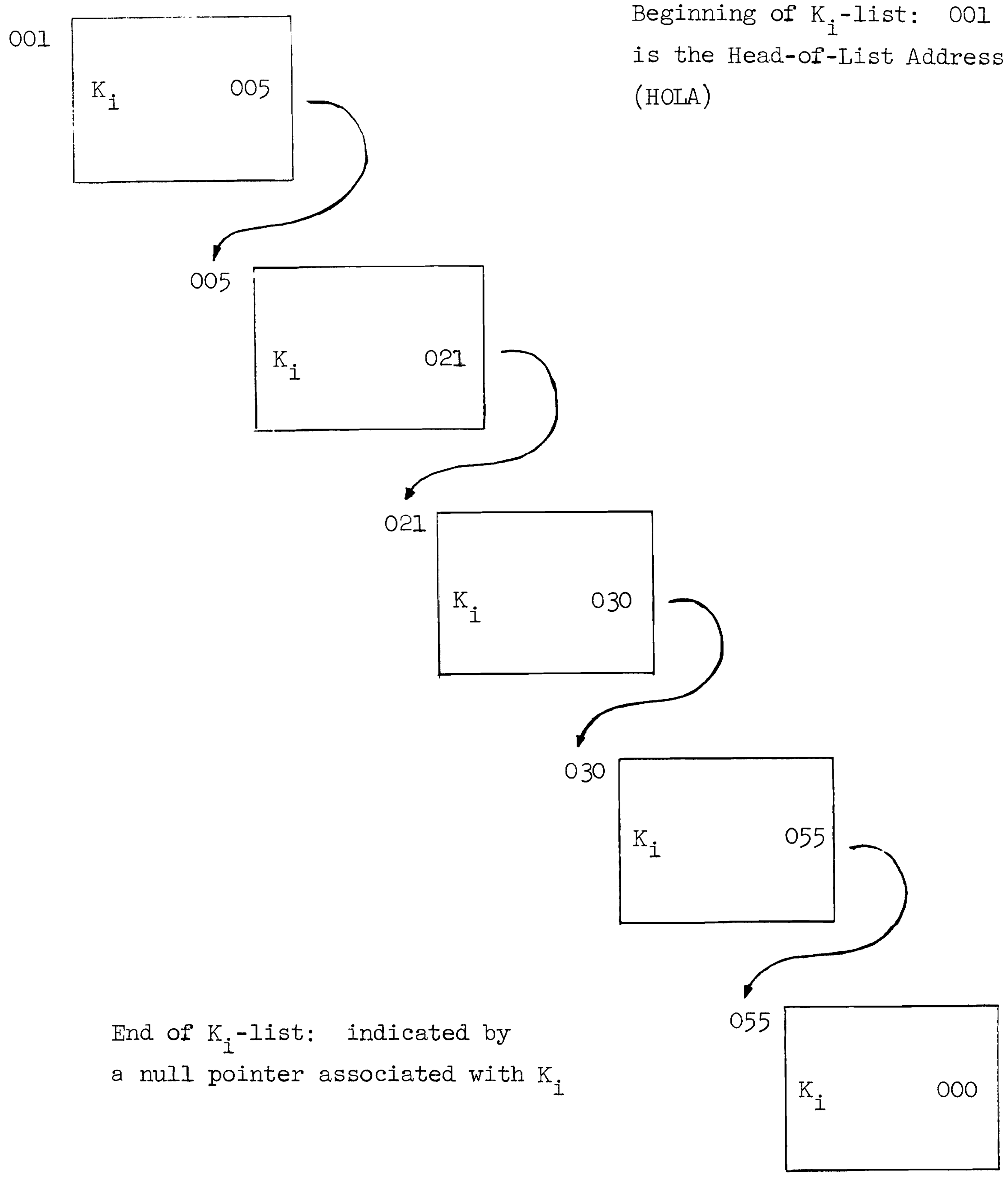
ii) the record at the address $a_j$ contains a $K_i$-pointer $a_{j+1}$ for $j = 1, 2, \ldots, n-1$.

This means that for a given $K_i$, a record cannot be in more than one $K_i$-list. The address of the first record in a $K_i$-list is known as a Head-of-List Address or HOLA for short, and this term will be used hereafter when referring to the beginning address of any $K_i$-list. In Figure 1, a typical $K_i$-list is illustrated, showing the beginning and the end of the list and the pointers which chain the records together.

2.1.5 File and Directory

A file is a set of records which completely contains all the $K_i$-lists made up of those records. In other words, a file is a set, whose elements are records, which is the union of all the $K_i$-lists which contain the records. The HOLA's of all the $K_i$-lists in a given file must be carefully noted and kept separate from the HOLA's of the $K_i$-lists in another file because the same keyword, but with different meanings, can occur in both these files.

This leads us to the concept of a directory for a file. The directory associated with a file contains the HOLA's of all the $K_i$-lists in that file. For each keyword $K_i$ used in the file, there is one entry in the directory, the form of the entry being shown in Example 4. More precisely, a directory for a file is a sequence of m such entries where m is the number of different keywords used in the file.



Beginning of $K_i$-list: 001 is the Head-of-List Address (HOLA)

End of $K_i$-list: indicated by a null pointer associated with $K_i$

Figure 1: An Illustration of a $K_i$-list

$(K_i, n_i, h_i; a_{i1}, a_{i2}, \ldots, a_{ih_i})$

| | | |
|---|---|---|
| $K_i$ | - | the ith keyword in the file F. |
| $n_i$ | - | the number of records in F containing the keyword $K_i$. |
| $h_i$ | - | the number of $K_i$-lists in F. |
| $a_{ij}$ | - | the HOLA of the jth $K_i$-list in F. |

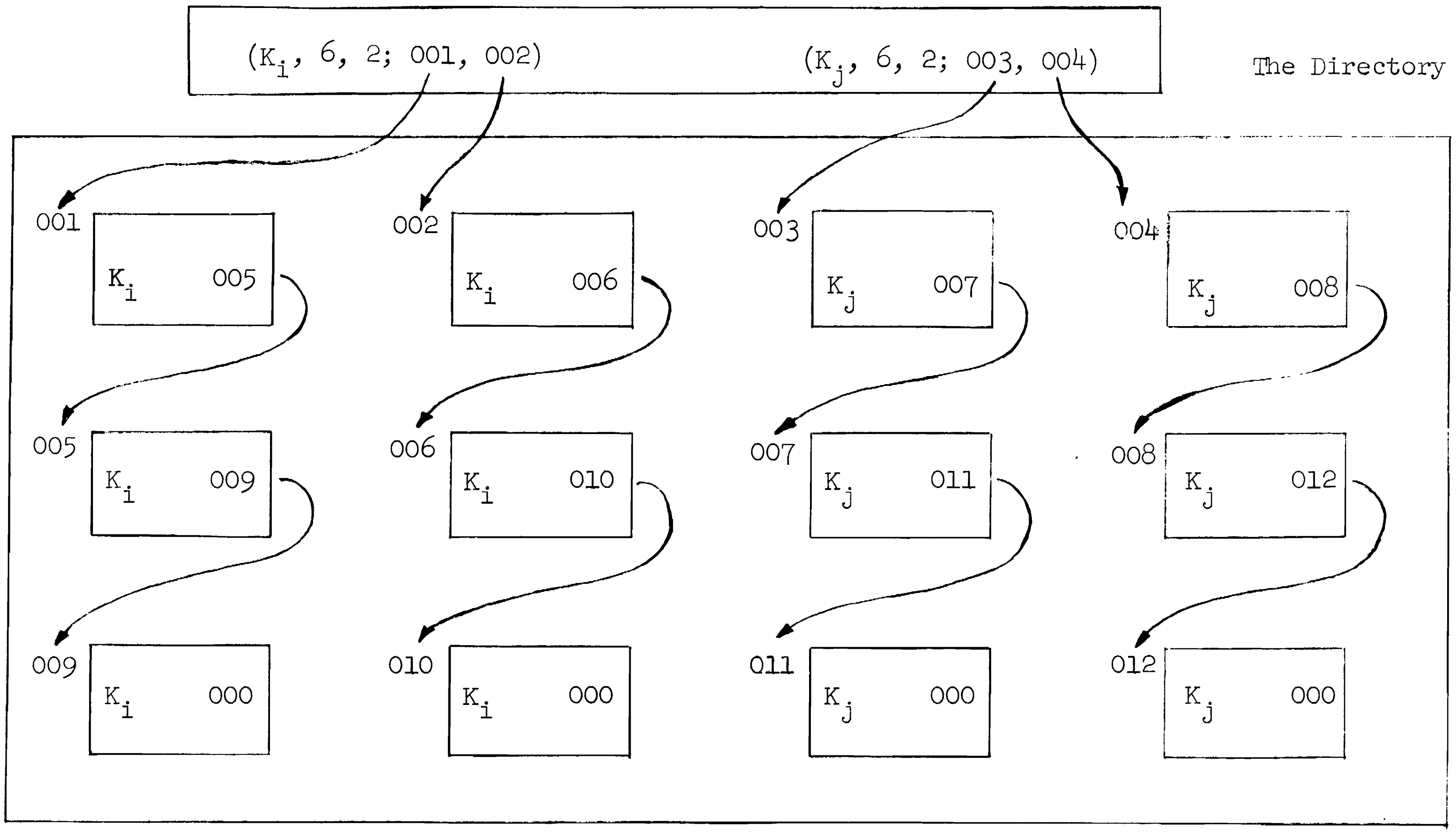
Example 4: Format of a directory entry

## 2.1.6 Generalized File Structure

We can now define a <u>generalized file structure</u> as a file with its directory. This file structure is called generalized because it can be shown that many commonly used file structures such as inverted, index-sequential, and multilist are actually special cases of the generalized file structure [8]. An example of a generalized file structure is shown in Figure 2.

As was evident in the directory format, there may be more than one list corresponding to a particular keyword $K_i$, but these lists are mutually exclusive because of the definition for lists presented previously. In other words, a record containing the keyword $K_i$, cannot be in two different $K_i$-lists.

However, since a record may have more than one keyword, it may be in more than one keyword list. A record containing the keywords $K_i$ and $K_j$ (with $i \neq j$), is a member of <u>one</u> $K_i$-list and <u>one</u> $K_j$-list simultaneously. For example, if a record contains both the keywords AUTHOR = KRAMER and YEAR = 1964, then that record would be in both an AUTHOR = KRAMER list and in a YEAR = 1964 list. This is illustrated in Figure 3, where the



The File

Figure 2: Example of a Generalized File Structure Showing the Logical Relationship Between the Directory Entries and the Keyword Lists

AUTHOR = KRAMER list consists of records located at the addresses 050, 101, 121, and 165, and the YEAR = 1964 list consists of records located at the addresses 060, 101, 112.

### 2.1.7 Request Description

When a person accesses a file, rarely does he want to see all of the records in the file. Rather, he usually wants to see only that part of the file which interests him, not caring about the rest of the file. Such a partition can be accomplished by listing the addresses of the records which he wants. This, however, is cumbersome and requires much research on the user's part to find the addresses of the records he is interested in. Another way to partition the file would be to describe the records of interest by listing their characterizing keywords in the form of a Boolean expression. This expression is called a user's request description. Using the propositional calculus, any Boolean expression can be uniquely written as a disjunct of conjuncts, known as the Disjunctive Normal Form (DNF). All request descriptions used hereafter will be assumed to be in Disjunctive Normal Form.

A record satisfies a user's request description when all the keywords in a single conjunct are in the record. A record containing only the keywords $K_1$ and $K_3$ satisfies the conjunct: $(K_1 \wedge K_3)$, but does not satisfy the conjunct: $(K_1 \wedge K_2 \wedge K_3)$. The problem of finding, in a file, the addresses of records which satisfy a user's request description now lies with the EDMF and not the user.

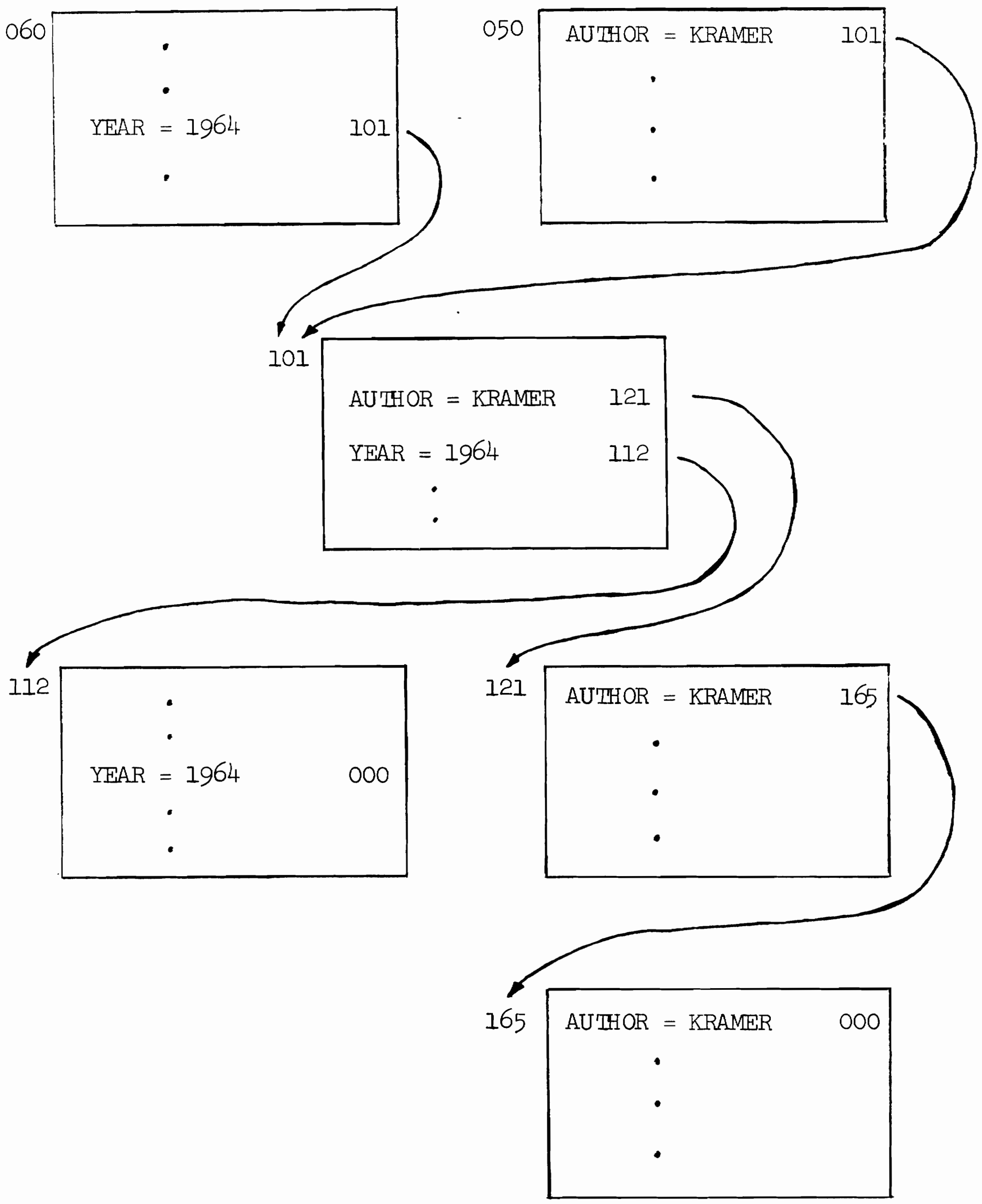


Figure 3: Example of Record Being in a $K_i$-list and a $K_j$-list simultaneously. In this case we have for $K_i$ - YEAR = 1964 and for $K_j$ - AUTHOR = KRAMER.

### 2.1.8 $K_i$-list Process

To find records containing a certain keyword $K_i$, two steps must be executed. First the HOLA's corresponding to $K_i$ must be found so they can be used for retrieval. Given $K_i$, the directory is searched for the entry corresponding to this keyword. The production of the HOLA's of the $K_i$-lists is called decoding the keyword. Second, all the records in the $K_i$-lists must be retrieved. The $K_i$-list process involves repeatedly, a given address or pointer of a record in a $K_i$-list, the retrieval of the record and the extraction of the $K_i$-pointer from that record. The occurrence of a null pointer signifies that the process has been completed. At the end of the process, all the records in that $K_i$-list are retrieved.

## 2.2 Retrieval Algorithm Used in the File Search Function

The retrieval algorithm is based on the application of the $K_i$-list process to the lists of the generalized file structure. There are two novel features of this algorithm which deserve elaboration. The first one involves the selection of the shortest lists for retrieval and the second involves the parallel processing of the shortest lists selected.

### 2.2.1 The Selection of the Shortest Lists for Retrieval

The key to the first feature lies in the fact that although a record may be a member of many different lists, in looking for a record containing some particular conjunct of keywords, it would search the lists corresponding to that keyword which appears in the least number of records. Recalling the format of the directory entry in Example 4, this means that the lists for the keyword with the smallest $n_i$ are searched. The keyword is called the prime keyword and its

$K_i$-lists result in the smallest partition of records from the file which could possibly satisfy the user's request description. Thus the prime keyword lists are the "shortest". This minimizes the file searching effort because, for any conjunct, all the records being sought must be included in the prime keyword lists.

For an example of the prime keyword idea, assume that we would like to see all the records which satisfy the following description:

( AUTHOR = KRAMER $\wedge$ YEAR = 1964 $\wedge$ TOPIC = LOGIC )

Let us assign the following numbers of records to each keyword:

5 records contain the keyword AUTHOR = KRAMER

15 records contain the keyword YEAR = 1964

8 records contain the keyword TOPIC = LOGIC

We see that the keyword AUTHOR = KRAMER is contained in the least number of records, so we select this as our prime keyword. This means that _at most_ 5 records can satisfy our description so we want to retrieve only those records for analysis, instead of using either one of the other two keyword lists for retrieval. Of course before any list processing can begin, the prime keywords must be decoded in order to get their HOLA's.

### 2.2.2 The Parallel Processing of the Shortest Lists

In a DNF request description, each conjunct will have a prime keyword which is to be searched on. Since each prime keyword has at least one associated list and each DNF description has at least one conjunct, we now have to process multiple lists in order to find all the records satisfying the request description.

Here a question arises as to exactly how we will process these multiple lists. One way is to take the lists in order and process each one to the end before going on to the next list. This is a serial list processing method because, the lists are processed in a serial (sequential) fashion. This method is very inefficient for a moving head random access device because it requires the head mechanism to reset itself after all the records in a single list are retrieved. Furthermore, a record which has been retrieved as a member of one $K_i$-list may be retrieved again as a member of a different $K_i$-list.

An example of this kind of retrieval redundancy can be seen in the serial processing of the AUTHOR = KRAMER and YEAR = 1964 lists illustrated in Figure 3. First, the processing of the AUTHOR = KRAMER list results in retrieval of the records at the addresses 050, 101, 121, and 165. Then the processing of the YEAR = 1964 list results in retrieval of the records at the addresses 060, 101, and 112. The record at the address 101 was retrieved twice, the first time to extract the AUTHOR = KRAMER pointer and second time to extract the YEAR = 1964 pointer.

To eliminate the redundant retrieval of records and to minimize the file searching effort, the notion of parallel processing of lists is introduced. The more efficient parallel processing of lists first arranges the addresses of the records monotonically. This results in a more ordered motion for the moving head of a random access device and hence a shorter seek time. To do this we may have to retrieve a record from one list and the next record from another list. Thus all the lists are being processed 'simultaneously' and resulting in parallel list processing. Since a record can be in many different lists, we must guard against having to retrieve a given record more than once, something which can happen in serial list processing.

In serial list processing, the redundant retrieval of a record occurs when more than one prime keyword pointer must be extracted from the record, as shown in the preceding example. This means that when the record is retrieved, it must be processed in such a way that all of the prime keywords are searched for and, the corresponding pointers, if any, are extracted. Once all the prime keyword pointers have been extracted, there is no reason to retrieve that particular record again.

This is effectively checking a record for membership in many different lists, although it was retrieved as a member of one particular list. As we shall see later, under our implementation it does not matter which list's processing resulted in the retrieval of the record, just as long as all of the pointers for all the lists of which the record is a member are extracted. This extraction of all the pointers ensures the continuity of the prime keyword lists and assures that all the records which could possibly satisfy the user's request description are retrieved and checked.

In summary, the File Searching Function is that part of the Supervisor which actually controls the retrieval of the records from the files of the EDMF. It is the implementation of the parallel processing of the $K_i$-lists, which is a part of the general retrieval algorithm used for retrieving records in the EDMF. The selection of the prime keywords for the user's request description and the decoding of these prime keywords is managed by the Retrieval Optimization Function of the Supervisor as described in [4].

## 2.3 The Implementation

### 2.3.1 Steps of the File Searching Function

The work of the File Searching Function can be broken down into three steps. In the first step it selects an address which is to be used to retrieve the next record. In the second step it directs the primitive storage and retrieval routines [6] to retrieve the record at the address selected in the previous step. In the third step it searches the record just retrieved for pointers associated with the prime keywords. The selection of the addresses in the first step is done monotonically to allow parallel processing. Steps 2 and 3 together are responsible for the parallel processing of the $K_i$-lists as it was described in Sect. 2.2.2. The following sections will further elaborate these three steps.

#### Selection of a Record Address

When all the prime keywords have been decoded, there are many HOLA's available with which to begin the list processing. To maximize the efficiency of moving the read-write head of a random access device, we would want to select these addresses in a monotonic order.

To decide in which order - ascending or descending - the addresses should be selected, we must take a look at the generalized file structure which is implemented in the EDMF. We note that the lists are arranged in such a way that the pointers are in descending order. That is, the record at the head of the list has a numerically larger address than any other record in the list. In general it is true for the files of the EDMF that:

$K_i$-pointer of R < Address of R

This ordering results from a new record containing $K_i$ being added to the head of a $K_i$-list. The address of the new record is higher than the addresses of the other records in the list because the new record is located further from the beginning of the file space than the previous records. Therefore, the addresses are selected in a monotonically descending order. This means that at any given time, the highest address not yet used to retrieve a record, is selected for the retrieval of the next record.

## Record Retrieval

The beginning of the list processing is done by the second step of the File Searching Function. The address selected by the previous step is now given to a storage and retrieval subroutine which will locate the record at the given address and move the record from its secondary storage location into core memory. This is considered a retrieval by the Supervisor only and not by the user, because the record has not been made available to the user. When the record becomes available to the user, then the record is considered retrieved by the user.

## Extraction of Pointers

The extraction of pointers from a retrieved record is an important step in the File Searching Function. This step ensures that the lists are being processed continuously until their terminators (the null pointers) are reached. This processing of all the lists constitutes the file search.

Since a record is retrieved only if it is a member of at least one of the prime keyword lists, the record contains at least one prime keyword but it may contain more than one, making it a member of more than one list. This record can be the beginning of one prime keyword list while at the same time being imbedded in another prime keyword list. Thus the pointers corresponding to all the prime keywords which are in the record must be extracted.

To extract these pointers, the record is checked for the presence of a prime keyword $K_i$. If $K_i$ is found in the record, the $K_i$-pointer is extracted and added to those addresses which have not yet been used for retrieval. Regardless of whether or not $K_i$ was found in the record, the record is subsequently checked for the presence of the next prime keyword of the user's request description. This process is repeated for all the prime keywords, resulting in the extraction of all relevant pointers.

Extracting all of the prime keyword pointers after the first retrieval of a record eliminates the necessity to retrieve that record again. Although a record is retrieved as a member of one prime keyword list, the same record need not be retrieved again as a member of another prime keyword list because the pointer corresponding to the other prime keyword is also extracted. As records in general contain multiple keywords and user's request descriptions consist of many conjuncts, the saving of retrieval time and effort due to the elimination of redundant retrievals is considerable.

## 2.3.2 Routines of the File Searching Function

The implementation of the File Searching Function involves two routines and the primitive storage and retrieval subroutines as described in [6]. Step 1 is completely implemented in the routine RETALG. Step 2 is also implemented in RETALG, but it requires the services of the primitive storage and retrieval subroutines. Step 3 is implemented in the routine RCDCHK, which also serves an important role in the Record Validating Function.

### 2.3.2.1 Address Selection Mechanism

This mechanism, which implements the first step of the File Searching Function, is responsible for selecting the address of the next record to be retrieved. It also notes the addresses which have been used for retrieval, so that the same record is not retrieved again.

#### ISAM Keys

The EDMF utilizes the Indexed Sequential Access Method (ISAM) for device level input/output operation on the RCA Spectra 70/46, the computer system on which the EDMF has been implemented. In ISAM, the number assigned to a record as an address is five bytes long and is in packed decimal format. This address is called a key or more precisely an ISAM key, and these terms will be used hereafter when referring to a record's address. Note that the term "key" or "ISAM key" is used to denote a record address in this implementation and it should not be confused with the term "keyword" which is an attribute-value pair in a record.

#### KEYQUE Format

In order to avoid repeated retrievals of a given record, the processing of the lists is done in a unified manner. That is, all the keys which are obtained as HOLA's and all the keys which are extracted as pointers, are stored in one single area. Duplicate keys are eliminated and unique keys are then marked as they are used for retrieval.

Since this area is a queue of ISAM keys, it is given the name KEYQUE.

At the head of KEYQUE, there is a five byte area named SERKEY. This is a temporary storage location for the key which is being considered for addition to KEYQUE or the key which is the highest unused key during the selection process. SERKEY is followed by a series of entries for the ISAM keys which are generated and extracted during the list processing. Each six byte entry consists of one byte for control information and five bytes for storing an ISAM key. KEYQUE is terminated by an End-of-File (EOF) mark in the control byte following the last entry. The format of KEYQUE is illustrated in Figure 4 below.

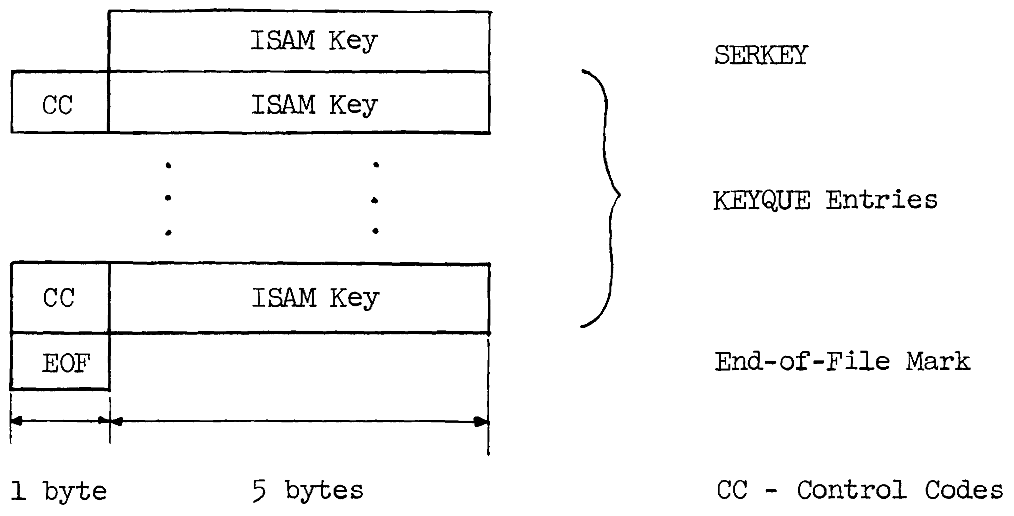


Figure 4: Format of KEYQUE Showing the Individual Entries

The control byte contains a code which indicates the current status of the accompanying ISAM key. A key in KEYQUE can be in one of the five states listed below:

1. The key is available for retrieval (i.e. it has not been used yet).
2. The key has been used for retrieval.
3. The key has been used for the retrieval of a record which was validated (i.e. given to the user).
4. The key has been deleted from KEYQUE.
5. The retrieval on this key resulted in an error condition.

The actual hexadecimal codes for these states can be found in Appendix A.

## Adding Keys to KEYQUE

KEYQUE is a dynamically growing area because keys are constantly being added to it as the processing of the lists continues. KEYQUE must contain all the keys which have been retrieved on and all the keys which are available for retrieval. In the adding of keys, the HOLA's which are obtained from the directory and the pointers which are extracted from the records are treated the same way. In fact, the key addition process is not aware of the source of the keys to be added.

Keys to be added are in a temporary queue. This queue is pointed to by an eight byte area which also contains the length of this temporary queue. If there is more than one temporary queue, these pointers are listed sequentially with an EOF marker. In Figure 5, the format of the temporary queues and their respective pointers is shown.

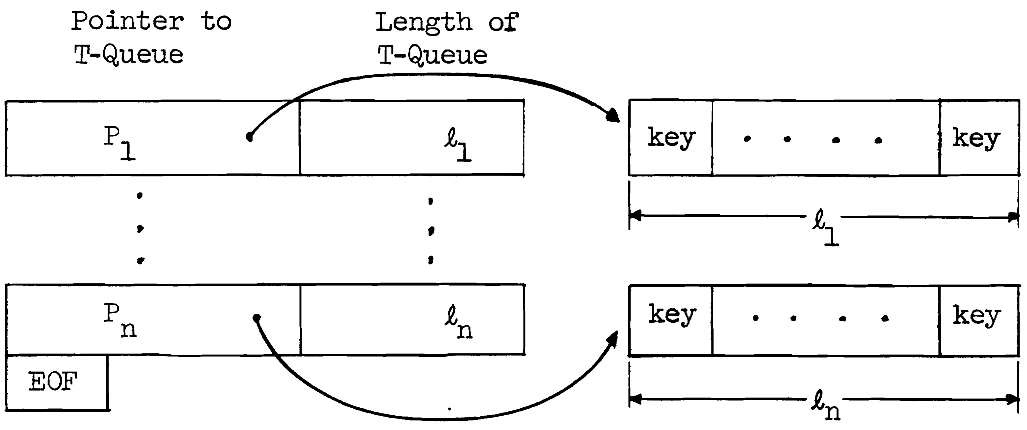


Figure 5: Temporary Queues (T-Queues) for Holding Keys to Be Added to KEYQUE

The key addition process first handles the pointer to the temporary queue and checks whether there are any keys in it by examining the length parameter. A non-zero length indicates there are keys to be added, and it goes to the temporary queue and begins the addition process.

The first key is taken from the temporary queue and put into SERKEY. It is then compared with each previous member of KEYQUE, including those keys which have already been used for retrieval. If the key in SERKEY is found in KEYQUE, the process returns to the temporary queue to find the next key to be inserted. In this way, the keys which are already in KEYQUE are not inserted again. Keys which are not duplicates are added to the end of KEYQUE, followed by the relocation of the EOF marker.

Null pointers, which are put into the records to indicate the end of a keyword list, are not put into KEYQUE because they do not point to anything. If a null pointer is found in the temporary queue, it is ignored, and the next key is picked up.

At the beginning of list processing, KEYQUE consists of only SERKEY and the EOF marker. The HOLA's are then taken by the addition process from the temporary queues and put into KEYQUE. Every HOLA put into KEYQUE is unique, the duplicate ones being eliminated by the addition process.

## Selecting Keys From KEYQUE

The largest unused key must be found in KEYQUE in order to determine which record is to be retrieved next. The selection process goes to the beginning of KEYQUE, and scans the control bytes sequentially until it finds a key which has not been used. This key is then put into SERKEY and the location of this key in KEYQUE is also recorded. The search continues from this point to find the next available key. The next available key in KEYQUE is then compared with the key in SERKEY. If SERKEY is larger than the key in KEYQUE, the search continues. If the keys are equal, this means that a duplicate has found its way into KEYQUE. The duplicate key in KEYQUE is deleted by changing its control byte, making that area available for the addition of a key. The search now continues also. If, however, SERKEY is less than the key in KEYQUE, the larger key replaces the key in SERKEY. When the end of KEYQUE is reached, SERKEY contains the largest key which has not been used yet. The control byte of the key in SERKEY is changed to indicate that the key has been used for retrieval.

The File Searching Function is terminated for a given description when all the keys in KEYQUE have been used to retrieve records. This occurs when SERKEY does not contain any key after a search of KEYQUE.

### 2.3.2.2 Record Retrieval Mechanism

The record corresponding to the key selected from KEYQUE must now be retrieved to realize the second step of the File Searching Function. The actual retrieval is done by the primitive storage and retrieval routine RETRREC [6] which is called by the routine RETALG.

Before RETRREC can be initiated, a parameter list must be prepared, indicating the key of the record to be retrieved and the core address at which the retrieved record is to be put. The parameter list also contains specific system information which is also needed by RETRREC. When the parameter list has been prepared, it is passed on to RETRREC to initiate its processing.

The routine RETRREC returns control to RETALG after it has put the record specified by RETALG in core. RETRREC also returns a code to indicate if the retrieval was successful, or if it resulted in an error condition. If the return code indicates that the record is too large for the core area specified, RETALG will allocate enough space to accommodate the record and will reinitiate the retrieval.

When the record is in core and control returns to RETALG, the record is considered retrieved by the Supervisor, although at this point, the user does not know that the record exists. The record in core is now readily accessible by RETALG which will now initiate the third step of the File Searching Function.

### 2.3.2.3 Pointer Extraction Mechanism

The pointers associated with the prime keywords in a record are extracted by the routine RCDCHK, which is called RETALG to realize the third step of the File Searching Function. As in the implementation of step 2, RETALG must first prepare a parameter list before making a call for RCDCHK. This parameter list contains the address of the record to be checked and the address of the prime keywords for which the checking is to be performed.

Format of the Record

The record brought into core by the retrieval mechanism is in an internal format of the EDMF. This format allows for the removal of the actual attributes from the record and for keeping the attributes in an area called the Record Format Block (RFB). Each of the attributes in the RFB is coded uniquely using a format number. That is, no two attributes have the same format number. The values in the record are associated with their proper attributes by means of these format numbers. By removing the attributes from all the records in a file, this method of storing records can reduce the physical storage size of the records and consequently reduce the physical size of the file as well [9].

The internal format of the record consists of two parts. First there is the Record Control Block (RCB) which contains the format numbers of all the attributes used in the record. These numbers may not all be different if there is more than one occurrence of a particular attribute in the record. Then there is the text of the record which contains the actual values and the keyword pointers associated with them.

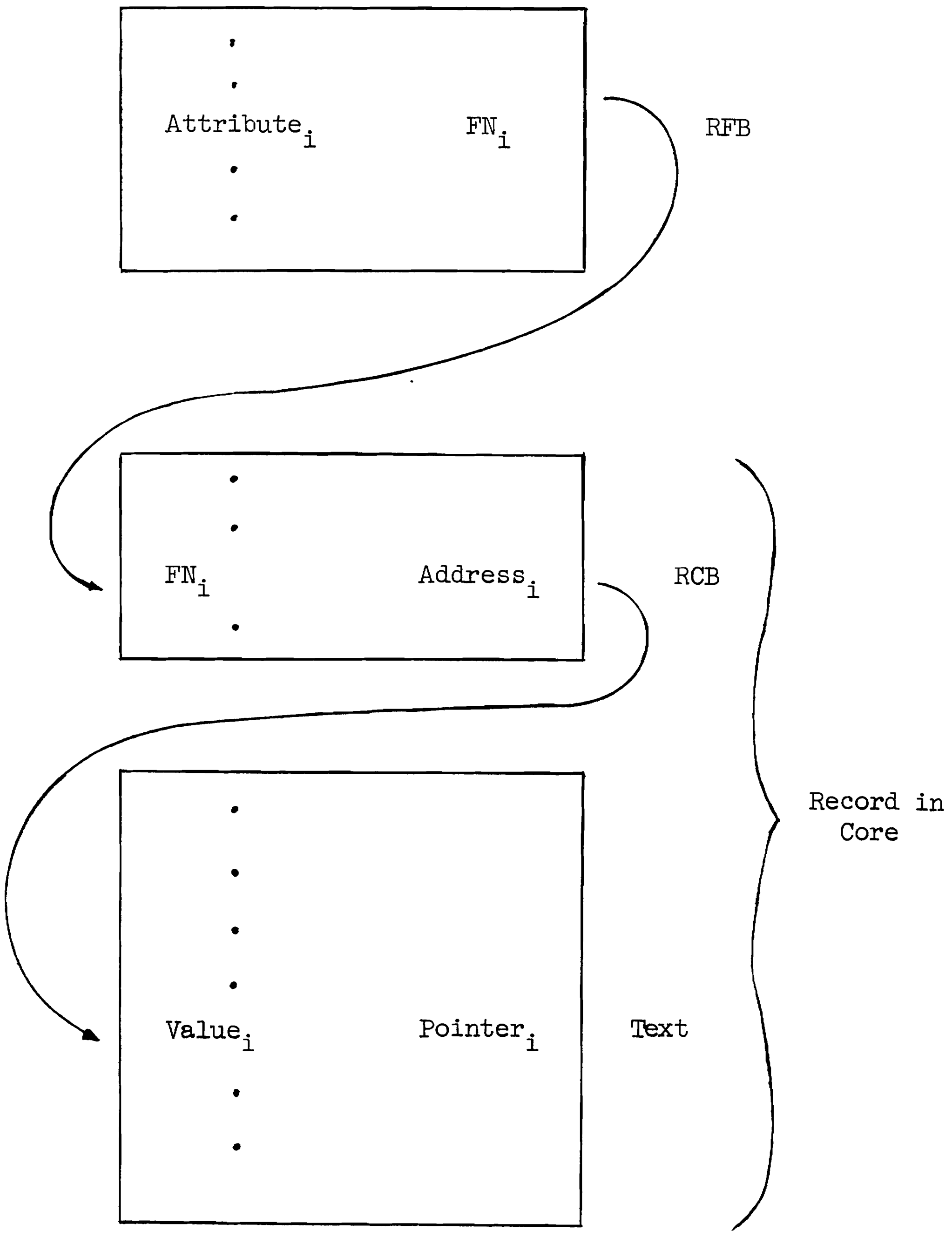
In the RCB, associated with each format number, there is an address which points to the value in the text which corresponds to the attribute represented by the format number. The relationships between these various areas can be seen in Figure 6, and the reader is referred to Appendix B for the detailed specifications of the RFB and the internal form of the record.

Prime Keyword Search

The prime keywords from a user's request description are put into a form which is compatible with the internal format of the record. In the record, a keyword actually exists as a format number and an associated value so the actual attributes of the prime keywords are also put into format number form. This is done by the Retrieval Optimizing Function [4] of the Supervisor after the prime keywords have been found.

The prime keywords are in a stack whose beginning address is given to RCDCHK in the parameter list. The search begins when a prime keyword is taken from the stack. The format number of this keyword is then compared sequentially with all the format numbers in the RCB. If the format number of the prime keyword is not present in the RCB, then that prime keyword is not in the record and the next prime keyword is taken from the stack.

However, if the format number of the prime keyword is present in the RCB, then the attribute half of that keyword is present in the record. Now the value of the prime keyword is compared with the corresponding value in the record text. If the values match, then the prime keyword is present in the record, and the pointer must be extracted. If the values do not match, then the prime keyword is not present, and the format number search of the RCB resumes.



FN - format number

Figure 6: The major parts of the internal format of a record in the EDMF. Figure shows how the keyword: Attribute$_i$ = Value$_i$ is stored.

This process is repeated for every prime keyword in the stack, so that all relevant pointers which are in the record can be extracted.

## Extraction of the Pointers

As each prime keyword is found in the record, the corresponding pointer is taken and put into a temporary queue. When all the prime keywords have been checked for, then all relevant pointers have been extracted. An area pointing to and containing the length of the temporary queue is now set up so that the pointers can be added to KEYQUE. This completes the third step of the File Searching Function and now RCDCHK returns control to RETALG and passes along the address of the temporary queue.

# CHAPTER 3

# THE RECORD VALIDATING FUNCTION

## 3.0 Introduction

In any data management facility, the security and integrity of the records are as important as the ease with which the records may be retrieved. A good system is one in which the security precautions are enough to ensure reasonable record integrity while at the same time not unduly encumbering the retrieval mechanism. The assurance of the integrity of the files encourages users to add records to the data management facility, thus enhancing the dynamic growth of the data base. Easy retrieval will encourage frequent use of this data base, leading to an orderly, yet efficient way for information dissemination. In general, these security arrangements will require that each record be "validated" or cleared by some authority before it is given to the user. This is the job of the Record Validating Function.

The "validity" of a record or its clearance to be outputted is a function of an agreement made by two parties. One of these parties is the user who wishes to see records satisfying a certain description. The other party is the owner of the file who would like to control the access of other users to his records. Therefore, a record is valid if it is one which the user would like to see and if that same record is one which the file owner will allow the user to access. Only when the wishes of the owner and user coincide (intersection, in the language of set algebra), does a record get outputted from the system. In this way, the integrity and security of the file are upheld.

## 3.1 The Implementation

### 3.1.1 Steps of the Record Validating Function

The Record Validating Function consists of two steps which determine if a record is valid and can be given to the user who requests it. One of these steps is checking the record to see if it satisfies the user's request description. This step is called record checking. The other step is called security checking and is used to determine if the user is allowed to read or write the record.

The record checking is done first. It will examine a retrieved record and compare it with the request description given by the user. A record which is satisfactory from the user's point of view will then be subject to the security check by the Supervisor. This means that only those records which satisfy the user's request description will have to be security checked. In other words, every record which is eventually given to the user must be security checked.

The security checking done by the EDMF is at the software level only. Security problems related to hardware and electronics, and the personnel in the computer center are subject to more broad measures which are not included in this study [10].

### 3.1.2 The Authority Item

In order to validate a user's authorization to access a file, the EDMF must somehow obtain the authority information concerning the user's access to that particular file. This information can be stored in a record associated with the file listing all the users who are allowed to access the file and their extents of access. This way of organizing and using the authority information is called a file oriented security information system. Another way would be to assign to each user a special record, which would be kept in a system file. This record contains the names of all the files to which the user is allowed access (and possibly some to which he is denied access) and the extent of access allowed in each file. The record is known as the Authority Item (AI), because it contains the user's authorization to access the files in the EDMF. This way of organizing and using the authority information is termed as user oriented file security.

The security in the EDMF is user oriented and the latter type of storing authority information is used. When there is to be a change in the user's authorization, the updating is simple because all of the user's authority information is kept in one place. The Authority Items are set up in the same internal format as records in the rest of the EDMF so that the Supervisor can use the regular retrieval routines to retrieve a particular Authority Item from the system file. By keeping the authority information in a system file on a user oriented basis, the security of the authorization is better protected because the system file can be accessed only by the Supervisor. This makes it more difficult to tamper (i.e. change illegally) with the Authority Items. Since all the Authority Items are in the exact same format, the job of the security checking routine becomes easier because it is working with a single format. Furthermore, the EDMF does the actual setting up and modifying of the AI's, because the file owner cannot access the system file. The file owner merely specifies to the system what authorization he wants to give to any user. After checking the validity of the file owner, the EDMF then alters the specified AI's.

This also insures that the format of the AI's remains standard. A detailed specification of the Authority Item is included in Appendix C.

### 3.1.3 Record Checking

The record checking step is done to ensure that the retrieved records are those which satisfy the user's request description. Since the request description is in DNF, the checking consists of examining the record for the presence of <u>all</u> the keywords specified in a conjunct of the request description. A record containing all of the keywords is said to satisfy the description as defined in Sect. 2.1.7.

<u>Use of Prime Keywords</u>

The routine RCDCHK, which was used for finding the prime keywords and extracting the pointers, is also used for checking the record, because both checking processes overlap and are therefore incorporated into one routine. The finding of prime keywords in the records helps expedite the record checking by virtue of the fact that each conjunct has only one prime keyword. Only when the prime keyword of a conjunct is found (and the pointer is extracted) is the record checked for the rest of the keywords in that conjunct. This way, a lot of useless checking is eliminated because conjuncts are not checked for unless there is a positive indication (such as finding the conjunct's prime keyword in the record) that the record may satisfy the conjunct. To further reduce unwarranted checking of the record, there is a provision in RCDCHK such that once the record has satisfied a conjunct in the user's request description it is considered a satisfactory record and no further conjunct checking is done. The record, however, continues to be checked for the rest of the prime keywords so that the corresponding pointers may be extracted, as necessitated by the File Searching Function.

Checking for the Conjuncts

The records being checked reside in core and have the internal format which was described in Sect. 2.3.2.3. This means that the user's request description has to have format numbers for attributes just as the prime keywords did. In actuality, the whole request description is translated into an internal form by the Query Language Assembler [3], and is then put in a single location. Keywords which become prime keywords are appropriately marked. Format numbers are added to every keyword in the description, prime or not, by the Retrieval Optimizing Function of the Supervisor [4]. The actual checking for the presence of a given keyword is identical to the checking for a prime keyword, this process having been described in Sect. 2.3.2.3.

When a prime keyword is found, this means that one of the keywords in a conjunct is in the record. Now the other keywords must be checked for.

The routine goes to the beginning of the conjunct and takes the first keyword (unless it is the prime keyword, in which case the next keyword is taken). The record is then checked for the presence of this keyword. If it is present, the record has partially satisfied the conjunct up to this point in the checking process and is called a partially satisfactory record. Then the next keyword in the conjunct is taken (if it is the prime keyword, it is skipped) and the record is checked for the presence of that keyword. If it is present, the record continues to be partially satisfactory. The next keyword in the conjunct is taken and the checking is repeated. If there are no more keywords in the conjunct, then the record has become satisfactory. All the keywords specified in the conjunct are in the record, and the record does not have to be checked for any more conjuncts. If, however, during the checking of the record for the keywords in a conjunct, a keyword does not appear in the record, the record is unsatisfactory and has failed the test. In this case the routine goes back to check for the prime keywords.

When all the prime keywords have been checked for, the record is deemed either satisfactory or unsatisfactory. A satisfactory record goes on to the second step of the Record Validating Function, while the unsatisfactory one is erased from core, having served its purpose in the File Searching Function. This keyword checking routine assures that the retrieved records satisfy the user's request description by preventing superfluous records from being sent to the output routine. It also keeps the lists intact by extracting all the relevant pointers from the records retrieved. This allows an efficient search and retrieval mechanism to be implemented but not at the expense of the convenience to the user. The first step of the Record Validating Function, that of assuring the user that he will receive only those records which he asks for, is thus completed.

### 3.1.4 Security Checking

In the EDMF, all the security checking is controlled by the routine AUTHCHK. The protection mechanism implemented in AUTHCHK can operate at three levels corresponding to the logical levels of the file structure in the EDMF. These are the file level, the record level and the field level (a field being an attribute-value pair in the record).

In general, a file level check is concerned with the security of the file as a whole, and therefore will control any access whatsoever to the file. If a user has no authorization to access a certain file, the processing of his request is immediately terminated. The file level check will also involve setting up the parameters which will be needed in record and field level checks if the user is allowed into the file. The record level check is concerned with the security of individual records in the file. The check done at this level will ascertain if the user is allowed access to the record which has been retrieved and deemed satisfactory by RCDCHK. If the user is allowed to access the record, it is output to him via the device he has selected. If on the other hand, the user is not allowed to access the record, then the record is discarded, and the File Searching Function resumes its processing. The field level check is concerned with certain parts of the record and whether or not the user is allowed to see them. If the user is not allowed to see certain parts of the record, these parts are deleted before the record is given to the user. A detailed description of AUTHCHK can be found in Appendix A.

Of course, any file owner can incorporate security measures above and beyond those offered by the EDMF, but that is not required for basic file security. In this case the file owner would probably have, in some part of the file, a list of users whom he will allow to access his file along with a program to check these lists for the name of a user wishing to access the file.

## The File Level Check

The file level security check (or a user's authorization check) is necessary to control any access to a file and to set up the parameters which will be needed for the subsequent record and field level checks. This makes the file level security check the most involved and complex procedure of the three checking level procedures. The user's Authority Item is the standard from which all file level controls (record, and field level also, for that matter) are derived for a given user.

At the file level, a user's AI is examined to determine if he has any access to the file he requests records from. If he does have access, a further examination of the AI will determine the exact extent of this access. Specifically, two major entries are examined at the file level. One of these tells whether a user is the owner or a sharer of the file and the type of access control which applies to him. In the EDMF, a file owner will have complete and unrestricted access to the file, making further checking unnecessary. A sharer's access will be checked at each level to determine the extent of the access which he is allowed.

Another item in the file level check is determining whether the user has used in his request description any keywords which he is not allowed to retrieve on. This is to protect the file's integrity at the field level, for if the user is forbidden to see certain fields in a record he must also be prevented from retrieving on these keywords.

The importance of this check can be illustrated in the following example. Suppose a person is doing a medical survey and is allowed to access the medical records of the patients in a hospital, but with the patient's personal data deleted. The records would be output without any identifying data such as name, address, social security number, etc. Without the restriction on request descriptions, he could ask for a retrieval using a patient's name as the keyword. Receiving the record, he finds the patient's name deleted. However, by virtue of the fact that the record was retrieved in response to a name keyword, the user now knows the contents of the deleted name field. Thus, a field which was to be left unknown to this user, has had its security compromised. If the user were not allowed to request records using a keyword not to be seen by him, his request would be rejected should it ever enter the EDMF. This preserves the security of fields not to be seen by a user. Such an access control is called directory protection because it will not even allow the directory to process the request.

An important, but optional, part of the file level check is to see whether the owner has also implemented some kind of access control to his file. If there is such a control program, it must be satisfied by the user (using passwords, etc.) before any further processing is done by the EDMF. If the file owner's control program rejects the user's request for access, then the EDMF will also reject the user and block any of his attempts to access the file. The file owner can also specify that only his access control is to be used to protect his file. In this case, there is no further checking by the security checking routine, and the EDMF relies solely on the owner's control program for security checking.

Part of the file level security check also involves setting up parameters for subsequent checks at lower levels. If it is determined that the user is a sharer of the file, further checks are made to ascertain what limitations have to be imposed at the record and field level. If there is a description which will limit him to records in one part of the file, then this description (which is in the same form as a description issued by a user) must be changed into an internal form which can be used by the record checking routine RCDCHK. The file level security check first calls for the Query Language Assembler [3] to translate the limiting description into internal form. It then calls a routine of the Retrieval Optimizing Function of the Supervisor [4] which make the internal form usable for a record level check. The address of the file limiting description is then stored so it will be available for use during a record level check.

The file level security check returns to the Supervisor with a code indicating whether or not the Supervisor should proceed in handling the user's request. It also returns the address of the area where the description limiting the user's access at record level is stored. Thus the file level check has completed its job of checking a user's access to the file and setting up the parameters for subsequent checks on lower levels.

## The Record Level Check

The record level check is used to verify if retrieved records belong to that part of the file to which the user is allowed access. At this level it checks first whether or not the user is the owner of the file. If he is, then no more checking is necessary, and it is assumed that the user is allowed access to any record in the file.

Checking at this level occurs for each record which is retrieved on the user's request description after it has been determined that the record is satisfactory. Then the record is examined to see if it included in the limiting description which was set up in internal form by the file level checking routine. The routine RCDCHK is used to determine if a record satisfies the limiting description because the checking procedure for this step is the same as it was for the record checking step. After determining whether or not the record is in that part of the file which is partitioned by the limiting description, the checking routine will see whether the description was an inclusive or exclusive expression. An inclusive expression describes that part of the file which the user is allowed access to, while the exclusive expression describes that part of the file which the user is _not allowed_ to access. Therefore a record satisfying an exclusive expression is not allowed to be seen by the user and is not validated. If that same expression were an inclusive type, and a record satisfied it, then that record would be validated and allowed to be output to the user.

The record level check is set up in such a way that RCDCHK is also utilized for checking if the record satisfied the AI's limiting description. This allows for an efficient use of system resources, and for standardization of the various checking procedures for both the user's convenience and the owner's privacy.

## The Field Level Check

After a record has been validated (released for output) it is formatted and made ready for outputting. However, before it is actually given to the user, the field level check must be applied to determine if the record output format has to be altered to conform to the user's authorization.

The field level check is based on the assumption that the record has been retrieved using keywords other than those forbidden to the user for request descriptions. The field level check then examines the user's AI to see which fields have to be deleted before the record is given to the user. When the fields which must be deleted are known, the output format is scanned for the presence of these fields. If they are present, the value or the whole keyword may be deleted, depending on the circumstances surrounding the restriction. This is done for all the forbidden fields until only that part of the record is left which the user is authorized to see.

The field level check completes the security measures taken to preserve the file owner's privacy. The capability of being able to check security on these three levels, allows the file owner a wide range of options so that he can selectively protect the integrity of his file. These options encourage a greater use of the EDMF because the owners know they can protect the privacy of their files while allowing users of different authorization levels to access selected parts of the file and records.

# CHAPTER 4

# THE RECORD FORMATTING FUNCTION

## 4.0 Introduction

Although a record has been retrieved by the File Searching Function and has been validated by the Record Validating Function, it is not of any use to the user. The record is still in the internal format described in Sect. 2.3.2.3, which makes it meaningless to the user. The Record Formatting Function transforms this record into a format which the user can interpret and understand. There are two types of formats available for giving the record to the user. First, there is the output format which is used if the record is to be printed out for the user on some outputting device. Second, there is the core format which is used if the user wants to process the record in core memory.

## 4.1 Record Output Format

This is the first type of format available for giving records to the user. It is used for actually outputting the records to the user, and for this reason it is oriented toward human interpretation.

There are two major types of devices which would be used to output the records to the user. These would be a high speed printer or a low speed terminal, either a teletypewriter or a video console. Because of the greater speed and capacity of the printer, the format sent to it will be slightly different than the format sent to the low speed devices.

The basic output format is to have a maximum of one attribute-value pair on a physical line of the output device. If the size of this attribute-value exceeds one line, as many additional lines are used as are needed. Each attribute-value pair is preceded by a line number which also denotes the ordering of that attribute-value pair in the record.

Each attribute-value pair consists of the complete attribute name, followed by an equal sign which is then followed by the complete value. As an example we may have the following entry printed out:

AUTHOR = KRAMER, J. A.

Here the complete attribute name is "AUTHOR", and the complete value contained in a certain record is "KRAMER, J. A." The rest of the values in the record are printed out in a similar manner.

### 4.1.1 Record Number

At the beginning of each record which is printed out, there is a record number. This number is used when referring to that record for the purposes of updating or modifying it in an interactive mode. The record number allows the updating routines to get the corresponding internal form of the record so that the necessary changes can be made. In the EDMF, a coded form of the record's ISAM key is used as the record number.

### 4.1.2 Line Numbers

Line numbers are put in front of each attribute-value pair to indicate the sequential order of the attribute-value pair in the record retrieved. The purpose of line numbers is to aid the user in picking out and referring to certain parts of the record which he wishes to update or otherwise modify in an interactive mode. The line numbers are assigned in multiples of one hundred, thereby allowing the user to insert additional attribute-value pairs between already existing ones by assigning the corresponding line numbers to new attribute-value pairs.

As an example we show that a part of the output for a record was retrieved:

```
000900      AUTHOR = KRAMER, J. A.
001000      TOPIC = LOGIC
001100      YEAR = 1964
```

The user can change any of these attribute-value pairs by referring to their line numbers and inputting the new value. An insertion of an additional attribute-value pair can be accomplished by assigning this pair a line number which is numerically between the line numbers of the attribute-value pairs which are to surround the new attribute-value pair. A change and insertion to the part of record output illustrated above would take the following form:

```
000900      AUTHOR = CRAMER, K. B. (change existing line)
001050      MONTH = MAY            (insert a new line)
```

With the above changes, the part of the record illustrated would have the following output format on subsequent retrievals.

```
00900       AUTHOR = CRAMER, K. B.
001000      TOPIC = LOGIC
001100      MONTH = MAY
001200      YEAR = 1964
```

It must be noted here that although the attribute-value pair MONTH = MAY was inserted with a line number of 001050, its line number became 001100 upon retrieval. There is no discrepancy here. The number 1050 indicated that this attribute-value pair was to be inserted between the attribute-value pairs with line numbers 1000 and 1100 in the old record. Since the line numbers indicate the sequential order of an attribute-value pair at the time of retrieval a subsequent retrieval finds that the attribute-value pair MONTH = MAY is now the 11th item in the record and is assigned the line number 001100, with the line numbers of other attribute-value pairs adjusted accordingly.

## 4.1.3 Output Formatting Mechanism

The mechanism for transforming the record into output format is implemented in a routine called RCDFRMT. This routine is called by RETALG to initiate the Record Formatting Function for the output type of format. The routine RCDFRMT has two parts which transform a validated into printed output for the user.

The first part of RCDFRMT takes the record in the internal form and the Record Format Block (RFB), and collates the attributes in the RFB with the proper values from the internal form of the record. This is done by taking the first value in record and decoding the control information to find out which attribute in the RFB is to be associated with that value. When the proper attribute is found in the RFB, it is moved to an assembly area. The attribute is followed by an equal sign and the value from the record. The attribute and value have now been collated in the assembly area and are in the form which is printed out. This attribute-value pair is separated from the next one in the assembly area by a delimiter which immediately follows the value. Then, the next value in the record is accessed and the process is repeated until all the values have been associated with their proper attributes in the assembly area. This part of the formatting is the same, regardless of the output device.

The second part of RCDFRMT consists of moving the attribute-value pairs from the assembly area into the device output buffer along with the appropriate line numbers. The physical line images for the output devices are formed by this part of the mechanism. In this part, the current line number is moved into the device buffer. This is immediately followed by the next attribute-value pair from the assembly area. If the attribute-value pair exceeds the line image, it is truncated and the remainder is put into the next line image without a line number. When the record has been put into the buffer of an output device, it is said to be in <u>output format</u>. When the device outputs the contents of its buffer, the record is given to the user. In the case of a low speed terminal, this occurs after each line image has been formatted, resulting in a record being output line-by-line. In the case of the printer, the outputting occurs when the entire retrieval is completed, so that all the records are outputted all at once.

## 4.2 Record Core Format

This is the second type of format available for giving records to the user. It is called <u>core format</u> because the record is not outputted but actually remains in core memory so that it can be processed by a user or a system program. For this reason, the format is oriented toward programmable manipulation.

This format is basically very simple, consisting of a <u>header</u> which is followed by a series of <u>attribute-value entries</u>. The basic layout is illustrated in Figure 7.

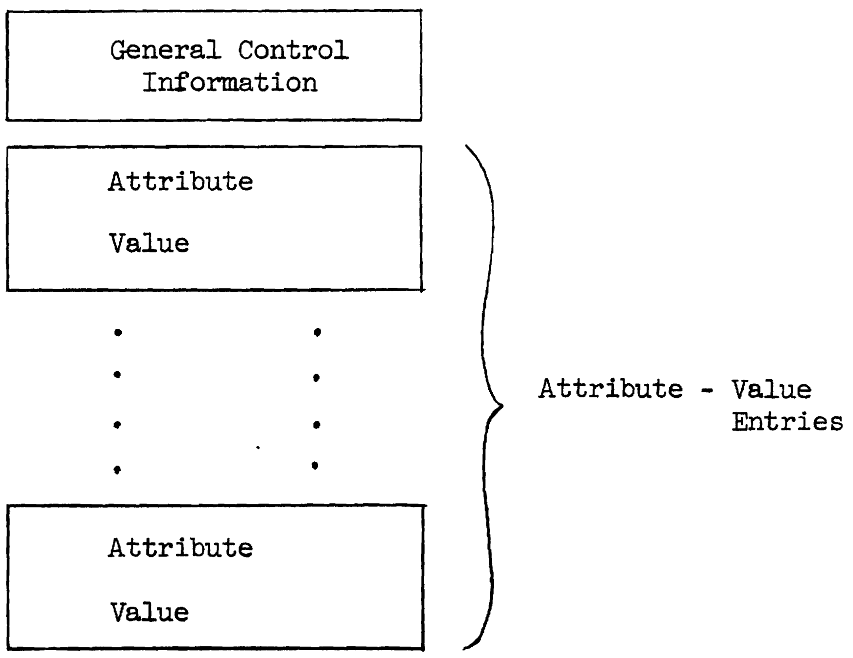


Figure 7: Layout of the Core Format for a Record

Each attribute-value entry contains the complete attribute name and the complete value associated with the attribute. The lengths of both the attribute and the value are also included in the entry as well as other control information to make the core format machine oriented.

The core format is assembled by the routine COREFMT, and its processing is somewhat similar to RCDFRMT. The routine COREFMT takes each value in the internal form of the record and then finds the associated attribute in the RFB. Then both the attribute and value are moved into the core formatting area. The length parameters are added along with the control information and the attribute-value entry is completed. This process is repeated until an attribute-value entry has been formed for each field in the record.

The record, now in core format, can be put into a location specified by either the user or the Supervisor. If the record is in a user specified area, the Supervisor notifies the user that the record is released to him. If the record is in an area specified by the Supervisor, the user is given the address of that area as notification that the record is ready.

The completion of a user's request service occurs when the desired records are given to the user in an intelligible form. This form can be either the output format or the core format, both of which were discussed above.

# CHAPTER 5

# SUMMARY AND CONCLUSION

The Extended Data Management Facility (EDMF) was implemented to provide a general purpose data management system for the orderly dissemination of information. The EDMF utilizes a generalized file structure and retrieval algorithm for efficient data management. The user can cause the Facility to retrieve records by describing the contents of these records as a Boolean expression of keywords. The files in the EDMF are protected by elaborate security measures which allow the individual file owner to impose a wide variety of access controls on the users of his file.

The five functions of the Supervisor are employed to direct the Facility's handling of a user's request and to enforce the access control on this request for the file owner. The Access Controlling and Retrieval Optimizing functions which partially fulfill the roles of 'doorman', 'administrator', and 'foreman', are described in [4]. This thesis has described the File Searching, Record Validating, and Record Formatting functions of the Supervisor.

The File Searching Function fulfills that aspect of the 'foreman' which deals with the retrieval of individual records from the files of the EDMF. Implemented in the File Searching Function is a novel feature which allows the parallel processing of record lists in a generalized file structure. This feature eliminates the redundant retrieval of records thus reducing the file searching effort and also results in an orderly motion for the moving head of a random access device. Thus the parallel list processing capability of the File Searching Function assures an efficient utilization of system resources by the Supervisor.

The Record Validating Function fulfills that aspect of the 'administrator' which is concerned with the validation of individual records. This validation consists of two steps, the first of which checks if the record satisfies the user's request. The second step checks if the record is allowed to be accessed by the user. A record becomes valid only if it can pass both of these checks.

The Record Formatting Function fulfills the role of the 'dispatcher' and is concerned with giving the actual records to the user. This function first puts the record into one of two formats which are intelligible to the user and then outputs the record. When all the records which were requested have been given to the user, the processing of a user's request is completed.

These three functions constitute the basic components of the Supervisor. With additional time and effort, improvements in the coding of the routines and better use of the systems facilities can be made. However, the design of the implementation has been made flexible enough to allow for modifications and additions in the future. Some suggestions of improvements are listed below:

(1) In supervising the retrieval of records, the File Search Function must direct the primitive storage and retrieval routines to retrieve one record after another. Error conditions may exist between the retrieval of one record and another. It is important that error recovery routines supplied by the user or the systems programmer can be easily incorporated so that continuation of the retrieval process is assured. Currently, a very simple error recovery routine

is used. It is hoped that more elaborated recovery routines can be added to the File Search Function for diagnostic, debugging and monitoring purposes.

(2) The KEYQUE mechanism which selects prime keywords for retrieval of records can be improved. The current mechanism employed which is a sequential search of the keyqueue may result in considerable search time if the queue is long. Consideration of binary and hashing algorithms may be helpful.

(3) As part of the Record Validating Function, the routine RCDCHK presently compares the keyword in the user's request description with the keyword in the retrieved record to see if they match. However, the EDMF design has allowed for relations other than 'equal' to be used between the attribute and the value, namely, the relation 'inclusion'. This makes the comparing procedure more complex because RCDCHK now has to interpret the keyword in the request description and then decide whether the keyword in the record is part of those keywords covered by the relation. This requires a more extensive interpretation mechanism to be included in the RCDCHK.

(4) There is the need of implementing a mechanism by which a user can find out from his Authority Item the exact extent of his access to the files of the EDMF. If he finds that he needs more access to a certain file, he can contact the owner in order to ask for access privileges. Otherwise, he would use the EDMF only to get an error message that he is denied access. This feature would be very important to those users

whose accessibility to the EDMF is very limited.

(5) At present the Record Formatting Function has only two fixed formats which the user can select from. It is not clear to the author whether these two formats can be easily accepted by other language processors as input so that more elaborate output format can be generated. One such processor is the Report Generator. Another one is SNOBOL.

These suggestions may improve the working of the Supervisor and enhance the use of the EDMF - leading to a more orderly accumulation and dissemination of information.

## BIBLIOGRAPHY

1. Desiato, B., "Directory Constructing and Decoding in a Generalized File Structure," M.Sc. Thesis, The Moore School of Electrical Engineering, University of Pennsylvania, August 1970.

2. Dodd, G. G., "Elements of Data Management Systems," Computing Surveys, Vol. 1, No. 2, June 1969.

3. Gana, J., "A Command and Query Language Assembler for an Extended Data Management System," M.Sc. Thesis, The Moore School of Electrical Engineering, University of Pennsylvania, August 1970.

4. Hirsch, J., "The Access Control and Retrieval Optimization Functions of the Supervisor for an Extended Data Management Facility," M.Sc. Thesis, The Moore School of Electrical Engineering, University of Pennsylvania, August 1970.

5. Hoffman, L. J., "Computers and Privacy: A Survey," Computing Surveys, Vol. 1, No. 2, June 1969.

6. Horton, M., "Reading, Writing, Creating and Updating Records and Files in a Generalized File Structure," M.Sc. Thesis, The Moore School of Electrical Engineering, University of Pennsylvania, August 1970.

7. Hsiao, D. K., "A File System for a Problem Solving Facility," Ph.D. Dissertation, The Moore School of Electrical Engineering, University of Pennsylvania, May 1968.

8. Hsiao, D. K. and Harary, F., "A Formal System for Information Retrieval From Files," Communications of the ACM, Vol. 13, No. 2, February 1970.

9. Manola, F., "An Extended Data Management Facility for a General Purpose Time Sharing System," M.Sc. Thesis, The Moore School of Electrical Engineering, University of Pennsylvania, August 1970.

10. Petersen, H. E. and Turn, R., "Systems Implications of Information Privacy," Proc. of SJCC 1967, pp. 291-300.

11. Wexelblat, R., "The Development and Mechanization of a Problem Solving Facility," Ph.D. Dissertation, The Moore School of Electrical Engineering, University of Pennsylvania, December 1965.

# APPENDIX A

# ROUTINES

## A.1 Routine RETALG

The routine RETALG serves as the part of the Supervisor which directs the retrieval and validating of individual records. It is the most important part in the implementation of the File Searching Function.

### A.1.1 Entry Point

RETALG is entered always at the location named ESTAB.

### A.1.2 Exit Points

RETALG has two exit points. One is used where an Authority Item has been retrieved from the system files. This exit procedure is begun at the location SEXOK. This exit point is used only for the special Authority Item retrieval and is bypassed during normal functions.

The normal exit point begins at the location SPVEND and is used when record retrieval has been completed. Preceding this, starting at location NOKEY, is a procedure which prints informatory notices to the user.

### A.1.3 Input Parameter List

The input parameter list which is passed to the routine RETALG has the format specified below. The parameter list is assumed to begin on a fullword boundary.

| Bytes | Content |
|---|---|
| 0 - 3 | Address of File Control Block (FCB) |
| 4 - 7 | Address of Key Information Buffer (KIB) |
| 8 - 11 | Address of prime keyword stack |
| 12 - 15 | Address of RFB |

| Bytes | Content |
|---|---|
| 16 - 19 | Address of T-Queue pointers |
| 20 - 21 | Number of records requested |
| 22 | Code of service requested |
| 23 | Code for AUTHCHK level |
| 24 | Code for record output device |

A.1.4 Register Conventions

The registers in RETALG are assigned in the following manner:

| Register | Utilization |
|---|---|
| 0 | Not used. |
| 1 | Address of parameter list given to called subroutine. |
| 2 | Counter for valid records. |
| 3 | Base for RETALG |
| 4 | Address of T-Queue from RCDCHK. |
| 5 | Address of T-Queue entries. Temporary storage for setting up parameter lists. |
| 6 | Pointer to KEYQUE entries. |
| 7 | Address of SERKEY. |
| 8 | Address of T-Queue pointer stack. Address of ISAM key currently in SERKEY. |
| 9 | Address of the end of a T-Queue. Address of parameter list for RETRREC. |
| 10 | Length of T-Queue. |
| 11 | Address and base of input parameter list. |
| 12 | Address and base of RETALG work area. |
| 13 | Address of RETALG save area. |

| Register | Utilization |
|---|---|
| 14 | Return address in RETALG. |
| 15 | Subroutine call address. |

A.1.5 Internal Work Area

The internal work area used by RETALG also contains the parameter lists for some of the routines called by RETALG. It is used to store data from the input parameter list which must be passed on to other routines. The work area has the following format:

| Bytes | Content |
|---|---|
| 0 - 71 | Save area for RETALG. |
| 72 - 75 | Address of parameter for AUTHCHK. |
| 76 - 79 | Address of FCB. |
| 80 - 83 | Address in KEYQUE of most recently used key. |
| 84 - 87 | Address of T-Queue area of RCDCHK. |
| 88 - 91 | Size of area where retrieved record is put. |
| 92 | Code of requested service - RETCODE. |
| 93 | Code for record printout - PNTCODE. |
| 94 | Control byte to indicate no records retrieved - DXCODE. |
| 95 | Area for analysis of return codes from subroutines. |
| 96 - 103 | Area for WROUT parameter list. |
| 104 - 183 | Area for the line image of a V-form record. |
| 184 - 191 | Packed form of record count. |
| 192 - 199 | Unpacked form of record count. |
| 200 - 211 | Unassigned. |
| 212 - 227 | Parameter list for privileged REQM. |

| Bytes | Content |
|---|---|
| 228 - 243 | Parameter list for RETRREC. |
| 244 - 263 | Parameter list for RCDCHK. |
| 264 - 279 | Parameter list for RCDFRMT. |

## A.1.6 Internal Codes

There are many coded control bytes used in RETALG to serve as switches and store information. The bit positions of the eight bit bytes are indicated by the use of hexadecimal digits.

Mode Byte

| | |
|---|---|
| X'0A' | Routine in Primary mode |
| X'06' | Routine in Secondary mode |

KEYQUE Control Byte

| | |
|---|---|
| X'01' | ISAM key has been used for retrieval. |
| X'02' | ISAM key is available for retrieval. |
| X'03' | ISAM key resulted in valid record being retrieved. |
| X'04' | ISAM key has been deleted. |
| X'08' | End of KEYQUE marker. |
| X'10' | ISAM key resulted in non-retrievable record. |

RETCODE (Service request codes)

| | |
|---|---|
| X'21' | Retrieve single record. |
| X'22' | Retrieve records. |
| X'26' | Delete records. |
| X'2A' | Restore records. |
| X'2C' | Store records. |
| X'42' | Open file for reading. |

| | |
|---|---|
| X'43' | Open file for writing. |
| X'48' | Close file. |
| X'44' | Create a file. |
| X'46' | Delete a file. |
| X'1*' | All auxiliary function where * - 0, ..., F. |
| X'81' | Retrieval of Authority Item. |

PNTCODE (Print codes indicating output device)

| | |
|---|---|
| X'00' | Output on Low Speed Terminal (LST). |
| X'02' | Output on high speed printer. |

DXCODE (Records output indicator)

| | |
|---|---|
| X'00' | Records have been output. |
| X'02' | No records have been output. |

### A.1.7 Flowchart

The flowchart for RETALG is shown in Figure A.1. This figure consists of five parts, each part dealing with a separate aspect of RETALG's processing.

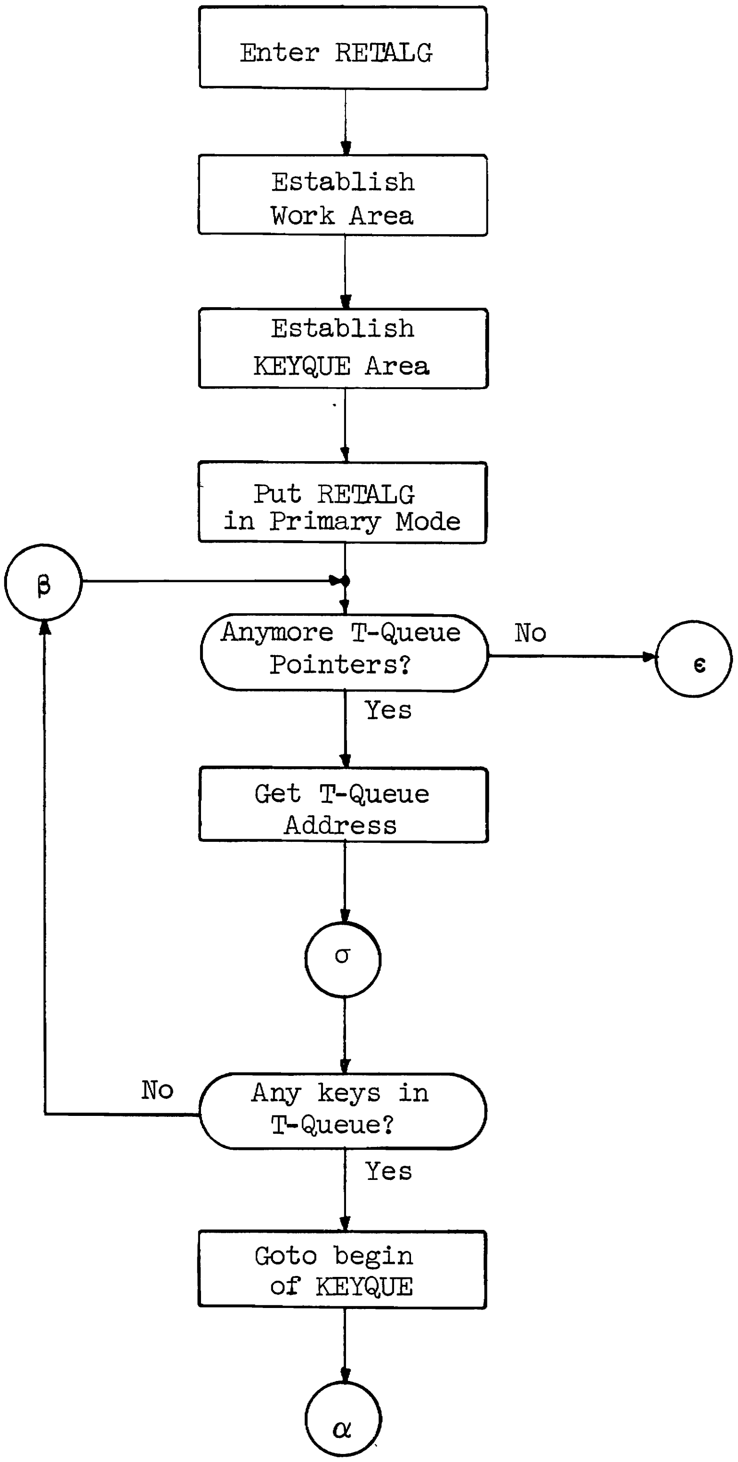


Figure A.1.a: RETALG Initialization

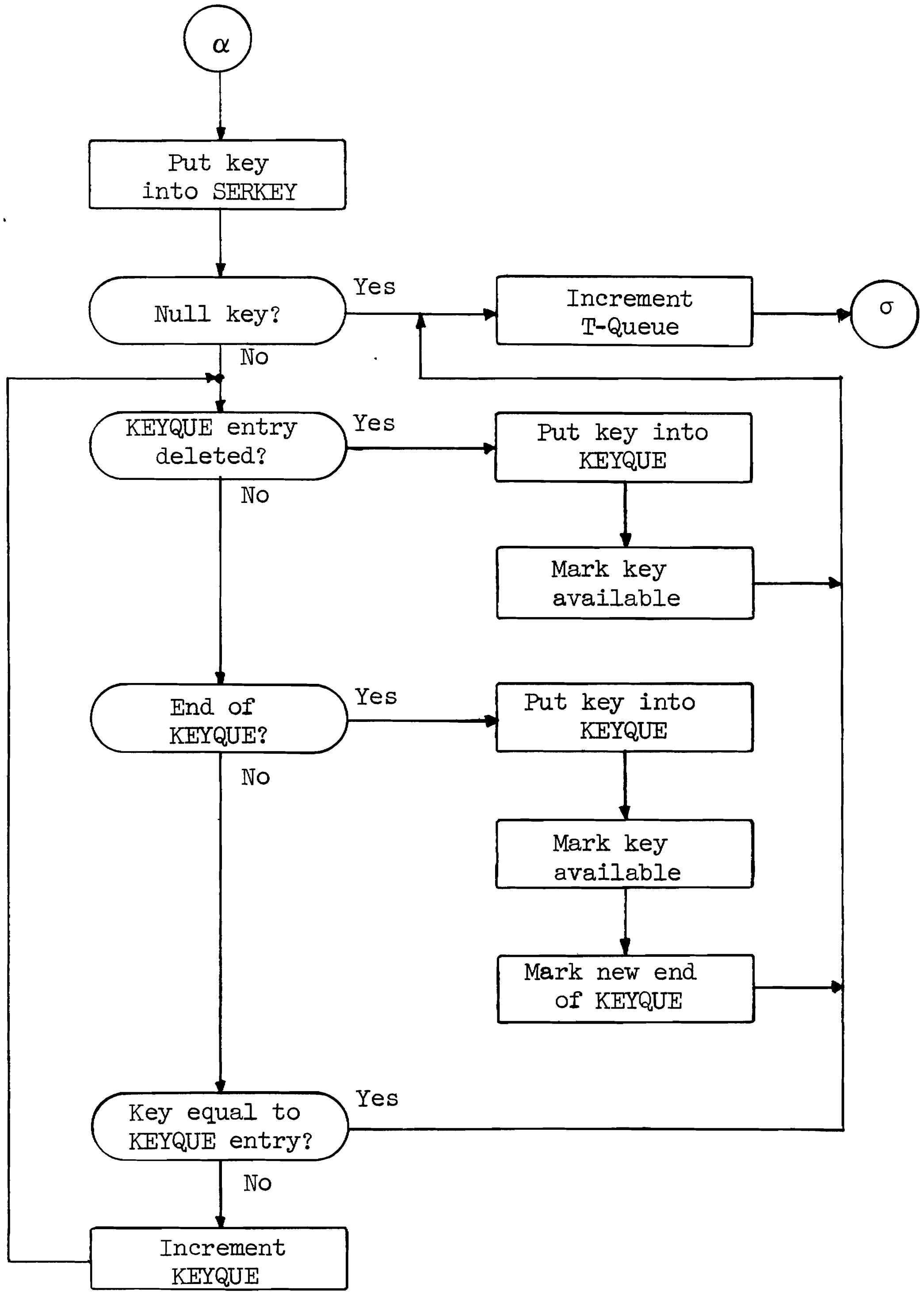


Figure A.1.b: Key Addition Mechanism for KEYQUE

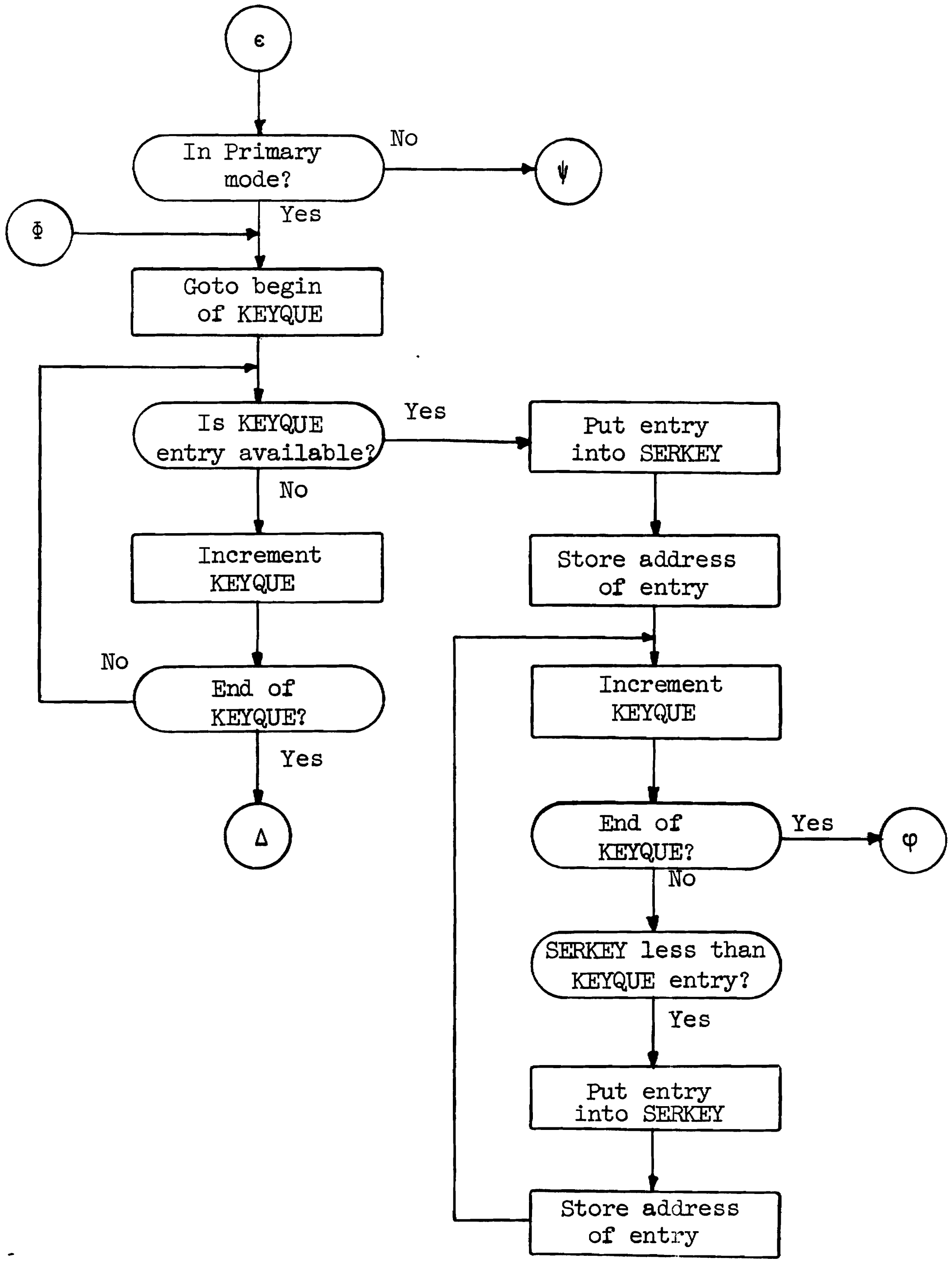


Figure A.1.c: Key Selection Mechanism for KEYQUE

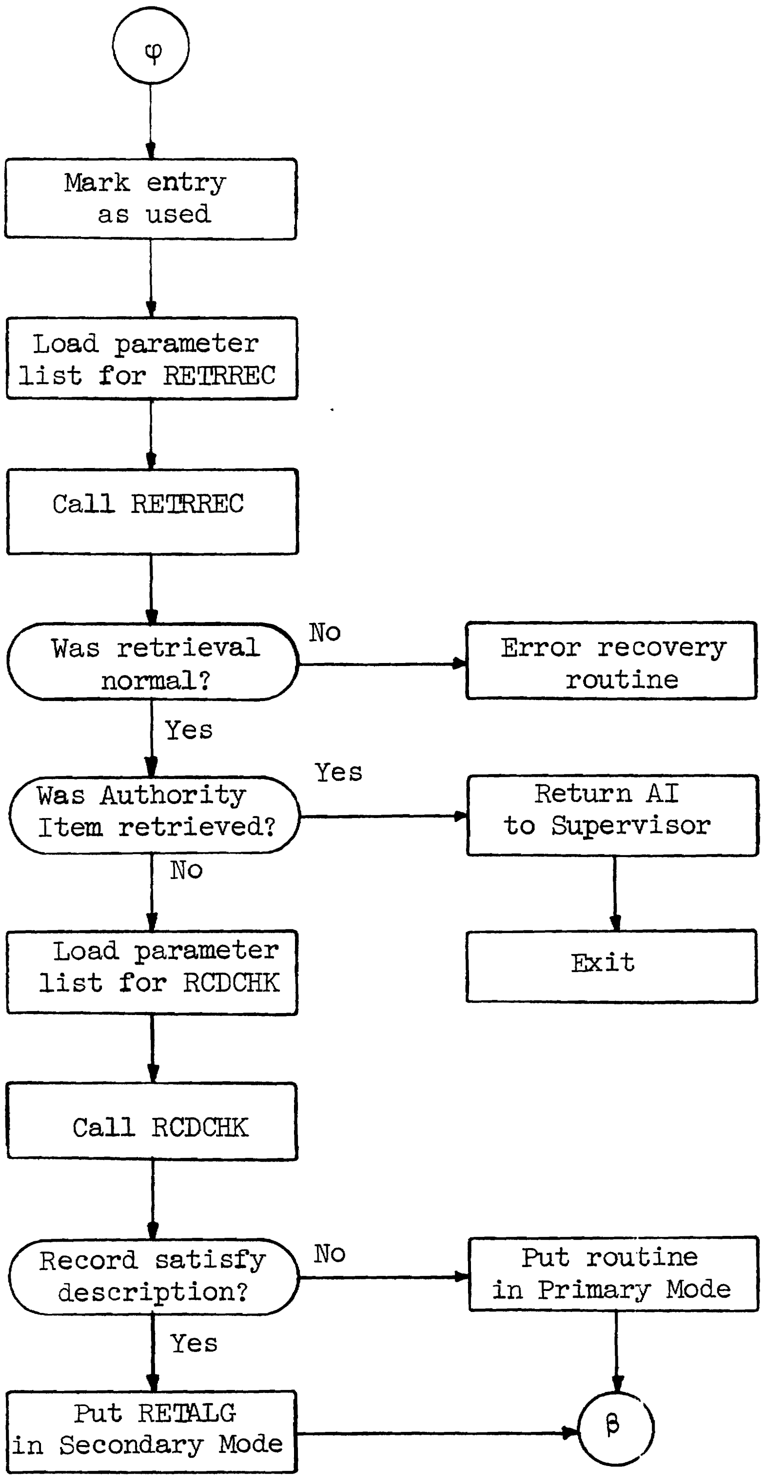

φ
Mark entry as used
Load parameter list for RETRREC
Call RETRREC
Was retrieval normal?
No
Error recovery routine
Yes
Was Authority Item retrieved?
Yes
Return AI to Supervisor
Exit
No
Load parameter list for RCDCHK
Call RCDCHK
Record satisfy description?
No
Put routine in Primary Mode
Yes
Put RETALG in Secondary Mode
β

Figure A.1.d: Record Retrieval and Checking for Satisfying User's Request Description

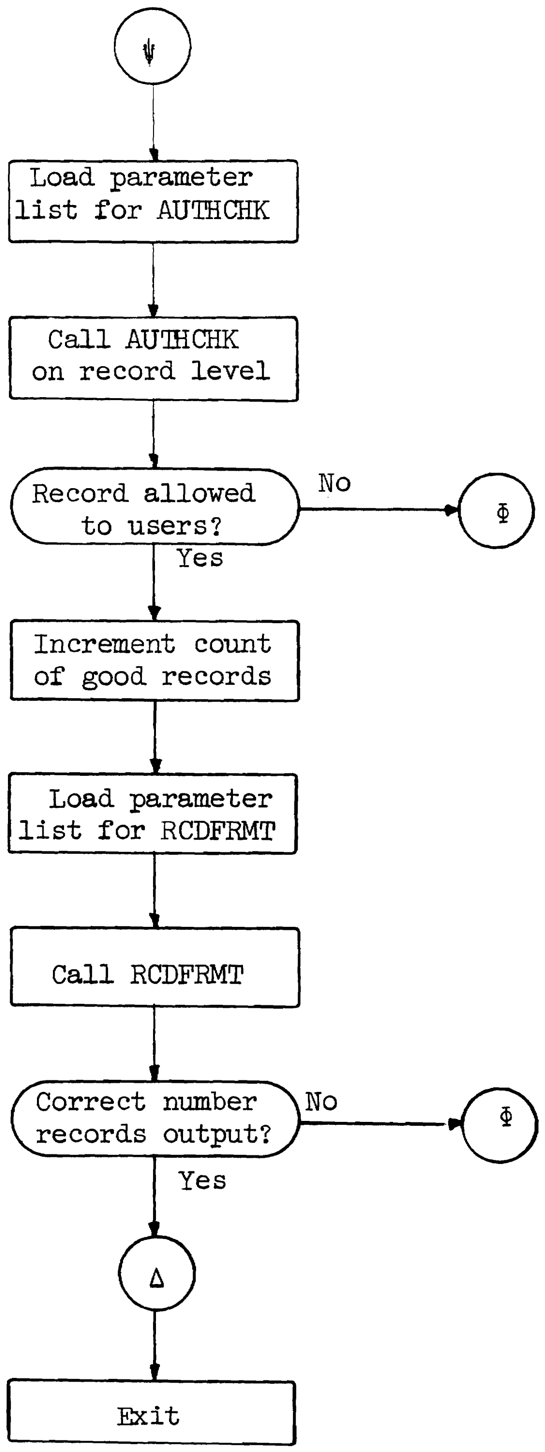

ψ
Load parameter list for AUTHCHK
Call AUTHCHK on record level
Record allowed to users?
No
Φ
Yes
Increment count of good records
Load parameter list for RCDFRMT
Call RCDFRMT
Correct number records output?
No
Φ
Yes
Δ
Exit

Figure A.1.e: Record Authorization and Outputting

## A.2 Routine RCDCHK

The routine RCDCHK is used to check if a record satisfies a given description. This description can be either from a user's request or from the Authority Item limiting a user's access.

### A.2.1 Entry Point

The routine RCDCHK is entered at the location RCDCHK. This is the only entry point and is used whenever RCDCHK is called.

### A.2.2 Exit Point

There is only one exit point in RCDCHK which is used for all purposes. The exit sequence begins at the location RETURN.

### A.2.3 Input Parameter List

RCDCHK is given a parameter list in the following format. Included in the list is space to be used by RCDCHK for storing its control bytes. This eliminates the necessity of RCDCHK having to request core memory. The parameter list is assumed to begin on a fullword boundary.

| Bytes | Content |
|---|---|
| 0 - 3 | Address of prime keyword stack. |
| 4 - 7 | Address of the record in internal format. |
| 8 | Byte to indicate if record is deleted - DELB. |
| 9 - 11 | Address of T-Queue to be used by RCDCHK. |
| 12 - 15 | Address of Key Information Buffer. |
| 16 | Pass byte - PASSB. |
| 17 | Mode indicator byte - MODEB. |

## A.2.4 Register Convention

Registers in RCDCHK are assigned in the following manner:

| Register | Utilization |
|---|---|
| 0 | Unassigned. |
| 1 | Address of input parameter list. |
| 2 | Pointer to prime keyword in Description Control Block. |
| 3 | Pointer to entries in T-Queue. |
| 4 | Pointer to prime keyword stack. |
| 5 | Pointer to RCB entry. |
| 6 | Pointer to value entry in record text. |
| 7 | Base for input parameter list. |
| 8 | Base for RCDCHK. |
| 9 | Pointer to actual value in Key Information Buffer. |
| 10 | Pointer to current entry in the Description Control Block. |
| 11 | Used in character insertion macro. |
| 12 | Used in character insertion macro. Contains branching code for comparison of keywords. |
| 13 | Address of calling routines save area. |
| 14 | Return address of calling routine. |
| 15 | Temporary storage. |

## A.2.5 Internal Codes

The internal codes in RCDCHK are all one byte long and their bit settings will be indicated by hexadecimal digits.

<u>Delete Byte - DELB</u>

| | |
|---|---|
| X'00' | Record is good. |
| X'04' | Record has been deleted; only pointers are extracted. |
| X'08' | Record is security checked for limiting description. |

<u>Pass Byte - PASSB</u>

| | |
|---|---|
| X'00' | Record does not satisfy given description. |
| X'04' | Record satisfies given description. |

<u>Mode Byte - MODEB</u>

| | |
|---|---|
| X'00' | Not in conjunct mode; checking for prime keywords only. |
| X'04' | In conjunct mode, checking for a complete conjunct in a record. |

## A.2.6 Flowchart

The flowchart for the routine RCDCHK is shown in Figure A.2.

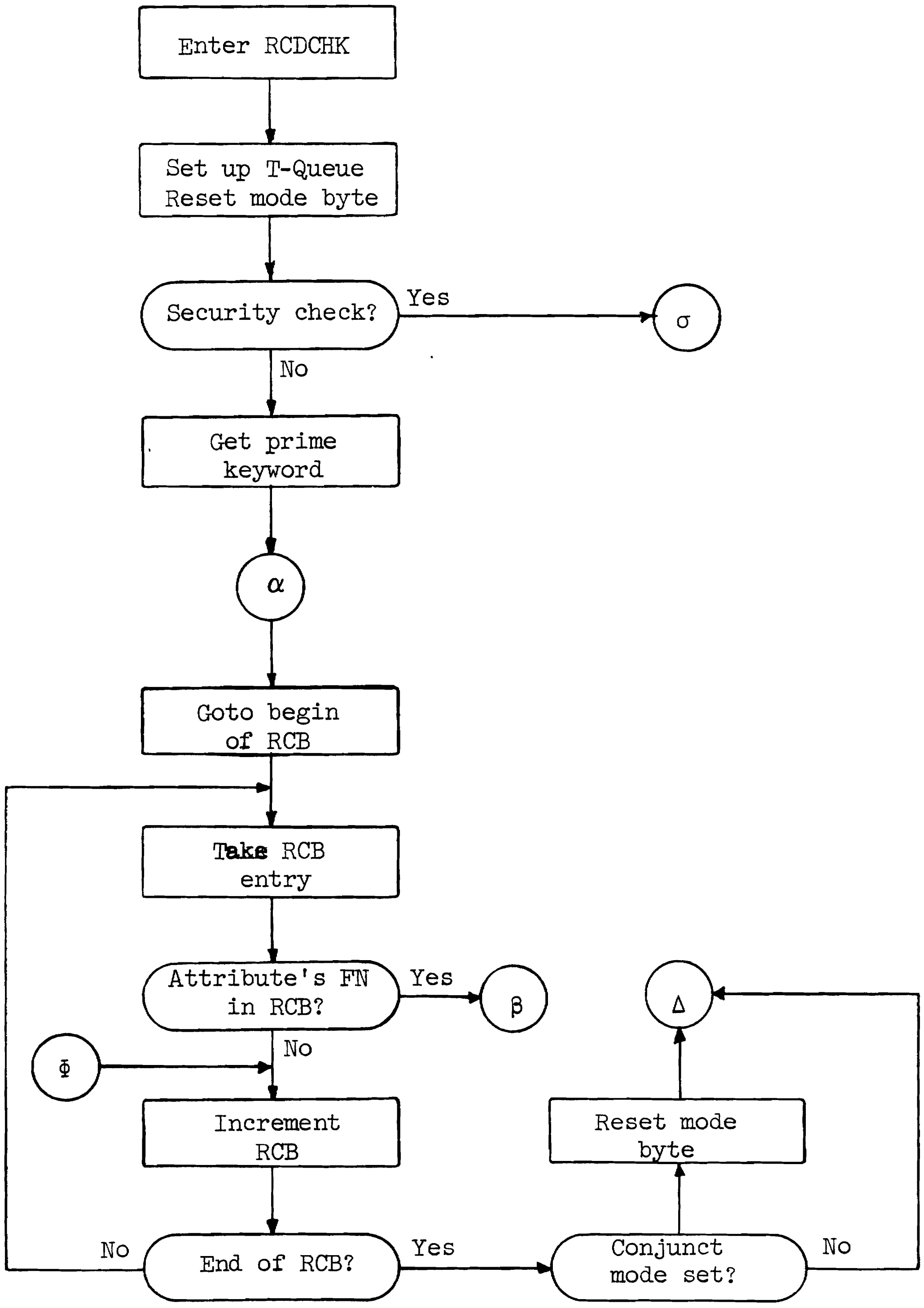


Figure A.2.a: Initialization of RCDCHK and Finding a Format Number (FN) in the Record

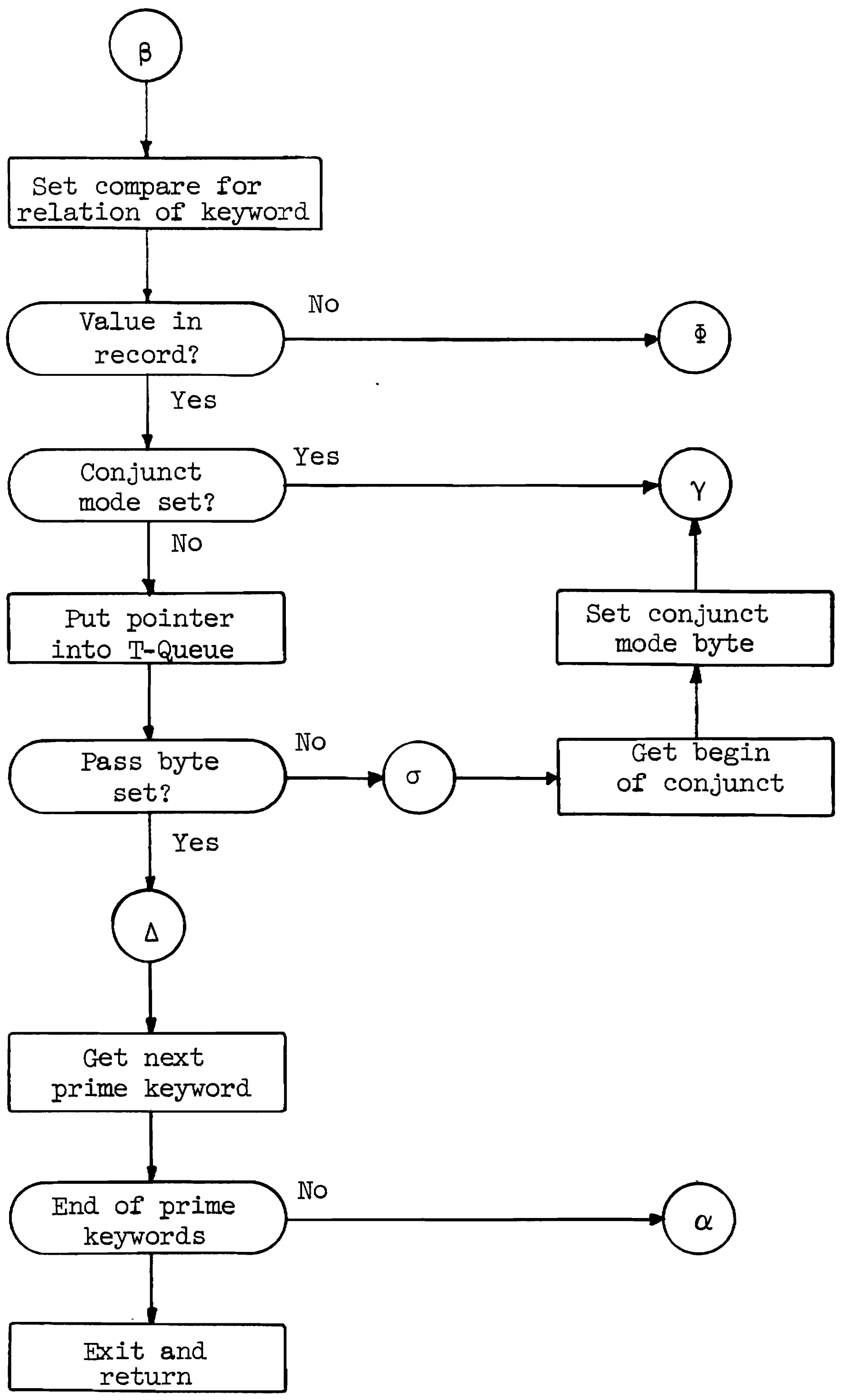


Figure A.2.b: Finding the Value in the Record and Pointer Extraction Mechanism

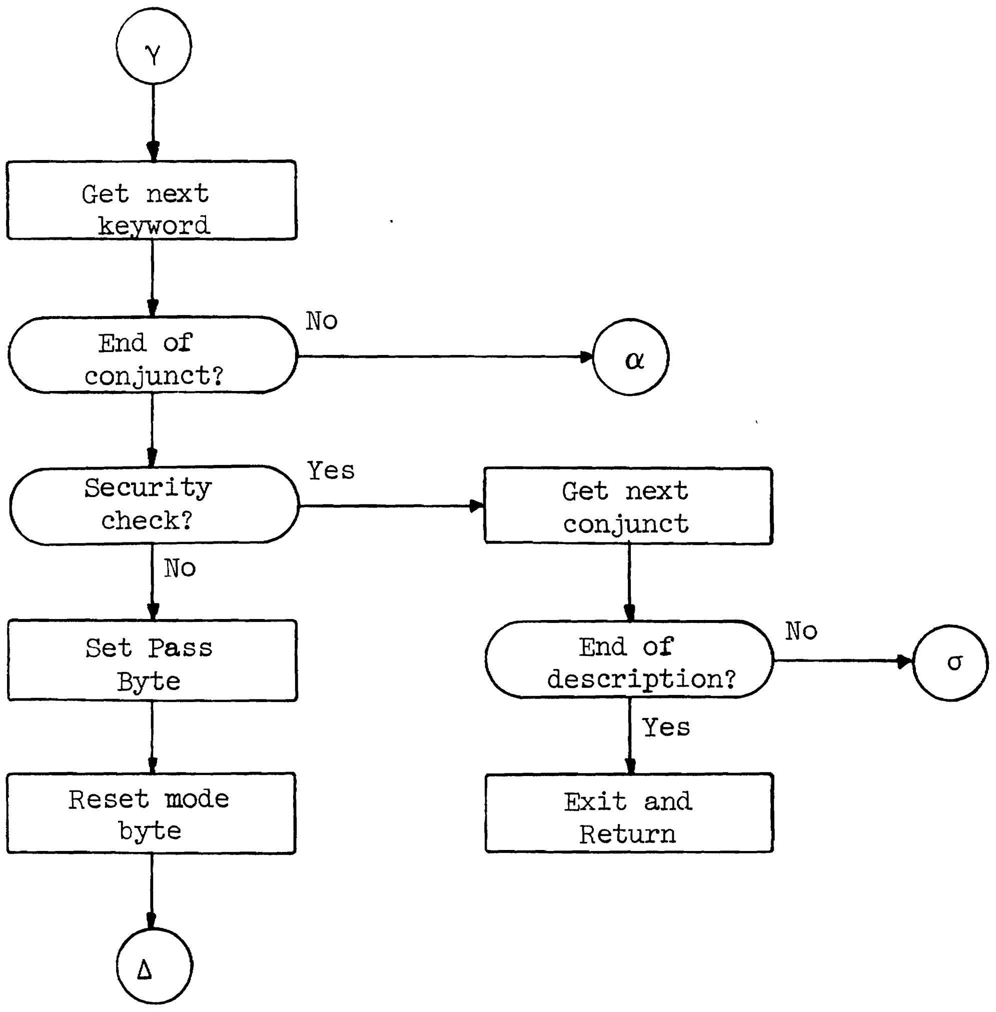


Figure A.2.c: End of Conjunct Check

## A.3 Routine AUTHCHK

The routine AUTHCHK is used to implement the security checking in the EDMF. All checking levels are monitored by AUTHCHK although other subroutines may be called. The input to AUTHCHK indicates what has to be checked for.

### A.3.1 Entry Point

There is a single entry point to AUTHCHK at the location AUTHCHK. The initial processing is the same for all security checks.

### A.3.2 Exit Point

A single exit is used by AUTHCHK for all security checks. The exit processing begins at location AEXIT.

### A.3.3 Input Parameter List

The parameter list given to AUTHCHK has the following format. It is assumed to start on a fullword boundary.

| Bytes | Content |
|---|---|
| 0 - 3 | Address of user's Authority Item. |
| 4 - 7 | Address of record to be checked. |
| 8 - 9 | Length of file name. |
| 10 - 63 | Filename of file whose access is to be checked. |
| 64 - 67 | Address of file open description (a X'FF' in byte 64 indicates that the description is present). |
| 68 - 71 | Length of file open description |
| 72 | Service request code |
| 73 | Checking level code |

| Bytes | Content |
|---|---|
| 74 - 75 | Control information about limiting description. |
| 76 - 79 | Address of internal form of limiting description. |
| 80 - 83 | Address of Key Information Buffer for limiting description. |

A.3.4 Register Conventions

The registers in AUTHCHK are assigned as described in the following specification:

| Register | Utilization |
|---|---|
| 0 | Unassigned. |
| 1 | Address of parameter lists to be given to called subroutines. |
| 2 | Base for AUTHCHK. |
| 3 | Address and base for the parameter list given to RCDCHK. |
| 4 | Temporary storage area. |
| 5 | Unassigned. |
| 6 | Pointer to beginning of value entry in the text of the Authority Item. |
| 7 | Pointer to beginning of text in the Authority Item. |
| 8 | Pointer to file control information in the Authority Item. |
| 9 | Pointer to RCB entry in the Authority Item. |
| 10 | Address and base for input parameter list. |

| Register | Utilization |
|---|---|
| 11 | Address of the temporary storage location DSTORE. |
| 12 | Address and base of work area for AUTHCHK. |
| 13 | Address of save area for AUTHCHK. |
| 14 | Return address in AUTHCHK from called subroutine. |
| 15 | Address of called subroutine. Passing return codes between routines. |

### A.3.5 Internal Work Area

The internal work area for AUTHCHK contains the save area for AUTHCHK as well as the parameter lists for the subroutines called by AUTHCHK. The internal work area has the following format, and like the parameter list, it begins on a fullword boundary.

| Bytes | Content |
|---|---|
| 0 - 71 | Save area for AUTHCHK. |
| 72 - 75 | Temporary storage area DSTORE. |
| 76 - 87 | Extra storage for use by AUTHCHK. |
| 88 - 107 | Parameter list for RCDCHK. |

### A.3.6 Internal Codes

The internal codes used by AUTHCHK for control and storage are given here. Codes which are in the Authority Item to specify a user's access rights can be found in Appendix C. The codes here are in hexadecimal notation.

Checking Level

| | |
|---|---|
| X'01' | Field level check is to be done. |
| X'02' | Record level check is to be done. |
| X'04' | File level check is to be done. |

Return Codes

| | |
|---|---|
| X'00' | User is allowed access which he requested. |
| X'01' | User is denied the requested access. |
| X'03' | User request denied because access data on the requested file was not in his Authority Item. |
| X'10' | Error in user's Authority Item format. |
| X'20' | Input data is in error. |

A.3.7 Flowchart

The flowchart for the routine AUTHCHK is shown in Figure A.3. Each part of the flowchart deals with a separate area of the security checking.

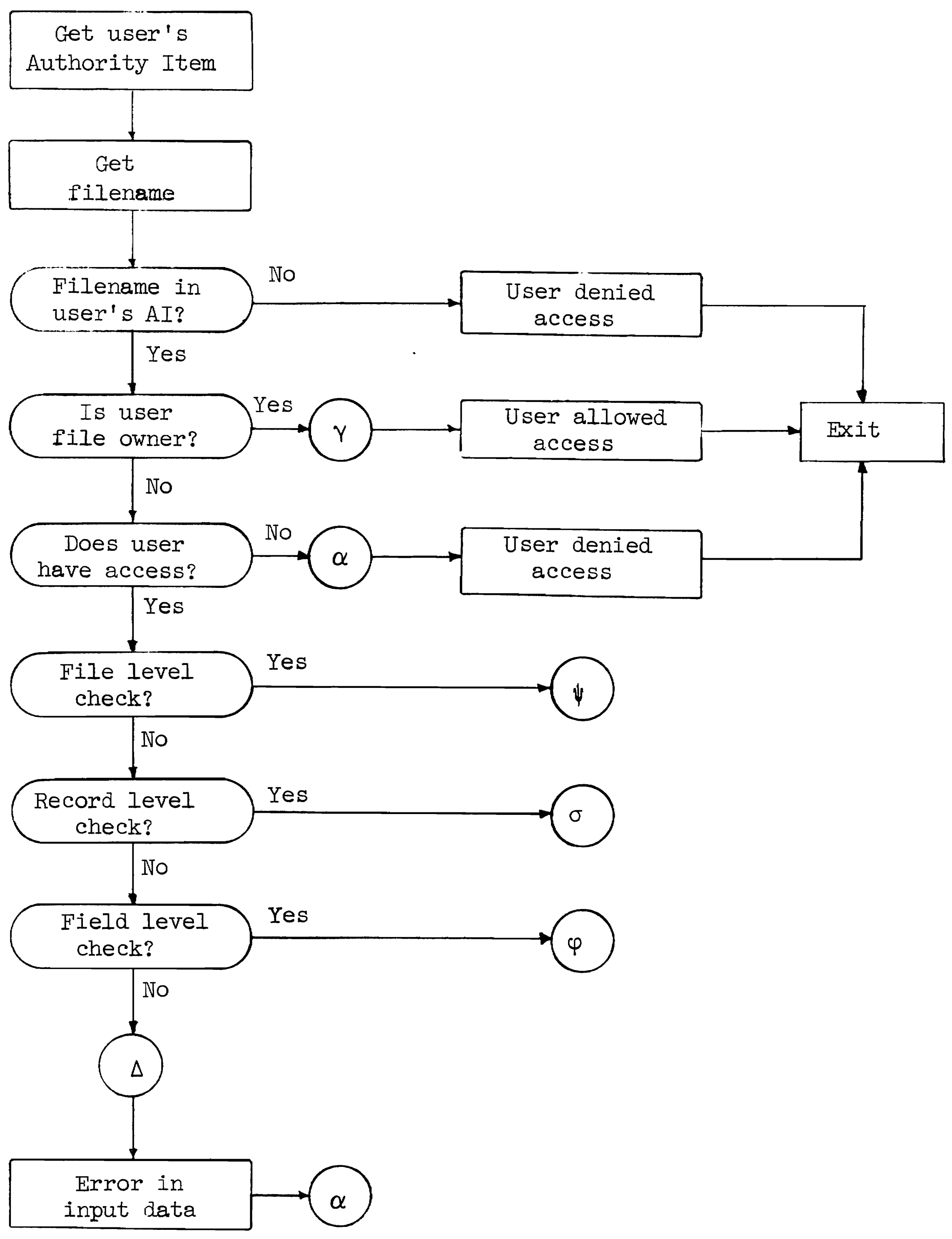


Figure A.3.a: Beginning of AUTHCHK and Initialization of Security Checking

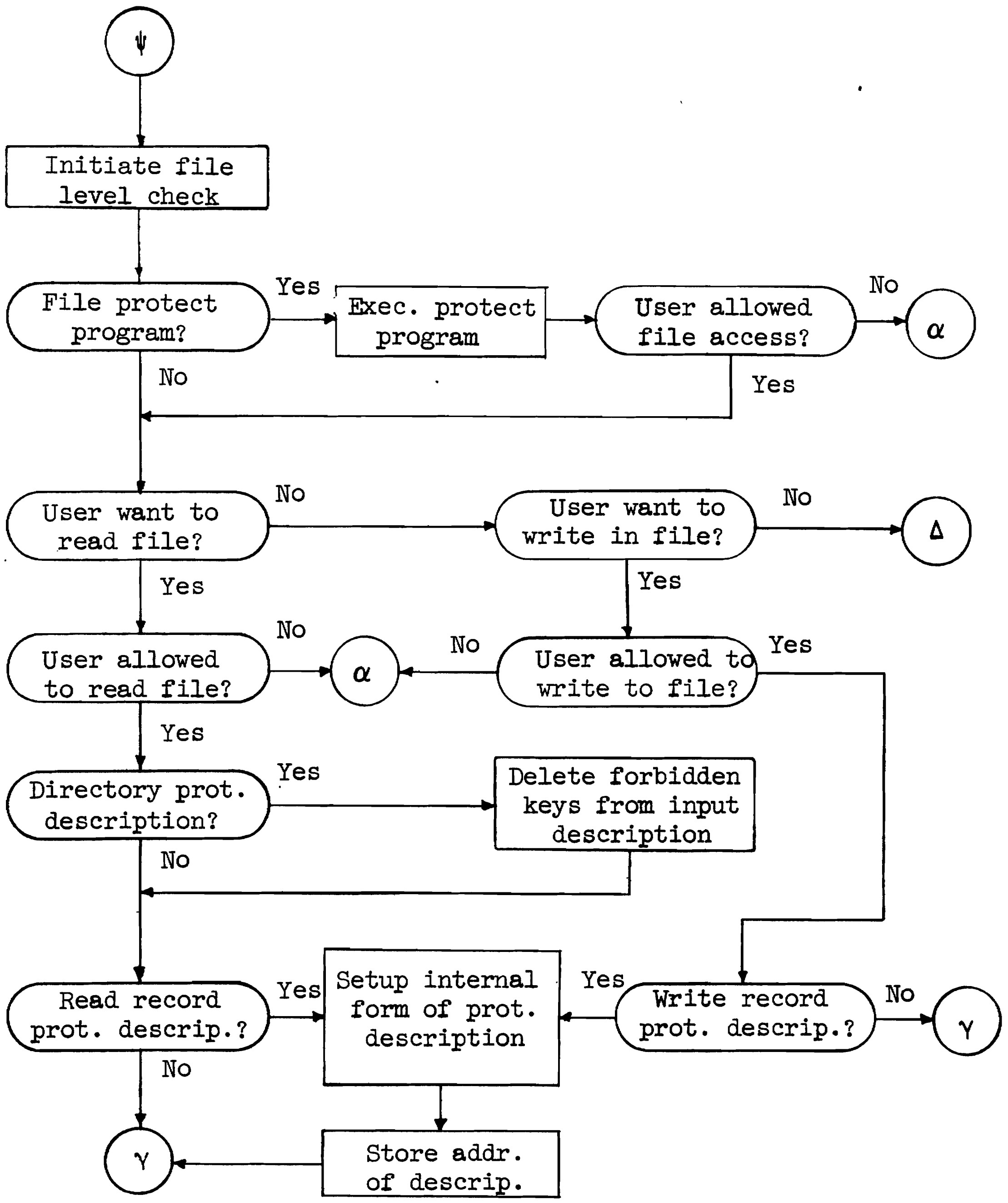

ψ
Initiate file level check
File protect program?
Yes
Exec. protect program
User allowed file access?
No
α
Yes
No
User want to read file?
No
User want to write in file?
No
Δ
Yes
Yes
User allowed to read file?
No
α
No
User allowed to write to file?
Yes
Yes
Directory prot. description?
Yes
Delete forbidden keys from input description
No
Read record prot. descrip.?
Yes
Setup internal form of prot. description
Yes
Write record prot. descrip.?
No
γ
No
γ
Store addr. of descrip.

Figure A.3.b: File Level Security Checking

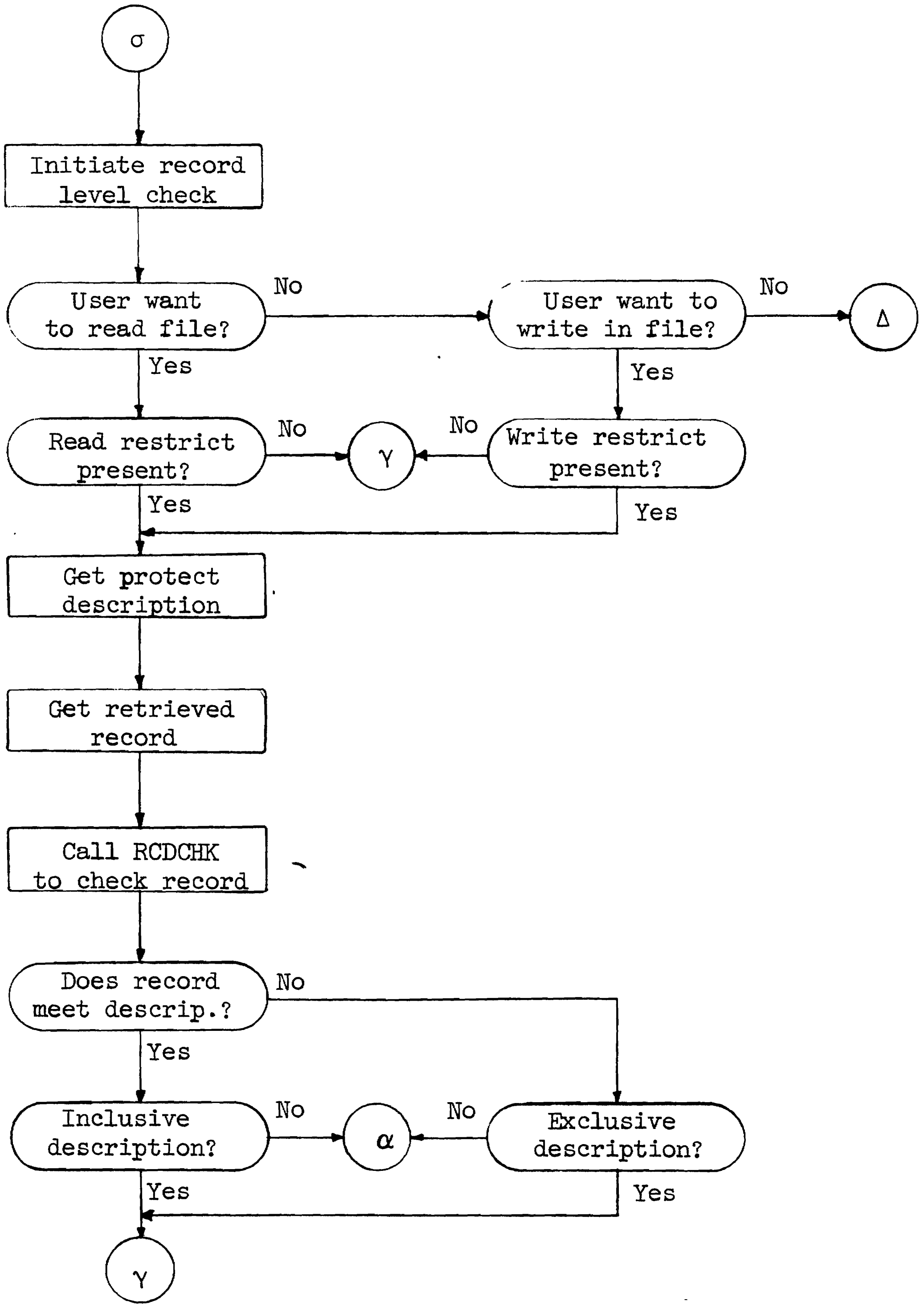


Figure A.3.c: Record Level Security Checking

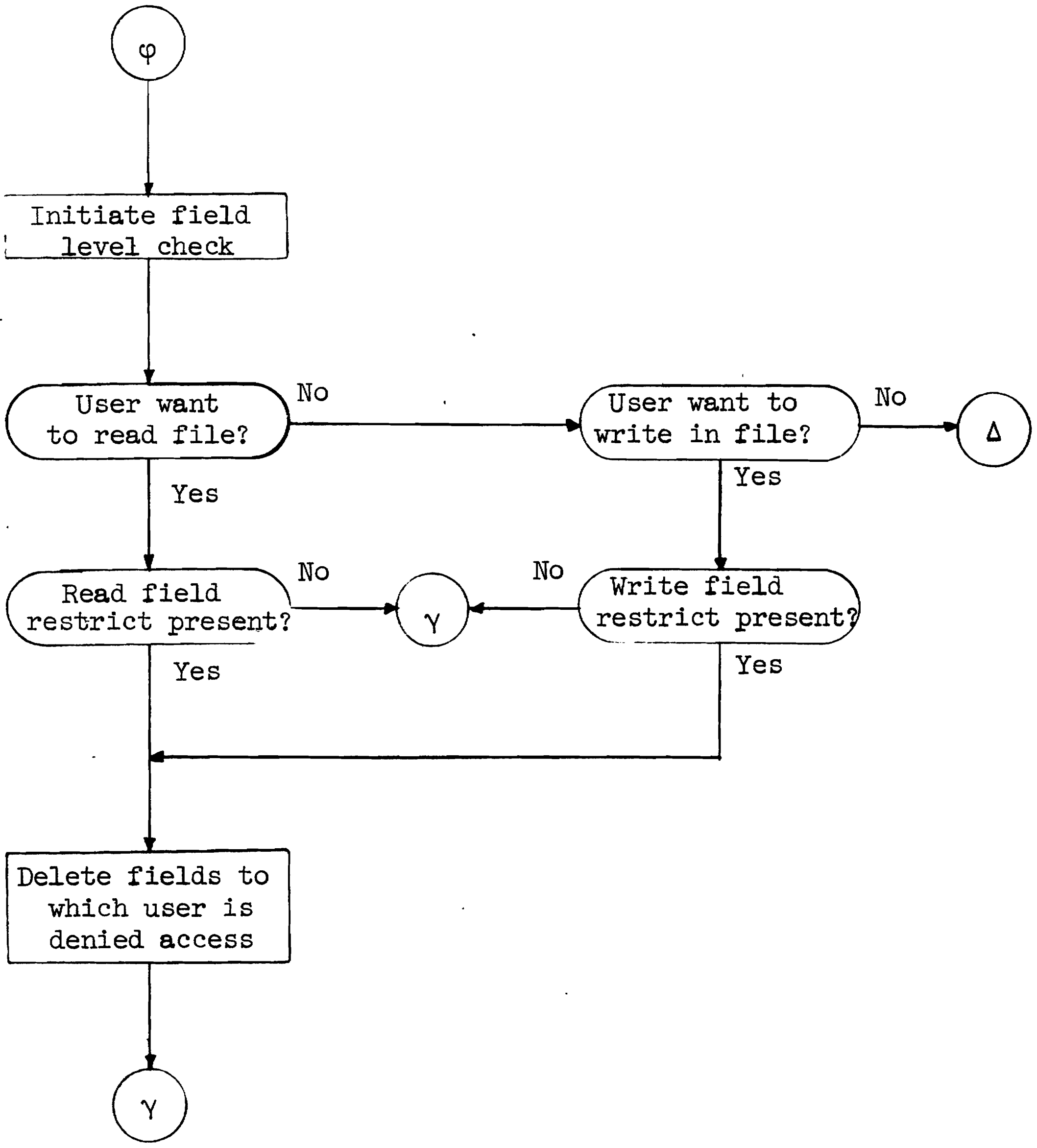

φ
Initiate field level check
User want to read file?
No
User want to write in file?
No
Δ
Yes
Yes
Read field restrict present?
No
γ
No
Write field restrict present?
Yes
Yes
Delete fields to which user is denied access
γ

Figure A.3.d: Field Level Security Checking

## A.4 Routine RCDFRMT

The routine RCDFRMT is used to put records into a format which is given to the user on an output device. Records can be output on a Low Speed Terminal (LST) or on a high speed printer.

### A.4.1 Entry Point

There is one entry point to the routine RETALG which is used for all types of format processing. This is at the location RCDFRMT.

### A.4.2 Exit Point

There is only one exit point to RCDFRMT. The actual exit occurs at the location ERREXT, although the exit sequence begins at location ERROUT so that the allocated memory can be released.

### A.4.3 Input Parameter List

The parameter list given to RCDFRMT is of the format specified below. It is assumed that the format area begins on a fullword boundary.

| Bytes | Content |
|---|---|
| 0 - 3 | Address of the RFB. |
| 4 - 7 | Address of the internal format of the record. |
| 8 - 11 | Address of the area where the formatted record is assembled. |
| 12 | Print code indicating the output device wanted by the user. |
| 13 - 15 | Unused. |

### A.4.4 Register Conventions

The registers in RCDFRMT are assigned according to the following specification:

| Register | Utilization |
|---|---|
| 0 | Unassigned. |
| 1 | Address of parameter list given to RCDFRMT. |
| 2 | Base for RCDFRMT. |
| 3 | Address of the RFB. Temporary storage for loading parameter lists. |
| 4 | Address of the internal form of the record. |
| 5 | Pointer to RCB entry. Beginning address of line image. |
| 6 | Pointer to beginning of the format entries in the RFB. |
| 7 | Pointer to the beginning of the text in the internal form of the record. Length counter for the move instruction. |
| 8 | Pointer to entry in Table of Contents for RFB. Pointer to assembly area. |
| 9 | Pointer to format assembly area. End of assembly area after collating is completed. |
| 10 | Pointer to format entry of RFB. |
| 11 | Address and base of input parameter list. |
| 12 | Address and base of the work area for RCDFRMT. |
| 13 | Address of save area for calling program. |
| 14 | Return address in the calling program. |
| 15 | Address of entry point in RCDFRMT. |

### A.4.5 Internal Work Area

The internal work area of RCDFRMT contains space for the formation of the line images which are passed to the output device. The work area is in the format specified below:

| Bytes | Content |
|---|---|
| 0 - 71 | Save area for RCDFRMT. |
| 72 - 79 | Parameter list for output macro. |
| 80 - 219 | Area for line image of the V-form output record. |
| 220 - 223 | Work area for RCDFRMT. |
| 224 - 227 | Beginning address of formatting area. |
| 228 - 231 | Highest line number used up to present time. |

### A.4.6 Internal Codes

These codes are used internally in RCDFRMT for controlling the flow of processing.

<u>Print Code</u>

| | |
|---|---|
| X'00' | Record is to be printed out on LST. |
| X'02' | Record is to be printed out on high speed printer. |

<u>Internal Delimiter</u>

| | |
|---|---|
| X'FC' | This code is used as the delimiter to separate. |

### A.4.7 Flowchart

The flowchart for RCDFRMT is shown in Figure A.4. Each aspect of the processing is shown in a separate part of the figure.

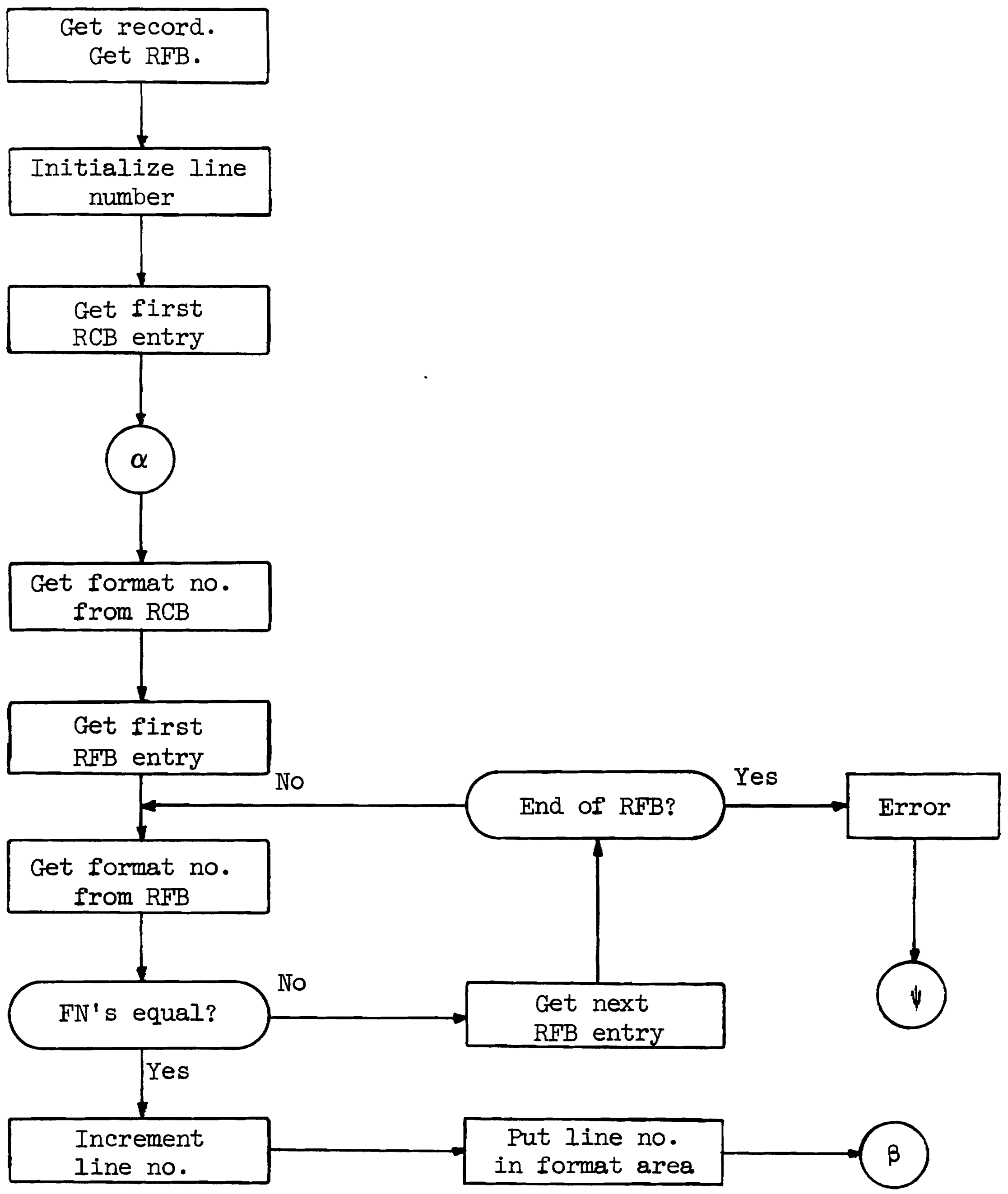


Figure A.4.a: Initializing RCDFRMT and Checking Getting the Attribute

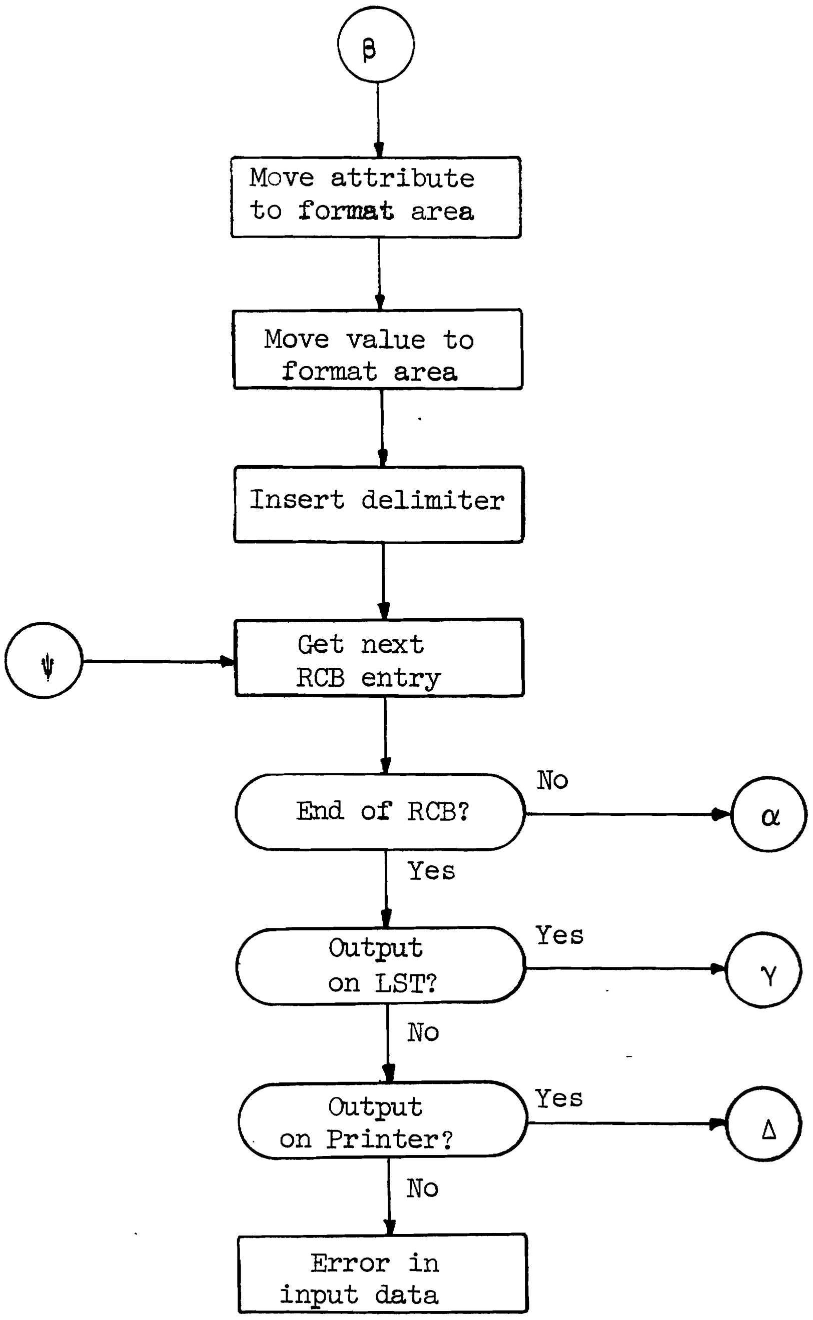
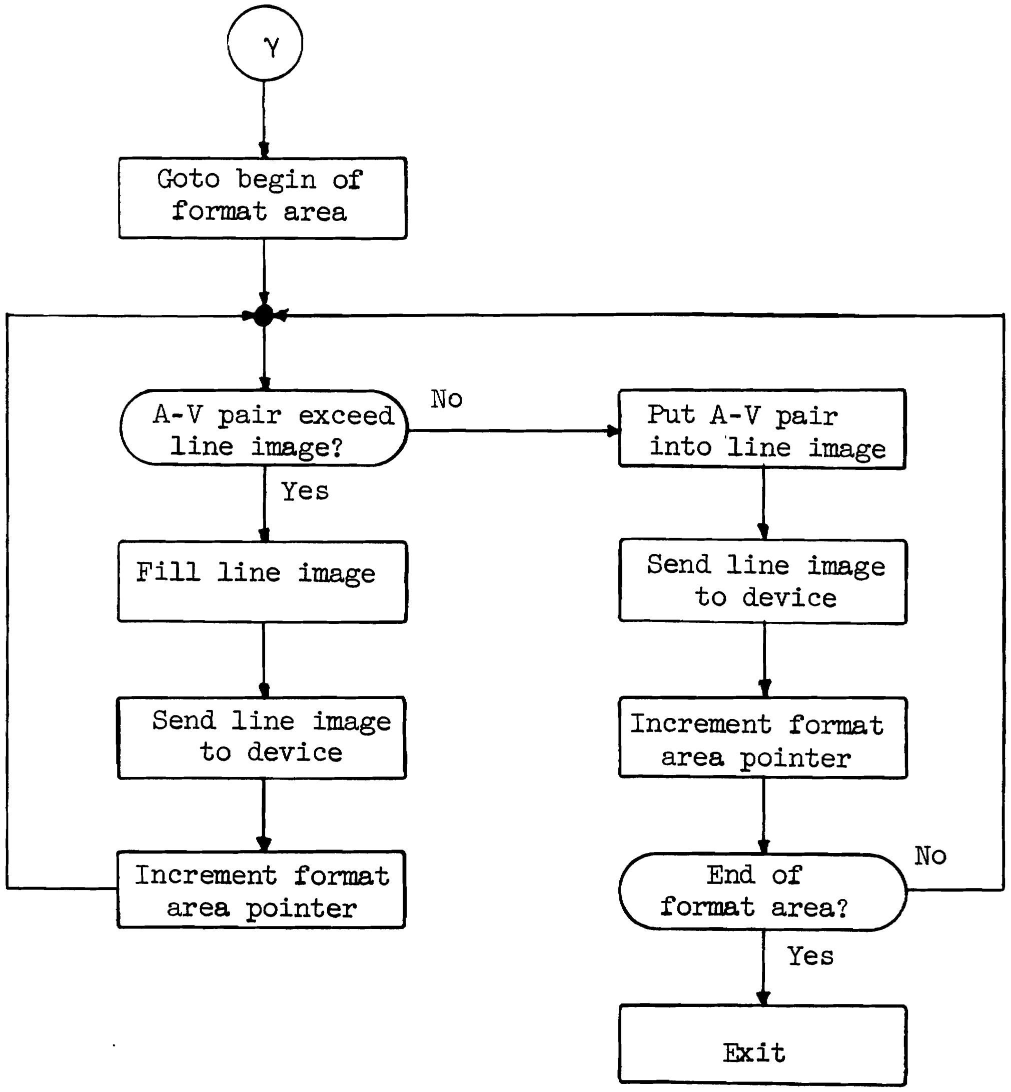


Figure A.4.b: Moving Attribute and Value into Formatting Area



Y
Goto begin of format area
A-V pair exceed line image?
No
Yes
Put A-V pair into line image
Fill line image
Send line image to device
Send line image to device
Increment format area pointer
Increment format area pointer
End of format area?
No
Yes
Exit

Figure A.4.c: Outputting Records on the LST

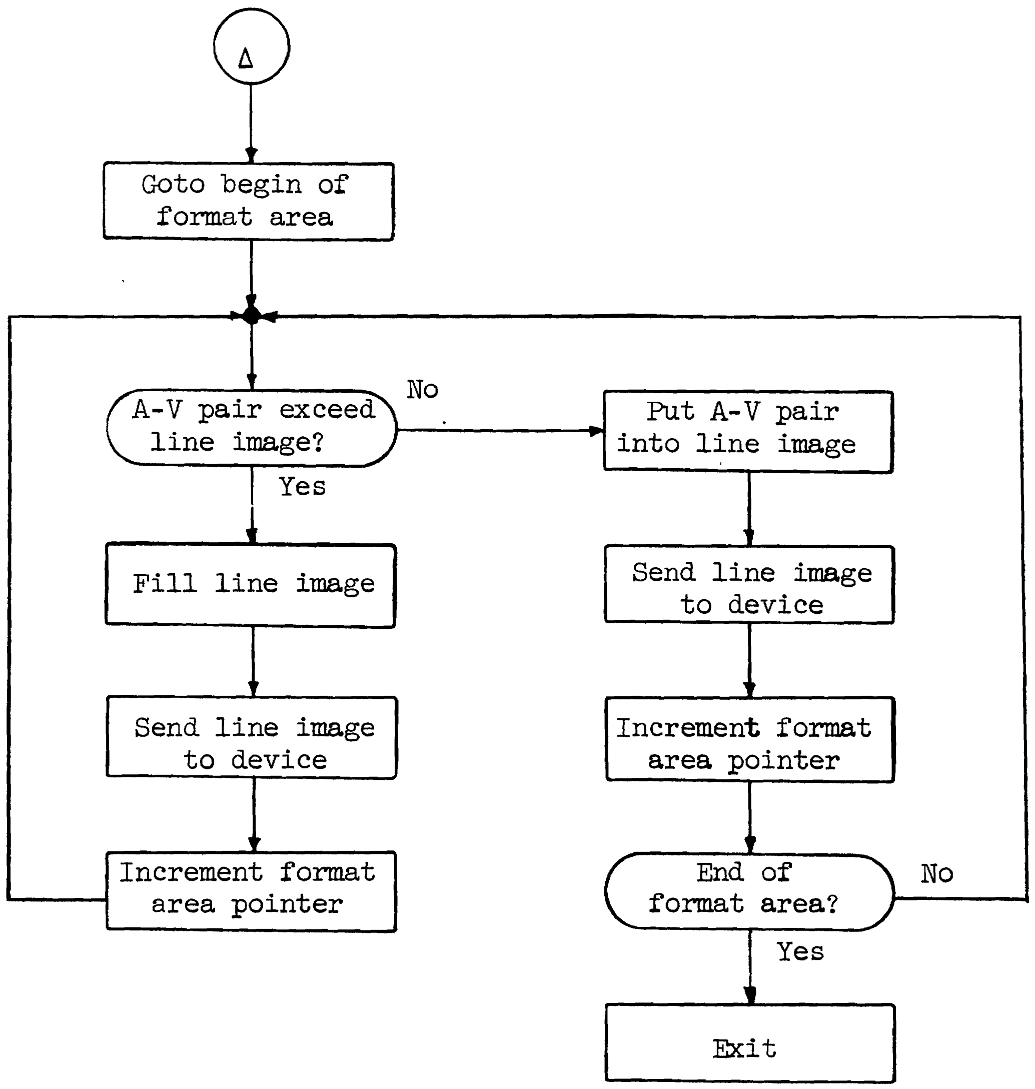

Δ
Goto begin of format area
A-V pair exceed line image?
No
Yes
Put A-V pair into line image
Fill line image
Send line image to device
Send line image to device
Increment format area pointer
Increment format area pointer
End of format area?
No
Yes
Exit

Figure A.4.d: Outputting Records on the High Speed Printer

## A.5 Routine COREFMT

The routine COREFMT is used to change the record from its internal format to the core format.

### A.5.1 Entry Point

There is only one entry point to the routine COREFMT at the location of the same name. This entry point is used for each time that COREFMT is called.

### A.5.2 Exit Point

There is one exit point for the routine COREFMT. The exit sequence begins at the location ERRENTER, and it releases the memory allocated by COREFMT.

### A.5.3 Input Parameter List

The parameter list for the routine COREFMT is in the format specified below. The parameter list is assumed to begin on a fullword boundary.

| Bytes | Content |
|---|---|
| 0 - 3 | Address of the internal format of the record. |
| 4 - 7 | Address of the RFB. |
| 8 - 11 | Address of the core format area. |
| 12 - 15 | Length of the core format area. |

### A.5.4 Register Conventions

The registers are assigned according to the following specification:

| Register | Utilization |
|---|---|
| 0 | Unassigned. |
| 1 | Address of parameter list given to COREFMT. |
| 2 | Base for COREFMT. |

| Register | Utilization |
|---|---|
| 3 | Unassigned. |
| 4 | Address of the RFB. |
| 5 | Pointer to format entry in RFB. Pointer to the table of contents for the RFB. |
| 6 | Pointer to record text entry. |
| 7 | Unassigned. |
| 8 | Address of the record. Pointer to RCB entry. |
| 9 | Address of core format area. |
| 10 | Pointer to Attribute-Value entry in the core format. |
| 11 | Temporary storage. |
| 12 | Address and base for COREFMT work area. |
| 13 | Address of the calling routine's save area. |
| 14 | Temporary storage. Address of return to calling program. |
| 15 | Temporary storage. |

### A.5.5 Internal Work Area

The work area for COREFMT also contains a save area for its registers should it need to call subroutines. The work area is assumed to begin on a fullword boundary and is in the format specified below:

| Bytes | Content |
|---|---|
| 0 - 71 | Save area for COREFMT. |
| 72 - 75 | Work area. |
| 76 - 79 | Address of first format entry in the RFB. |
| 80 - 83 | Address of the first text entry in the record. |

## A.5.6 Flowchart

The flowchart for COREFMT is shown in Figure A.5. The two major phases in the processing are illustrated.

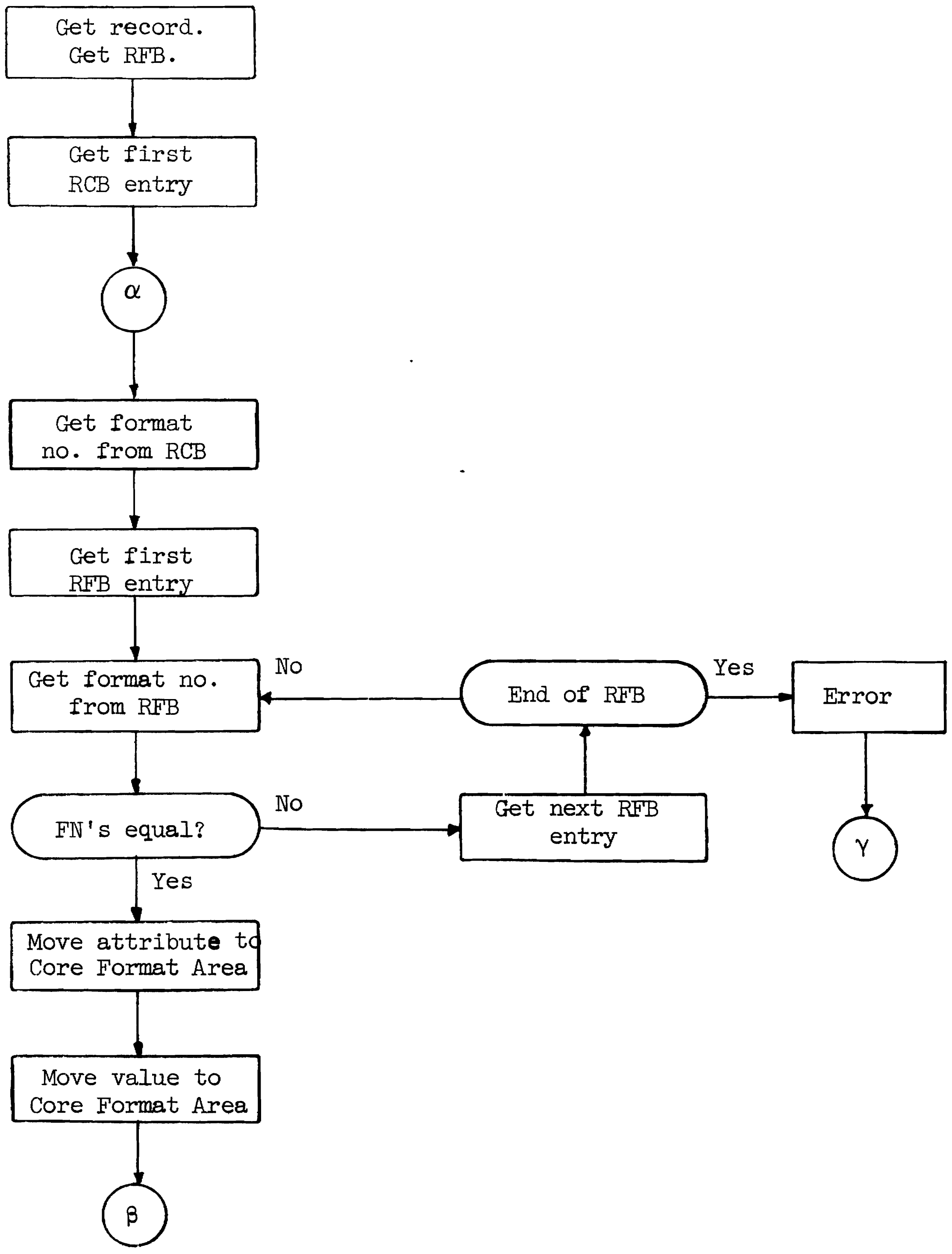


Figure A.5.a: Beginning of COREFMT and Moving the Attribute and Value to the Format Area.

FN - Format Number

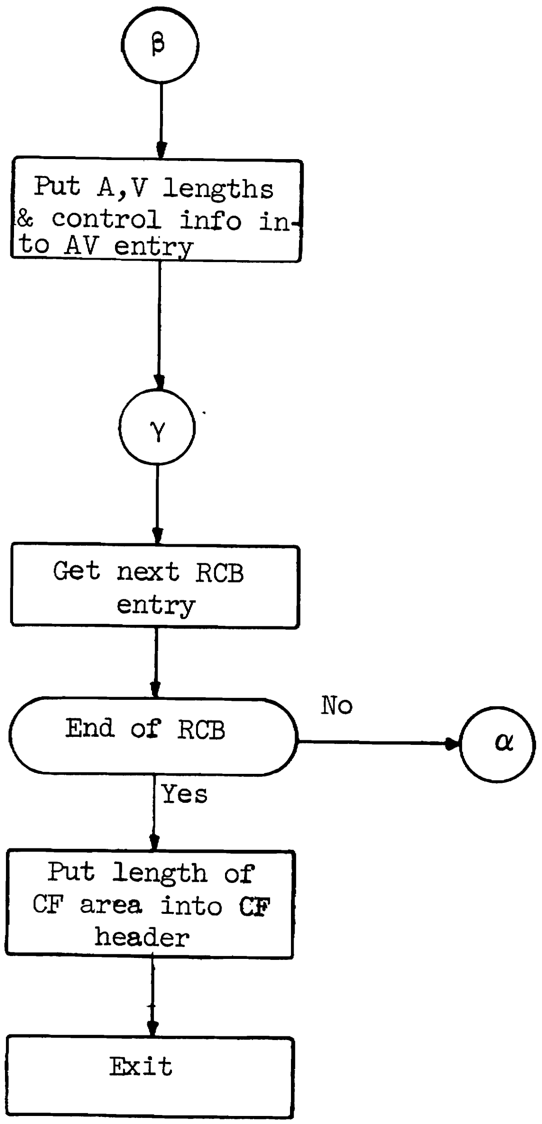

β
Put A,V lengths & control info into AV entry
γ
Get next RCB entry
End of RCB
No
α
Yes
Put length of CF area into CF header
Exit

Figure A.5.b: Finalizing the Attribute-Value (AV) Entry, and the Exit

## APPENDIX B

## RECORD FORMATS

### B.1 Internal Format of the Record

| | | |
|---|---|---|
| 3 bytes | Size of Record | HEADER |
| 2 bytes | Count | |
| 5 bytes | Disk Address (ISAM Key) | |
| 2 bytes | Pointer to Test (values) relative to 1st byte of record | |
| 2 bytes | Control Information | |
| 2 bytes | Format number | RECORD CONTROL BLOCK ENTRY |
| 3 bytes | Size of value | |
| 1 byte | blank | |
| 1 byte | Control information | |
| 3 bytes | Relative address of first value | |
| 2 bytes | Format number | |
| 3 bytes | Size of value | |
| 1 byte | blank | |
| 1 byte | Control information | |
| 3 bytes | Relative address of second value | |
| | ⋮ | |
| 5 bytes | Pointer (ISAM Key) | RECORD TEXT ENTRY |
| n bytes | Value of attribute | |
| | ⋮ | |

Notes on the Internal Format of the Record:

1. The Count field is used when a record is broken into several pieces for I/O.
2. The Control information in the header is used among other things to indicate that a record is to be deleted. This is also true for the Control information in the Record Control Block.

3. The Size of Value in the RCB appears even though the format may be fixed-length.

4. Control information in the RCB entry is one byte long with the following specification:

   ab00 0000

|  |  |  |
|---|---|---|
| a: | 0 | Entry is not a keyword |
|  | 1 | Entry is a keyword |
| b: | 0 | Keyword is active |
|  | 1 | Keyword is not active |

## B.2 Record Format Block (RFB)

| | | |
|---|---|---|
| 4 bytes | Control Information | |
| 2 bytes | Pointer to first format relative to first byte of RFB | HEADER |
| 2 bytes | Last format number assigned | |
| 2 bytes | Format number | |
| 2 bytes | Control information | |
| 2 bytes | Relative address of first format | TABLE OF CONTENTS |
| 2 bytes | Format number | |
| 2 bytes | Control information | |
| 2 bytes | Relative address of second format | |
| | • • | |
| 2 bytes | Format number | |
| 4 bytes | Type of format | |
| 2 bytes | Level number | |
| 2 bytes | Repetition number | |
| 3 bytes | Size of value | FORMAT ENTRY |
| 1 byte | Control information | |
| 2 bytes | blank | |
| 4 bytes | Field protection data | |
| 2 bytes | Length of attribute | |
| n bytes | Full attribute name | |
| | • • • | |

Notes on the Record Format Block:

1. All relative addresses in the Table of Contents are relative to the first byte in the first format, hence a pointer to the first format is placed in the header. This arrangement obviates the need for changing relative addresses in the Table of Contents if new formats are added to the block.
2. Format numbers appear in the Table of Contents in order of their appearance in file records.
3. The Type of Format field may be used to indicate a program which processes the format.

4. Like the size of value entry, the repetition number will not appear in the format if the format may repeat a variable number of times. Variable repetition is indicated by a bit in the control information.

5. Control information in the format entry is one byte long with the following specification:

abcd ee00

| | | |
|---|---|---|
| a: | 0 | Repetition number is variable |
| | 1 | Repetition number is fixed |
| b: | 0 | Value size is variable |
| | 1 | Value size is fixed |
| c: | 0 | Attribute is not in the directory |
| | 1 | Attribute is in the directory |
| d: | 0 | Attribute optionally appears in a record |
| | 1 | Attribute appears in every record |
| ee: | 00 | Value is packed decimal |
| | 10 | Value is alphabetic |
| | 01 | Unassigned |
| | 11 | Unassigned |

## B.3 Core Format of the Record

| | | |
|---|---|---|
| 3 bytes | Size of Record | CORE FORMAT HEADER |
| 5 bytes | Reference number, unpacked | |
| 1 byte | Control information | |
| 3 bytes | Length of attr.-value entry | |
| 1 byte | Control information | ATTRIBUTE VALUE ENTRY |
| 1 byte | Number of Directory Lists | |
| 2 bytes | Length of attribute | |
| variable | Attribute | |
| 3 bytes | Length of Value | |
| variable | Value | |
| 3 bytes | Length of attr.-value entry | |
| 1 byte | Control Information | |
| 1 byte | Number of Directory Lists | |
| 2 bytes | Length of attribute | |
| variable | Attribute | |
| 3 bytes | Length of Value | |
| variable | Value | |
| | •<br>•<br>•<br>•<br>• | |

Notes on the Core Format of the Record:

1. "Number of Directory Lists" field is used for those attribute-value pairs which are used as keywords when file characteristics allow a variable number of directory lists. Field is ignored for other attribute-value pairs.
2. The length specified in the 3 byte "Size of Record" entry includes the 9 byte Header size.

# APPENDIX C

# THE AUTHORITY ITEM

The Authority Item (AI) is a record in the system file which contains the access control information for a single user. The AI is set up in the same internal format as the other records in the EDMF. This allows the Supervisor to use the same routines to handle both the AI and the user records, resulting in an efficient use of system resources.

## C.1 Attributes

The attributes which are used in connection with the AI are the following:

1. User ID
2. General Control Information
3. Filename
4. File Access Control Information
5. File Access Program Name
6. File Opening Description Entry
7. File Blocking Description Entry
8. Record Protection Description Entry
9. Record Protection Program Entry
10. Field Protection Description Entry
11. Field Protection Program Entry
12. Directory Protection Description Entry
13. Directory Protection Program Entry

## C.2 Logical Format

The logical format of the AI reflects the organization of the access control data. The format given below in Figure C.1 would be the way an Authority Item would appear if it was printed out.

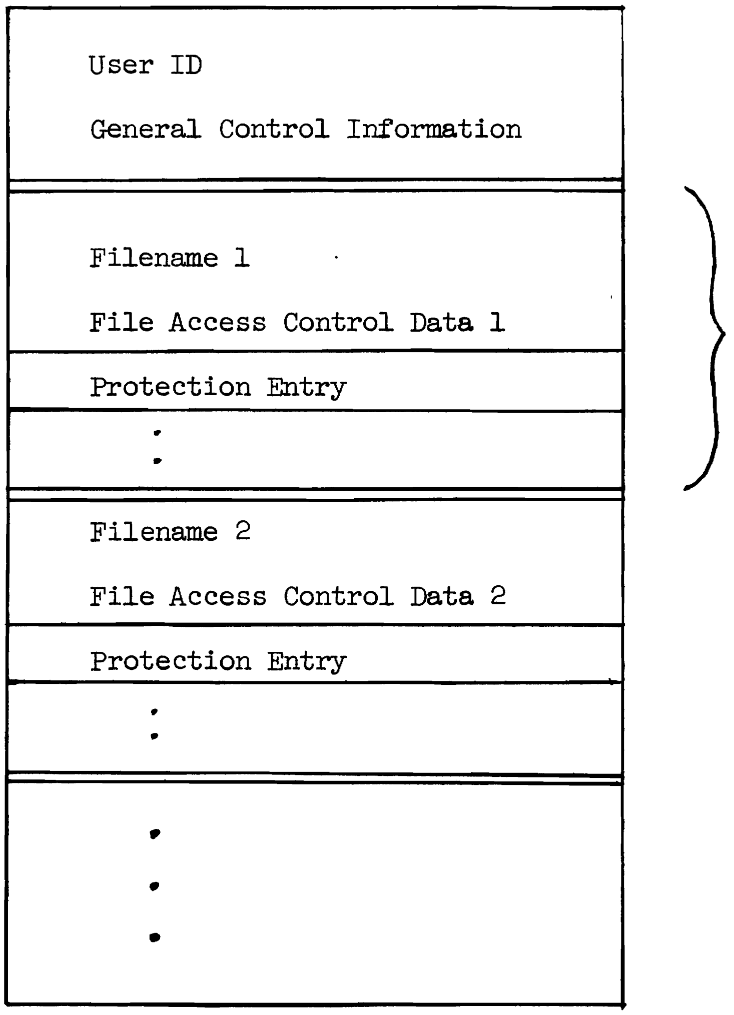

User ID
General Control Information
Filename 1
File Access Control Data 1
Protection Entry
Filename 2
File Access Control Data 2
Protection Entry
Data for one file

Figure C.1: Logical Format of the Authority Item

## C.3 File Access Control Data

The actual file access control data is in the AI as a value would be in a user record. This value is associated with the attribute 'File Access Control Information' through the use of a format number in the RCB of the AI.

This control data is allocated one fullword on the RCA Spectra 70/46. Since there are four bytes in a fullword, each byte will correspond to a particular level of authority and will contain the information pertinent to that level of checking. The bytes are allocated as follows:

Byte 1 - Data concerning user's ownership extent and type of access control.

Byte 2 - Control data pertinent to a file level check.

Byte 3 - Control data pertinent to a record level check.

Byte 4 - Control data pertinent to a field level check.

The format of the fullword containing the File Access Control Data is the following:

```
aabb 0000    cdee fghi    jk00 000    lmn0 0000
```

| | | |
|---|---|---|
| aa: | 00 | User unauthorized |
| | 01 | User shares file |
| | 11 | User owns file |
| bb: | 00 | No access to file |
| | 01 | Standard Access Control |
| | 10 | Program access control |
| | 11 | Standard the program access control |
| c: | 0 | File protection program present |
| | 1 | File protection program absent |

| | | |
|---|---|---|
| d: | 0 | Open description present |
| | 1 | Open description absent |
| ee: | 00 | No read or write to file |
| | 01 | Read only file |
| | 10 | Write only to file |
| | 11 | Read and write to file |
| f: | 0 | Directory protection description present |
| | 1 | Directory protection description absent |
| g: | 0 | Directory protection program present |
| | 1 | Directory protection program absent |
| h: | 0 | User may not block file |
| | 1 | User may block file |
| i: | 0 | Blocking description present |
| | 1 | Blocking description absent |
| j: | 0 | Record protection description present |
| | 1 | Record protection description absent |
| k: | 0 | Record protection program present |
| | 1 | Record protection program absent |
| l: | 0 | Standard field protection active |
| | 1 | Standard field protection inactive |
| m: | 0 | Field protection description present |
| | 1 | Field protection description absent |
| n: | 0 | Field protection program present |
| | 1 | Field protection program absent |

## C.4 Entry Type

Each protection entry consists of one control byte followed by the description or program which limits the access. Again, this entry is stored as a value in the AI corresponding to one of the attributes which describe the entry.

The code for the entry type shall consist of one byte (8 bits) and is divided into two parts. The first 4 bits of the byte denotes what kind of service (Read or Write) the entry type is applicable to and the extent of applicability. The second four bits contain a code which denotes the type of entry that it is.

The format of the entry type code is the following:

aabb xxxx

| | | |
|---|---|---|
| aa: | 00 | Not applicable to read |
| | 10 | Deny read according to entry |
| | 11 | Allow read according to entry |
| bb: | 00 | Not applicable to write |
| | 10 | Deny write according to entry |
| | 11 | Allow write according to entry |
| xxxx: | 0000 | Entry deleted or no longer active |
| | 0001 | File access program pointer |
| | 0010 | File opening description |
| | 0011 | File blocking description |
| | 0100 | Record protection description |
| | 0101 | Record protection program pointer |
| | 0110 | Field protection description |
| | 0111 | Field protection program pointer |

| | |
|---|---|
| 1000 | Directory protection description |
| 1001 | Directory protection program pointer |

Security Classification

**DOCUMENT CONTROL DATA - R & D**

*(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)*

| | |
|---|---|
| 1. ORIGINATING ACTIVITY *(Corporate author)* The Moore School of Electrical Engineering University of Pennsylvania Philadelphia, Pa. 19104 | 2a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED |
| | 2b. GROUP |

3. REPORT TITLE

THE FILE SEARCHING, RECORD VALIDATING AND RECORD FORMATTING FUNCTIONS OF THE SUPERVISOR FOR AN EXTENDED DATA MANAGEMENT FACILITY

4. DESCRIPTIVE NOTES *(Type of report and inclusive dates)*

Technical Report

5. AUTHOR(S) *(First name, middle initial, last name)*

Agu Raymond Ets

| | | |
|---|---|---|
| 6. REPORT DATE August 1970 | 7a. TOTAL NO. OF PAGES 106 | 7b. NO. OF REFS 11 |
| 8a. CONTRACT OR GRANT NO N00014-67-A-0216-0014 | 9a. ORIGINATOR'S REPORT NUMBER(S) Moore School Report #71-04 | |
| b. PROJECT NO. NR 049-153 | | |
| c. | 9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)* None | |
| d. | | |

10. DISTRIBUTION STATEMENT

Reproduction in whole or in part is permitted for any purpose of the United States Government.

| | |
|---|---|
| 11. SUPPLEMENTARY NOTES None | 12. SPONSORING MILITARY ACTIVITY Office of Naval Research Washington, D.C. |

13. ABSTRACT

The purpose of the Supervisor in an Extended Data Management Facility (EDMF) is to direct the Facility's handling of a user's request for service. The Supervisor employs the five main functions of Access Controlling, Retrieval Optimizing, File Searching, Record Validating and Record Formatting in order to accomplish its task. This report is concerned mainly with the design and implementation of the File Searching and Record Validating Functions, although it also covers the Record Formatting Function. The File Searching and Record Validating Functions form that part of the Supervisor which actually controls the retrieval of records from the files of the EDMF. The major part of the report is concerned with discussing the File Searching Function because of the novel feature which has been implemented. This feature is the parallel processing of record lists in a generalized file structure, which eliminates redundant retrievals while at the same time reducing the access time of the device on which the records are stored. The Record Validating Function checks the record for compliance with the user's request and verifies the user's authority to access the record. A validated record is then subject to the Record Formatting Function which outputs it to the user.

Security Classification

| 14 KEY WORDS | LINK A | | LINK B | | LINK C | |
|---|---|---|---|---|---|---|
| | ROLE | WT | ROLE | WT | ROLE | WT |
| Access control | | | | | | |
| File searching | | | | | | |
| Data management facility | | | | | | |
| Record validation | | | | | | |
| Record formatting | | | | | | |
| Retrieval optimization | | | | | | |
| Inverted file structure | | | | | | |
| Index-sequential file structure | | | | | | |
| Generalized file structure | | | | | | |
| Multilist | | | | | | |
| Time shared system | | | | | | |
| Key word listing | | | | | | |
| Parallel processing | | | | | | |
| List processing | | | | | | |
| File searching | | | | | | |
| Prime key word search | | | | | | |
| Security checking | | | | | | |
| User oriented systems | | | | | | |
| Query language | | | | | | |
| Directory protection | | | | | | |
| Output format | | | | | | |
| Core format | | | | | | |
| Boolean expressions | | | | | | |