# AN INVESTIGATION TO STUDY THE FEASIBILITY OF ON-LINE BIBLIOGRAPHIC INFORMATION RETRIEVAL SYSTEM USING AN APP

by

RANA DATTAGUPTA

supervised by

R. M. LEA

A thesis submitted for the degree of Doctor of Philosophy at Brunel University, Uxbridge.

July 1977

# CONTENTS

## ABSTRACT

This thesis reports an investigation on the feasibility study of a searching mechanism using an APP suitable for an on-line bibliographic retrieval.operation, especially for retrospective searches.

From the study of the searching methods used in the conventional systems it is seen that elaborate file- and data- structures are introduced to improve the response time of the system. These consequently lead to software and hardware redundancies. To mask these complexities of the system an expensive computer with higher capabilities and more powerful instruction set is commonly used. Thus the service of the system becomes cost-ineffective.

On the other hand the primitive operations of a searching mechanism, such as, association, domain selection, intersection and unions, are the intrinsic features of an associative parallel processor. Therefore it is important to establish the feasibility of an APP as a cost-effective searching mechanism.

In this thesis a searching mechanism using an 'ON-THE-FLY' searching technique has been proposed. The parallel search unit uses a Byte-oriented VRL-APP for efficient character string processing.

At the time of undertaking this work the specification for neither the retrieval systems nor the BO-VRL-APP's were well established; hence a two-phase investigation was originated. In the Phase I of the work a

bottom up approach was adopted to derive a formal and precise specification for the BO-VRL-APP. During the Phase II of the work a top-down approach was opted for the implementation of the searching mechanism.

An experimental research vehicle has been developed to establish the feasibility of an APP as a cost-effective searching mechanism. Although rigorous proof of the feasibility has not been obtained, the thesis establishes that the APP is well suited for on-line bibligraphic information retrieval operations where substring searches including boolean selection and threshold weights are efficiently supported.

# CHAPTER I

# INTRODUCTION

Information is collection of knowledge. This collection of knowledge is used in directing the further advancement and organization of knowledge. The rapid growth of science and technology has led naturally to a corresponding growth of knowledge. The enormous size and complexity of information from various sources has reached the point where alarm is felt in regard to the potential loss of knowledge due to increasing difficulties of retrieving it. This may lead to unnecessary and expensive duplication of research and consequent stultification of research and development. To the individual scientist, the main problem is to find out the documents which contain useful information and to obtain copies of these. The growing importance of information accessibility has recently drawn fresh attention to this problem. Thus the need for an automated solution to this problem is now well appreciated.

In general, an information retrieval system[1-11] can be used for:

1. Data retrieval
2. Fact retrieval
3. Document retrieval

Data retrieval:- A set of data is retrieved in response to a query for helping managers to produce reports, statistics and future projection for day to day decision making problems.

Fact retrieval:- The function of this is similar to an encyclopedia or engineerin handbook which provides an answer to a simple question.

Document retrieval:- This is generally a two-step operation. In the first step a set of documents is located. In the subsequent operation the desired documents are retrieved from the storage.

The present discussion is mainly aimed at the library bibliographic information retrieval system, the function of which is similar to the document retrieval operation. The performance of such a retrieval system in regard to the total number of documents retrieved should be flexible enough to carry out specific and exhaustive searching. For example a user may intend to get only a few references in response to his query - that is, he may be interested in the specific retrieval. On the other hand a researcher would like to perform an exhaustive search on the data-base to retrieve all documents related to his area of interest. Such an exhaustive search may be of two types. These are:-

1. retrospective search
2. current awareness service

The retrospective search informs the user of all past works which are relevant to his particular field of research.

The current awareness service allows a user to keep abreast with the currently produced literature.

In the current awareness service an immediate response is generally not required. Hence a number of queries can be grouped together to form a batch of profiles. A sequential search on the current collection of the data-base is then carried out with the batch of profiles to generate respective outputs since this is a regular service, the contents of the output of each user can be monitored and there is an opportunity of refining user's query to get maximum relevance.

The retrospective search however, is carried out only once. Hence, an online retrieval operation is desirable for retrospective search. Since the response of the system is immediate and the profile can be interactively modified, the best result is obtained. Another feature of on-line retrieval system is 'browsing'. This allows a vaguely defined user's query to gain precision by interactive refinement of the profile.

Commonly, in an information storage and retrieval system a user specifies his need for some facts by selecting a set of search keys. The collection of documents which is stored in the data-base, is also assigned with similar keys. The retrieval of information is then carried out by a simple association of the users and documents Keys, which is essentially an associative process. This primitive operation of association is not an intrinsic feature of a conventional computing system. In such a system elaborate techniques are employed to create an artificial association. This increases the complexities of the data-structure and computation of the retrieval system. Hence to provide an acceptable grade of service, a computing system of higher performance, which masks these complex operations by faster and more expensive hardware, is used. Consequently the cost of retrieval service becomes expensive enough to deter many potential users from availing this facility.

To simplify the function of an information retrieval system, it can be divided into three major phases of operations. These are:

1. Indexing:- Assigning a set of Keywords to a document and storing it in the data-base. These keywords classify a document according to the subject matter of its contents; so that in future the documents can be accessed by the subject index.

2. Query or user's profile:- A user wishing to retrieve a set of documents should express his information need by specifying a query. A user's query

or profile consists of a set of Keywords describing a field of specialization.

3. Search:- In this phase of the operation[11-17] the data-base is searched for the selection criterion as specified in the user's profile. When all relevant documents satisfying the search criterion are located, they can be physically retrieved from the storage.

The first two operations (indexing and query formulation) are in the domain of information science and still require a lot of human decision-making capabilities. The scope of the present discussion is limited to the search phase of the information retrieval system.

The simplest method of the searching operation is to scan the entire document file from the beginning to the end to find out the occurrance of the search-key. A noticeable improvement of the required number of comparisons for a search operation is obtained by simple ordering of the Keys of the document file. Further improvement can be achieved by a tree structure, such as, binary or multiway tree. The number of key-comparisons involved in a tree-structured file then depends on the total number of entries. More improvement can be achieved with hash-coding, where the number of comparison is independent of the size of the data-base, but it may involve many computations. As the complexity increases, it may be required to locate a document by more than one key or by cross-references. To facilitate this, an inverted file is often employed. The performances of an inverted file system are functions of the datastructure employed. Each of these has its relative merits and disadvantages. In general, for fast retrieval operation the data-base of an information retrieval system, using conventional computers, should be highly structured. Then it is likely that the updating will be more difficult.

On the other hand a computer system which embodies association as a primitive operation may be efficiently used for an information retrieval system[47-54, 74-81, 151-159].
In the simplest form of an associative retrieval system, the data-base is stored in a content-addressable memory and a parallel search is then carried out to locate the desired Keys. In practice a large associative memory array suitable for storing a reasonable size of data-base is difficult to produce.
Alternatively two other techniques can be adopted, where:

1. A part of the data base is held in the associative memory.
2. The search data (user's profile) is held in the associative memory (ON THE FLY)

The first organisation has the disadvantage of continued loading of the associative memory, but it allows more complex manipulations to be carried out. Although the converse of these advantages and disadvantages is true for the second method, it has the primary advantage of cost. As a keyword or record may contain a variable number of characters, a provision for incorporating this feature should also be included in the system. This type of data-organization is well supported by a byte-orientated variable record length associative parallel processor (BO-VRL-APP). Hence, for the current investigation 'ON THE FLY' search technique using a BO-VRL-APP is chosen. In the 'on the fly' technique, records containing indices are passed over the top of a 'parrallel search' unit to filter-out the relevant documents. This process is continued until the end-of-the file is encountered.

The major advantages of an associative retrieval system[151] are:

1. Minimal data-structuring
2. More efficient searching
3. More efficient updating
4. Improved flexibility
5. Minimal storage redundancy

The basic objective of the present investigation is to evaluate these indications and prove that APP can support efficient and flexible keyword searching.

Unfortunately, until now, research in neither information science nor associative parallel processor is so established that it can provide an exact system specification. However, a top-down design of information system and bottom up development of APP system is considered to be the most sensible approach to encounter the lack of information in these fields. Moreover to exploit the full capabilities of the hardware, it is considered that the associative retrieval system should be implemented with low level associative instructions. The other constraint of the present research is the inadequacy of resources. At the beginning of the work, except for a nand-gate implemented associative memory array, no hardware or software facilities to support the development of an associative retrieval system was available at Brunel University. Hence, it was essential to divide the present investigation into two different phases. It was decided that in the first phase of work, an interactive experimental set up would be developed to specify the instruction set for a byte orientated VRL-APP. In the next phase, the results obtained in the first phase would be utilized to implement a research vehicle for an associative retrieval system. The purpose of experiments carried out in this phase would be to develop algorithms for information retrieval operation to demonstrate the flexibility, efficiency and simplicity of a retrieval system based on an associative parallel processor and to compare its performances with its conventional counter parts.

This thesis discusses the problems related to the implementation of research vehicles for associative retrieval system. The proposed system uses an 'ON THE FLY' searching technique utilizing a byte-orientated variable record length associative parallel processor. The purpose of the present investigation is to study and demonstrate feasibility of such a system. The experience gained by this investigation may be the basis of future development of associative retrieval systems.

# CHAPTER 2.

# INFORMATION RETRIEVAL SYSTEM

2.

The user of an information retrieval system[1-11] is concerned to get facts about his query. It is expected that the information centre is capable of catering to the information needs of its users. That is, the collection of the information centre should cover the set of documents required for a group of users. The operation of retrieval of documents from an information center is basically resolving the relevance of documents stored in it with the query. A document can be retrieved by its author's name, title or contents. In the first two cases, it is assumed that the user is informed of the existence of a document designated by these keys. On the other hand, when a user does not have any prior knowledge of the author's name or the title of a document, he can locate the desired document by its contents (subject index). The subject matter contained in a document is indicated by an index term. The index terms classify documents into different subject categories. Indexing is the process of assigning an index term to a document, and is generally carried out according to a predefined rule.

As the retrieval of documents involves matching index terms assigned to a document with the index terms cited in the user's query, it is expected that the user would express his query in a language similar to that used for indexing.

When the user's query is presented to the information centre, the index terms referred to in the query are searched, either manually or by a mechanised device, within the data-base. Once a match between the index terms is found, the relevant documents are retrieved from their physical locations. This concludes a retrieval operation.

Thus, it can be seen that three major operations are involved in an information retrieval system. These are:

1. Indexing
2. Query formulation
3. Searching

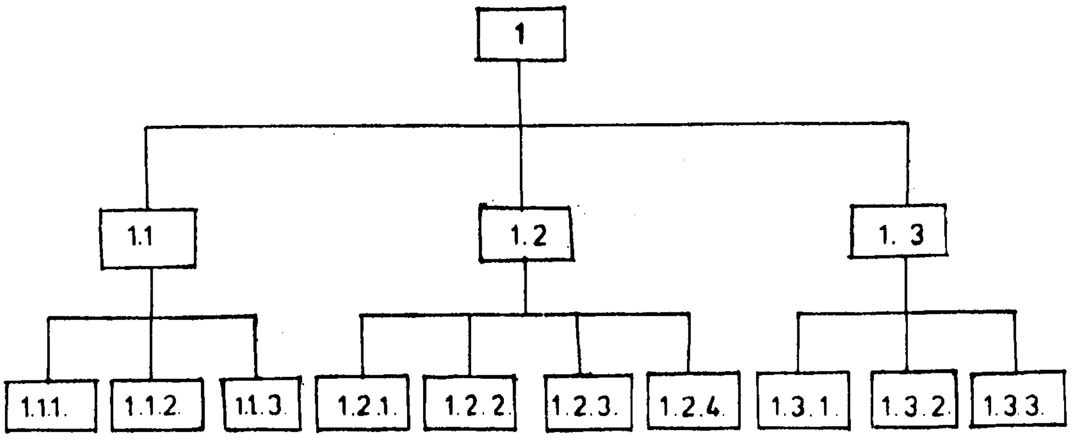
Although these processes look trivial, there are many problems associated with each of these operations. In the following sub-sections 2.1, 2.2 and 2.3 some aspects of these operations are discussed.

## 2.1 INDEXING:-

The process of indexing[1-10] is required to classify a document into a set of subject categories which are contained in a document. Although mechanisation of indexing is possible to a certain degree of success, it still involves to a large extent human intellectual effort. Until now indexing is best done by human indexers. Indexers are responsible for evaluating the relevance of a document to a class of subject categories, to which it fits best. To express an indexer's comment about a document, that is, to assign an index term to it, he needs the help of an indexing language. The choice of an indexing language depends on two criteria; these are:

1. expressiveness
2. unambiguity

There is no doubt that the selection of a natural language for indexing would result in the best expressiveness of the index, but at the same time it would be most ambiguous. One approach to solve the problem of ambiguity is to use a hierarchical indexing[18-20] procedure (see Fig. 2.1(a), (b), (c) ). In this approach an authority list of all possible subject categories is produced. This consists of all generic terms at the first level of the hierarchy, and subsequent detail



a) Tree representation of a Hierarchical index.

| | |
|---|---|
| 600 | Technology (applied science) |
| 620 | Engineering |
| 629 | Other branches of Engineering |
| 629.13 | Aeronautics |
| 629.138 | Uses of Aircraft |
| 629.138 8 | Space of flight. |

b) The Authority list of Dewey's decimal language.

| | | |
|---|---|---|
| Annealing | Vm | Heat Treatment |
| See Also Black Annealing | Vmb | Harriogenization |
| Bright Annealing | Vmd | |

c) An Authority list of a Facet Subject Classification

Fig. 2.1 Hierarchical Indexing

aspects of a generic term are expressed in lower levels of the hierarchy. In order to assign an index term to a document a strict syntactic rule must generally be followed.

One of the most commonly encountered problems in a library is that the majority of its collections use only a few of the approved index terms. This often leads to complex subdivisions of these index terms. Sometimes, it becomes extremely difficult to accommodate a new subject concept within the existing authority list. This problem could be partially solved by a continuous updating of authority list, that is, by including a new index term as soon as the subject is well established. Another difficulty arises when a document under consideration, covers a number of mutually unrelated subjects. In those cases, it is virtually impossible to partition these documents under only one broad category. As a solution to this problem, a number of index terms[21] are attached to the document, each according to corresponding subject concepts. The resulting index terms are then permutated[22] to provide a full index (Fig. 12). This allows equal accessibility of a document when this document is intended to be retrieved from any subject's point of view. In many cases it is observed that the complete permutations of index terms are not essential, even then, this increases the size of the index to a large extent. This is particularly true when the number of index terms exceeds two.

The index thus obtained for a set of documents is co-ordinated[23] during the time of indexing. Hence this cannot be changed during the searching phase. This type of indexing is called 'pre-co-ordinated' system. In the other type of indexing, 'post co-ordinated'[24] system, correlations of classes of documents are done during searching time. This leads to a flexible indexing system, as the entire mode of classification could be modified by the user of the system. Here an indexer is allowed to assign any number of index terms, called 'keyword's' to a document, which he thinks are relevant to it. The final co-ordination of these keywords is done by logical interconnections among these keywords. A major difficulty arises here due to free

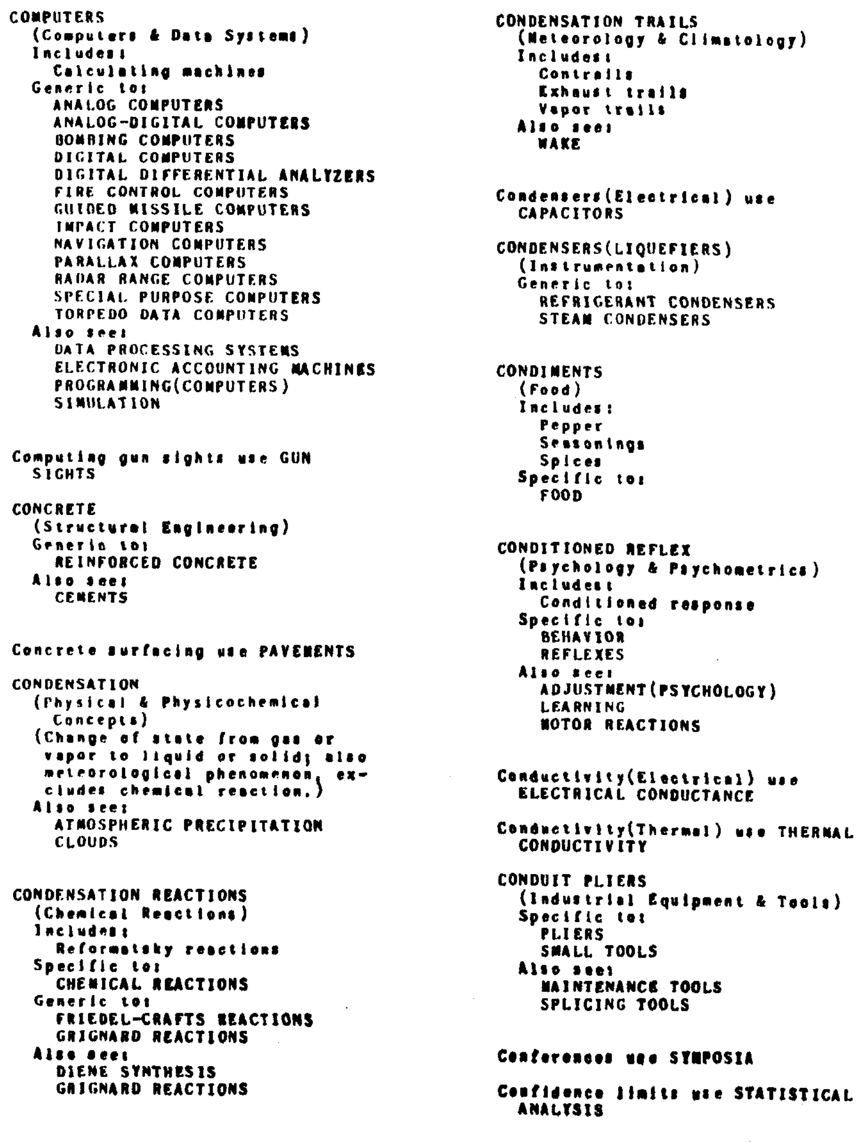
621.762 : 546 . 65
546.65 : 621 . 762

a) Colon Classification

|  |  |  |
|---|---|---|
| Kgb | Bqt | Ac |
| Bqt | Ac | Kgb |
| Ac | Kgb | Bqt |
| Kgb | Ac | Bqt |
| Bqt | Kgb | Ac |
| Ac | Bqt | Kgb |

b)

Fig. 2.2. Permutated Indexing



```
COMPUTERS                                   CONDENSATION TRAILS
  (Computers & Data Systems)                  (Meteorology & Climatology)
  Includes:                                   Includes:
    Calculating machines                        Contrails
  Generic to:                                   Exhaust trails
    ANALOG COMPUTERS                            Vapor trails
    ANALOG-DIGITAL COMPUTERS                  Also see:
    BOMBING COMPUTERS                           WAKE
    DIGITAL COMPUTERS
    DIGITAL DIFFERENTIAL ANALYZERS
    FIRE CONTROL COMPUTERS                  Condensers(Electrical) use
    GUIDED MISSILE COMPUTERS                  CAPACITORS
    IMPACT COMPUTERS
    NAVIGATION COMPUTERS                    CONDENSERS(LIQUEFIERS)
    PARALLAX COMPUTERS                        (Instrumentation)
    RADAR RANGE COMPUTERS                     Generic to:
    SPECIAL PURPOSE COMPUTERS                   REFRIGERANT CONDENSERS
    TORPEDO DATA COMPUTERS                      STEAM CONDENSERS
  Also see:
    DATA PROCESSING SYSTEMS
    ELECTRONIC ACCOUNTING MACHINES          CONDIMENTS
    PROGRAMMING(COMPUTERS)                    (Food)
    SIMULATION                                Includes:
                                                Pepper
                                                Seasonings
Computing gun sights use GUN                    Spices
  SIGHTS                                      Specific to:
                                                FOOD
CONCRETE
  (Structural Engineering)
  Generic to:                               CONDITIONED REFLEX
    REINFORCED CONCRETE                       (Psychology & Psychometrics)
  Also see:                                   Includes:
    CEMENTS                                     Conditioned response
                                              Specific to:
                                                BEHAVIOR
Concrete surfacing use PAVEMENTS                REFLEXES
                                              Also see:
CONDENSATION                                    ADJUSTMENT(PSYCHOLOGY)
  (Physical & Physicochemical                   LEARNING
    Concepts)                                   MOTOR REACTIONS
  (Change of state from gas or
   vapor to liquid or solid; also
   meteorological phenomenon, ex-           Conductivity(Electrical) use
   cludes chemical reaction.)                 ELECTRICAL CONDUCTANCE
  Also see:
    ATMOSPHERIC PRECIPITATION               Conductivity(Thermal) use THERMAL
    CLOUDS                                    CONDUCTIVITY

                                            CONDUIT PLIERS
CONDENSATION REACTIONS                        (Industrial Equipment & Tools)
  (Chemical Reactions)                        Specific to:
  Includes:                                     PLIERS
    Reformatsky reactions                       SMALL TOOLS
  Specific to:                                Also see:
    CHEMICAL REACTIONS                          MAINTENANCE TOOLS
  Generic to:                                   SPLICING TOOLS
    FRIEDEL-CRAFTS REACTIONS
    GRIGNARD REACTIONS
  Also see:                                 Conferences use SYMPOSIA
    DIENE SYNTHESIS
    GRIGNARD REACTIONS                      Confidence limits use STATISTICAL
                                              ANALYSIS
```

FIG. 2.3. Sample from thesaurus.

selection of keywords vocabularies. A document, depending on time and mood of indexers, may be indexed quite differently. This requires a dictionary to control free selection of keywords. Even though a single document is indexed by more than one synonym , a cross reference within the dictionary is created to cover all identical ideas, so that there would be little problem in retrieving the intended documents. As no rigid syntax exists in the simple form of post-co-ordinated indexing system, there is a likelihood of losing the relevance or semantic information[25] of the keywords. This is obvious especially in cases of homographs and the words which have varied implications in different subject concepts. Hence a large number of false co-ordination is expected in this simple system. To solve this problem,links, and role indicators[26]( a syntactic device) are incorporated within the system. This again leads to a different kind of authority list called 'thesaurus'[27](Fig.2.3). In a well designed information retrieval system a combination of both pre- and post- co-ordinations are used to improve unambiguity and expressiveness of the indexing language.

Several attempts have been made to mechanise the process of indexing. In the simplest form of a machine generated indexing process, the technique has been used generally on the basis of the title of bibliographic items. The computer initially ignores all syntactical words from the title. The remaining words of the title are selected as index terms. The result of machine manipulation is an index of keywords printed in alphabetical sequence, together with text immediately surrounding each term. This is called 'keywords In Context (KWIC)[28-31]Indexing' (see Fig.2.4). The success of this method of indexing is totally dependent on the descriptive quality of the titles. To improve reliability of machine indexing, a variation on the KWIC index is attempted. In this type of indexing, KWOC (Keywords Out of Context) index[32,33] index terms are selected from the entire content of a document and presented along with the title of the article. The selection of keywords from the text depends on the statistical frequency of occurrence of a word or the relative frequency of co-occurrence of some words and on linguistic and textual

Pulse, Digital and Switching Waveforms

| | |
|---|---|
| Pulse, | Digital, and switching waveforms |
| Pulse, Digital | and switching waveforms |
| Pulse, Digital and switching | waveforms |
| Pulse, Digital and Switching waveforms | |

(a) Selection of Keywords.

| | |
|---|---|
| Pulse, Digital | and switching waveforms |
| Pulse, | Digital and switching waveforms |
| Pulse, Digital and switching | waveforms |
| Pulse, Digital and switching waveforms | |

(b) Final Index

Fig. 2.4 Key Word In Context.

Header (ID, Priority, File Access Key)

Command (Retrieve, Update, Report ...)

Output Device (Typewriter, Display, Lineprinter)

| | |
|---|---|
| Keys | Processing (A function of K1, K2 ...) |
| K 1 (Key Name/Value) | . Logic functions of Key |
| K2 " | Inter-record processing |

Output FORMAT

- Titles
- Print format

At present, a suitable means for scanning text directly from printed documents does not exist, and hence this method of indexing is expensive. Thus mechanisation of indexing, until now, is only of theoretical interest. And this domain of the problem is still left to the human intellect and decision-making capabilities.

## 2.2. QUERY:

The user of an information retrieval system seeks some facts regarding his query. He communicates his demand for certain types of information with the information centre through a query.[1-11] It has been discussed earlier that the retrieval of documents is a process of matching a user query with a document file. And the formation of the document file is carried out by indexing. Hence to match the vocabularies and the syntax of the user's query with the keywords of the document file, every effort should be made so that the terms used in the query have a one-to-one correspondence with the indexed record. A user generally does not have any prior knowledge of the vocabularies and the syntax of the indexing language. To expose him to the environment of the system, there is a need for a dictionary[2,3]. Consultations with such a dictionary allows a user not only to correct errors within the query, but also to inform him whether or not the terms used in the query are included in the indexed records. When a user discovers that a keyword used in the query is not present in the document file, he could use other synonyms. Moreover, he could extend the coverage of the query by selecting a set of synonyms and the relevant terms. These relevant terms could have been selected on a statistical analysis of the co-occurrence of a set of keywords.

The other aspect of the dictionary, hierarchical relationships of index terms, allows browsing for a vaguely defined query. This also enables users to modify their query for searching a document file with a desired precision ranging from an exhaustive to specific retrieval of documents.

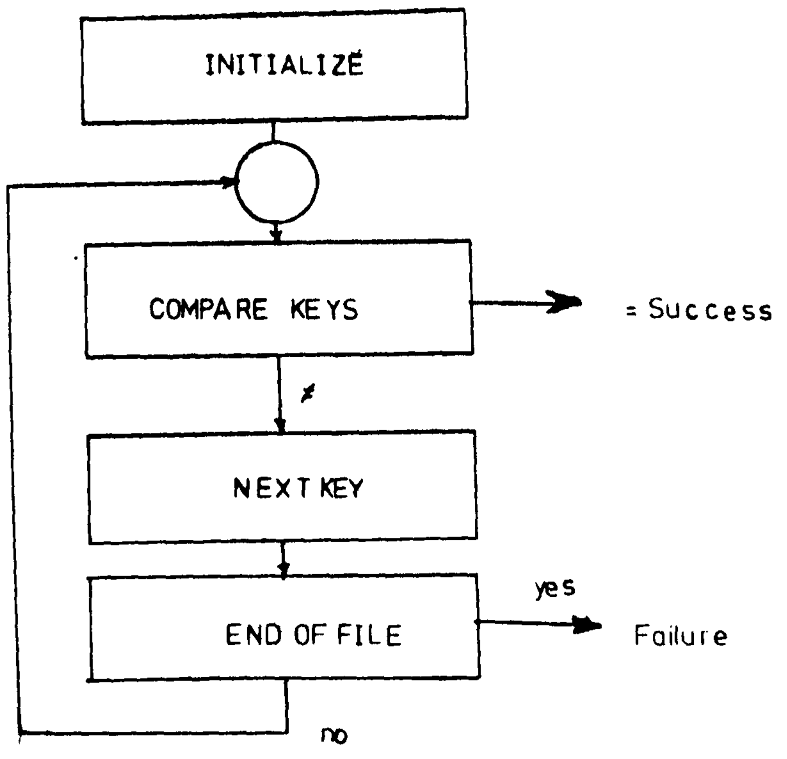
In addition to formulating a searching strategy, a query may also incorporate some control instructions[11]. These could include: user's name and priority, allocations of output device, commands (retrieve, update/delete) and instructions for report generations. An example of such a query is shown in Fig.2.5. The generation of the report may involve some inter-record processing within the retrieved records. On the basis of the result of these operations a part of the retrieved documents are selected for output. These selected documents are then sequenced in a desired order and presented to the output device according to a specified output format. It is expected that on-line updating and report-generation would receive more attention in future from both system designers and users.

## 2.3 SEARCHING

### 2.3.1 Searching on primary keys:

The retrieval of information from a data-base is a process of locating[11-17] documents which are relevant to the query. In the simplest form, it could be considered that all documents are identified by a unique indicator, called a 'Key', and no keys within the document file are duplicated. It is also assumed that the user's query is represented by a single key. In this case, the searching would involve matching the keys in the query and the document file. This process would begin at the beginning of the document file and continue until either the key under search has been detected or the entire data file has been scanned. In the first case, the result of the search is a success and in the other case, it is a failure. The flow-chart for such a searching strategy is shown in Fig.2.6 and it is referred to as a sequential search on unordered file.[12-14, 34-36]

Equations[12,13] 2.1 and 2.2 give the average number of comparisons involved in a successful and unsuccessful Key-searches respectively,



FI G.2 6. Sequential search on unordered file

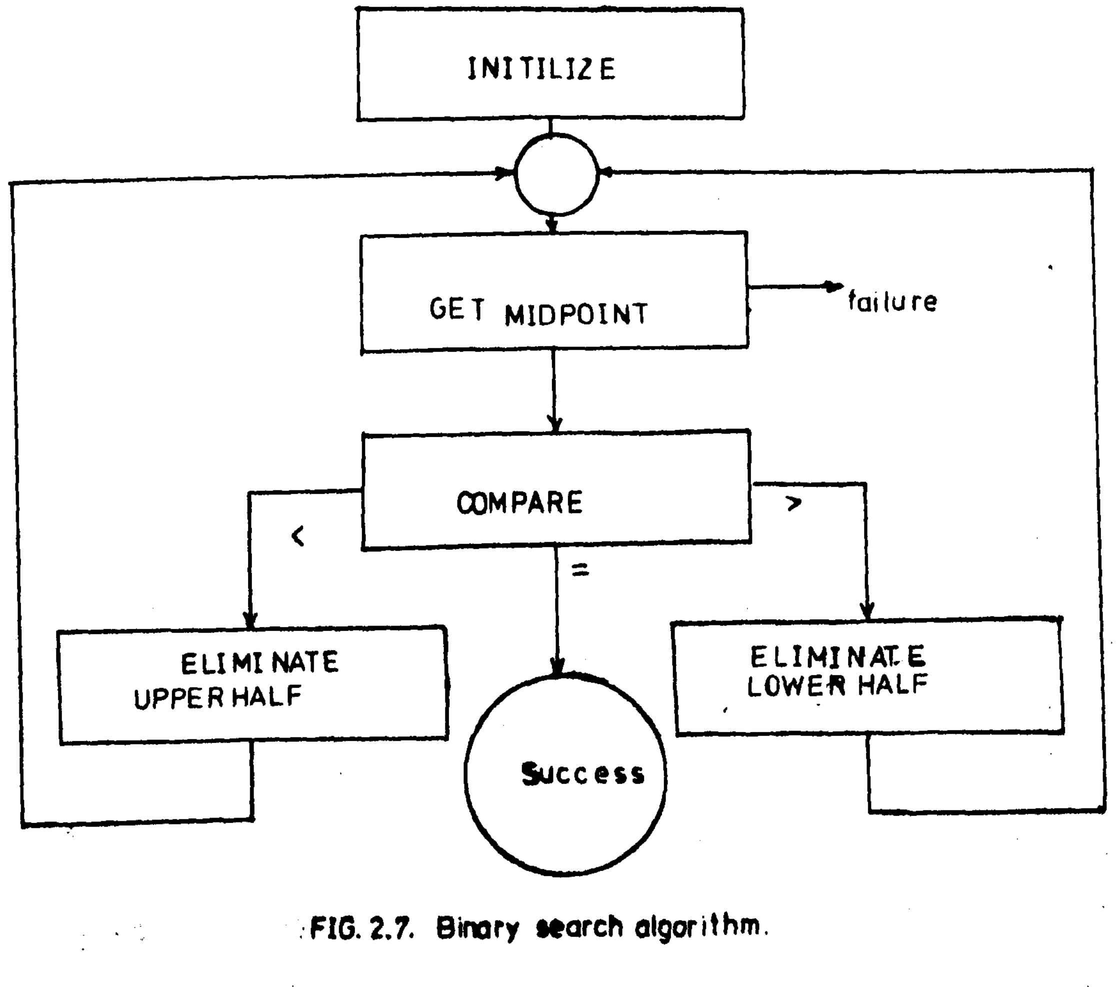


FIG. 2.7. Binary search algorithm.

for success;

$$C_s = \frac{N+1}{2} \qquad \text{..........} \qquad 2.1$$

for failure:

$$C_f = N \qquad \text{..........} \qquad 2.2$$

where N = total number of keys in the data file.

2.3.1.1 Sequential search on Ordered File :

It is well understood[12-14,34,35] that the problem of key searching becomes simpler when the document file is ordered in a pre defined sequence. The simplest form of ordering could be achieved by sorting all Keys of the document file in ascending order of their numerical values.

The matching of Keys starts at the beginning of the document file, but it terminates either when a Key has been found or when a currently compared Key is numerically greater than the search Key.

Following equations[12,13] 2.3, 2.4 and 2.5 give the number of comparisons involved during searching a Key on an ordered file.

For a successful search, the number of comparisons performed depends on the position of the key in the document file. Therefore if $K_1$ is to be located in an ordered file, then

$$C_{s1} = 1 \qquad \text{..........} \qquad 2.3$$

number of comparisons is to be performed before locating the Key '$K_i$'. Here it is assumed that none of these keys are duplicated in the data file.

On the average the number of comparisons performed per successful Key is:

$$Cs = \frac{N+1}{2} \qquad \text{..........2.4}$$

Similarly the average number of the Keys to be compared during an unsuccessful search is:

$$C_f = \frac{N}{2} + 1 \qquad \text{..........2.5}$$

where N = Total number of Keys in the data file.

One of the advantages of this method is quick termination of scanning for the Keys which are not included in the document file. The other significant improvement is achieved when more than one key are simultaneously searched. For example, it is assumed that the total number of Keys in the data file are N; and the number of Keys to be searched are m. Then in the case of sequential search on unordered file the total number of comparisons[13] 'P' would be

$$P = mN \qquad \text{.......... 2.6}$$

But when both the index and profile (query) are ordered in the same way the number of comparison[13] 'P' reduces to

$$P = N \qquad \text{.......... 2.7}$$

An 'm' times improvement in searching speed is hence obtained.

There are some other alternative ways of ordering[4,12,36] an index file. These are based on special properties of the information need of the users, and are called 'self organizing' files. In some cases where the frequency of access of all Keys are known, the file could be organised in such a way that the keys which are more likely to be referred to are placed near the beginning of the file. In the other case, where current information is more likely to be referred to, the file is arranged in such a way that all new entries are inserted at the beginning of the index file.

## 2.3.2. Searching By Comparison of Keys:

It has been seen in the case of sequential search on ordered file that, although there is an improvement in aborting a search for a non-existing Key, it does not benefit a Key which is present in the document file. However, this problem of sequential searching on ordered file could be rectified by comparing Keys instead of matching. In this approach a Key from the document file is compared with the search Key. If as a result of this comparison, it is found that the key on the document file is numerically greater than the search Key, all Keys beyond that key on the document files are eliminated. The comparison is then continued with the rest of the Keys. Repeating this process of elimination, either the search Key would be located in the document file or it would be terminated when further elimination is not possible.

### 2.3.2.1 BINARY SEARCH :

The binary search[12-14, 37] is the simplest form of implementation of the above mentioned searching strategy. In this method, the comparison of the keys starts at the middle of the document file. If this matches the search Key, the desired record is found. Otherwise, depending on the result of the comparison, one half of the Keys in the document file are eliminated, and the next comparison is made with the Key, which is situated at the middle of the remaining half of the document file. This process is repeated until the desired key is found.

The average number of comparisons 'Ca' is given by the equation[12,13] 2.8

$$Ca = \lceil \log_2 N - 1 \qquad \text{..........} \quad 2.8$$

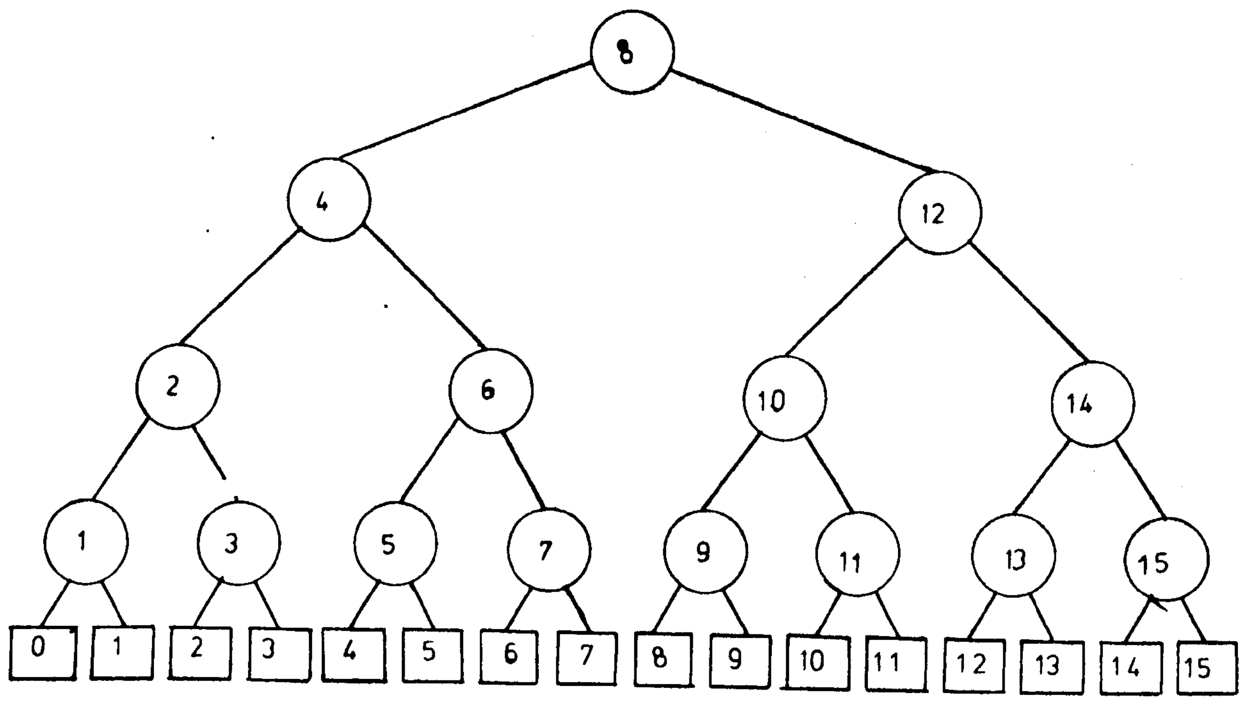


FIG. 2.8.Binary decision tree.

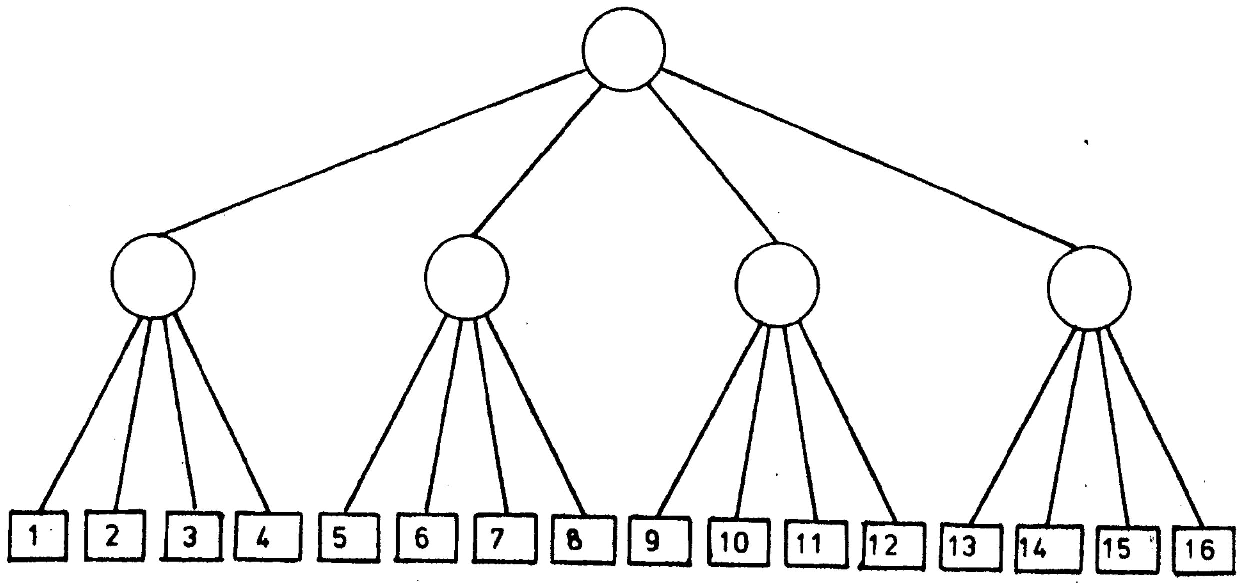


FIG. 2.9. Multiway tree.

The maximum number of comparisons[12,13] 'Cm' required to establish the non-occurrence of a search key is

$$Cm = \lceil \log_2 N + 1 \qquad \ldots\ldots\ldots\ldots \quad 2.8(a)$$

Where N = total number of Keys in the data file.

$\lceil x$ is the next higher integer when the value of x is a fraction.

The simplified flow-chart for the binary search method is shown in Fig.2.7. It does not explain how the probing is terminated during an unsuccessful search operation. The structure of data organisation in a binary search file is shown in Fig 2.8. It looks like a binary decision tree. Each of the nodes and leaves are represented in this figure by circles and squares respectively.

## 2.3.2.2 Multiway Tree Search:

The binary search method is very useful when all index Keys are stored in a fast random access storage. But as the number of index Keys increases, it becomes impractical to store all of these Keys simultaneously in the core memory. In this situation index keys are stored in some direct access devices such as a disc or a drum. Now if binary search technique is applied, it would require a large number of probes - depending on the number of levels of the tree - into the direct access devices. During each probe, it would have to wait for a long access time of the device. This access time problem could be solved by reducing the number of levels in a tree, that is, by increasing the number of branches at each node of the tree as shown in Fig. 2.9. This file structure is called multiway-tree[11,12,38] structure. At each level of the tree the appropriate branching could be selected by either sequential or binary search technique.

For an 'm' way tree, the maximum number of levels of the tree 'n' is given by

$$n = \lceil ( \log_m N) \qquad \text{..........} \quad 2.9$$

The number of comparisons[12,13] 'c' is given by

$$C = W ( \lceil \log_m N ) \qquad \text{..........} \quad 2.10$$

Where N = total number of Keys

W = number of comparisons required to search each level of the tree.

## 2.3.2.3 Indexed Sequential Search:

The advantages of both sequential and direct access to records in a file can be achieved in an indexed sequential file organisation[35,36]. This file organisation (see Fig 2.10) comprises of two files, index and record file, and these two files are arranged in sequential order. Each index contains the address of a record in the file. Thus a record can be directly accessed by locating its index without reading the entire file. On the other hand it can also be accessed by the sequential search on the record file.

This file organisation is well suited for storing in the direct access storage device (disc) where a three-level tree[39] for index decoding can be adopted (see Fig 2.10). The first level of branching determines the cylinder address of the disc; the second level determines the appropriate tracks within that cylinder; and finally the third level contains the records.

Due to the sequential arrangement of the files, some difficulties arise during the update operation. During the insertion of a new record, it is required to

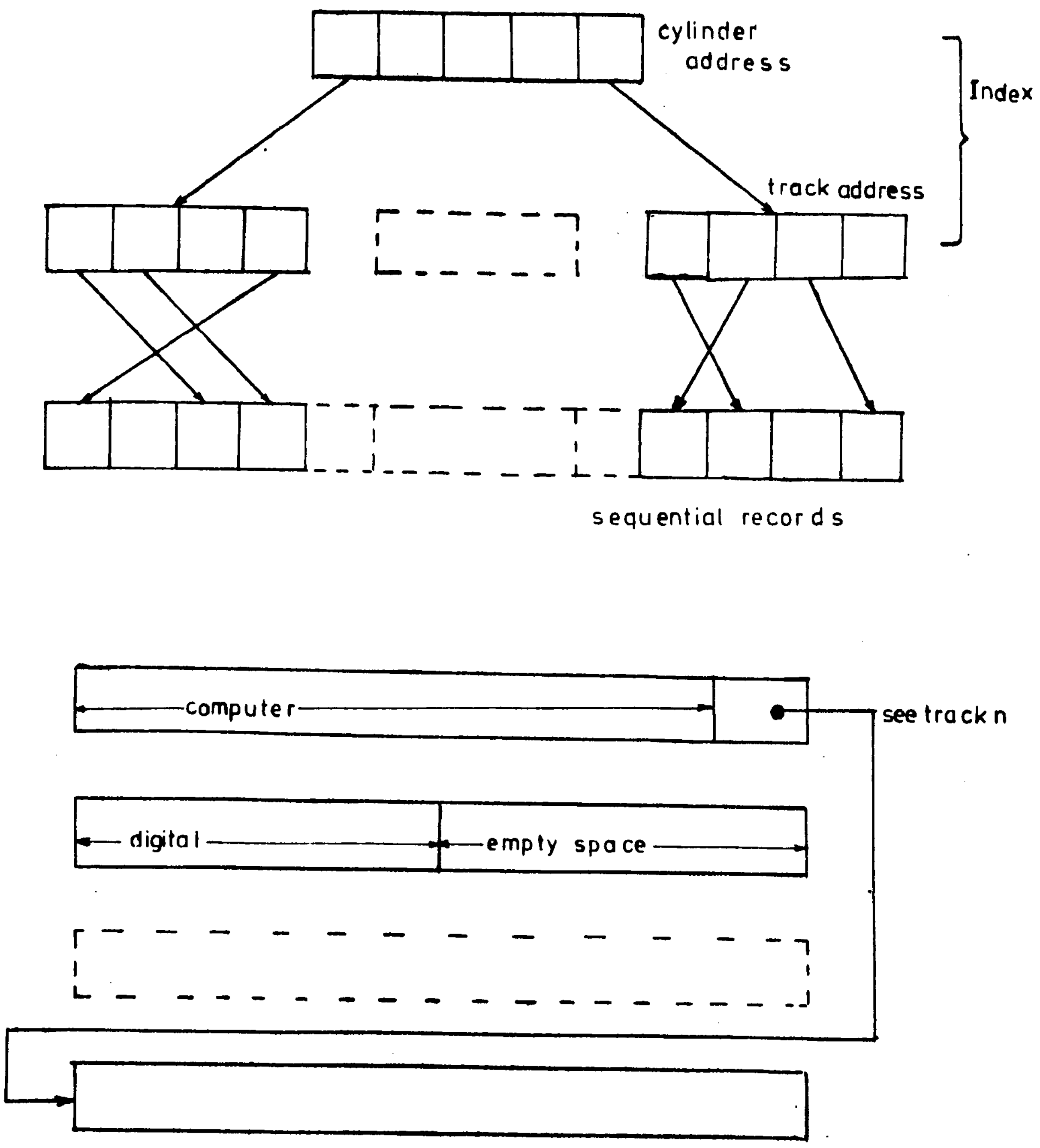


FIG. 2.10. Indexed sequential file

maintain the order of the file which needs to rearrange the entire file. To restrict this rearrangement process within a locality, sufficient space is left empty at the end of each track (bucket). However, when an overflow occurs, a new track can be allocated. In this case, a link address is stored at the end of the old track to point out the new track.

In general a sequential or tree-search technique requires the data base to be in a strictly ordered sequence. A new entry cannot be made unless the correct position for this item has been found. This can be a very time-consuming operation. Thus, with a conventional computer, if the file is structured for fast retrieval operation, it is likely that the updating will be more difficult.

### 2.3.3. Searching by Hash Tables:

The retrieval mechanism so far discussed relies on successive comparisons of search-Keys with index Keys. The number of such comparisons depends on the size of the index file. Thus it reveals that as the size of the data-base increases, more search time would be necessary for retrieving a record from it. Alternatively, a different approach for the storage and retrieval of keys should be adopted, which would make the searching time independent of the size of the data-base. The underlying principle of this method is described below. The problem of information storage and retrieval is to store keys within a specified range of memory. This is then followed by subsequent retrieval of a Key from its storage location. In theory, it is possible to map all non-duplicating Keys to unique locations of memory within a specified range by a suitable transformation rule[11-12,40]. Both storage and retrieval of keys would equally benefit from these transformations. The transformation of Keys to their respective memory locations is called randomizing or hashing (Fig. 2.11) and the corresponding transformation rule is referred to as hash-functions; they generally involve some arithmetic processing and manipulations of Keys. These include

| Keys | Transformed Address |
|---|---|
| JONES | 4 |
| SMITH | 2 |
| BLACK | 1 |
| JOHNSON | 9 |

| Address | Item |
|---|---|
| 1 | BLACK |
| 2 | SMITH |
| 3 | |
| 4 | JONES |
| 5 | |
| 6 | |
| 7 | |
| 8 | |
| 9 | JOHNSON |

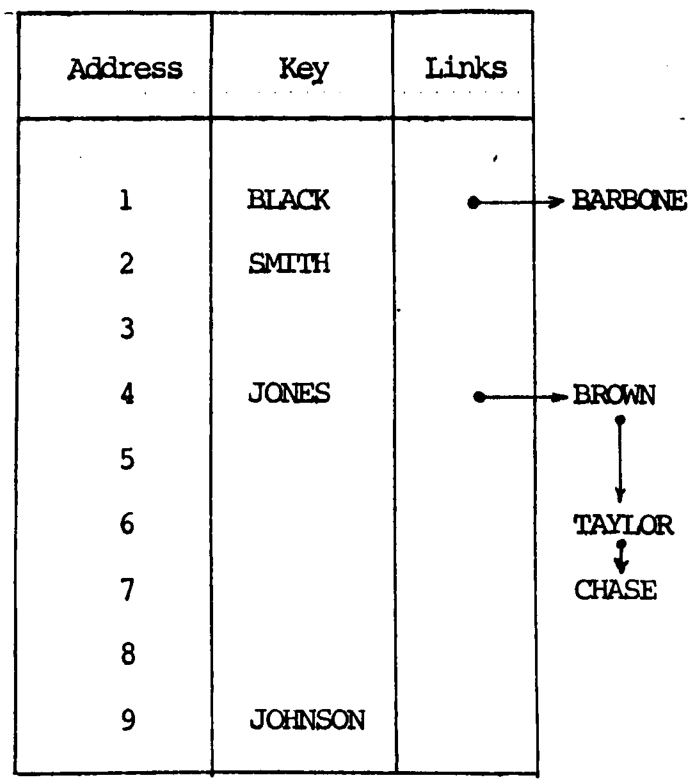
Fig. 2.11 Hash Table

| Keys | Transformed Address |
|---|---|
| JONES | 4 |
| SMITH | 2 |
| BLACK | 1 |
| BROWN | 4 |
| JOHNSON | 9 |
| TAYLOR | 4 |
| BARONE | 1 |
| CHASE | 4 |

(a) List of transferred addresses with synonyms.

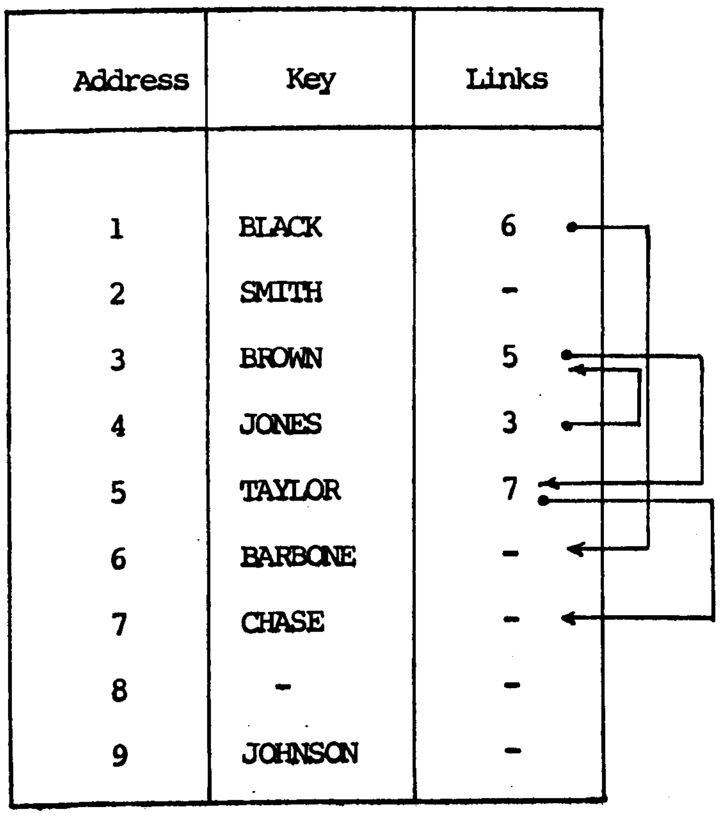
| Address | Key | Links |
|---|---|---|
| 1 | BLACK | - |
| 2 | SMITH | - |
| 3 | - | - |
| 4 | JONES | - |
| 5 | - | - |
| 6 | - | - |
| 7 | - | - |
| 8 | - | - |
| 9 | JOHNSON | - |

(b) File after First Stage.



| Address | Key | Links |
|---|---|---|
| 1 | BLACK | |
| 2 | SMITH | |
| 3 | | |
| 4 | JONES | |
| 5 | | |
| 6 | | |
| 7 | | |
| 8 | | |
| 9 | JOHNSON | |

(c) File after second stage separate chaining



| Address | Key | Links |
|---|---|---|
| 1 | BLACK | 6 |
| 2 | SMITH | - |
| 3 | BROWN | 5 |
| 4 | JONES | 3 |
| 5 | TAYLOR | 7 |
| 6 | BARBONE | - |
| 7 | CHASE | - |
| 8 | - | - |
| 9 | JOHNSON | - |

(d) coalesced chaining.

Fig. 2.12 Collision resolution by chaining.

a) Squaring Keys and selecting the middle portion of data

b) Modulo - division

c) Selecting nearest prime-numbers and many others.

The requirements of an ideal hash-function are:

a) A minimum time for computing hash function.

b) Should produce the unique address for all Keys.

None of the known hash-function could guarantee the uniqueness of transformation for a given set of Keys. Moreover, there is no formal method for selecting a suitable hash-function. Consequently there is a possibility of more than one Key transferred into one address. This is often called a synonym or collision. There are many methods[12,13,40] of resolving synonyms; these are:

1) Chaining method

2) Open addressing method

3) Bucketing method

The first two methods of collision resolution are suitable for internal searching where all keys are stored in the core memory, and the last method is suitable for external searching using direct access storage.

1) <u>Chaining method</u>:

The simple method of collision resolution is that of chaining, where a link field is maintained with each address locations as shown in Fig 2.12. The transformation of Keys are carried out in two stages. During the first stage of operations all non-synonym keys are entered and then the synonyms are entered in the available empty places. Whenever a synonym is entered the link field of the preceding entries are loaded with an appropriate address to point to their successors.

At the search time, a key is first transformed to its hash-address. It is then compared with the content of that location. If a match is found, the search terminates successfully. Otherwise successive links are traced and the contents of each traced locations are compared until a match is hit or the termination of the link is encountered.

2) Open Method:

In this method of collision resolution, a key is first transformed to its normal hash-address. If this location is occupied, a probe to the next location is made until an empty position is found. The new entry is entered in this first empty location. The sequences of these next addresses could be derived in different ways. In the simple version,the next address could be obtained by incrementing the hash-address linearly to form a cyclic probe sequence (Fig 2.13). In the other method,a second hash-function could be applied to resolve synonyms; this is known as open address with double hashing.

At the time of file searching the probing is continued, following an identical address generation rule, to compare keys. This would result either in a success or would end with an empty place, establishing a failure.

3) The Bucketing Method:

When a searching is carried out on Keys, stored in direct access devices, a penalty in time is associated at each re-access. To avoid such situations a number of empty places are allocated to each address of the direct access device to accommodate synonyms (Fig. 2.14). The selection of the size of bucket depends on two criteria.

1) Conservation of storage media

2) Reduction of successive accesses

| Address | Keys |
|---|---|
| 1 | BLACK |
| 2 | SMITH |
| 3 | BARBONE |
| 4 | JONES |
| 5 | BROWN |
| 6 | TAYLOR |
| 7 | CHASE |
| 8 | - |
| 9 | JOHNSON |

Fig. 2.13 Collision resolution by Linear open addressings.

| Address | Keys | | | |
|---|---|---|---|---|
| 1 | BLACK | BARBONE | | |
| 2 | SMITH | | | |
| 3 | | | | |
| 4 | JONES | BROWN | TAYLOR | CHASE |
| 5 | | | | |
| 6 | | | | |
| 7 | | | | |
| 8 | | | | |
| 9 | JOHNSON | | | |

Fig. 2.14 Bucket Method

In general a compromise figure for the size of bucket is accepted. In cases of bucket-overflow a secondary or a tertiary bucket is used by establishing proper link.

The performance of a hash-coded system depends on:

1) <u>Bucket Size</u>

The size of the bucket '$\beta$' is defined as the number of entries (Synonyms) allowed to be transformed to an address.

2) <u>Load factor of the hash-table:</u>

It is the defined as the ratio '$\alpha$' of the number of records entered 'N' to the total number of possible entries 'M' in a hash-table, that is

$$\alpha = N/M$$

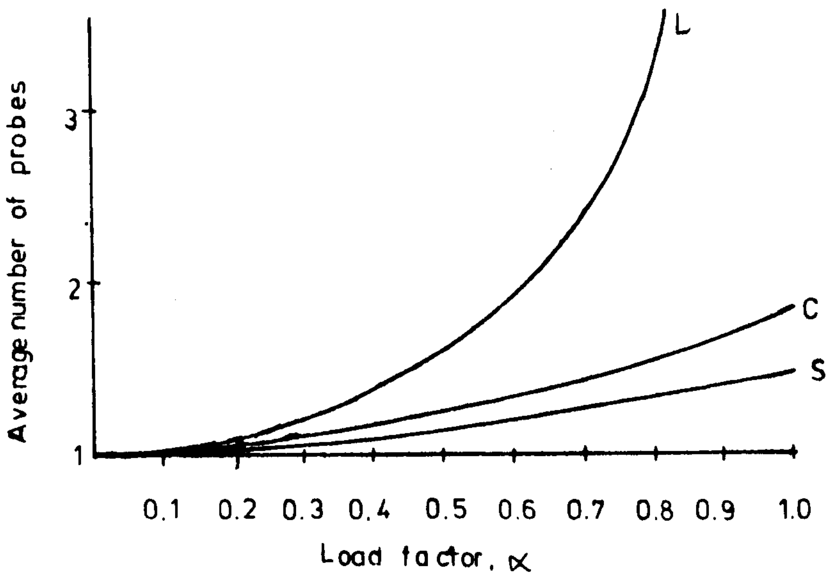
Considering a chain-organised hash-table of bucket size of 1, the average number of probes[13] 'Ca' is given by

$$Ca = 1 + \frac{\alpha}{2} \qquad \ldots\ldots \qquad 2.11$$

when the table is full, that is = 1 the equation 2.11 reduces to

$$Ca = 1.5 \qquad \ldots\ldots \qquad 2.12$$

In the worst case, that is when the table is full, it is found that on the average only 1.5 probes are required to locate any Key. The comparisons of collision resolution methods[12] for both successful and unsuccessful keys areshown in Fig. 2.15 .

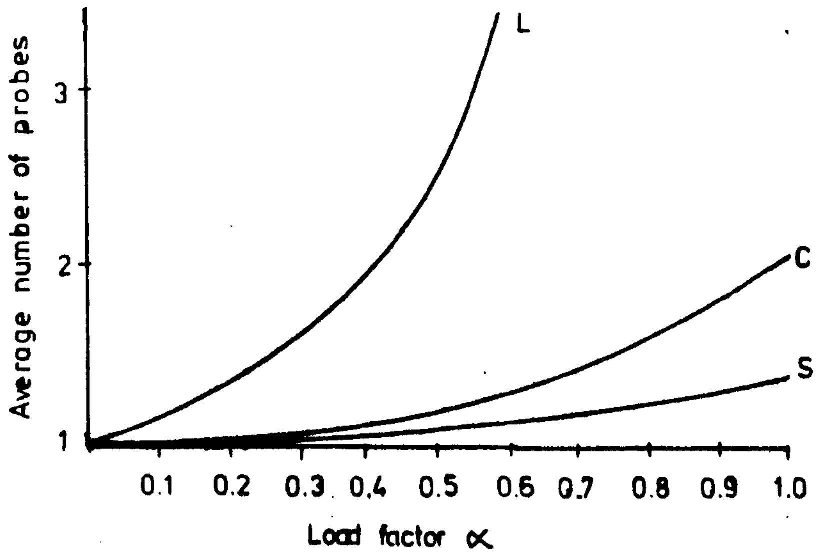


(a) Successful search

L = Linear probing

C = Coalesced chaining

S = Separate chaining



(b) Unsuccessful search

FIG. 2.15. Comparison of collision resolution methods.

It is also seen that the hash table requires less space to store all Keys of the document file. Thus, from these points of view, hash-coding is the most economic method for searching, as well as being the fastest.

On the other hand there are some problems associated with hashing. These are discussed in the following paragraphs.

1) In the case of retrieval of Keys from the hash-table, the search algerithm for an unsuccessful Keyword search terminates when it encounters the first empty place. Hence special care must be taken during deleting an entry from a hash-table. If an empty state is entered in place of a deleted item, the Keys beyond that deleted entry get lost. This problem could be solved by marking the deleted items by a different symbol. This leads to difficulties in maintaining a file where the rate of deletion is high.

2) The searching operation in a hash-table is fast provided it is carried on a precisely defined key. On the other hand, it becomes impossible to retrieve documents by searching Keys, which lie between some limits.

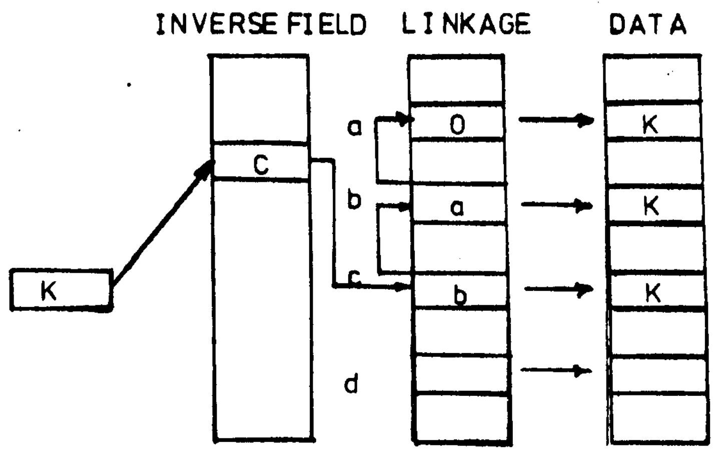
3) It has been stated earlier that the performance of a hash-coded index degrades with increased number of synonyms and load-factor. Hence to maintain the grade of service, these two factors should be monitored. One method of solution would be to allocate a large memory area to hold the hash-table. This leads to an inefficient utilization of storage media. The other method involves rehashing the entire index whenever the performance falls below some acceptable value.

4) Although the average search time for a hash-table is minimum, in a particular case the number of probes required to locate a Key is unpredictable, and it could be large in some cases. In real time application, where it is

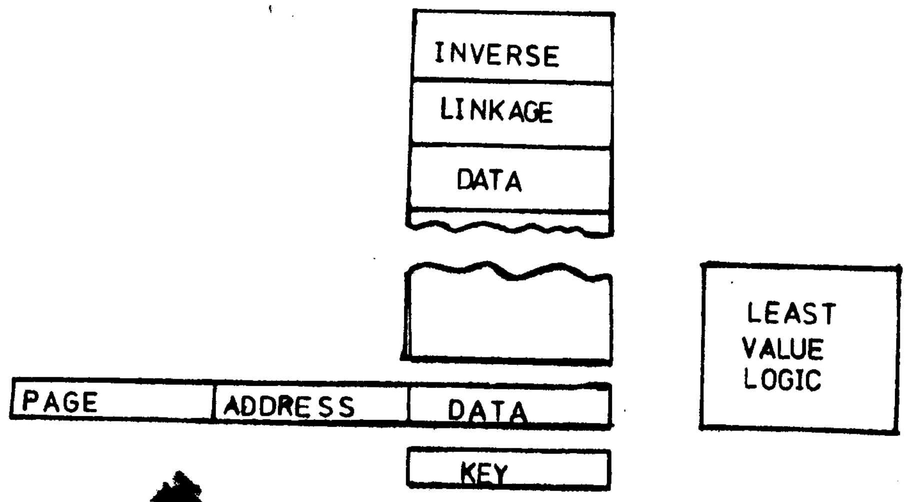
essential to complete a search within a guaranteed time, the performance of the hash-coding scheme becomes adversely affected.

## 2.3.3.1.

A content addressable memory using conventional memory elements has been proposed by Bowden[41]. This is basically a hardware solution to the problem of mapping an 'n' bit code onto $2^n$ locations of a memory space. Where the search keys are divided into smaller sections. Each of these sections is associated with a three-field column. These three fields of a column are inverse field, linkage field and data field (see Fig. 2.16). A section of a search Key (K) directly points to the Kth location of the inverse field. The inverse field points to the most recently entered data in the linkage field. The problem of multiple occurrences of a Key is handled by the linkage field. The linkage field has a one to one correspondence with the data field and contains the addresses of previous occurrences of the Key. The data field contains data which is arranged in any convenient format. In general a search-Key consists of a number of sections. In such a case, all linkage addresses are first obtained from the inverse field of the individual sections. The least linkage value is then evaluated by an external logic unit. The location, as indicated by the 'least value logic', is accessed and compared with the search-key. If the match fails, the next address is evaluated by traversing the links and the process is continued. The main advantages of this method are that any part of the search Key can be masked and the same memory space can be used as both associative and conventional memory. The disadvantages of the method are that the search process is sequential and may require a number of probes which are very much dependent on the data base. Hence the primitive feature of parallelism of associative memory is lost.



Three-field column.



Serial unit.

FIG. 2.16. Associative memory using conventional memory elements.

## 2.4 Searching on Secondary Keys

In the foregoing discussion a simplified document file was considered. Where it was assumed that all records are identified by a single Keyword and no Keywords, within the file, are duplicated. But in practice, as it has been shown in section 2.1, a number of Keywords could be assigned to a record. Moreover, many records could also be indexed by a Keyword. It is desirable that a document should be equally accessible by any of the Keys assigned to it. And it is also desirable that the searching should retrieve all documents which are indexed by the same Keyword.

One method of solving these problems would be to copy each record as many times as the number of Keywords assigned to it, and to order them under each of these Keywords. This not only increases the size of the index, but also imposes problems when a criterion for selecting a set of documents is the inclusion of a boolean equation of Keywords. An alternative solution to these problems is to generate another file called 'directory file'[1-11, 42-45] (see Fig.2.17) in addition to the usual document file. The information structure of the directory file is the inverse of the original document file. That is, instead of listing a set of Keywords contained in a record, the directory file (inverted file) contains Keywords, which maintain a list to point all records associated with these Keywords.

The retrieval of information, from such a system, is accomplished in two stages. In the first stage all Keywords, present in a query, are retrieved. This is known as 'directory decoding'. The result of directory decoding would provide a list of the relevant documents. The second stage of operations would be carried out to retrieve documents from the document file.

From the point of view of storage utilization it would appear that the directory file is redundant, because it merely duplicates the document file. But on the other hand, the presence of the directory file improves the overall performance.

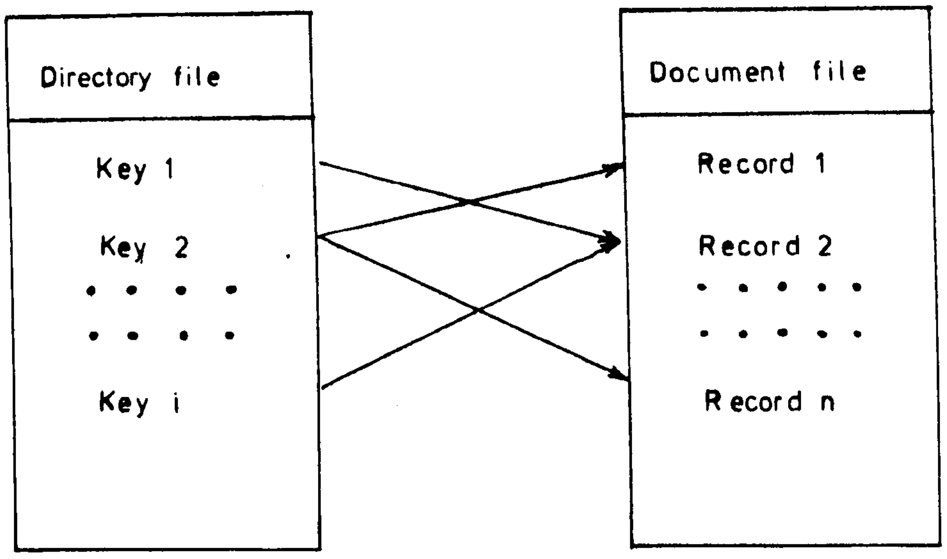


FIG. 2.17. Inverted file.

| W | | X | | Y | Z | | |
|---|---|---|---|---|---|---|---|
| A6 | A19 | A3 | A19 | A9 | A15 | A27 | |
| A7 | A23 | A7 | A20 | A14 | A17 | A37 | — — — |
| A9 | A35 | A14 | | A16 | A22 | | |
| A12 | | A15 | | A21 | A25 | | |

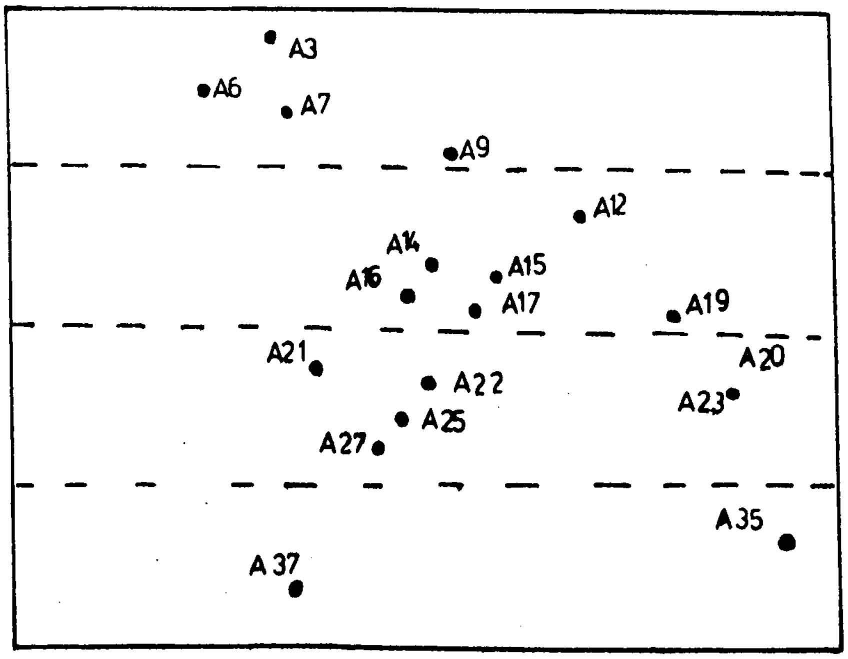


FIG. 2.18. Inverted list.

of the system. As the size of the inverted file is generally smaller than the document file, more efficient searching technique could be adopted for directory decoding. This would equally benefit the non-existing Keywords, where absence of those keywords would be quickly reported. The other advantage of the inverted file is that boolean operations on Keyword could be more easily performed.

The decoding operation of a directory, stored in a Direct Access Storage Device[11], comprises two processes. These are:

1) Transfer of data from DASD to the core memory
2) Processing of data in the core mamory.

The total time required for data transfer operation[11] from DASD to the core memory comprises

a) Time for head positioning (for movable head devices)
b) Latency delay
c) Data read time
d) Revolutions lost during the processing of data in the core memory.

The processing of data in the core memory for determining appropriate branching can be carried out according to any of Keyword searching techniques described in section 2.3. This in-core data processing time is very small compared to the time required for transferring data from DASD to the core memory. Hence the total average directory decoding time can be derived, ignoring this data processing time. In general the decoding time $T_n$ for an n-level tree is given by equation[11] 2.13 to 2.16.

1) **First level in core, successive levels in same cylinder of a movable head disc**

$$Tn = P + (2n - 2.5) R \quad \text{for } n > 1 \quad \ldots\ldots \quad 2.13$$

2) First level in core, fixed head disc.

$Tn = (2n - 2.5)R$, for $n > 1$ ..... 2.14

3) First level in movable head disc, successive levels in same cylinder

$Tn = P + (2n - 0.5)R$, for $n > 0$ ..... 2.15

4) First level in movable head disc and successive levels not in same cylinder.

$Tn = n\ (P + 1.5R)$ for $n > 0$ ..... 2.16

Where P = time required for positioning the head of a movable head disc

R = Rotational time of the DASD

L = R/2 = average latency delay.

The total average decoding time for a hash coded directory stored in DASD is

$T_h = P + 1.5\ R$ ..... 2.17

The actual retrieval of the document, of course, depends on the structure of the records of the inverted file.

A record of the inverted file consists essentially of two major parameters; these are:

1) a number of documents associated with the Keyword
2) pointers to the relevant documents

There are four major data-structures[11,35,36] used in a record of an inverted file; these are:

1) inverted list
2) chained list
3) controlled chained list
4) Cellular list

Some explanations of these data-structures along with corresponding document retrieval techniques are discussed briefly in the following sub-sections.

### 2.4.1 Inverted list:

In this type of data structure (shown in Fig. 2.18), a complete list of the document pointers is included in the record of the directory file. These pointers could be directly used to retrieve documents from the document file. As a record on the document file could be pointed from many Keywords, this data-structure could use larger storage. On the other hand logical operations could be more easily performed on the list of document pointers. Hence the pre-search statistics,which indicate an upper bound on the ultimate retrieval is much better for an inverted list.

The total time to retrieve the desired documents from the storage is a function of the data-structure employed in the inverted file. Commonly, the total retrieval time comprises

1) directory decoding time
2) List or cell intersection time
3) List or cell search and record transfer time

Each of these processes is again a function of parameters related to the characteristics of file, query and storage devices. Hence a set of these parameters are defined in table 2.1

For an inverted list file-structure the time required for above mentioned processes are given by equation[11] 2.18 to 2.21.

1) Directory decoding time $= Nt\ Tn$ ..... 2.18

TABLE 2.1

File related parameters

| Symbol | Definition |
|---|---|
| V | number of Distinct Keys in vocabularies |
| Np | number of Records in system |
| Nk | Number of Keys/record (Average) |
| L | Average list length $\frac{NrNk}{V}$ |
| Cf | Character/file (logical) Record (Average) |
| Rc | record/cell (Average) |
| Ck | Cells/Key (Average) |

Query related parameters

| Symbol | Definition |
|---|---|
| Nt | number of terms in a single query |
| Np | number of nonnegated terms in a single query |
| Ls | shortest list length in query (Average) |
| $\rho$ | Ratio of query response to Ls (Average) |
| $\alpha$ | Ratio of query cell responses to Ck (Average) |

Device related parameters

| Symbol | Definition |
|---|---|
| A | number of file record addresses per DASD physical record |
| Tr | Random access time of DASD (Average) |
| Rt | Transfer rate of DASD (B/s) |
| R | Rotation time of DASD (Sec) |

2) List intersection time $= \lceil \frac{L}{A}$ Nt (tr + 1.5R) ..... 2.19

here $\lceil \frac{L}{A}$ is next higher integer for a fraction

3) List search and record transfer time

$= \rho$ Ls (T +1.5R) ..... 2.20

4) Total retrieval time = NtTn + ( $\lceil \frac{L}{A}$ Nt + $\rho$ Ls)(Trt 1.5R) ..... 2.21

## 2.4.2 Chained List:

A chained list data-structure is shown in Fig. 2.19. This data-structure, instead of listing pointers for individual documents, points to the head of the document list. The subsequent members of the same list are chained by providing a link address inside the document file. Any record in the document file could be simultaneously a member of more than one list. Thus a threaded multilist is formed. The main advantages[36] of multilist data-structure are:

1) requirement of storage space of the directory file is less because each index contains only the address of the head of list.
2) programming is simpler and updating is flexible because it uses the list data-structure.

The disadvantage of the multilist data-structure is that as any prior information about the membership of a list is not known. the boolean operations could not be performed before retrieving all relevant documents from the data-base. Some improvement could be made in the case of logical 'And' operations of Keywords. Here, a list with the least membership is selected and is traced. During this list-tracing operation the content of each record is interrogated to find whether or not the record under examination is also a member of other lists. A

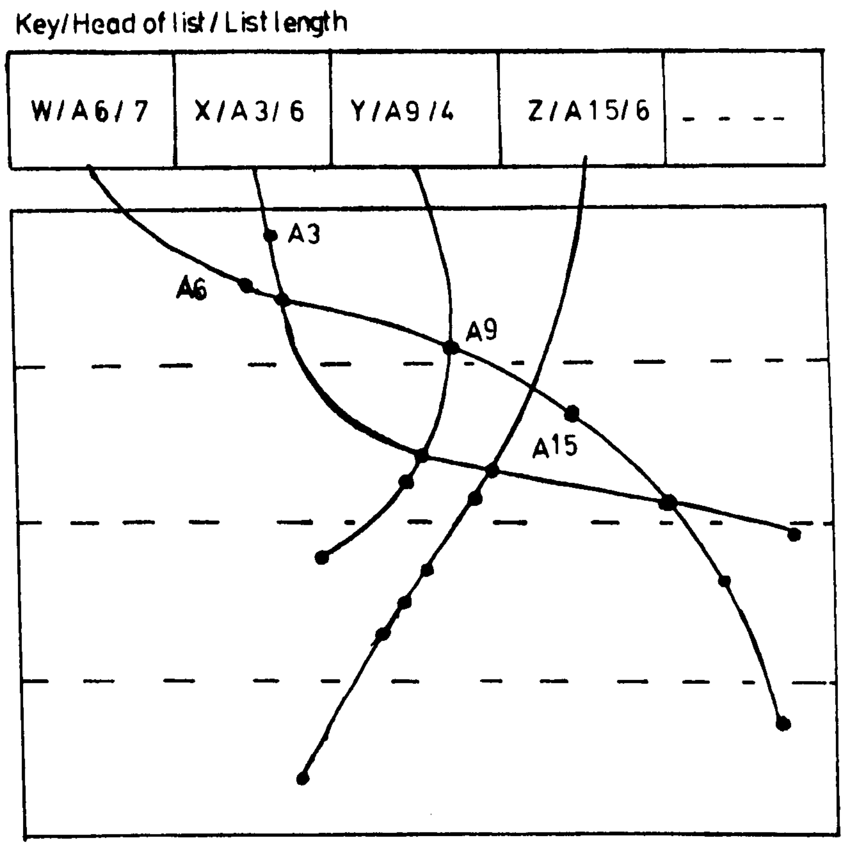


FIG. 2.19. Multilist.

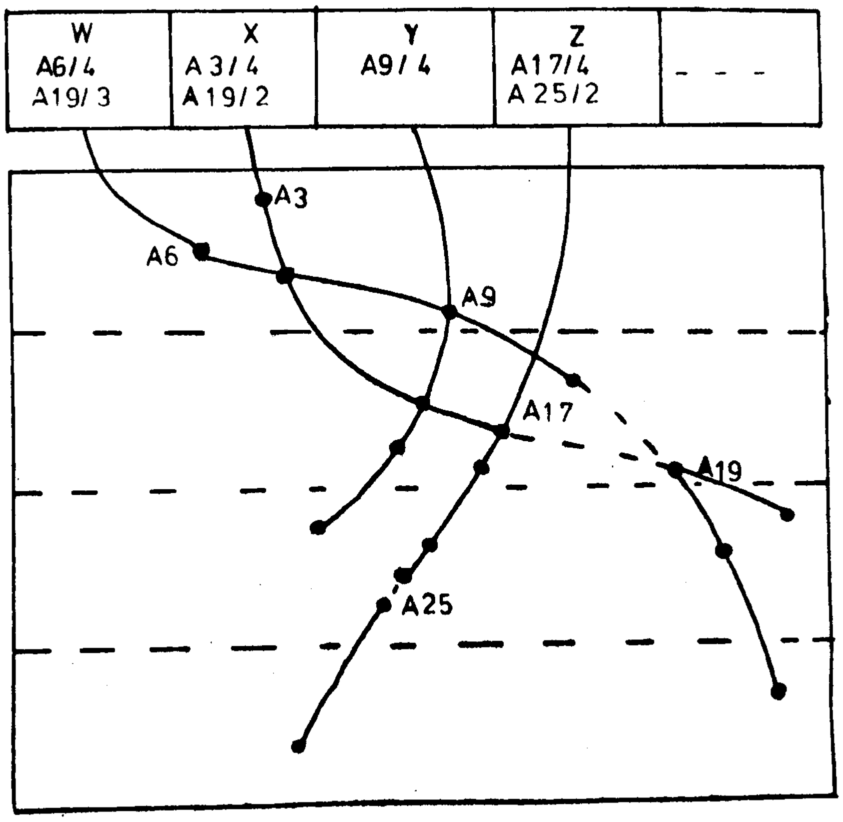


FIG. 2.20. Controlled multi-list

lot of time is wasted during this link-tracing operation, thus it degrades the retrieval time of the system.

For a multilist data structure, the corresponding equations for the retrieval time of the desired documents are given below.

1) Directory decoding time = NpTn ..... 2.22

2) List intersection - is not possible in a multilist data structure.

3) List search and record transfer time $= Ls\ (T_r + 1.5R)$ ..... 2.23

4) Total retrieval time $= NpTn+Ls\ (T_r + 1.5R)$ ..... 2.24

## 2.4.3 Controlled multilist:

To combine the benefits of both inverted and chained lists, a new generalised data-structure, as shown in Fig. 2.20 is suggested. As in the case of chained lists, it starts with a pointer showing the head of a linked list. But in this case the total number of memberships of the chained list is controlled to a specific number. Whenever the membership of a chained list exceeds this number a new head of list is inserted in the record of the directory file. This is a generalised data-structure, because when the control number is set to infinity it produces a chained list. And when the chain length is restricted to one, it produces an inverted list. Here the retrieval of documents are done by the combination of techniques utilised by multilist and inverted list data-structures.

## 2.4.4. Cellular partition:

It has been seen from the above discussions, that no effort has been made to order the data-structure of the records of the directory file so that optimal retrieval speed could be achieved. In general a large document file is stored on direct access devices such as discs. The main factor for the data-transfer from such a device is access time. Moreover, during each transfer operation a block of data is loaded to the core memory. To reflect these properties of storage device a cellular partition data-structure, as shown in Fig 2.21, is suggested.

This data structure, instead of specifically pointing location of documents, contains the addresses of the blocks of the storage device where the relevant documents are present, thus pointing to a cell of data.

The logical operation could be carried out as easily as an inverted list data-structure to access only relevant blocks of the storage device. Then the final selection of documents could be carried out, in high speed core, by comparing each record. Although the final selection is done by sequential searching, it saves many unnecessary accesses as in the case of multilist. And it also saves storage space in the directory file as this would otherwise have been required by the inverted list data-structure.

For a cellular serial file structure the retrieval time is given by the following equation[11]

1) Directory decoding time = $NpTn$ ..... 2.25

2) Cell intersection time = $\lceil \frac{Ck}{A} \quad Np \quad (T_r + 1.5R)$ ..... 2.26

where $\lceil \frac{Ck}{A}$ is next higher in case of a fraction

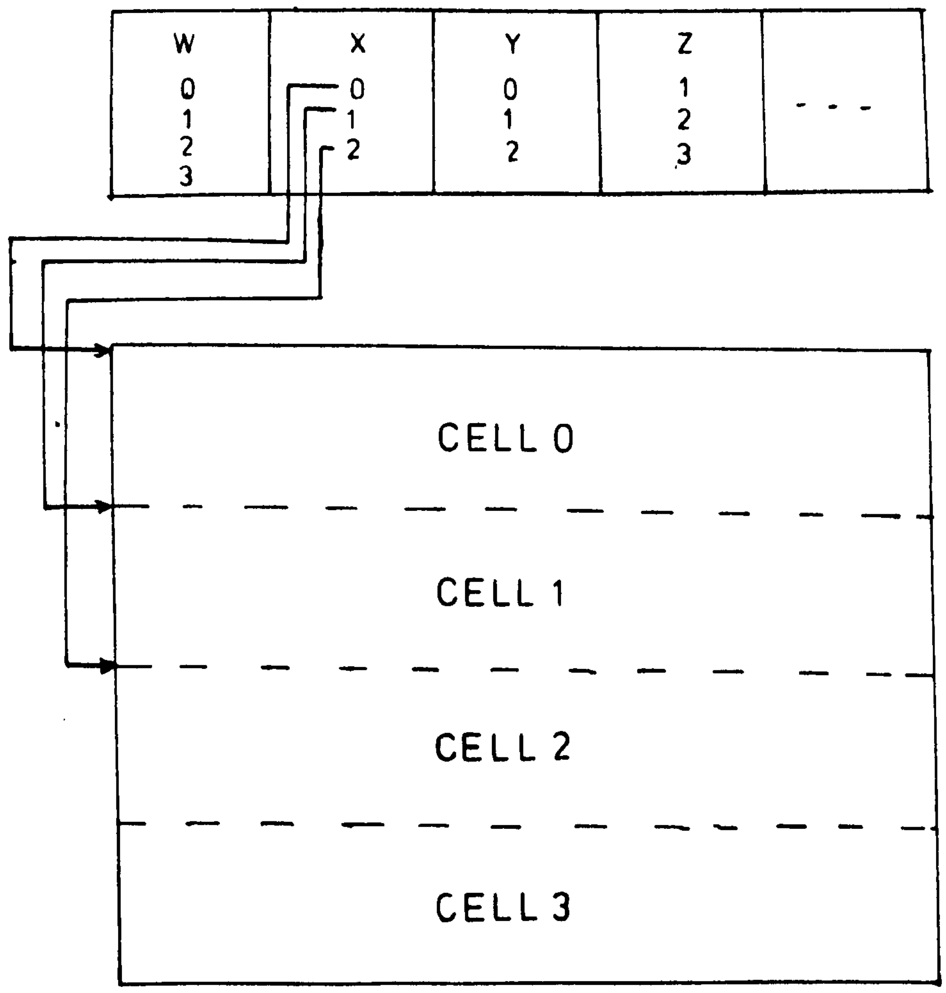


FIG. 2.21. Cellular serial file.

3) Cell searching and record retrieval time

$$= \alpha\ Ck \quad (T_r + \frac{RcCf}{Rt}) \qquad \ldots\ldots \qquad 2.27$$

4) Total retrieval time

$$= Np\ Tn + \sqrt{\frac{Ck}{A}} \quad Np \quad (T_r + 1.5R)$$

$$+ \alpha\ Ck \quad (T_r + \frac{RcCf}{Rt}) \qquad \ldots\ldots \qquad 2.28$$

## 2.5 File Update and Maintenance:

Apart from the retrieval operation, an information retrieval system must be capable of performing file update and maintenance operations. The file-update operations can be classified into five categories; these are:

1) whole record addition
2) Whole record deletion
3) Addition of Key
4) Deletion of Key
5) Addition/Deletion/modification of non-key data

The structure of the file has an immense effect on the flexibility and ease of these update operations. This is demonstrated in table 2.2, 2.3 and 2.4. Hence a system designer should take proper care to select a file structure.

Another important function of retrieval systems is maintenance of the file. That is, collection and re-usage of empty spaces of file. This operation is called garbage collection. The system designer must also consider a suitable garbage collection scheme, especially when the data-base is dynamic.

TABLE 2.2

Update Inverted List

| Process | whole record addition | whole record deletion | Deletion of n Keys | non key modification (without relocation) | non Key modification with relocation | addition of n Key |
|---|---|---|---|---|---|---|
| Directory decoding |  | T3 | T3 | T3 | T3 | T3 |
| Access Record |  | TA | TA | TA | TA | TA |
| Update directory | Nk T3 | Nk T3 | n T3 |  | Nk T3 | n T3 |
| Update Inverted list | NkTL [1] | NkTL | nTL |  | NkTL | n TL |
| Store Updated data | TA | TA [2] | TA | TA | TA | TA |

[1] $T_L = (Tr + \frac{1}{2} \lceil \frac{L}{A} \rceil R)$ Add R; if read after write verification is required.

[2] Required only if delete bit is used. Delete bit is required only if space brooming is used

$T_A = T + 1.5R.$

TABLE 2.3

Update timing for multilist File Structure

| Process | whole record addition | whole [2] record deletion | Deletion of n Keys | non key modification (without relocation) | non key modification (with relocation) | addition n Keys (without relocation) |
|---|---|---|---|---|---|---|
| Directory decode |  | T3 [3] | T3 | T3 | T3 | T3 |
| Access Record |  | TA | TA | TA | TA | TA |
| Update Directory | NkT3 |  |  |  | NkT3 | nT3 |
| Store updated Record | TA [1] | TA | TA | TA | TA | TA |

[1] TA = $T_r$ + 1.5R     Add R; if read after write verification is required

[2] Assumes that directory list lengths are physical; therefore, directory update is not required when Key or record delete bit is set

[3] T3 = Decoding time for three level tree.  Add R; if read after write verification is required.

TABLE 2.4

Update: Cellular Serial File

| Process | Whole[1] record addition | whole record deletion | Deletion[2] of n Keys | non Key modification without relocation | non Key [3] modification with relocation | Addition of n Key (without relocation) |
|---|---|---|---|---|---|---|
| Directory decode | | T3 | T3 | T3 | T3 | T3 |
| Access records | | TA | TA | TA | TA | TA |
| Update Directory | | | | | | |
| Update inverted list | | | | | | |
| Store updated record | TA | TA | TA | TA | TA | TA |

1 Assume that all keys in record are already represent in cell

2 Keys not deleted from the cell

3 Assume that record is relocated within the cell.

TABLE 2.5

Update Comparisons Among Three File Structures

| Update Type | Multi list | Inverted List | Cellular serial |
|---|---|---|---|
| Whole record Addition | Nk T3 + TA | NkT3 + TA + Nk TL | TA |
| Whole record deletion | T3 + 2 TA | T3 + 2TA + NK (T3 + TL) | T3 + 2TA |
| Deletion of n Keys | T3 + 2 TA | T3 + 2TA + n (T3 + TL) | T3 + 2TA |
| Non Key modification without relocation | T3 + 2 TA | T3 + 2TA | T3 + T2A |
| non Key modification with relocation | (Nk + 1) T3 + 2 TA | (Nk + 1) T3 + 2TA + NkTL | T3 + 2TA |
| Addition of n Keys | (n + 1) T3 + 2 TA | (n + 1) T3 + 2TA + nTL | T3 + T2A |

## 2.6 Summary:

The average searching time of a primary Key for various methods has been given by the following equations;

1) for sequential search on unordered file

$$Cs = \frac{N+1}{2} \quad \ldots\ldots \quad 2.1$$

2) for sequential search on ordered file

$$Cs = \frac{N+1}{2} \quad \ldots\ldots \quad 2.4$$

3) for binary search

$$Ca = \lceil \log_2 N - 1 \quad \ldots\ldots \quad 2.8$$

4) for a 'm' way tree

$$C = W \ ( \lceil \log_m N) \quad \ldots\ldots \quad 2.10$$

where N = total no. of records

W = No. of comparisons required to search each level of the tree

5) for a hash-table

$$Ca = 1 + \frac{\alpha}{2}$$

where $\alpha = \frac{N}{M}$ = Load fraction

From these equations it can be seen that, in the case of hash-coding method the average number of comparison is minimum. But the number of probes required to establish non-occurence of a Key is unpredictable and it may be quite large. Although the average searching time for a sequential searching method is highest, it offers flexibility of easy updating. In the binary search method both average and maximum searching time of a Key are predictable. But this requires all Keys to be simultaneously resident in the core memory. As the number of Keys grows, it becomes impracticable to satisfy this criterion of the binary search method, and Keys are generally stored in a DASD (Direct Access Storage Device). Here, the total number of access-requests to the DASD is more important than the number of Keys compared. The main objective, in such a situation, is to reduce the number of levels of decoding tree by increasing the number of branches at each level. For an 'n' level tree, the decoding time Tn is given by the equations

1) The first level of tree in the core memory, successive levels in the same cylinder of a movable head disc.

   $Tn = P + (2n - 2.5)R$ , for $n > 1$ ..... 2.13

2) The first level in core, fixed head disc

   $Tn = (2n - 2.5)R$, for $n > 1$ ..... 2.14

3) The first level in movable head disc, successive levels in the same cylinder.

   $Tn = P + (2n - 0.5)R$ for $n > 0$ ..... 2.15

4) The first level in movable head disc and successive levels not in the same cylinder

   $Tn = n\ (P + 1.5R)$ for $n > 0$ ..... 2.16

5) For hash-coded directory

   $Tn = P + 1.5R$ ..... 2.17

where P = time required for head positioning

R = Rotational time of the DASD

L = R/2 = Average latency delay

It has been seen in equations 2.13 - 2.17 that the total decoding time of a directory is only a function of the parameters of the DASD and the contribution of the data-processing time within the core memory is insignificant.

It has been also found in section 2.4 that the flexibility of a retrieval system is enhanced by the use of an inverted file. In such a system, due to the presence of a two-level hierarchy, the total retrieval and update time becomes a function of the data-structure employed within the inverted file. The total retrieval time for various data-structures is given by the following equations:

1) For an inverted list

$$\text{Total retrieval time} = NtTn + \left(\left\lceil \frac{L}{A} \right. Nt + \rho\, Ls\right)(Tr + 1.5R) \quad \ldots\ldots \quad 2.21$$

2) For a multilist

$$\text{Total retrieval time} = NpTn + Ls\ (Tr + 1.5R) \quad \ldots\ldots \quad 2.24$$

3) For a cellular serial file-structure

$$\text{Total retrieval time} = NpTn + \left\lceil \frac{Ck}{A} \right. Np\ (Tr + 1.5R)$$

$$+ \alpha Ck\ Tr + \frac{RcCf}{Rt} \quad \ldots\ldots \quad 2.28$$

where $\lceil x$ indicates next higher integer in case of a fraction and for legend see table 2.1

The total update time for various data-structures is shown in Table 2.5.

To summarise the performances of the various file-structures the table 2.6 is given[11]. Here the lower value of an entry indicates an optimal performance. Although the performance figures shown in this table are not precise, these are indicative of making general assessment of the various file-structures. From table 2.6 a number of plots (Fig. 2.22 - 2.25) can be drawn to show the relative merits of the individual file-structures. Here an entry near the origin indicates

TABLE 2.6

Summary of Performances of File-structures

| | Inverted List | multilist | cellular serial |
|---|---|---|---|
| Total retrieval time | 1 | 4 | 1 |
| No. of file Random accessions per query | 2 | 4 | 1 |
| Presearch retrieval statistics | 1 | 3 | 4 |
| Programming complexity | 3 | 1 | 1 |
| Update time | 3 | 1 | 1 |
| DASD memory requirement | 3/1* | 3 | 1 |

* With Keys in the inverted list file record/ without keys the inverted list file records.

N.B. Lower value of the entries indicates more optimal property value.

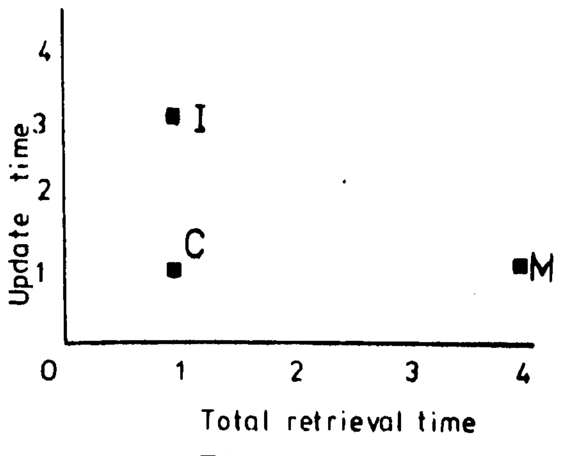


FIG. 2.22.

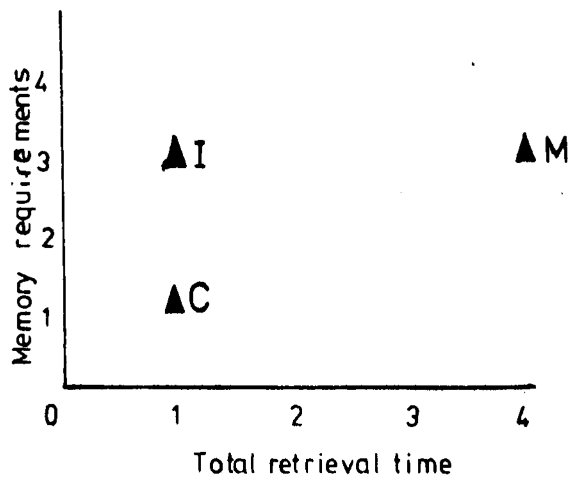


FIG. 2.23.

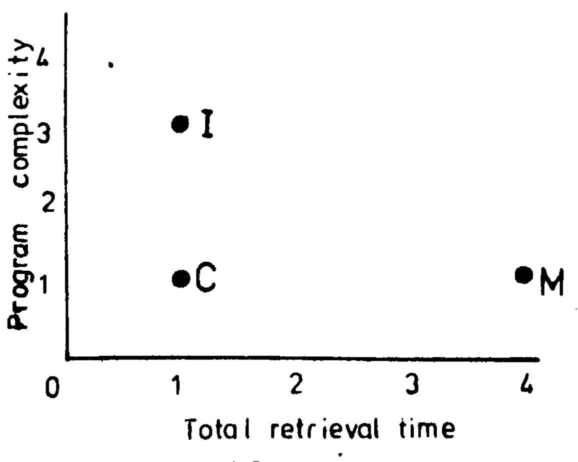


FIG. 2.24.

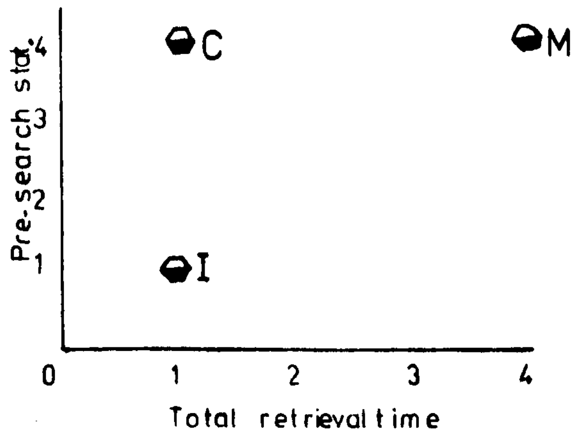


FIG. 2.25.

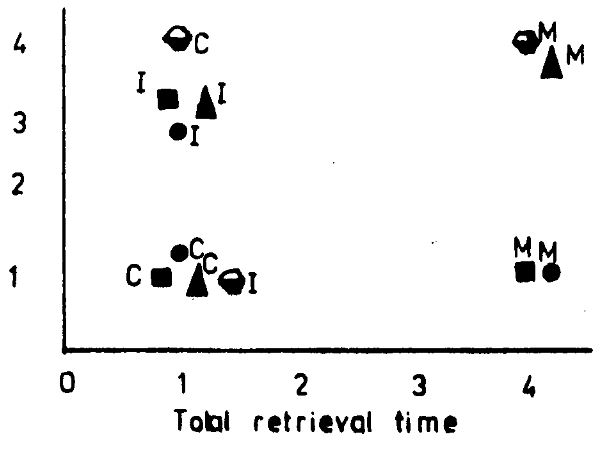


FIG. 2.26.

LEGEND

| | |
|---|---|
| ■ | UPDATE TIME |
| ▲ | MEMORY REQ |
| ● | PROG. COMPL. |
| ⬣ | PRE-SEARCH STAT. |
| I | INVERTED LIST |
| M | MULTI LIST |
| C | CELLULAR SERIAL |

optimal performance. The Fig. 2.26 shows that, except for the presearch statistics (see section 2.4.1.), the overall performance of the cellular serial file structure is optimal.

## 2.7 DISCUSSIONS:

It has been seen that to improve speed of response, precision and flexibility of retrieval system the data should be highly structured. It not only increases the complexity of data-base during its creation, but also makes the update and maintenance of data-base more difficult. The complexity of data structure is also reflected in the retrieval algorithm. Hence to cope with these requirements of performances, use of the data-processing Unit of a better and higher performance figure becomes essential. The other overhead of complex data-structure is requirement of larger storage media, basically to store unnecessary links, pointers or tables. The overall effects of these are increased investment and running cost. Thus the service of an information retrieval system becomes expensive and generally rises beyond the capabilities of many potential users such as research students and design engineers.

On the other hand it is interesting to observe that a human, wishing to select documents with the help of a short list of related Keys, would recognise and retrieve the appropriate information, regardless of their positions. This form of pattern-matching is the basis of the associative retrieval system. This is perhaps the right way of handling the problems of retrieval operation which is essentially nothing but the problem of association of Keys and documents.

In the following chapter some aspects of using associative parallel processer for information retrieval systems are reviewed. This also discusses the architecture and organization of systems, based on content-addressable memories.

# CHAPTER 3.

## Associative solution of the information retrieval problem and an overview of associative parallel processor.

## 3.

In Chapter 2, it has been seen that, the retrieval of information primarily involves association of the Keys in the user's profile with those in the document file. This basic property of association of Keys is not inherent in a conventional computer[46]. Instead, a lot of software effort and housekeeping functions are incorporated to establish an artificial associative property within a conventional computing system. This obviously leads to increased complexity and cost of the system. On the other hand, a system based on content-addressable memory[47-55] has an implied property of association. This eliminates the need for any extra effort to create an artificial association. Moreover, the natural parallelism of the content-addressable memory yields a faster search and retrieval operation.

## 3.1 Associative solution of the information retrieval systems:

The simplest approach of solving the problem of Keyword searching is to store the entire document file in an associative memory array and presenting the Keys of the profile to the 'search data' part of the data input register of the memory array (see Fig. 3.1). Then a parallel search is carried out over the entire contents of the document file. As a result of this parallel search operation a number of documents which satisfy the search-Key, are selected and subsequently retrieved.

Although this provides a simple solution for the retrieval operation, the major

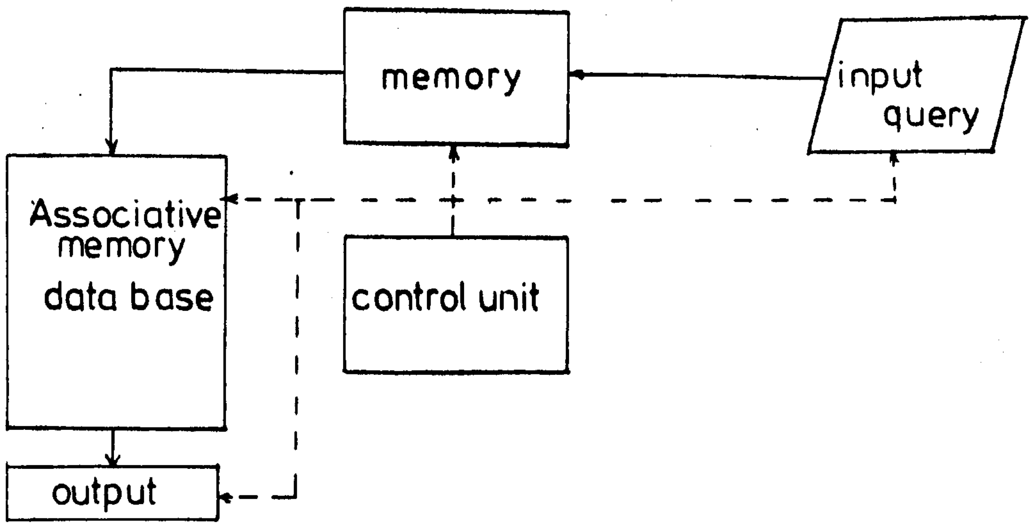


FIG.3.1. Associative retrieval system

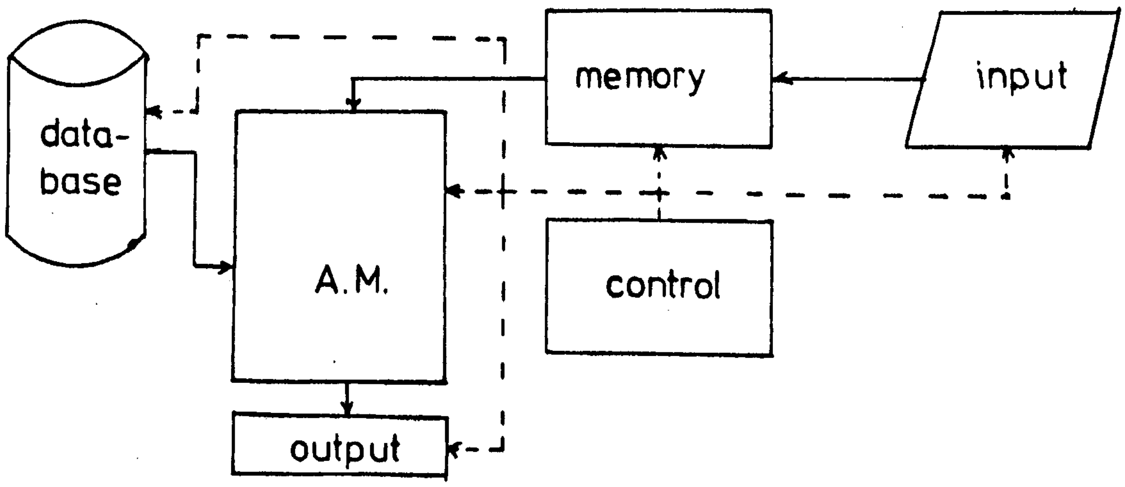


FIG. 3.2. Part of data-base in A.M.

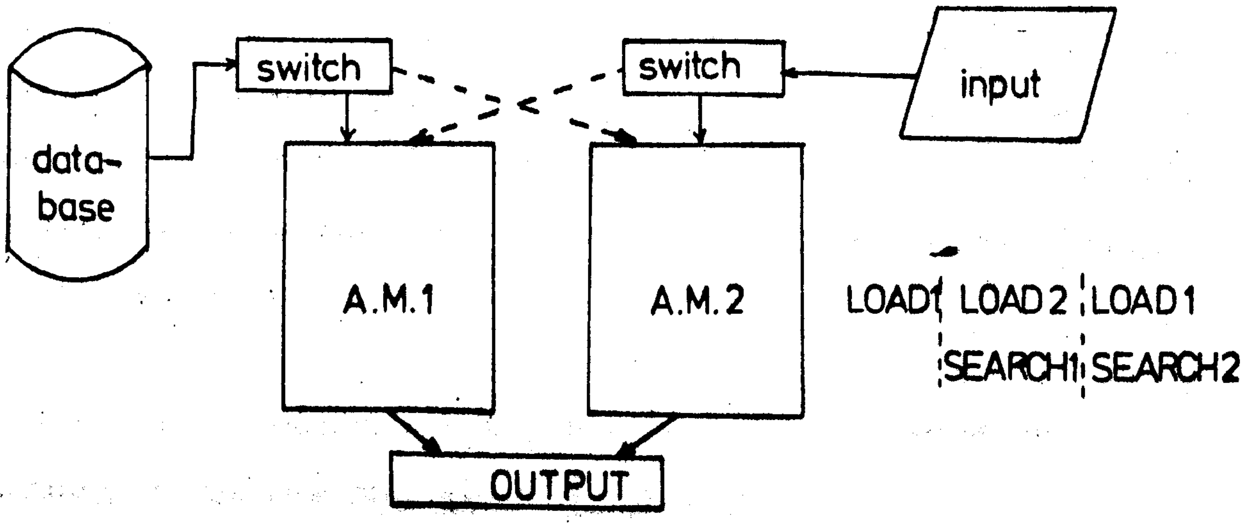


FIG. 3.3. Two memory solution

difficulty arises when more than one document matches the search-Key. Special attention must be given, in this multiple response case, to resolve each of these matching documents so that they can be individually retrieved. As the size of the data-base grows, it becomes difficult to store the entire data-base simultaneously in an associative memory array. This is mainly because of hardware problems;to-date, a cheaper solution to produce a large associative memory array is not available.

In an attempt to solve this problem, an alternative method of retrieval system[151,15] is suggested. In this method (see Fig. 3.2), a part of the data-base (an integral number of records) is initially loaded in the associative memory array; a search operation is carried out to locate the relevant records from this portion of the data-base. This loading and search operation is continued for the rest of the data-base, until the scanning operation is complete. The loading of a part of the data-base in the associative memory obviously introduces a delay in the search operation. This is because the search operation should be in-operative during the loading time of the associative memory. A multiple associative memory system could be adopted to solve this problem. A system as shown in Fig 3.3, using two separate associative memories, could be implemented so that the searching operation is carried out in one of these associative memories while the loading operation is continued in the other associative memory. A criterion for the success of this method is that the time required by the algorithm for the loading and the searching operation should be balanced. Otherwise some unnecessary waiting time between successive operations would be encountered. The other disadvantage of this system is the requirement of two separate memory arrays, thus it becomes expensive.

In another obvious alternative system, the 'ON-THE-FLY'[151] method, (See Fig. 3.4) the strategy for the searching operation is reversed. In this system the user's profile is stored in the associative memory and the Keys of the documents, as

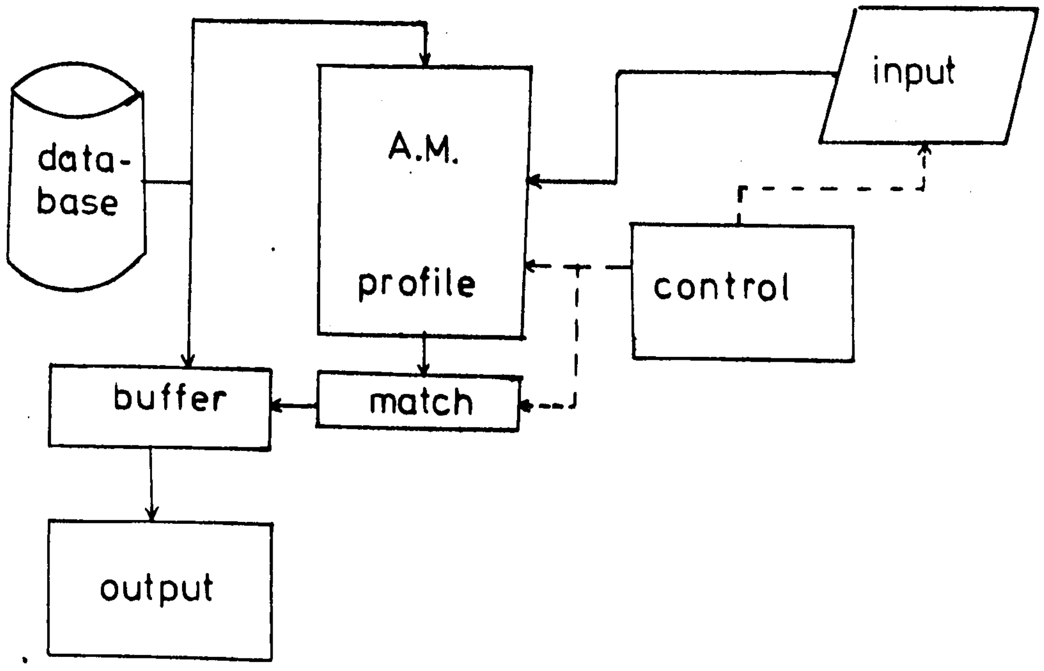


FIG. 3.4. ON-THE-FLY searching

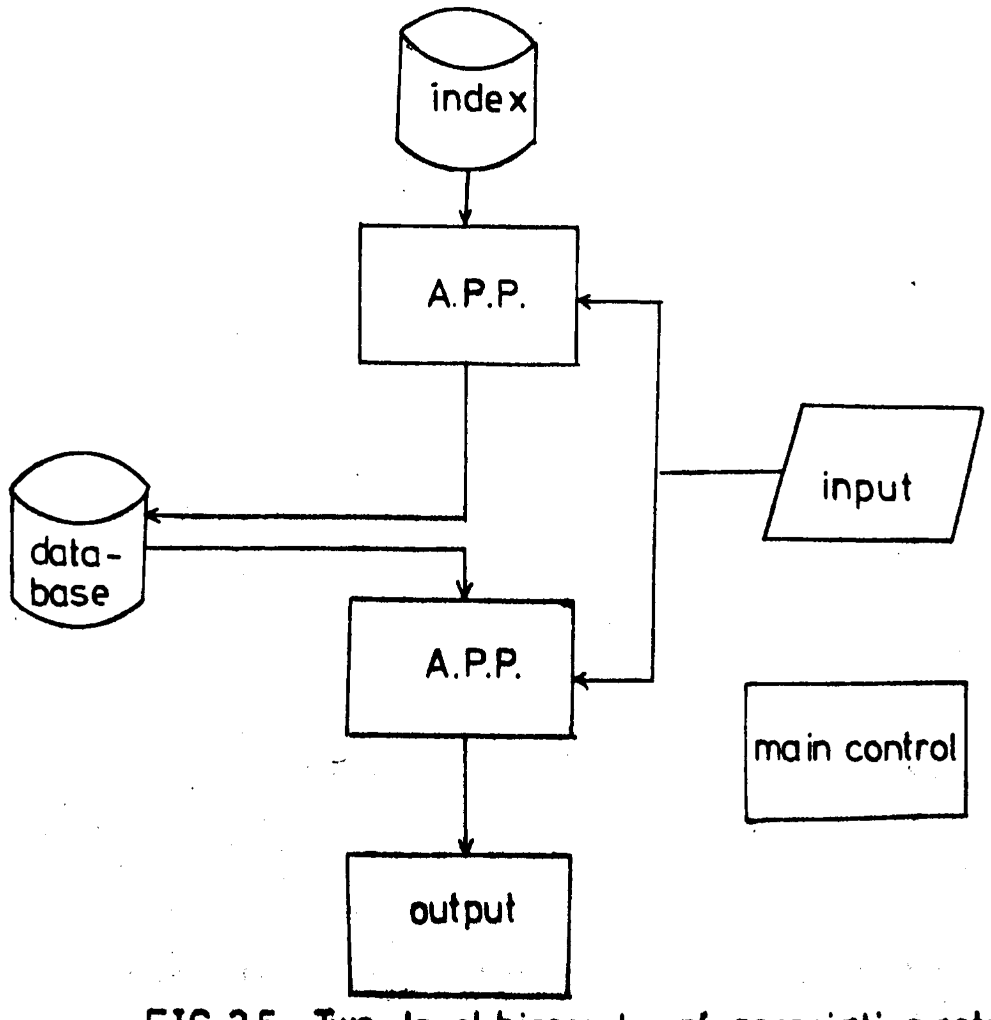


FIG. 3.5. Two-level hierarchy of associative retrieval

they appear on the read head of a rotating disc unit, are presented for matching operation. At the end of scanning a record, its relevance to the user's query is evaluated and the successful records are filtered out. The scanning of the document file is continued until an end-of-file mark is detected.

In the present investigation, On-The-Fly searching technique is chosen mainly because of its simplicity and inexpensiveness. It is realised that the serial scanning of the document file could impose a problem for a large data-base. In such cases a system similar to head-per-track content-addressable data-base or content-addressable file storage system could be proposed. The block diagram of such a system is shown in Fig 3.5. In this system, a two-level hierarchy of associative addressing[151] is adopted. In the first level of the hierarchy an index file, the size of which is a small fraction of the entire data-base, is scanned. This index file comprises a short description of all records in the data-base, and provides the information for selecting the best block(s) of the data-base to be further scanned for retrieving the desired records. Thus the burden of indexing is greatly reduced to that of pointing out the most probable areas of data-base where the relevant documents are likely to occur. In the second level of operation only those blocks of the data-base, as pointed out by the index, are associatively scanned for final retrieval operation.

Both of these tasks can equally benefit from associative processing. Thus it could be seen that the primary operation for all retrieval tasks is basically the same and On-The-Fly searching technique can be applied as a general solution.

As the content addressability and the natural parallelism of an Associative Parallel Processor (APP) are exploited in an associative retrieval system, the search operation would be more efficient and fast.[151,152] In such a system the searching is not restricted to any predefined Keys, but instead the entire data-base is scanned. Thus the system could support a more flexible keyword-searching strategy. Finally due to the absence of any rigid file- and data- structure, it would be very easy to create, update and maintain the data base. These indicate that an associative retrieval system would have better performance over its conventional counter parts.

Before any further discussion on associative retrieval system, some aspects of the associative parallel processor are reviewed in section 3.2

## 3.2 An Over-view of Associative Parallel Processor

The use of computers in modern society extends beyond the usual arithmetic and logical operations of data to the area of non-numeric applications. The conventional computers[46] are especially designed for an efficient numerical operation. On the other hand the non-numerical text-processing applications need efficient sub-string search and string manipulations. Thus when these conventional systems are used in non-numerical applications, they become obviously inefficient. As an alternative, associative memory, which could efficiently support these primitive operations, could be used for non-numerical applications.

Use of content-addressable memory in a computing system leads to the development of the Associative Parallel processor[47-67] Before proceeding further, some terms in this context are defined. The following definitions are due to Parhami[47].

### Associative Memory:

An associative memory is a storage device that stores data in a number of cells. These cells can be accessed or loaded on the basis of their contents.

### Associative Processor:

An associative processor is an associative memory in which more sophisticated data transformation can be performed on the content of a number of cells selected according to their contents.

Associative Computer:

An associative computer is a computer that uses an associative memory or processor as an essential component for storage or processing respectively.

## 3.2.1 Associative Processor Architecture:

A generalised block diagram of an associative processor[51,52] is shown in Fig. 3.6. Two distinct functional units, Arithmetic and memory, of a conventional computer architecture and replaced here by a single associative memory array. where data are processed in-situ. In addition, each word in the array is accessed by its contents, rather than by physical location.

The functions of the control and the input/output units are similar to that of the conventional system. A brief description of the unfamiliar associative memory array is included in the following paragraphs.

The organization of an associative memory unit[51] is shown in Fig. 3.7. The associative memory unit shown here is an array of identical one bit cells. Each cell, in addition to its normal read/write operation, is capable of comparing its contents against an external comparand. These cells are usually organised in a group to form a word-row. Each word is generally partitioned into two. One part of the word is reserved for data storage. The other, called activity or control field, is used for storing flags. This control field is used as a temporary markers for processing, or as a permanent marker forming an extension to the data in each word, in order to improve the flexibility of access.

The unmasked portion of the comparand, which is stored in the data-input register, under the control of bit-select logic is applied to the memory array as a search Key. The result of this parallel search operation is then staticized in a tag register. The match reply signal is usually generated to provide a conditional

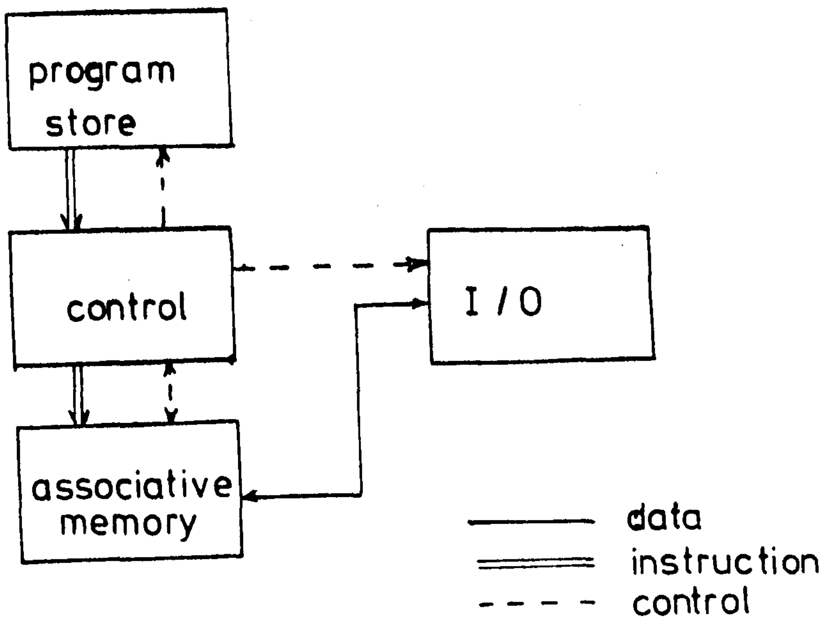


FIG. 3.6. Block diagram of an A.P.P.

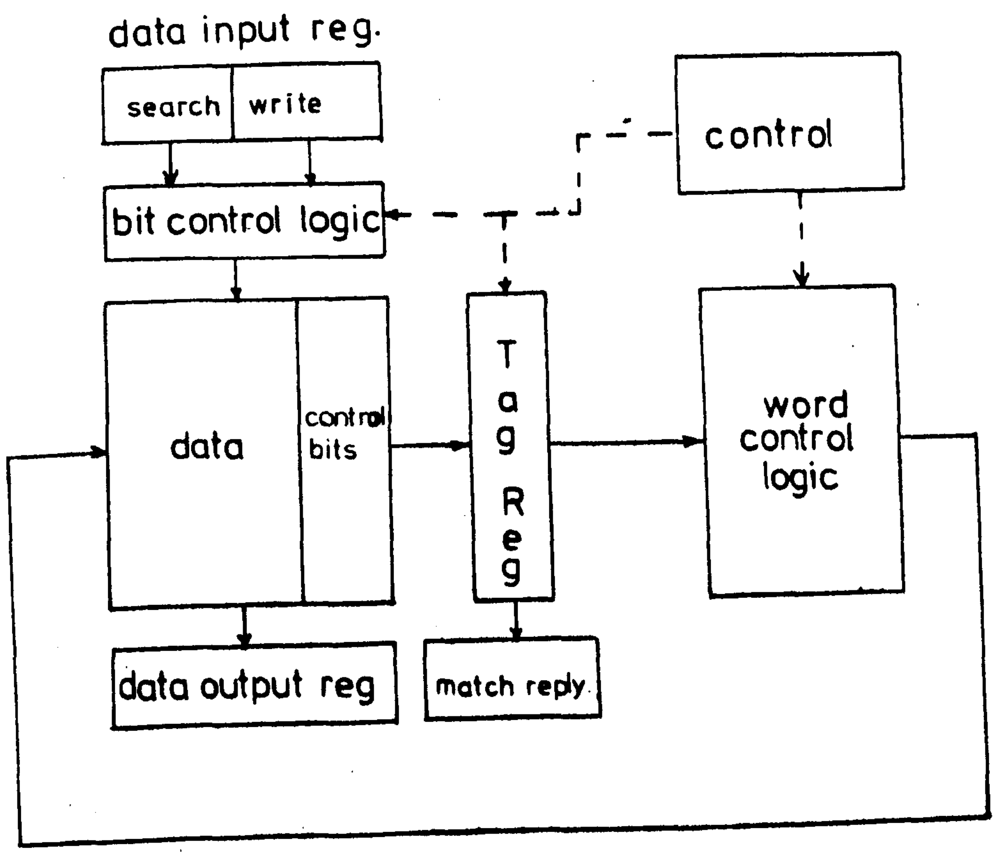


FIG. 3.7. Associative memory unit.

branching, which depends on the outcome of a search.

The word selection logic, in conjunction with the tag register and the mode control, enables a number of words for subsequent read/write operations. An additional feature of inter-word communication is provided in some systems. This communication is generally limited between neighbouring words. This facility can be utilized to provide a bit serial bi-directional shifting capability.

Further characteristics of a particular APP depends on the type of memory organization used in that system. These memory organizations fall in several categories, which are discussed below.

## 3.2.1.1: Fixed Record Length:

In the fixed record length[51,56] APP (See Fig 3.8) one word-row of the associative memory is allocated to each record. In this mode of operation the communication between words is not provided. The FRL organization is suitable for data which has a fixed-length format, such that each word in the array can be processed independently. A disadvantage of the fixed-record-length memory is that for certain applications for which records are of dissimilar length, some redundancy can exist within the array.

## 3.2.1.2. Variable record length:

In the variable record length memory organization, one word-row of the associative memory is allocated to each item of a record. In this organization the communication between the neighbouring words is provided for an easier extension of a logical record. The memory organization is suitable for non-numerica

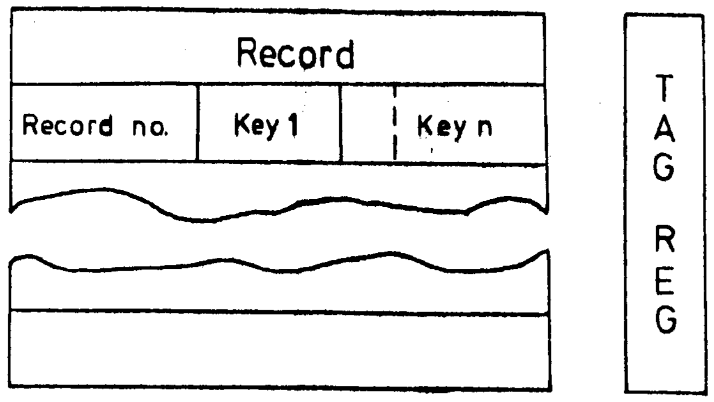
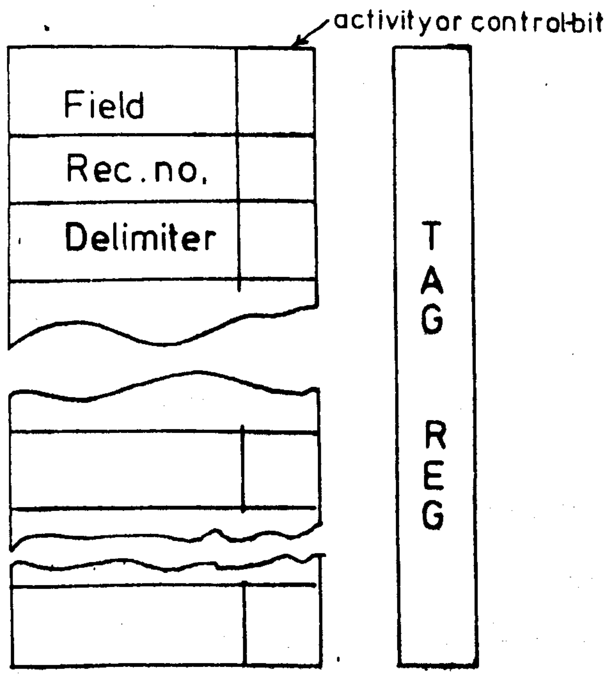


FIG. 3.8. Fixed Record Length



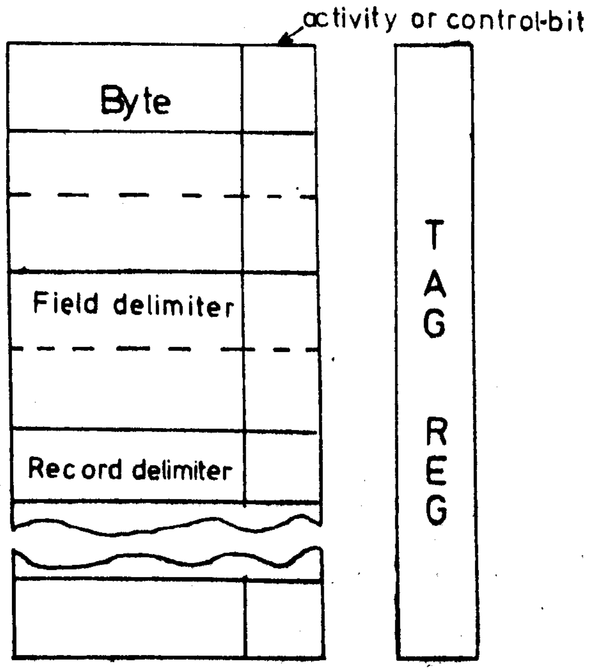


FIG. 3.9. Field-organized VRL

FIG. 3.10. Byte-organized VRL

computing. Variable record length memory organisations can be further divided into two categories.

a) Field-Orientated Variable Record Length

In the FO-VRL memory organisation[151] (Fig.3.9) a row of memory word is allocated to store a field of record along with some control bits.

b) Byte-Orientated Variable Record Length

In the BO-VRL memory organization[51,52,151] (See Fig. 3.10) each word in the associative memory array has sufficient storage for one character and a number of control bits. In this type of organization the data is stored as a one-dimensional character-string.

### 3.2.1.3. Word-Oriented

In the word oriented APP[47-49] each word in the memory array can store more than one character. The typical word length varies between 32 to 256 bits. In this type of data organization, the mode of access is either bit-serial, byte-serial or fully parallel depending on the particular hardware design.

### 3.2.1.4. Bit Serial

In the bit serial APP all words in the associative memory are accessed in only one bit position at a time. The STARAN system[68-70] was built using conventional memory elements to produce a word-parallel bit-serial APP.

Because sorting operations (maximum, minimum, between limits etc.) and arithmetic both use bit-serial processing operations on an APP, no time penalty is incurred for these tasks when this type of memory is used.

### 3.2.1.5. Word Serial

In the word serial APP, each word is accessed by content, and operated upon serially[71] at very high speed. The relative merits of this type of organization are faster instruction decoding and use of high data-rate low-cost circulating memories.

### 3.2.1.6. Associative file store

In the associative file store[153-156] the data is stored in a head-per-track disc (See Fig. 3.11). Where individual head is provided with sufficient logic to compare the incoming data against the searching criterion. This provides an effective means for high-speed searching on a large data-base.

### 3.2.1.7. Distributed logic memory:

In the distributed logic memory array, in addition to the content addressibility, sufficient logic is provided in each memory word to enable logical operation to be performed under a global control. The distributed logic memory, as shown in Fig. 3.12 was first proposed by Lee[72-74]. He proposed a linear array of inter-communicating cells for the purpose of information retrieval. Each of the cells is capable of performing basic operations such as search, read and write. Communication between cells is provided

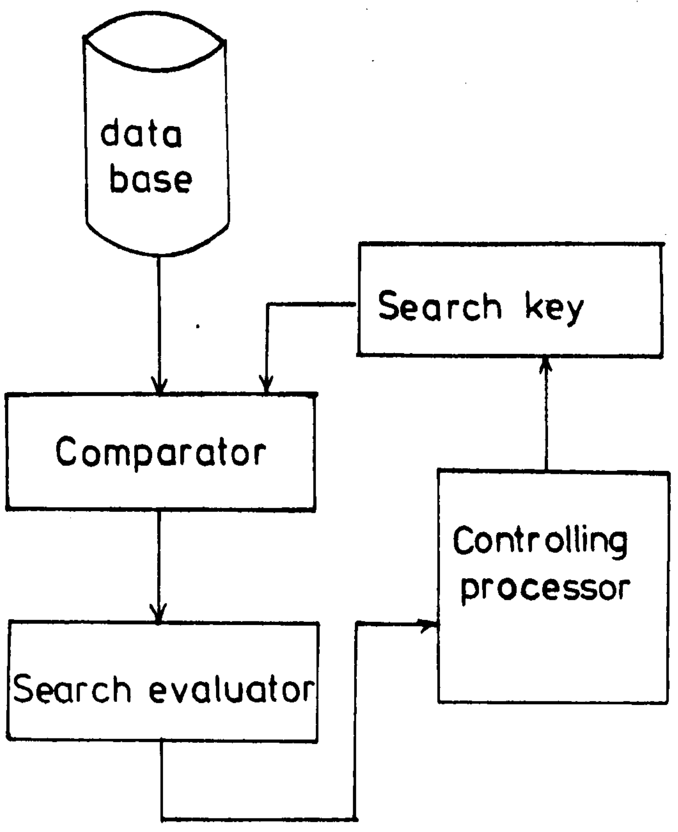


FIG. 3.11. Associative file-store.

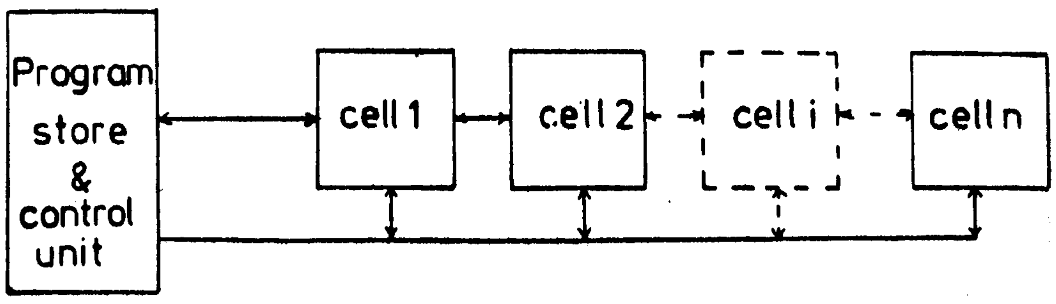


FIG. 3.12. Distributed logic memory

by the shifting left or right of an activity bit. All controls and data lines are common to each of these cells. This helps the modular expandibility of the memory array.

Extending Lee's idea of inter-communicating cell, struman[75-77] proposed a general purpose computer - where the program and the data share an uniform memory array. Lipovski[78-79] proposed a tree channel processor which solves the propagation delay problem of the DLM type array. Similar ideas are reported also by Crane[80], Kisylia[81] and Savit[122].

## 3.2.2. Basic Operations:

The basic operations[47,48,51-55] performed in an associative processor are

1) Search
2) Read
3) Write
4) Arithmetic and Logical operations.

### 3.2.2.1. Search

The simplest search operations are either equality or inequality. In this type of exact matching scheme, the unmasked portion of the search Key is compared with the content of the memory array. The result of the search operation is usually stored in a tag register, associated with each cell. The other types of possible search operations[82,157] are:

1) Less than

2) Greater than
3) Less than or equal to
4) Greater than or equal to
5) Between limits
6) Maximum value
7) Minimum value
8) Next higher
9) Next lower
10) Most frequent
11) Least frequent

## 3.2.2.2. Read:

The read operation is performed by either conventional or content addressing. In the latter case, if more than one word responds, the match resolves is used to isolate the first matching word.

## 3.2.2.3. Write:

Two types of write operations are possible. The simple write operation is similar to the read operation. In the multiple write operation, either the entire memory array or a number of selected words of the memory array are written simultaneously under the control of the word-selection logic.

## 3.2.2.4. Arithmetic and Logical Operations:

These could include.

1) Two's complement addition

2) Logical And/Or, Not, Nand/Nor exclusive
- Or and shift operations.

## 3.2.3. The multiple response problem:

The result of a search operation is usually fed-back to the control unit via the match reply line. Difficulties arise when a number of records satisfy a search Key. Different types of match reply methods are suggested to enable the control unit to take appropriate action. These include:-

1) Binary Reply[51]:- determines whether a memory array contains a matching word.
2) Tertiary Reply[85]:- indicates that the memory array contains no words, one, or more than one matching word.
3) Analogue estimate[86]:- provides an approximate number of matching words.
4) Exact count[87]:- gives the exact number of matching words.

The problem of isolating a single matching word is solved by either hardware or software method. In the hardware approach a parallel, logarithmic or ripple match resolver is added to the memory array. Although this provides fastest isolation of the first matching word, it is expensive and the cost increases with the size of the memory array.

In the simplest software approach the memory array is sequentially scanned in some direction until the first responder is encountered. Lewin[88] developed an algorithmn which requires two sense lines per bit column. This can isolate m matching words in 2m-1 cycles.

### 3.2.4. Hardware Element:

The basic characteristic of an associative memory is that it should be made up of a device which permits Non-Destructive read out. The earlier associative memory was developed using super conductivity[89-95]. It was projected that the cryogenic memory could be economically mass produced. But the problem associated with the maintenance and high initial cost of refrigeration caused some apprehension.

Some associative memory-using magnetic elements[96-98] have been fabricated. These include plated wire[99,100], thin film[101] and multi-aperture[102] core. With the advent of improved large-scale-Intergrated circuit technology, some content-addressable memories using MOS device[103-113] are proposed. But until to-day, an effective solution to this hardware problem has not been obtained to produce a large scale associative memory at a reasonable cost.

Good-year Aerospace[114] delivered a plated wire associative memory of 48 bit x 2K words to Rome Air Development Centre in 1968. A semi-conductor version of the associative processor, STARAN[68-70] (256 bit x 256 words) is now commercially available from Good Year. It uses bit-serial mode of I/O access. The processing of data in this system is also done in bit serial manner. STARAN utilises a PDP-11 as its sequential controller.

## 3.2.5. Software for APP:-

Research in the area of software development for associative processors is not significant. This is mainly due to the lack of associative processing hardware. The work so far done in the area can be broadly classified in two categories.

In one of these categories, much effort has been given to program associative processors at low level machine oriented languages, such as, assembly languages in simple mnemonic form or at microprogram level. An example of such machine oriented assembly language developed for STARAN, is APPLE [115-116](Associative processor Programming Language). Attempts have been made to extend some higher level languages, embedding the APP instructions, to support the operations of a specific APP hardware. Examples of such extended languages are JOVIAL[117] and PL/1[118].

In the other category, the Associative Processors are simulated either to demonstrate the feasibility of an associative processing hardware or to eliminate the expensive hardward altogether. These include AMPPL[119] (Associative memory Parallel Processor Language), APL[120,121] (Associative Programming Language), ASP[122] (Association-storing Process), LEAP[123,124] and TRAMP[125].

As all of these simulations are implemented on a conventional serial computer, the natural parallelism of the APP is lost - moreover, the content-addressability of these simulations is achieved by hash-coding, hence the search capabilities of an associative processor is restricted to simple equality search. Thus it can be seen that such software simulations are totally inferior to the envisioned hardware.

### 3.2.6. Applications:-

Numerous applications[137,138] ranging from commercial to military to scientific are suggested for implementation using APP systems. Some of these applications are:

#### 3.2.6.1 File Maintenance and Data-base Management

This includes[139-145] sorting, inventory control[176], table-lookup and telephone-directory services.

#### 3.2.6.2. Pattern Recognition:

This includes pattern and character recognition[146] and image processing [147-150].

#### 3.2.6.3. Information Storage and Retrieval

This includes[151-159] on-line data retrieval, cross-retrieval, catalogue searching, technical information retrieval and current-awareness services.

#### 3.2.6.4. Translation:

This includes language translation[160], code conversion[161], data compression and de-compression[172].

## 3.2.6.5. Military Application:

This includes Radar-track correlation[162], Radar-data processing[163], guidance and control[164].

## 3.2.6.6. Miscellaneous Applications

Some of these include Air Traffic Control[165], weather forecasting[166] and control functions[167-170] in computer.

In addition to use of associative memory as an associative processor, it can also be connected with a general purpose computer. The various possibilities of such configurations[47-49] are

1) a peripheral device
2) multi-processor
3) special I/O search unit

## 3.3. Research at Brunel University

Presently, research work of the APG Group at Brunel University is carried out on two different experimental hardware models[51]. One of which, built by GEC-Marconi, comprises a 32 bit x 128 words fixed record length associate memory array. The other comprises a 12 bits x 128 words associative memory array. This is structured as a Byte-oriented variable record length organization. The BO-VRL-APP is implemented with financial help from S.R.C. The current research interests of the group are in the fields of

1) APP Architecture
2) APP Hardware
3) APP Software
4) APP Applications

In the first two areas research is being carried out with two experimental research vehicles to specify the architecture and the instruction set of an associative parallel processor. Hardware implementation of associative memory, using both MOS technology[103-106] and nand gate structure[173] has been reported. Presently an ACTP contract is being undertaken to implement a Micro-APP[182] using Schottky $I^2L$.

Research in the software for Associative parallel processor is continuing to develop higher-level machine-independent languages. These include set theoretic and Intermediate Associative parallel processing languages.[127-129]

Besides the application of APP in the information storage and retrieval system, the work is being carried out in the following areas.

1) Text compression and decompression[172] unit, using both FRL and VRL memory organisation
2) Stock control[176]
3) Local text editing[171] terminal.

## 3.4 Discussion

In the foregoing discussions it has been seen that the primitive operations, such as, domain addressing, intersection and concatenation of sets are the basic requirements for an information retrieval system. It has also been seen that these primitive operations are intrinsic to an associative parallel processor. This reveals that the implementation of an information retrieval system could be very well supported by an associative processor. The other problem of the retrieval system is the unpredictability of length of fields and records. The byte-oriented variable record length data organisation of the APP could be efficiently employed to resolve this particular problem of the retrieval system. Finally, considering the simplicity and cost effectiveness the 'On-The-Fly' search technique, using BO-VRL-APP is chosen for the implementation of the present investigation on Keyword retrieval system.

The choice of the level of the programming language for the implementation of algorithms of the retrieval system is to be considered next. It is understood that the selection of a higher level language would lead to easy program writing. On the other hand, although the task of writing a program at a low-level language would be difficult and prone to error, the selection of a low-level programing language would provide the maximum flexibility of utilizing all features offered by the hardware system. Moreover, at the time of undertaking this investigation, neither of these programming facilities were available. Considering this to be the first attempt, a low-level associative processing instruction set is chosen for the implementation of a research vehicle for the retrieval system. It was decided that an appropriate set of associative processing instructions for the BO-VRL-APP would be first specified. These specifications could then be used as a basis for the development of algorithms for the proposed retrieval system.

# CHAPTER 4.

# The Objectives and Programme of Work

## 4.1

In an on-line information retrieval system the requirement of a simple, flexible and fast searching mechanism has been long felt[47-67]. It is well understood that the data-structure in an information retrieval system is inherently associative in nature[151-159]. The association of data in an information retrieval system, using conventional computer hardware, is implemented by several links, pointers and tables. Which often lead to excessive storage locations, unnecessary computations and slow response. The performance of the system worsens when flexibilities in terms of cross-reference, sub-string search and inexact correspondence are introduced. These inefficiencies are due to the fact that information processing needs efficient searching and non-numeric string processing, where as conventional computers are specifically designed for efficient arithmetic operations. These are strong indications[41,151] that an APP-based information retrieval system could achieve better performance when compared to its conventional counter part. But unfortunately, research in either information science or associative parallel processing is not well established to provide exact specification of an information retrieval system. Hence to bridge these gaps an inter-active experimental on-line retrieval system is proposed in this report. This proposed system is implemented with a Byte-oriented variable record length associative parallel processer (BO-VRL-APP) and utilises on-the-fly searching techniques.

## 4.2. Advantages of APP based retrieval system:

The major advantages of an APP based retrieval systems[41,151,152] are:

i) simple data-structure

ii) flexible search mechanism

iii) Faster response

iv) Lower system and development cost

### 4.2.1 Simple data-structure:

The content addressability of the proposed system does not impose any constraints to adopt a strictly defined transformation relation between logical and physical data. On the other hand this makes it much easier to map logical data-structures into their physical representation within a APP based system[151].

This eliminates any form of links and pointers. Since no extra storage location is required for links, pointers and directories, the estimation of requirements of storage is much simpler. And this also allows better utilisation of storage media.

This simple data-structure does not include any hierarchical structure, nested with links and pointer. Thus during implementation of the data-base hardly any preprocessing is required to generate and maintain a sophisticated addressing scheme. The same argument is also valid for file maintenance operation, where no complicated pointer modifications are required during update operation. Thus, it is easier to enter a new record in the data-base and is equally simpler to delete any existing record from it.

### 4.2.2. Flexibility:

Due to content addressing, the searching of data-base is not restricted any predefined primary or secondary Keys. Thus all search-Keys benefit from equally efficient searching. Alternatively a sub-string search can be easily performed on an entire data-base(KWIC). This results in an extremely flexible retrieval system, particularly, in the case of cross-references. Moreover, the data structure can be easily traversed and modified.

### 4.2.3. Fast response:

Content addressability leads to a simpler search mechanism. It does not require any complicated address computation and also eliminates unnecessary link-tracing. This reduces response time to a large extent. Moreover, the hardware is specially designed for high data rate, fast searching and efficient string manipulation. The facility is further augmented by high degree of parallelism of operations. Hence the proposed system is expected to provide a faster response time.

### 4.2.4. Cost:

Content addressability and parallelism yields more powerful instructions. These can eliminate many conventional routines, which are composed of low-level instructions. Moreover, the burdens of house-keeping programs are much more reduced. It also leads to a simpler software to be developed for the retrieval system. Thus the cost of software development is less expensive. Apart from this, it is expected that, a low-cost micro APP would be available in the near future. This indicates that, all features of the proposed system could be implemented at a reasonable cost.

## 4.3 Objectives:

The main objectives of the proposed work are to evaluate these indications. To prove validity of the claims that APP can support efficient and flexible text searching, a comparitive evaluation system would be constructed. This would furnish the necessary cost/performance statistics and the experiences of this experiment could lead to a tentative specification for associative information retrieval system.

## 4.4. Research Programme:

To fulfill the aims of the proposed work,research would be carried out according to the following programme.

### 4.4.1. System Design:

This would involve design and development of an on-line retrieval system to establish a research memory array. The system would enable successive records of selected fields of an Inspec data file to be transferred, character-by-character, to a search unit, which would store the search profile. The search unit would incorporate suitable buffering to enable matching records to be filtered out to an output file. Scanning of the input file would continue until an end-of-record mark is detected.

### 4.4.2. Searching Strategies:

An associative information retrieval system could support a number of different types of searching criteria; these are: equality, greater than, less than, between limits, maximum, minimum, most or least frequent and many other types. In this work a simple equality search is proposed, which includes boolean selection, Quorum and threshold searches on both word and text fragments.

## 4.4.3. System Evaluation:

Until now, sufficient information to substantiate any performance figure of an associative retrieval system is not available. But there are indications that the new system may have some superior performances over the conventional IS & R system. Hence the main aim of the present study would be to isolate the domain of problem area where this new system is most effective and also to locate its shortcomings. To evaluate these, performances of the proposed system would be compared to its standard counter parts (such as tree structures and inverted list etc.). The area of this comparative study would include

1) data-structure
2) total storage requirement
3) software
4) Instruction counts
5) flexibility and error tolerance
6) speed of response
7) Cost effectiveness

## 4.4.4. System Implementation:

These include programming, coding, testing and debugging of software to

1) implement control program to simulate on-line retrieval system

2) Handle the input and output files and the transfer of records between them.

3) Monitor running programs to generate evaluation statistics.

4) Display pertinent data (especially associative memory maps) for debugging and demonstration.

### 4.4.4.1. Algorithm Development

Design, coding, testing, debugging and modification of algorithms to implement the chosen searching strategies.

### 4.4.4.2. System Evaluation:

Operation of the on-line retrieval system with inspec data file to generate comparative evaluation statistics.

### 4.4.4.3. Algorithm Improvement:

Inter-active modification of algorithm to improve system performance.

## 4.5 The program of present work;

So far the advantages and flexibilities of an associative retrieval system have been discussed. But unfortunately, at the present time, no established specification of an APP is available. Hence, before designing an associative retrieval system, it is required to specify an APP system with the help of a research vehicle. To facilitate this, an inter-active experimental set-up to simulate byte-oriented variable record length APP is to be implemented first. This experiment would provide

a) Information responding data and instruction format.

b) A specification of Associative processing instruction set.

c) Micro-programs for the control Unit of APP system.

Thus it was decided that the present work would be divided into two major phases.

Phase I:- At the time of undertaking the current investigation, the BO-VRL-APP in development within APG was not sufficiently well specified to form the basis of the proposed system. Hence in this phase an interactive experimental set-up would be developed for simulating a BO-VRL-APP system. This would consist of hardware emulation of associative memory unit and software simulation of the remaining components of associative parallel processor. The hardware emulation[52,173] of the associative memory unit would comprise

1) AMA (associative memory array)

2) WCL (word control logic)

3) BCL (Bit control logic)

and 4) Data routing registers.

The software simulation[52] would comprise

1) Micro-order generation logic

2) Control Unit

3) Program store

and 4) I/O facilities

Experiments would be carried out to generate micro-order sequences to

1) prove the logical operation of the BO-VRL-APP[52]

2) test the feasibility of the proposed API[52] (Associative processing Instructions)

3) Consider modification of the logical structure and/or API before final specification.

4) achieve a precise, unambigious specification for the API.

5) estimate cost and performance statistics of practical BO-VRL-APPs.

<u>Phase II</u>:- On the basis of the results obtained in the Phase I of this work, a research vehicle to simulate an associative information retrieval system

would be constructed in this phase. Experiments would be carried out to demonstrate the feasibility of an associative retrieval system. This would also be employed as a useful tool to develop, varify, debug and improve the algorithms to implement the chosen searching strategies.

# CHAPTER 5.

## An Experimental Setup for the Simulation of a Byte-Oriented Variable Record Length Associative Parallel Processor.

### 5.0 Introduction:

The advantages of using Associative Parallel Processors (APP), particularly in symbol processing, have been indicated by many workers.[47-67] From these works a remarkable similarity of the basic system structure of associative parallel processors is observed. But unfortunately associative hardware of any sophistication has been always difficult to obtain. Hence the lack of first-hand experiences of using associative system has hindered further progress in research. In attempting to solve this problem, a number of simulation systems have been devised[117-133]. In general, most of these systems are very crude in comparison to the hardware structure of associative parallel processors. The software solutions of APP are usually implemented by either hash-coding processes or complex list structures on conventional serial machines. These have restricted the potential searching capabilities of associative parallelprocessors to a simple equality search and they do not have hardware support for some very important features such as parallel access and multiple match resolution. Thus in terms of capability, the simulations are totally inferior to the envisioned hardware. They do not provide anything close to a realistic associative processing environment nor the means to evaluate such an environment. Lea[52,54] and Wright [57] indicated the urgent need for hardware research vehicles to carry out further studies on experimental evaluation of associative parallel processing systems. It is expected that the role of such experimental research tools would be to allow the system designer, application engineer, software engineer and user to collaborate in the future progress in these fields.

## 5.1 Architecture of the Associative Parallel Processor:-

To assist the Associative Processing Group (APG) of Brunel University in carrying out further investigations on associative processor architecture, hardware, software and applications, Lea[52] proposed a generalised associative parallel processing system. The schematic block-diagram of Lea's associative parallel processing system is shown in Fig 5.1 This includes an associative memory, input/output unit and communication facilities which are under stored program control. The major differences of this type of architecture from the conventional system are

i) The program instruction and data are stored in physically seperate units.

ii) The data are accessed by content addressing rather than by conventional location addressing.

iii) The arithmetic and Logic Unit and data store of the conventional system are replaced by a single associative memory, where data are processed in-situ within the storage unit without transfer to an independent processing unit.

In his proposal Lea[52] suggested that the proposed system would be initially used for experimental evaluations and improvement of new system design

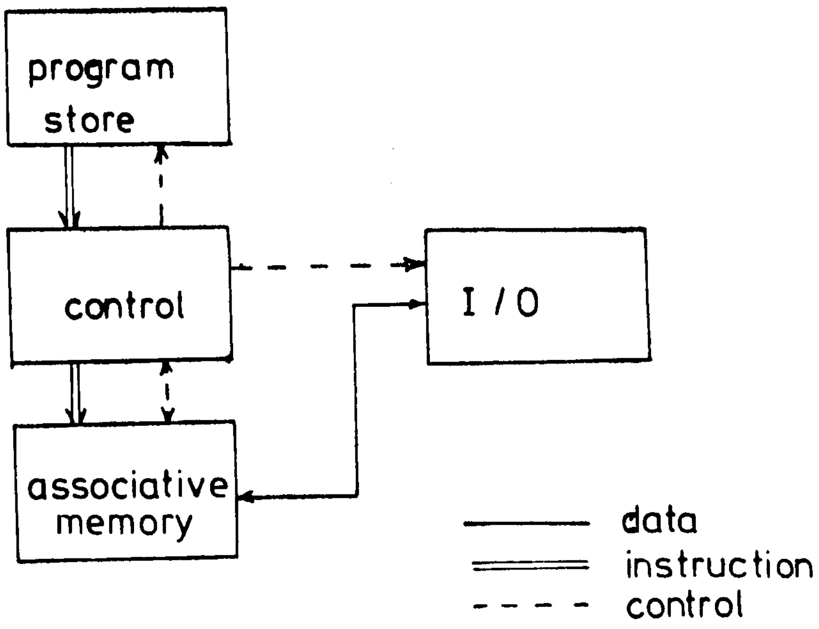


FIG. 5.1. Block diagram of a BO-VRL-APP.

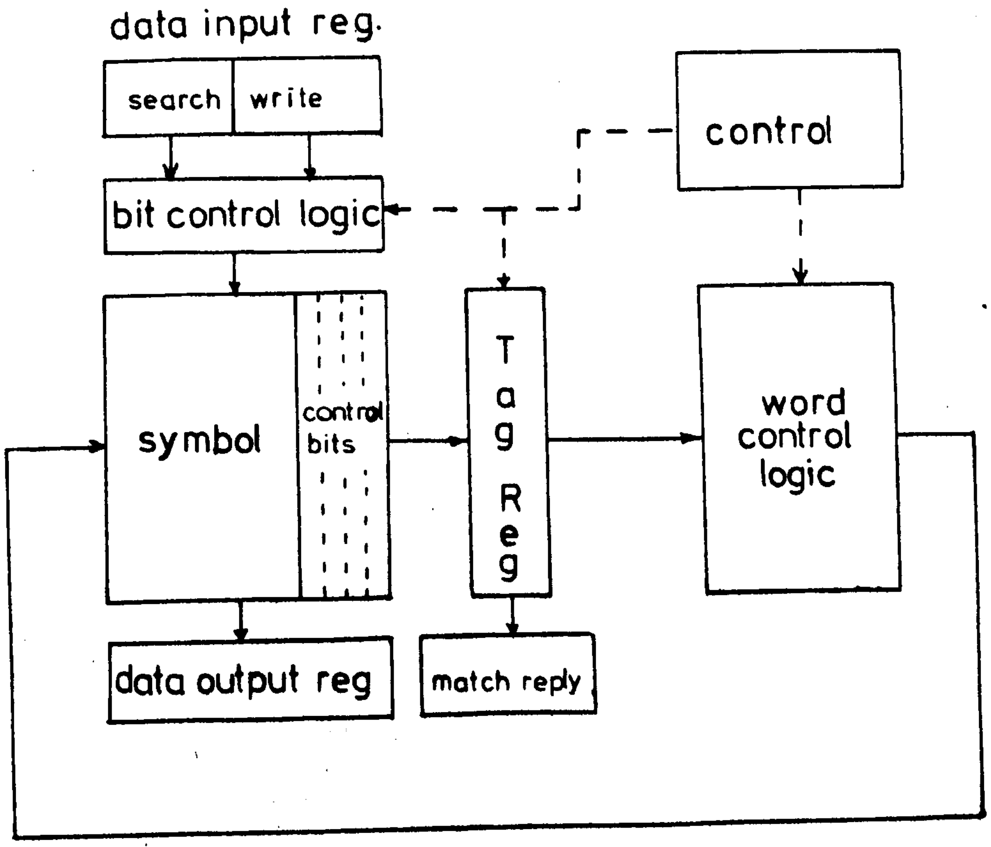


FIG. 5. 2 Associative memory unit.

concepts. It was also intended that the practical investigations should be restricted only to the exploratory phase of associative computer system design. From Fig. 5.1 it is seen that the functions of all consitituent blocks of the proposed system, excepting associative memory, are similar to conventional systems. Therefore no fresh attempt would be made to implement the whole system from scratch, rather a general purpose computer system would be used to simulate these conventional elements. Hence the proposed system would take the form of a prototype design, where a hardware associative memory array would be emulated. This emulation would consist of hardware, software and dedicated minicomputer. A two way interface unit would be included to facilitate communication between the hardware and the controlling system.

To enable further discussion, a brief introduction to the associative memory unit is included in the following sub-sections.

### 5.1.1. The structural organisation of an Byte-oriented variable record length associative parallel processor BO-VRL-APP.

The block diagram of the associative memory Unit is shown in Fig 5.2. The memory module consists of three basic units.

i) Memory array
ii) Address Unit
iii) Control Unit

#### 5.1.1.1. Memory Array:-

The associative memory is a two-dimensional array of identical cells as shown in Fig. 5.3. Each cell, one bit of processing element, can perform the functions of a read-write memory cell and in addition contains sufficient logic to compare its content with the corresponding bit of external data-input register. Each word-row of the byte-oriented VRL memory array comprises of twelve cells for storing a byte of information and forms a complex symbol. These complex symbols are partitioned into two fields:

i) symbol field

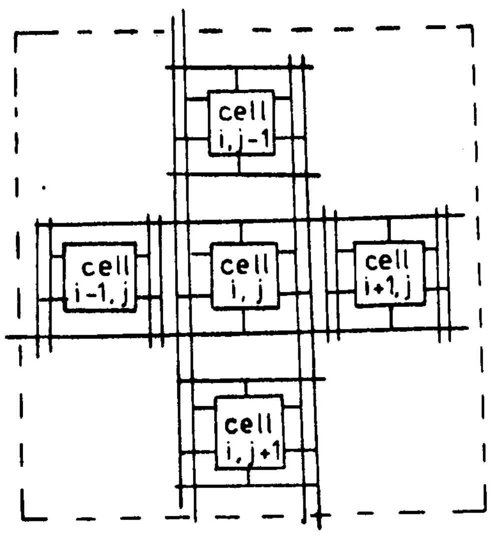
ii) control bit field

The first eight bits (a byte) stores alpha-numeric symbols and the remaining four bits store control bit informations. These control bits are used as either temporary low-level markers or symbol delimiters to provide a means for efficient symbol manipulation.

The memory array is word-organised, that is, twelve-bit complex symbol are connected to parallel input/output highways. All word-rows and bit-columns of the memory array can be accessed in parallel. Particular combinations of rows and columns can be selected by the addressing unit. A single tag bit is provided to indicate an exact match of the content of each complex symbol to the comparand. The tag bit contributes to the match reply line, which is common to all complex symbols and provides feed-back information from the memory unit to the control unit.

## 5.1.1.2. Addressing Unit:-

The addressing unit comprises of two logically seperate units. These are



| Symbol | Control-bit |
|---|---|
| 8-bits | 4-bits |

FIG. 5.3. Associative memory array,

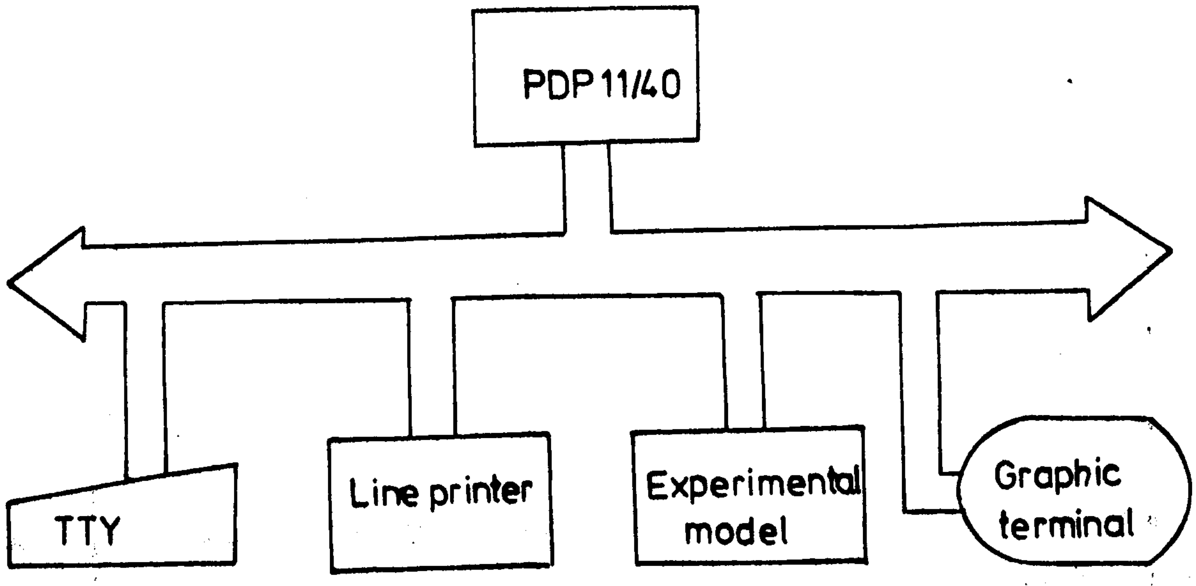


FIG. 5.4. Experimental set-up

i) Bit control logic

ii) Word control logic

Bit control logic:

A word within the memory array is accessed by its content. The contents of the data input register are composed of two complex symbols. One of those is the comparand, which is used for locating the pertinent words. The other is input data, which replaces the old contents of the accessed words. The bit control unit provides an automatic selection of desired bit-columns for both compare and write operations. The function of the bit-control logic, which is local to the memory unit, is to provide a proper set of data to the input high-way of the memory array under command of the control Unit.

Word control logic:-

The word-control logic includes a tag-register, which provides a link between search and read/write operations. Each bit of tag register is associated with a 'word match' line. A successful search operation on a word is marked by setting the corresponding bit of tag register. The read/write operation can now be performed on those words which either matched or mismatched on the preceding search operation. The provision for interw$\overset{o}{\cancel{a}}$rd communication between neighbouring words is also included. An additional feature of isolating and resolving a group of words is provided by run-generation logic.

An asynchronous control system is used for high-speed match-resolving operation. This match resolver, which is local to the memory array, provides an automatic facility for self and neighbour addressing of cells.

### 5.1.1.3. Control Unit:

The control unit consists of a synchronous control system. This is a medium speed indirect control. It consists of two sets of micro-orders, which provide a local autonomous control. These are:

i) Static micro-orders
ii) Dynamic micro-orders

i) Static micro-orders:-

The static micro-orders, as the name suggests, are a set of micro-orders which do not change during execution of an instruction. These specify the domain modification options of an Associative Processing instruction.

ii) Dynamic micro-orders:-

These are sequences of a set of micro-orders which control different steps of an instruction. The combination of these control signals enables the execution of search, read, write, propagate and run operations on complex symbols.

## 5.2. Approach for System Implementation and Objectives

The byte-oriented variable record length associative parallel processor of Brunel University has been described earlier. The proposed associative memory unit consists of a 12 bits x 32 words memory array. A provisional specification was proposed by Lea[52]. As the architecture of the BO-VRL-APP was in embryonic stage, some experiments were required before finalization of the specification. It was considered undesirable to proceed with the implementation to the provisional specification because it might lead to some inflexibilities in the system. On the other hand, an inter-active experimental approach of system evolution was much more attractive because the system configuration could be upgraded until full capability of the hardware was exploited. On the basis of this argument it was decided that an experimental set-up would be developed to derive a complete and final specification for a BO-VRL-APP system. This specification would then be utilized as a basis for hardware design of the control unit of a BO-VRL-APP system.

Before devising an experimental set-up for above purpose, it is worth reviewing the state of development of hardware at Brunel University. From the schematic diagram of Fig 5.2, it has been seen that the associative memory unit comprises

- i) memory array
- ii) Adressing Unit
- iii) Control Unit

Among these constituent elements, investigation on the design of associative memory cells has been thoroughly done by Lea. An associative memory cell, which is capable of performing primitive search, read, and write operations, has been implemented by utilising Nand gates[173]. There was also a fairly good knowledge of the functions of bit-control and word-control logics. Taking these as a basic design guide-line, the hardware for bit and word control logic circuits are made. In the design of bit and word control circuits enough oportunities for minor modifications are left open. However the major uncertainty was felt in the organisation of the control unit. Since modification of the proposed API set was possible, it was not timely to specify a set of sequences of low-level micro-orders. At this point it was realised that an experimental set up would be constructed using existing memory cell, bit and word control logic. Some extra logic would be added to enable a controlling computer to send proper low-level micro-orders. The effects of these micro-orders on the memory array would be monitored by suitable display unit. It was assumed that such an inter-active system would be helpful in developing a full set of API. The basic steps of this experiment would be to roughly define an API and then to derive an algorithm, which are combinations of SMO's and DMO's on paper. This set of micro-orders would be loaded in the buffer area of the controlling computer and sequentially transferred to the experimental model. The memory would be monitored to examine whether or not the initial definition of API is satisfied. Thus in this approach of design there is a possibility of getting important feed-back informations from the experiment. These feed-back informations could be utilized for modification of the hardware to improve overall performance of the system. Once a full set of API is specified, the set of control signals could also be precisely derived and verified on the real hardware system. The entire set of micro-orders would then give a basis for design of the micro-program of the control unit of the APP system. This control unit would eventually be translated into a corresponding hardware version.

The scope of the present project is to build a set-up for above experiment using existing hardware and computing facilities of Brunel University. The objective of this experiment would be to provide a completely specified API set and derive the corresponding micro-orders. The description of the experimental set-up built for this purpose is given in the following section.

## 5.3 The Experimental BO-VRL-APP set-up

The schematic diagram of Fig. 5.4 shows the interconnection of the associative memory with other peripheral units. The PDP 11/40 mini-computer[177] is utilized as the main controlling element. This stores program sequences of APP and also allocates two separate buffer areas to simulate I/O channel of the model. The mini computer is also employed to control the model explicitly by sending control signals down to micro-order level.

All man-machine communications are performed by the console tele-typewriter. The graphic terminal[178] (GT40) continuously displays the associative memory map, that is, the current contents of associative memory array. The line printer is employed for high-speed hard copy of the information displayed by the graphic terminal. The details of the hardware and interface descriptions follow:-

## 5.3.1. Hardware descriptions of the Experimental set up

The block diagram of the associative memory unit of the experimental hardware is shown in Fig. 5.5. The descriptions and functions of different hardware blocks are given below.

### 5.3.1.1. Associative memory cell:

A logic-circuit diagram for an associative memory cell is given in Fig 5.6. These cells are implemented by using Nand gates and reported by Lea. This report[173] also includes all relevent design consideration for practical implementation of associative memory cells. Three basic operations that can be performed by each memory cell are

i) Search

ii) Read

iii) Write

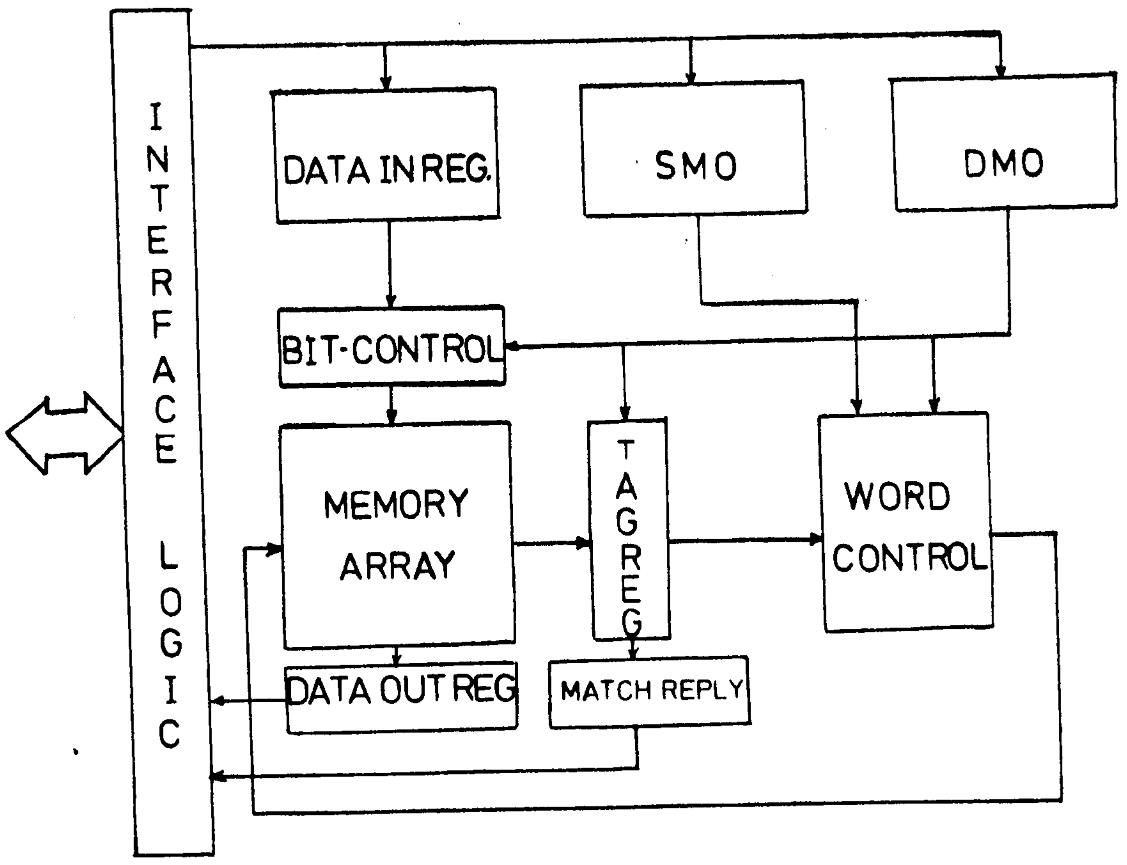


FIG. 5.5. Memory module

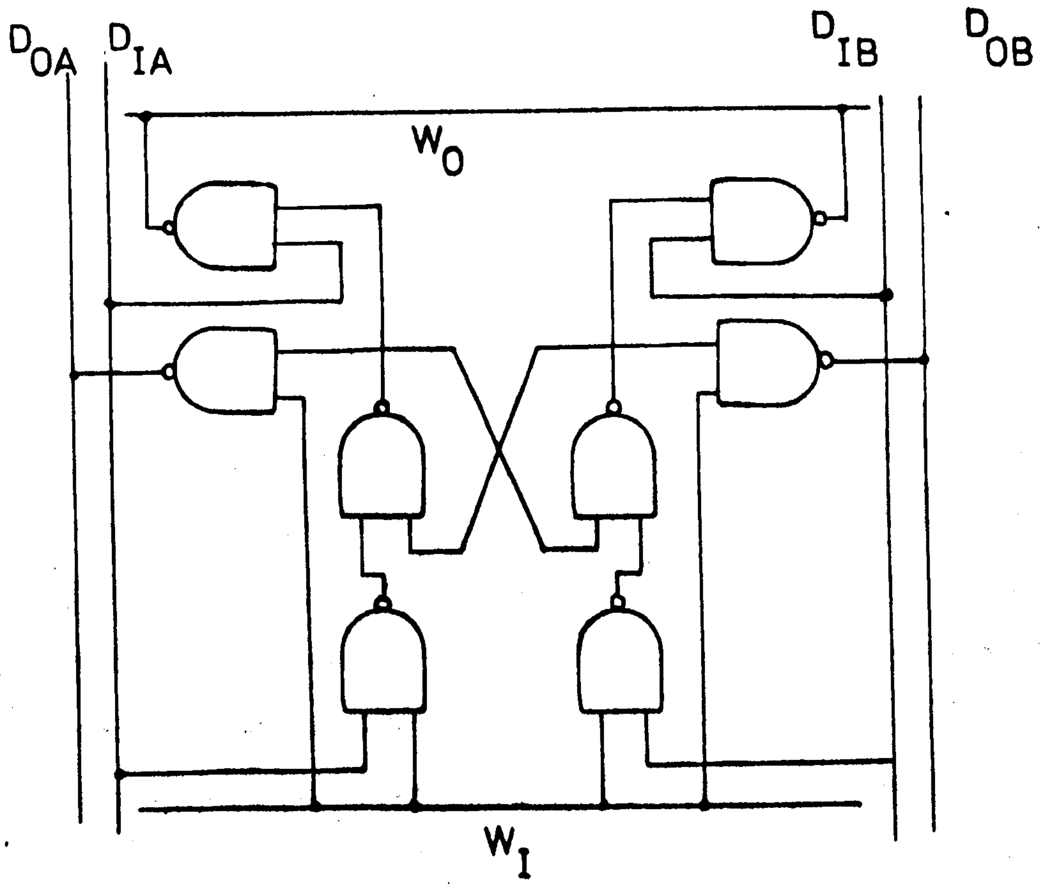


FIG. 5.6. Associative memory cell

Search:

In this operation all cells individually compare the logical content of their memory element with the corresponding information on lines $D_{1A}$ and $D_{1B}$. When the result of comparison is successful, the match output '$W_O$' goes to a logical high.

Read:

During read operation data input lines $D_{IA}$ and $D_{IB}$ are held at stand-by mode and word select line $W_I$ is enabled. Thus the information stored in the cell is made available at $D_{OA}$ and $D_{OB}$ lines.

Write:

In the write operation, input data are applied to $D_{IA}$ and $D_{IB}$ lines, the information is written into the cell by enabling $W_I$ line. When the input data is in the stand-by mode, that is, both $D_{IA}$ and $D_{IB}$ are held at logical zero, the previous contents of the cell are not altered by the execution of write operation.

## 5.3.1.2. Memory Array

The Fig 5.3 shows an associative memory array. The two-dimensional memory array consists of identical associative memory cell as described above. Common row and column connections are also shown in the Fig 5.3 Any combinations of row and column of the associative memory array can be accessed by the independent word- and bit- selection unit respectively. The exact match condition of selected colums in each word rows is indicated by the match output line.

## 5.3.1.3. Micro-order Register:-

As discussed earlier, the experimental model is controlled by a conventional computer. All control signals (micro-orders) are transferred from the controlling computer to the hardware. These micro-orders are needed to be locally stored. Hence two buffer registers, called micro-order registers, are incorporated in the hardware. These are

i) Static micro-order register
ii) Dynamic micro-order register

Static micro-order Register:
It contains a set of eight micro-orders which do not alter during execution of an instruction. The contents of SMO Register are shown in Fig. 5.7 (a)

Dynamic Micro-order Register:
The contents of the dynamic micro-order register are shown in Fig 5.7 (b). The dynamic micro-orders are a collection of sixteen low-level control signals issued to the hardware.

These DMO signals can be further partitioned into two categories
i) BC DMO's (Bit-control Dynamic micro-orders)
ii) WC DMO'S (Word-control Dynamic micro-orders)

The function of the bit control DMO's is to enable the bit control logic, and similarly the WC DMO's activates the word control logic.

The execution of an instruction is accomplished by sequencing an appropriate combination of micro-orders at different time slots.

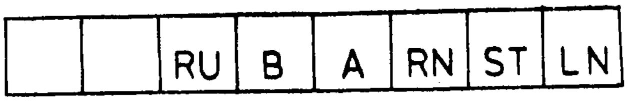


FIG.5.7(a). Static micro-order register

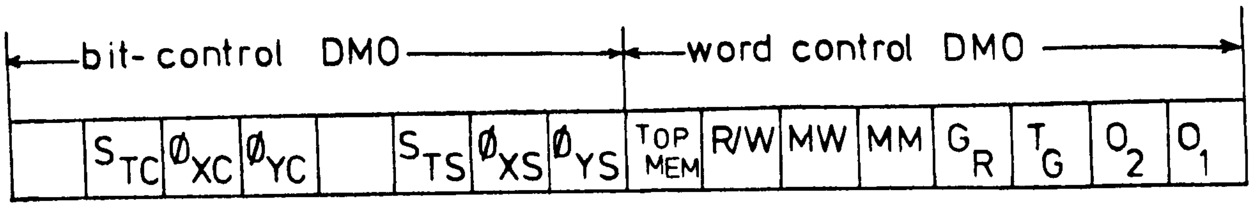


FIG. 5.7(b). Dynamic micro-order register

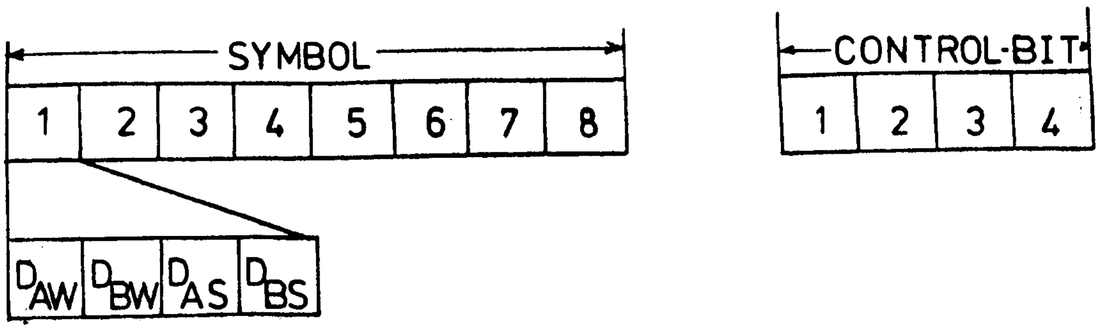


FIG. 5.8. Data input register

The descriptions of micro-orders could be found in APPENDIX A.

### 5.3.1.4. Data Input Register:-

It (Fig. 5.8) holds a twelve-bit complex symbol, which consists of alpha-numeric symbol and control bits. Corresponding to each bit position a four-bit code, namely $D_{AW}$, $D_{BW}$, $D_{AS}$ and $D_{BS}$, is allocated in the data input register. The bits $D_{AS}$ and $D_{BS}$ contain the information to be searched within the associative memory and the bits $D_{AW}$ and $D_{BW}$ contain the information to be written into the associative memory array. The code combinations used for write and search operations are given in Appendix A.

### 5.3.1.5. Bit control logic

The Figure 5.9 shows the logic diagram of the bit control unit, The bit-control logic is a simple four to two ways multiplexer circuit. In response to the micro-order signals $\emptyset_x$, $\emptyset_y$ and SI, the bit-control logic selects or masks a set data from the data input register. These selected data are finally applied to $D_{IA}$ and $D_{IB}$ lines of the associative memory array. The signals $\emptyset_x$ and $\emptyset_y$ are used to select search and write informations respectively. The signal $\overline{ST}$ is used to inhibit 'write one' during the clear phase of instruction cycle. This enables it to write sero only on those bit columns which were selected to search 'one' during the search phase of the instruction cycle.

### 5.3.1.6. Tag Registers:-

These (Fig 5.10) provide a link between search and read/write operations. Each bit of a tag register is associated with a word match line '$W_o$'. The

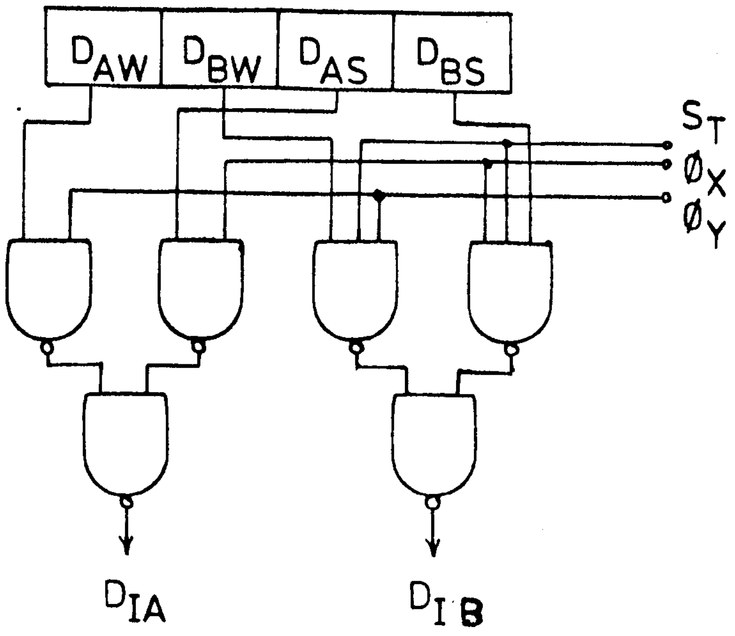


FIG. 5.9. Bit control logic

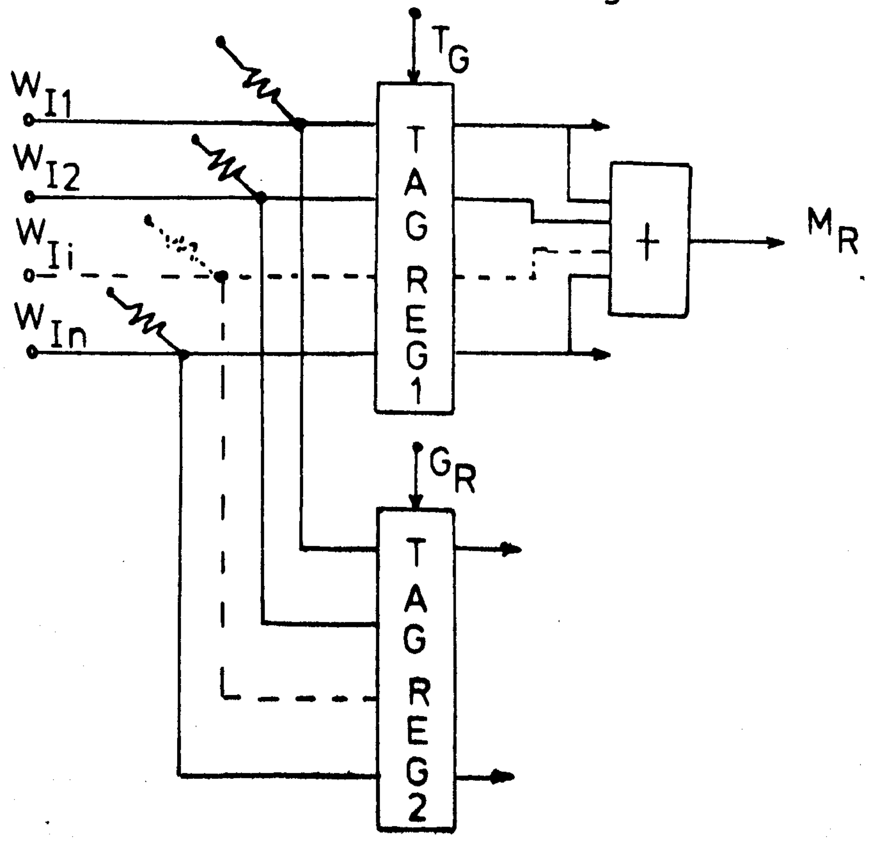


FIG. 5.10. Tag register

result of a parallel search operation is staticized in either Tag Reg 1 or Tag Reg 2 according to the dynamic-micro-order control signals $T_G$ and $G_R$ respectively. The match reply, which is the logical summation of all bits of tag register 1, indicates the presence of at least one tag bit.

## 5.3.1.7. Word Control Logic

The schematic diagram of the word-control logic is shown in Fig 5.11. The word-control logic, in conjunction with contents of tag registers, static and dynamic micro-order, is used to activate a set of word-rows for any specific read or write operation.

Multiwrite:-

The multiwrite line enables all word-rows in parallel. In this case all word-rows can simultaneously take part in any write operation.

Mode Control Logic:

The selection of a set of matched or mismatched word-rows is done by comparing the contents of tag register 1 with the mode signal 'MM'. The comparison of each bit of tag register 1 is performed by a set of exclusive-or gates; the output of which is distributed throughout the word-control logic. When the control signal $O_1$ is enabled, those word-rows with a logical 'one' at the output of mode control gate are activated.

Propagation logic:

The circuit diagram of propacation logic is shown in Fig. 5.11(b) where RN ST and LN (Right neighbour, stright through, and left neighbour) denote the

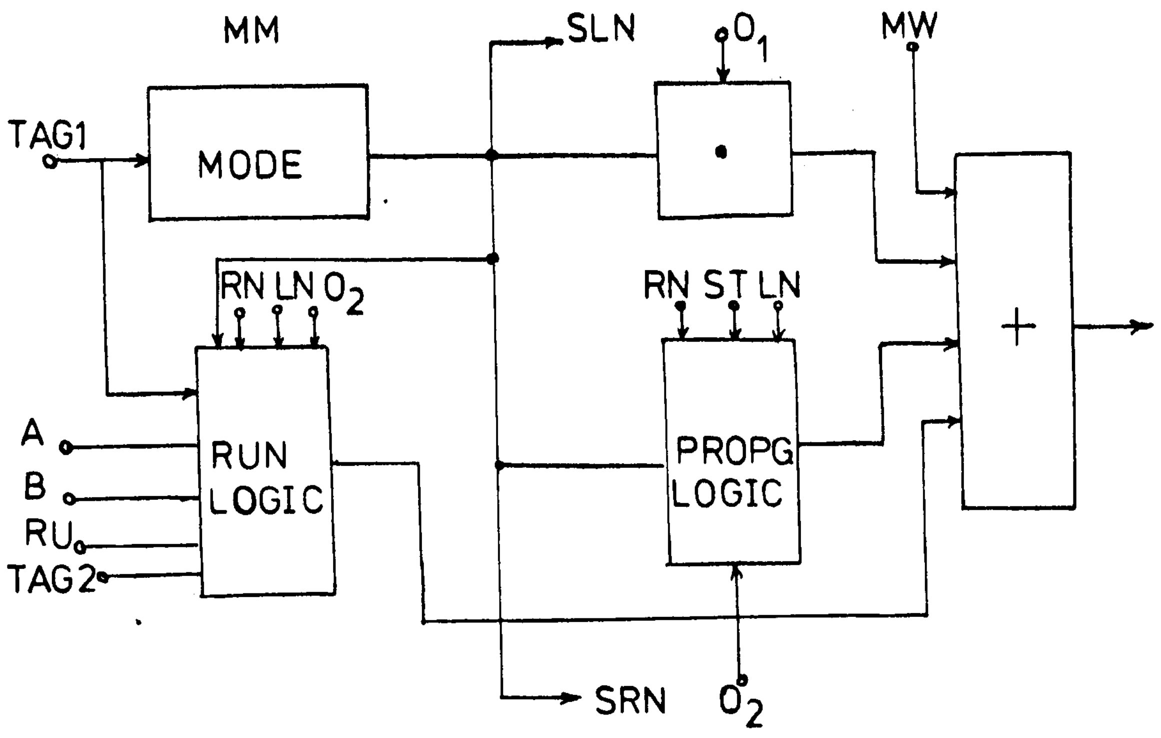


FIG. 5.11. Word control logic

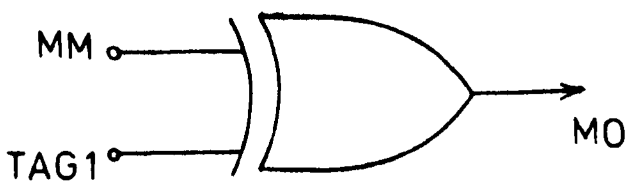


FIG. 5.11(a). Mode control logic

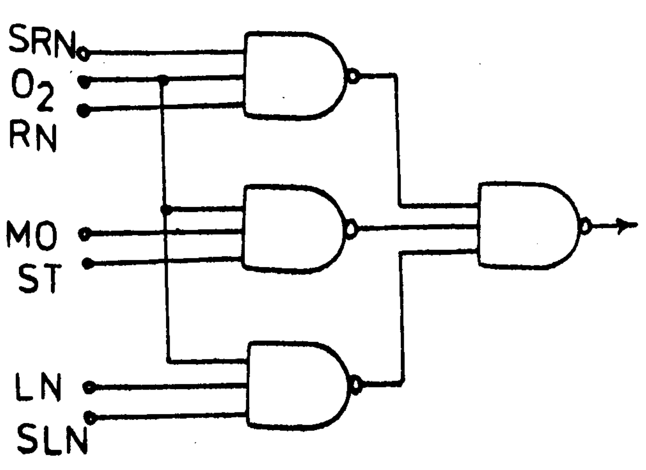


FIG. 5.11(b). Propagation logic

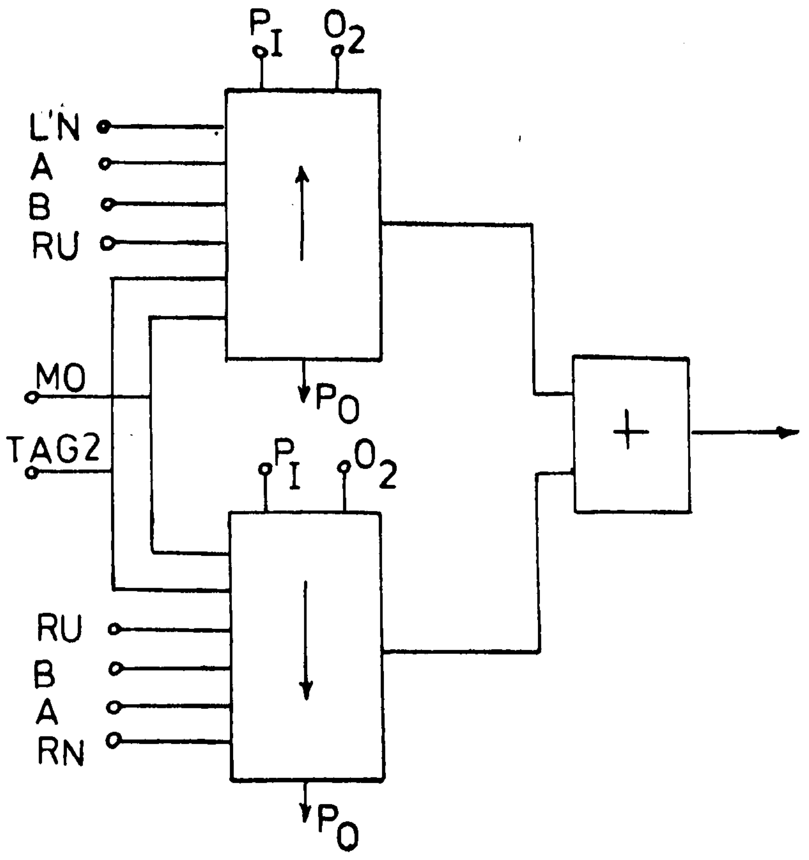


FIG.5.11(c). Run generation logic

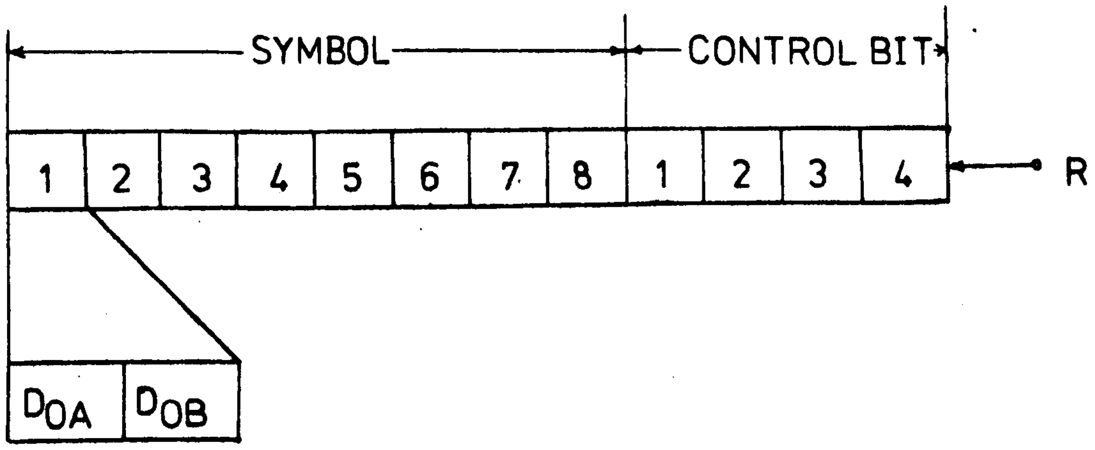


FIG.5.12. Read register

direction of propagations. The information from self and/or adjacent neighbours (SRN, SLN derived from mode control logic) are finally strobed by $O_2$ to select a set of word-rows.

Run Generator:

The schematic diagram of run-generation logic is shown in Fig 5.11(c). This enables to select a group of word-rows for write operation, which depends on the contents of tag registers. There are three different ways of run generations, these are top, bottom and group run - (see Appendix B). Each of these runs could be in either direction. These bi-directional runs are achieved by using two independent sets of parallel carry look-ahead generator trees. The proper control is derived by a combination of run codes, propagation specification and micro-order signals $O_{2R}$ and $O_{2L}$.

## 5.3.1.8. Read Register:-

The figure 5.12 shows a schematic diagram of read register, which hold a complex symbol (symbol + control bit). Each bit columns of the associative memory array has two rails of sense output; $D_{AO}$ and $D_{BO}$. Hence for each bit columns of memory array two bits are allocated in the read register. During the read operation, the contents of line $D_{AOi}$ and $D_{BOi}$ are staticized in the read register by enabling the control signal 'R'.

## 5.3.2. Interface Control Logic

Two way communications between the PDP 11/40 unibus and the associative memory hardware are performed by the DR 11-C[179]. The DR 11-C, a general purpose interface, provides the logic and buffer registers necessary for the program-controlled parallel transfer of 16 bit data between a PDP-11 system and an external device. The schematic diagram of Fig 5.13 shows the interface between the model and PDP 11/40.

### 5.3.2.1. Data Input:

The input highway (to the model) is shown in Fig. 5.13(a), where the low-order eight-bits are reserved for data information and two other lines carry control signals. These control signals provide interface initialisation and data routing informations.

When the signal 'S' is asserted the SMO and DMO counters are reset to the initial state. The signal PV enables output of either SMO or DMO decoder, depending on its logical value. The output of these decoders finally selects a portion of 'Data Input register' for loading input data. The counters (SMO and DMO counters) are automaticially updated at the completion of each data transfer.

### 5.3.2.2. Data Output:

During the output operation (from the model) contents of the read register along with feed-back signal OVA, OVB and Match reply are transferred to PDP 11. The Figure 5.13(b) describes the data output highway. Three transfer cycles are required to transmit the entire output informations. This sequential transfer is done by a multiplexer and a module-three counter. At the end of each transfer the counter is updated for proper data routing.

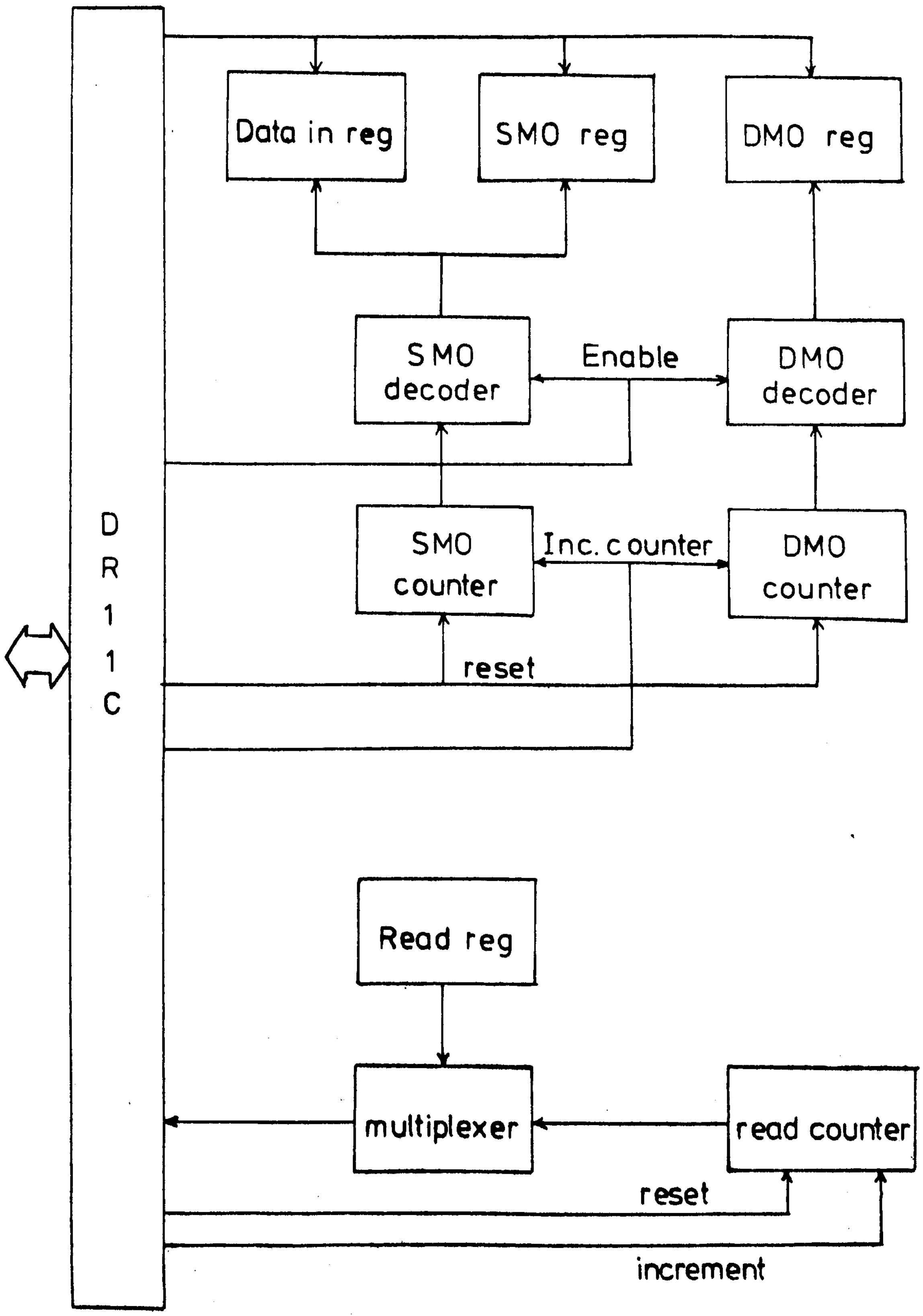


FIG.5.13. Interface logic

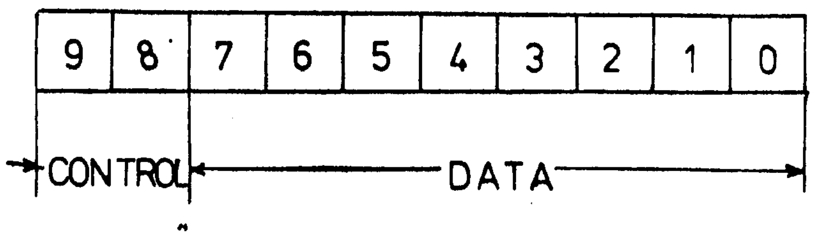


FIG. 5.13(a). Data input highway

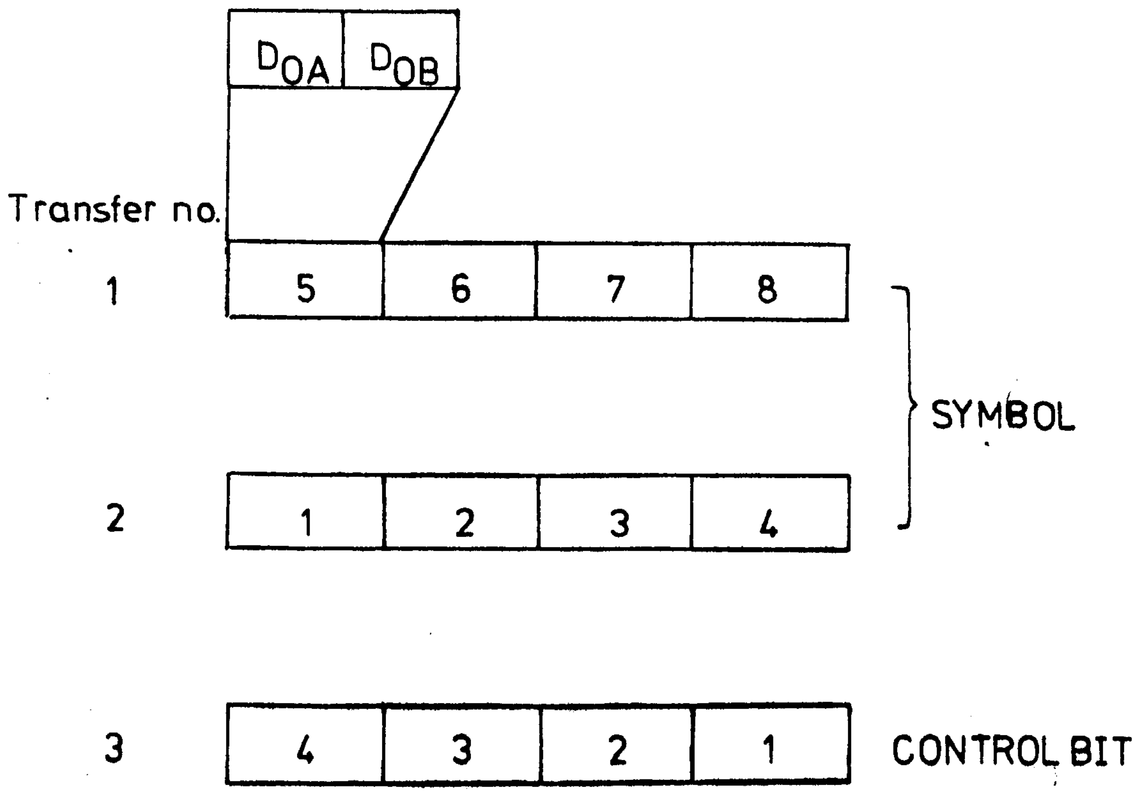


FIG. 5.13(b). Data output

For further descriptions of interface signals Appendix A may be referred to.

## 5.4. Associative Processing Instruction

The experimental hardware model described earlier, can execute a single Instruction on multiple data stream at a time. The API (Associative Processing Instruction) format and instruction execution cycle are described in this section.

### 5.4.1. Instruction Format:-

A generalised instruction format is shown in Fig 5.14. Each micro-instruction consists of function, address and modifier fields.

Function:- This comprises of two sub-fields; these are: op-code and data:

Op-Code: This indicates the nature of operation to be performed, such as read or write.

Data: This sub-field of the Function is interpreted according to the content of operation sub-field. During the read operation it indicates a 16 bit address within the buffer area of the program store, where the interrogated information is to be stored. During write operation it holds the information to be written in the selected words. During 'group run' (see Appendix B) it holds the data for second search operation.

<FUNCTION><DOMAIN ADDRESS><DOMAIN MODIFIER>

<FUNCTION>:<OP CODE><DATA>

FIG. 5.14. Instruction format

Address:- This field of the instruction contains the information to be searched in the initial domain search operation. This search operation is always associated with every instruction, which explicitly selects a domain of word-rows. This resembles the address field of the instruction format of a conventional system.

Modifier:- This field of the instruction modifies the entire addressing mode. The final specifications for the instruction of the byte oriented variable record length APP are included in Appendix B.

## 5.4.2. Instruction Cycle:-

Each micro-instructions (API) within the hardware model operates on a four beat cycles. These are domain search; domain modification (clear option) domain modification (propagate and/or run option); and function (read or write).

The function of the modifier field is to modify the addressing scheme and finally to enable a set of word-rows for further processing.

During the domain search operation, the contents of the address field are considered as the comparand, The result of the comparison is stored in tag registers.

During the clear option, the control bit field and/or symbol bit field of the selected words can be reset (write '0')

During the propagation and run generation, the addressing mechanism is modified to enable a proper operation.

During the last phase, a read or write operation (as indicated by the opcode) is performed.

## 5.5. Steps of the Experiment:

The objective of this experiment is to specify an instruction set for the byte-oriented VRL-APP. This objective could be achieved by an interactive experiment, as stated earlier. The steps which are to be followed during this experimentation are shown in flow-chart of Fig. 5.15, and a brief explanation is given below.

Step 1: API definition - an API is roughly defined.

Step 2: Algorithm development - an algorithm for execution of the API, as defined in Step I, is developed.

Step 3: Timing diagram generation:- A timing diagram for the entire set of micro-orders are generated on the basis of the algorithm developed in Step 2.

Step 4: Derivation of micro-order sequences:- The micro-order sequences are directly mapped from the timing diagram. These micro-order sequences are fed to the controlling computer. These micro-orders, which are stored in appropriate buffers, are the sequentially transferred to the hardware.

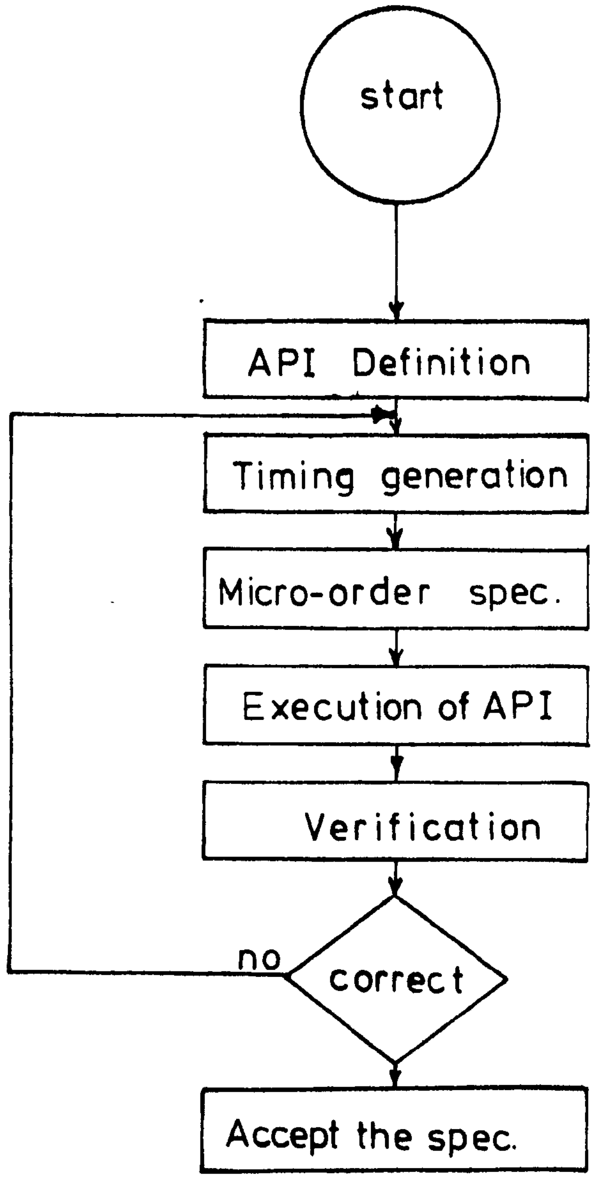


FIG.5.15. Steps of the experimentation

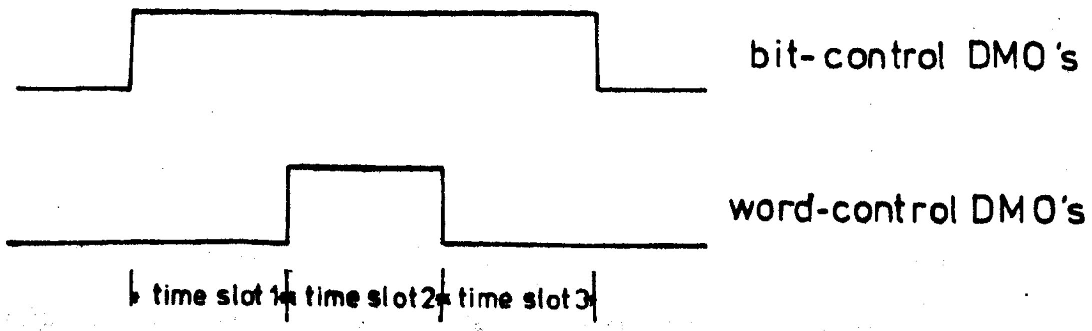


FIG. 5.16. Timing diagram

Step 5: Memory Map:- A memory map is produced to observe the effect of micro-order sequences(result of Step 4) on the contents of memory array.

Step 6: The memory map produced at Step 5 is compared with expected result. If some modifications are required, Step 1 to Step 6 are carried out with proper corrective measures. Otherwise the definition of API (Step 1) and micro-order sequences of Step 4 are accepted.

## 5.5.1. Timing diagram generation:-

The main consideration during the timing diagram generation phase is that all word-control dynamic-micro-orders should be covered by the bit-control DMO's. That is, no bit-control dynamic-micro-orders should change during the presence of a word-control DMO. Otherwise some criticalness of timing may occur, which may lead to an intermittent success of the operation. The solution of this problem for a beat of instruction cycle is shown in Fig. 5.16. The figure shows that three time slots are required for the transfer of every word-control DMO. Where the bit control DMO's are maintained for the entire sequences of T1, T2 and T3 ; the word-control DMO's are only enabled during time slots T2. From this timing consideration it is derived that twelve time slots would be required to execute four beats of any API cycle.

## 5.5.2. Memory Map

All relevant feed-back information from the hardware, which is required for any interactive experiment, is available from the memory map. The memory

```
 DATE: 8-APR-76

 TIME IS 13:03:40

API: W @01111110 0000 G XXXX 0000 0 000 N
                 STATIC MICRO-ORDER
     R CHARACTER  CB  MCCM C USD RR
                      WCBM       12
     0 @01111110 0000                 :FUNCTION
       G         XXXX 0000 0 000 00 :ADDRESS
         MEMORY MAP                          DYNAMIC M-O

WORD 1    10000000  0000                    0110011000000000
WORD 2    11000000  0000                    0110011000000100
WORD 3    11100000  0000                    0110011000000000
WORD 4    11110000  0000                    0101010100000000
WORD 5    11111000  0000                    0101010100000001
WORD 6    11111100  0000                    0101010100000000
WORD 7    11111110  0000                    0000000000000000
WORD 8    11001000  1000                    0000000000000000
WORD 9    11001001  0100                    0000000000000000
WORD 10   11001010  0010                    0000000000000000
WORD 11   11001011  1110                    0000000000000000
WORD 12   11001100  0000                    0000000000000000
WORD 13   11001101  1110
WORD 14   11001110  0000
WORD 15   11001111  1100                    READ REGISTER
WORD 16   11010000  0010                    10110110   0000
WORD 17   11010001  0000
WORD 18   11010010  0000
WORD 19   11010011  0000                    TAG REPLY =0
WORD 20   11010100  0000
WORD 21   11010101  0000                    OVER-FLOW1 =1
WORD 22   11010110  0000                    OVER-FLOW2 =0
WORD 23   11010111  0000
WORD 24   11011000  0000
WORD 25   11011001  0000
WORD 26   11011010  0000
WORD 27   10110001  0000
WORD 28   10110010  0000
WORD 29   10110011  0000
WORD 30   10110100  0000
WORD 31   10110101  0000
WORD 32   10110110  0000
```

FIG. 5.17. Memory map

map, that is the current contents of memory array, is continuously displayed on the graphic terminal. The general format of a memory map is given in Fig. 5.17. This consists of

1) current API description
2) current static micro-orders
3) current dynamic-micro-order descriptions
4) Contents of read register
5) Condition code output (such as overflow A and B; Match reply)
6) The current contents of the entire memory array.

This memory map enables to monitor the state of the associative memory array and provides a valid basis for experimental verification.

## 5.6. Software Package:-

A software package is developed to interconnect the model with the rest of the system. This simulates I/O unit, program store and control unit of VRL-APP system. This is also used to convert a micro-API instruction to machine code and to issue the low level control signals to the model. The program is written in Macro 11[180] and runs on the RT-11 F/B operating system[180]. In addition to normal experimental requirements, some extra facility such as initial clear, bulk loading and hard-copy print-out of memory map are also included. All functional operations are grouped into different modes - these modes of operation are:-

1) clear (AM)

2) load (AM)

3) Reload (AM)

4) specify (API and micro order)

5) Process (execute API)

6) Micro Instruction (specification followed by execution).

7) output (hard copy of memory map of console TTY)

8) Fast output (hard copy of memory Map on line printer)

9) Exit (End of session)

When the program is running, it initially waits for a mode control signal. Upon receipt of a mode control command, it starts a particular set of subroutines which corresponds to the mode command. At the completion of a mode of operation (except Exit) the control is transferred to the initial state of the program. A brief description of different modes is given below. The detailed operational steps are explained by the flow-chart in Appendix C.

Clear:- This mode initilizes the system by clearing the associative memory cells and the interface logic. The reset condition of the associative memory is displayed on the graphic terminal.

Load:- This mode requests the user to enter thirty two characters at the terminal. These input characters are first stored in a Load buffer area and then dumped into the successive associative memory wards. The graphic terminal displays the contents of the associative memory after this load operation.

Reload:- The old contents of the Load Buffer are reloaded in the associative memory. The GT displays the contents of the associative memory after re-load operation.

Specify:- During this mode of operation a new set of DMO's and API can be specified and stored in the respective buffer areas. The GT displays these new specifications along with the contents of the associative memory at the time of entering these specifications.

Process:- This executes the API, stored in the API buffers, according to the specified dynamic-micro-orders on the data set contained within the associative memory. The GT displays the API executed along with the dynamic micro-order set and the current contents of the AM after execution of this API.

Micro instruction:- This is a combination of a 'specify' followed by a'Process' mode. The newly specified API is stored and executed on the data stored in the associative memory. The contents of the graphic terminal are similar to that of the process mode.

Output:- The current contents of the graphic terminal are printed on the console tele-typewriter. At the top of this printed hard copy, the current date and time are also logged.

Fast Output:- The function of this mode is similar to that of the output mode. The gain in output speed is achieved by using a line printer instead of a TTY.

Exit:- At the end of an experimental session this mode is entered. This transfers the machine control to the Keyboard monitor of RT - 11 F/B operating system.

## 5.7. Results:-

The experiment is carried out according to procedure stated in section 5.6. As an illustrative example the following API is chosen.

API: W B 0001 A 1000 0010 0 00D N

Step 1. Definition of API:-
A symbol 'A' with control bit CBI would be searched; the control bit CBI of all matching words would be cleared; the propagation would be set for downward direction; finally a symbol 'B' with control bit CB4 would written in the selected words.

Step 2. The algorithm for the above definition of API is given in Fig 5.18(a)

Step 3. The timing diagram is generated from the algorithm of Fig 5.18(a) and is shown in Fig 5.18(b).

Step 4. The micro-order specifications are directly obtained from the timing diagram of Fig 5.18(b). The sequence of DMO's required for execution of the API, as defined in step 1, is given in Fig. 5.18(c).

Search complex symbol; set tag

Clear control-bit field of tagged word

Enable propagation down

Write complex symbol

FIG. 5. 18(a). API definition

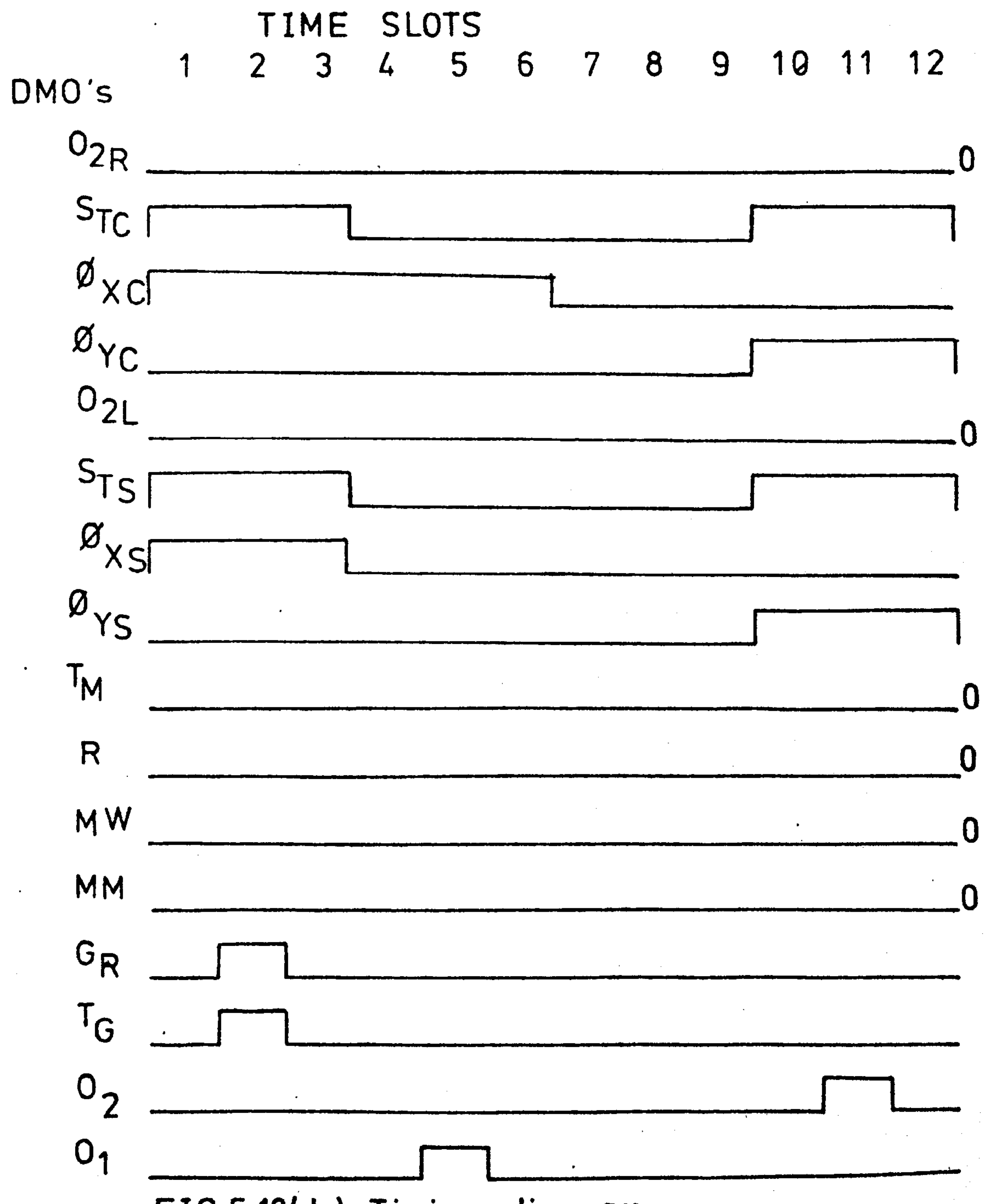


FIG 5.18(b) Timing diagram

DMO's

| Time Slots | $O_{2R}$ | $S_{TC}$ | $\phi_{xc}$ | $\phi_{yc}$ | $O_{2L}$ | $S_{TS}$ | $\phi_{xs}$ | $\phi_{ys}$ | TM | R | MW | MM | $G_R$ | $T_G$ | $0_2$ | $0_1$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 |
| 3 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 4 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 5 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 6 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 9 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 10 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 11 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 12 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Fig 5.18(c). DMO sequence.

```
  DATE: 8-APR-76

  TIME IS 13:29:49

API: W B 0001 A 1000 0010 0 00D N
                  STATIC MICRO-ORDER
     R CHARACTER  CB  MCCM C USD RR
                      WCBM       12
     0 B          0001                :FUNCTION
       A          1000 0010 0 001 00  :ADDRESS
        MEMORY MAP                         DYNAMIC M-O

WORD 1  11000001  1000                    0110011000000000
WORD 2  10000000  0000                    0110011000001100
WORD 3  10000000  0000                    0110011000000000
WORD 4  10000000  0000                    0010000000000000
WORD 5  10000000  0000                    0010000000000001
WORD 6  10000000  0000                    0010000000000000
WORD 7  10000000  0000                    0000000000000000
WORD 8  10000000  0000                    0000000000000000
WORD 9  10000000  0000                    0000000000000000
WORD 10 10000000  0000                    0101010100000000
WORD 11 10000000  0000                    0101010100000010
WORD 12 10000000  0000                    0101010100000000
WORD 13 10000000  0000
WORD 14 10000000  0000
WORD 15 10000000  0000                    READ REGISTER
WORD 16 11000001  0000                    00000000  0000
WORD 17 10000000  0000
WORD 18 10000000  0000
WORD 19 10000000  0000                    TAG REPLY =0
WORD 20 10000000  0000
WORD 21 10000000  0000                    OVER-FLOW1 =0
WORD 22 10000000  0000                    OVER-FLOW2 =0
WORD 23 10000000  0000
WORD 24 10000000  0000
WORD 25 10000000  0000
WORD 26 10000000  0000
WORD 27 10000000  0000
WORD 28 10000000  0000
WORD 29 10000000  0000
WORD 30 10000000  0000
WORD 31 10000000  0000
WORD 32 10000000  0000
```

Memory map 1.

```
  DATE: 8-APR-76

  TIME IS 13:31:53

API: W B 0001 A 1000 0010 0 00D N
                  STATIC MICRO-ORDER
     R CHARACTER   CB  MCCM C USD RR
                       WCBM        12
     0 B          0001                   :FUNCTION
       A          1000 0010 0 001 00     :ADDRESS
         MEMORY MAP                            DYNAMIC M-O

WORD 1   11000001   0000                      0110011000000000
WORD 2   11000010   0001                      0110011000001100
WORD 3   10000000   0000                      0110011000000000
WORD 4   10000000   0000                      0010000000000000
WORD 5   10000000   0000                      0010000000000001
WORD 6   10000000   0000                      0010000000000000
WORD 7   10000000   0000                      0000000000000000
WORD 8   10000000   0000                      0000000000000000
WORD 9   10000000   0000                      0000000000000000
WORD 10  10000000   0000                      0101010100000000
WORD 11  10000000   0000                      0101010100000010
WORD 12  10000000   0000                      0101010100000000
WORD 13  10000000   0000
WORD 14  10000000   0000
WORD 15  10000000   0000                      READ REGISTER
WORD 16  11000001   0000                      00000000   0000
WORD 17  10000000   0000
WORD 18  10000000   0000
WORD 19  10000000   0000                      TAG REPLY =1
WORD 20  10000000   0000
WORD 21  10000000   0000                      OVER-FLOW1 =0
WORD 22  10000000   0000                      OVER-FLOW2 =0
WORD 23  10000000   0000
WORD 24  10000000   0000
WORD 25  10000000   0000
WORD 26  10000000   0000
WORD 27  10000000   0000
WORD 28  10000000   0000
WORD 29  10000000   0000
WORD 30  10000000   0000
WORD 31  10000000   0000
WORD 32  10000000   0000
```

Memory map 2.

Step 5. By using mode 's' of the software the new DMO specifications of Fig 5.18(c) and API are entered. The memory map 1 shows the new specification of the DMO's and API along with the contents of the associative memory array at the time of entering them.

The mode 'P' of the software is entered to execute the API; and the memory map 2 is produced as a result.

Step 6. Comparison of the result with the expected operation is made. Once verified, it is accepted as a final specification.

Repeating the above procedure other API's are defined and corresponding DMO's are derived. The complete specification for the instruction set of the BO VRL APP, thus obtained, is included in APPENDIX B.

## 5.8. Discussion:-

An experimental set up for hybrid computer emulation of the byte-oriented variable record length associative parallel processor has been implemented. This system provides a research vehicle for user-oriented design of VRL-APP systems. The purpose of this investigation was to provide feed-back informations for the varification and improvement of the VRL-APP system design. The experimental investigations were mainly pointed:

i) to achieve a high degree of symbol-processing efficiency over a wide range of applications.

ii) to develop a well balanced and flexible instruction-set compatible with the VRL-APP systems and applications.

iii) to develop a basis for the design of the control unit of the APP system.

As stated earlier, an interactive design approach had been adopted to achieve these objectives. A modular software package has been included to facilitate the experiment . The general instruction format (API) of a VRL-APP system is given in section 5.4.1. These include function, address and domain modifier fields. The primitive operations of an associative memory are search, domain modification followed by read or write function. Extensive studies on read/write operations, with all possible domain modifications, on the data set stored in associative memory array have been carried out. The verification of proper executions of these fundamental operations are obtained from the sequence of memory maps of the associative memory.

The top, bottom and group runs are also verified in downward direction. It is expected that by inclusion of similar hardware, runs in upward direction could easily be implemented.

As an outcome of this experiment, a complete specification for the API set of the BO-VRL-APP along with corresponding micro-program for the control unit are produced.

This has also provided a basis for addition of an extra mode 'T' in the software package. This mode of operation accepts an API from the tele-type writer. The corresponding static- and dynamic- micro-orders are automatically generated by the software package to enable execution of the entered instructions.

Further improvements of the software have been carried out to buffer a set of up to sixteen API's. These buffered API's are then executed sequentially taking one instruction at a time. At the end of the buffer (when all instruction are executed) a switch register option is provided to repeat execution of the set of instructions, stored in the buffer, as an endless loop. An additional switch register option is included to provide hare-copies of the memory maps at the completion of each of these instructions.

Two application studies, using the extended version of the software, have recently been carried out. One of these, an implementation of on-line text editor, is done by Reynold[174]. The other,as reported by Ofulue[175],deals with the conversion of Intermediate Associative Programming language to a macro of low-level API.

These two application studies have demonstrated the flexibility and power of API in symbol processing environment. The following chapter describes another application of VRL-APP, where VRL APP system is used as a parallel search unit of an on-line information retrieval system.

# CHAPTER 6.

## On-Line Associative Retrieval System

### 6.0 Introduction

The discussion of the problems of on-line information retrieval, is mainly restricted to the area of a bibliographical information service of technical or national libraries[1-10]. As a general library deals with large numbers of different types of documents such as printed books, serials, maps, charts, paintings and musical records, it is not at all possible to store its entire collection in a computer system. Even if only printed books and serials are considered, the size of the information becomes so enormous that the storage of actual documents within a computer system is not economically feasible. Here the main purpose of the mechanisation is to locate the physical position of the document. Thereafter picking up of selected documents by an automatic system or manual intervention is a trivial mechanical aspect of the problem. Hence considering the cost of the system, the scope of this work is only limited to finding out the physical location of retrieved documents. Here, in response to a query, complete bibliographical information of all documents, as provided by a conventional card-catalogue file, along with information regarding physical location and current status (whether or not on loan) are provided in a suitable format. A hard-copy printout along with a visual display could also be obtained for future reference.

The main advantages of computer-based on-line catalogue searching systems are flexibility and speed of response. An interactive system could provide extensive cross-references and it is desirable since it allows poorly defined requests to gain precision from the results of subsequent searches. Another attraction of computer-based system is the possibility of SDI and current-awareness services, virtually without any extra effort.

The ultimate success of a system depends on the two major factors:

1) The way data-base is created
2) The mechanism used for searching

Indexing[1-10] attempts to bridge the communication gap between the searcher and the originator of a document. In fact the process of indexing is quite complex. This is because it is often difficult to describe a document by a single index term; and some index terms require to be further divided in sub-groups. Other difficulties arise due to variation in the values of the context of a document in user environment. A simple solution such as Keyword in context may usually lead to a large index, some times larger than the data[28-33]. Until now the process of indexing can be considered as an intellectual exercise. Hence in the following discussions aspects of indexing will be carefully avoided.

The other factor of a searching mechanism is highly dependent on the complex data structure of the file organisation.[35,36,42-45] These complex structures are required in a conventional system to establish an artificial link between Keys and document. As a consequence of this the system becomes inefficient in terms of usage of storage media and unnecessary computation. The natural property of association of attribute/argument of an Associative parallel processor[151,152] can be gainfully exploited in information retrieval applications. The simplest solution would involve the storage of an entire data-base in a large associative memory[158] with minimal data-structure. A subsequent retrieval of documents could be done by a comparison of Keys with the data-base. Currently a large assoicative memory is not an economic proposition. Many people[153-157] have suggested a hierarchy of associative memory organisation as used in a conventional computer system. According to these suggestions, a reasonably large data-base would be sotred in a system similar to Content Addressable File Storage System (CAFS)[153-156]. The CAFS system is a conventional magnetic disc unit with additional logic attached to it for rapid access of relevent data. It is revealed that the initial selection of most likely regions of data could be done by such a system. The final selection of documents would be performed by an array of associative memory. The scope of this work is limited to some investigations on retrieval of documents using an associative memory array.

There are two alternative search organisations using APP, where

1) a part of data- base is held in the associative memory [152].
2) The search data (user profile) is held in the associative memory. (On-The-Fly)[151].

The first organisation has the disadvantage of continued loading of the associative memory but it allows more complex manipulations to be carried out. Though the converse of these advantages and disadvantages are true for the second mothod, it has a primary advantage of cost. Hence, for simplicity, and from an economic point of view, the second organisation is chosen in the current investigation.

## 6.1 ON-THE-FLY Techniques of Searching:

In 'On-The-Fly' methods of searching technique, the records containing indices are passed over the top of a 'parallel search' unit. At the end of each record a test is carried out to examine whether or not the record, just passed over the search unit satisfies the searching criterion.

A simplified block diagram of the 'On-The-Fly'searching system is given in Fig.6.1 The user of the on-line retrieval system enters his profile through the terminal, which is subsequently stored in the associative memory of the parallel search unit. A character of information from the search file (Index File), as appearing at the read head of rotating disc, is passed over the parallel search unit and is also stored in a temporary buffer. When an 'end-of-recore' mark is detected, a special routine examines whether or not the current record satisfies the user requirement. If the result is successful, the contents of the temporary buffer are transferred to an output file, otherwise, the temporary buffer is cleared. This process is repeated until the 'end-of-file' mark is detected. Upon detection of an end-of-file mark, further searching is stopped and the resulting statistics and outputs are produced.

The parallel search unit mentioned here uses an associative parallel processer. As stated above, the user's profile is stored in associative memory for comparison against data-base. These profiles consist of a set of Keywords or indices. It is implied that the associative memory should have efficient

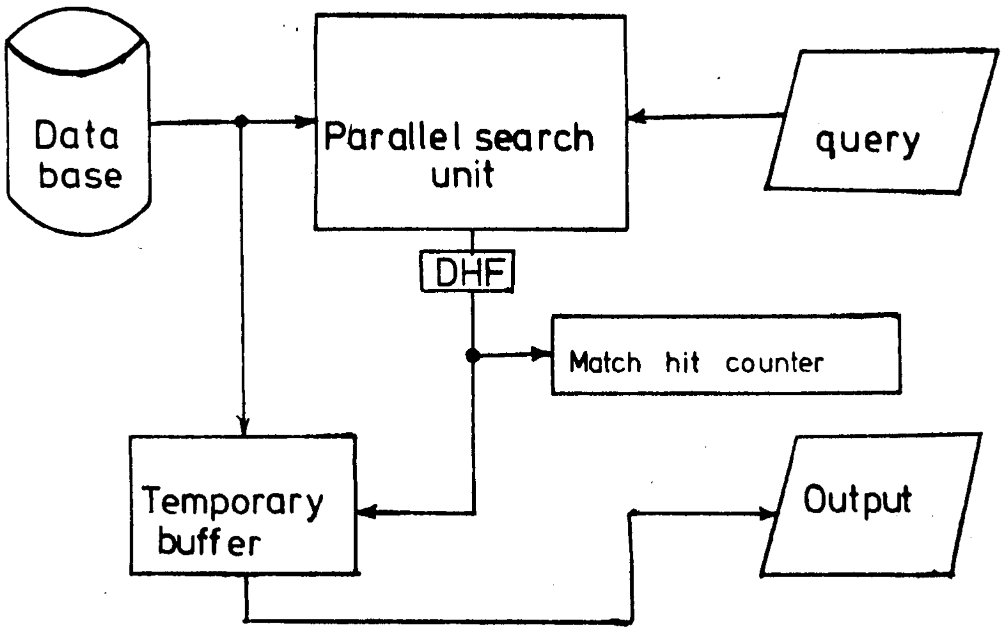


FIG. 6.1. ON-THE-FLY searching unit

1. (SOR) Document Identifier, KEY 1, KEY 2, KEY 3, KEY n, (EOR)

2. (SOR) Document Identifier, Document Name (EOR)

3. (SOR) KEY i, Document Identifier 1, ... Document Identifier N (EOR)

4. (SOR) Document Name (EOR)

<u>Fig. 6.2</u> <u>Data Formats</u>

capabilities of string manipulations. And the other requirements[52] of an associative processer are that no restrictions on the length of Keys should be imposed and it must incorporate a flexible serial search of input character string. This implies that a byte-oriented variable-record length APP is best suited for this application. In the following section variations of search criteria are discussed.

## 6.2 Search Criterion:

The users of an on-line retrieval system are generally allowed to search on the data-base with a user defined profile. These profiles consist of a set of Keywords or indices. A number of different search types[82,152,157] are permitted in a profile: these could include equality, greater than, less than, between limits, maximum, minimum and a number of others. In this report a simple equality search is considered. The different variations of equality searches are described below:

### 6.2.1 Simple Equality Search:

In its simplest form, the profile contains only one Key. The records which contain (or do not contain) this Key are retrieved. The profile can be extended to specify more than one Key; but the identification of a record is made by a simple match hit criterion of presence (or absence) of either all Keys or any of the Keys.

### 6.2.2. Combination of Boolean Terms:

In this type of search a number of Keys are used. Each Key is considered as a Boolean variable or terms. A search criterion can then be formulated to specify any predefined Boolean expression using these Keys[1-11].

### 6.2.3. Threshold Search:

In this case of search, all Keys in a profile are assigned to either equal or different weights. The respective weights of Keys present in a record are then summed up. The record is selected if the addition of weights exceeds a certain threshold value. In its simplest form, the threshold search[1-11] can be used for 'm' out of 'n' search criterion. The refinement can be made for a more complicated Boolean search expression or to introduce different emphasis among the Keys to reflect the user's view.

### 6.2.4. Interactive Search:

In this approach[1-11] a number of profiles starting from the general to the more specific are defined to describe a set of documents. Here the user expects to limit the number of retrieved documents within a user defined value. Initially the data-base is searched with the most generally defined profile. If this search results in a large number of documents, the searching process is then repeated with the next specific profile on the resultant data-base. This process is continued until either the number of documents less than the threshold value are retrieved, or all defined search profiles are exhausted.

## 6.3 Data Format

It has been stated earlier that the efficiency of an IS&R system depends on the data-format[35,36,152] of the index file. There are four important types of data format usually used in an IS&R. These are shown in Fig.6.2 and a brief description of this data format is given below.

i) The first format consists of a document identifier and a number of Keywords. The basic retrieval operation is to perform a Boolean search on a number of Keys, and to obtain those document identifier numbers which satisfy the search request.

ii) This format may be used in conjunction with the one discussed above. It consists of a document identifier and the full document name, which contains the information required by the user. Access is made by the identifier and the name is passed to the user.

iii) This format represents an entry in an index. As shown, the index is fully inverted. A Key is given together with a list of documents identifier which are associated with that Key. Typical identifiers would be disc addresses of records in format iv or a mixture of this and other formats.

An alternative interpretation of this format is produced when the document identifier list consists of a single item or a number of items less than the total of relevant documents. A multilist[11,35,36] system is then produced where the records in the document file are chained together after the first entry point has been given in the index. These files may also be sectioned to produce a cellular organisation[11,35,36]. In this type of index, the basic operation is to access on the key and then obtain records pointed to from disc which are then processed as for the other three formats.

iv) This format represents the absence of structuring within a record. The data consists of a single character string. Search will generally be based on a Boolean combination of substrings of the characters.

It could be seen that the search operations on records and indices are bascially the same, both entailing Boolean operation between sequences of character strings. The format iv. is chosen for index file of IS&R system of the present work. This requires minimum data-structuring[152], and at the same time the searching process is not restricted to only specified Keys. On the other hand full documents could be stored as index (KWIC)[28-32] and search criterion could be matched on entire contents of the document.

## 6.3.1. Index File:

The format of records in the index file is shown in Fig. 6.3(a). Where a record is a simple character string, pre-and de-limited by two special symbols.

The structure of the index file (search file) which is stored in a disc is shown in Fig 6.3(b). The indexfile is a collection of records as shown in the figure. Two special symbols are used to denote the start and end of the file.

Before starting any exercise on the IS&R system a realistic data-base (index-file) must be created according to the chosen format. It is also

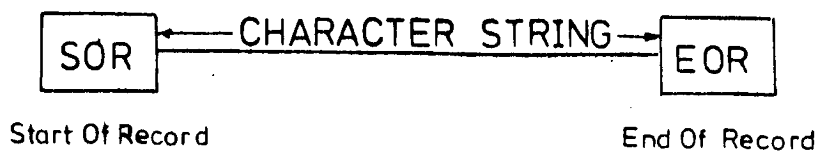


FIG. 6.3(a) Index record format

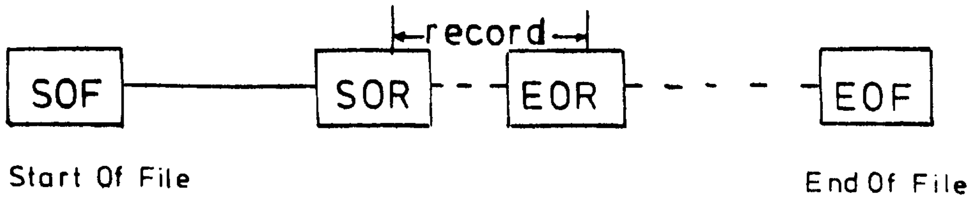


FIG. 6.3(b). Index file format

% KEY1≠% KEY2≠% KEY3≠ %KEY N≠(EOF)

FIG. 6.4. Profile format

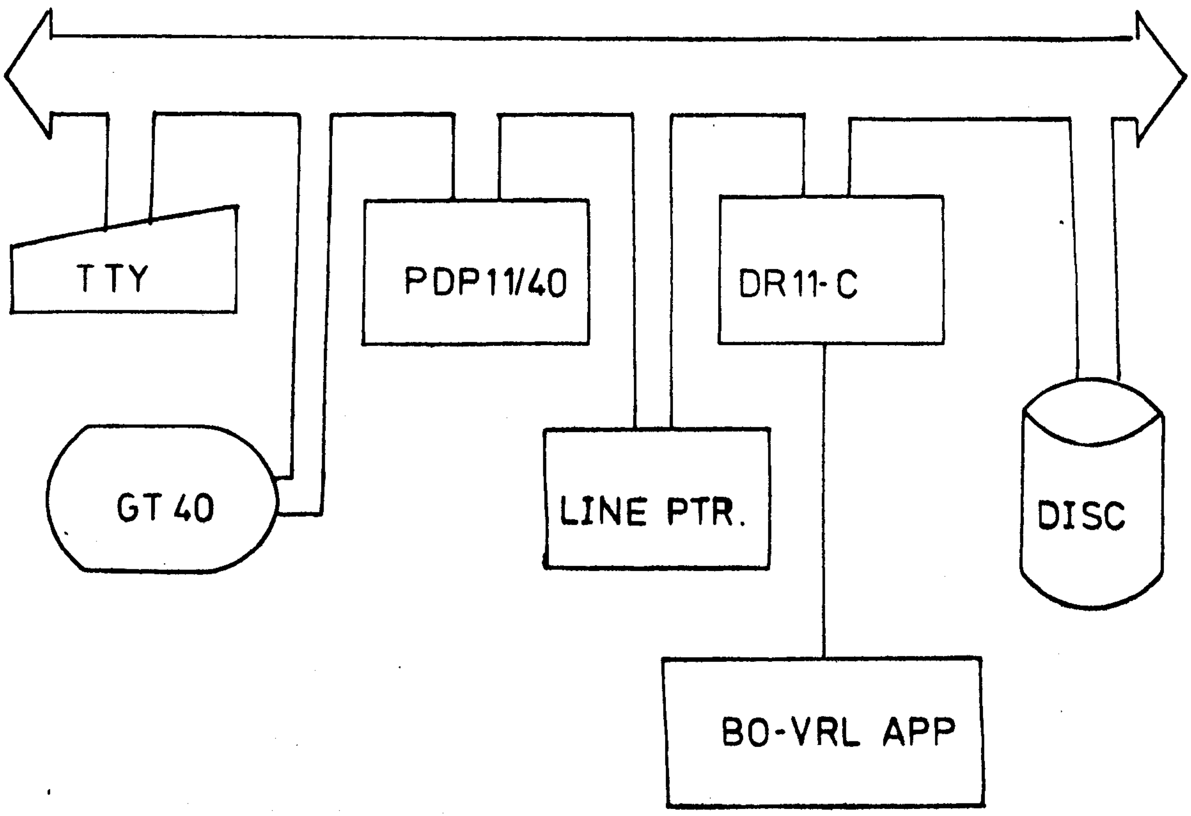


FIG. 6.5. Schematic diagram of the proposed associative searching unit

stated earlier that the cost of the process of creating an index file, is signigicant; hence considering the expense, it was decided that in this experiment an index file according to the chosen format would be derived from the data-base available from Inspec tape service.[181] The current and back issues of inspec data-base are available in magnetic tapes. The distribution format used for inspec tape services is based on ISO-2709, the international standard format for bibliographic data interchange.

The file and record layout of inspec system can be found in Inspec tape service manual. A brief description of the same is included in APPENDIX D. A file is a collection of bibliographic records. Each record in a file is sub-divided in a number of fileds. These fields are numeric and are arranged, within the directory, in ascending numeric sequence.

The general categories of fields include

1) Control field
2) Subject delineation
3) Personal names
4) Identifying codes
5) Volume and issues
6) Location
7) Number of pages
8) Organisation
9) Dates

and 10) File descriptions

Each of these fields is further divided into a number of sub-fields. And any sub-field may contain more than one attribute.

An algorithm to convert information from an Inspec data-base to a data-base of simple structure on a magnetic disc unit is developed. The detail descriptions of this algorithm looks for the presence of a number of selected Key-fields on each record. Once the desired fields are located the contents of those are extracted from the tape and stored in a disc-base file. Two special symbols are attached in the front and end of these character strings to separate individual records.

## 6.3.2. Profile:

It has been stated earlier that an information retrieval system requires its users to make known their information needs to it[1-10]. The query conveys the statement of required information to the retrieval system. It has also been seen that the query should be expressed in a language similar to the indexing language. Further various methods of changing the precision of the retrieved documents are also dicusssed. Finally, once a user decides what his intention is to be, he can formulate a search equation by selecting a number of Keywords which are connected by a set of logical operators. On the basis of this search equation a file called user's profile, can be created. To retrieve documents, this profile is matched against the records of the index file.

As an illustrative example, a simple search equation, as given below is chosen.

KEY 1 + KEY 2 + ..... + Key N

The contents of the profile for this search equation is shown in Fig.6.4 In figure 6.4 it is shown that each Key of the profile is both pre-and de- limited by two special symbols, and the delimiters of a negated Key are preceded by a 'minus' sign. In this case, it could be observed that one delimiting symbol would have been sufficient to delimit these Keys. But in other cases (threshold search etc) the character following a Key-delimiter may contain a control character (such as weights or Boolean operators) thus the end of a Key does not necessarily mean that the character following this delimiter will be the beginning of a new Key. Hence for generality, two special symbols, % and ≠ are used to both pre- and de- limit a Key. Finally, the entire string of these keys, that is, the profile is terminated by an end-of-file mark.

Some more examples of profiles could be found in sec 6.6.

In the following section some underlying philosophy for the implementation of an APP based on-line retrieval system is discussed.

## 6.4 Philosophy of Implementation of On-line IS & R system

There are two major alternatives to implement the proposed IS & R system. In the first approach a special purpose hardware could be designed with adequate software to achieve an efficient and dedicated IS & R system. The advantage of this system would be its better performance as an IS & R system, because it was specifically intended to perform this special work. And the disadvantages of any special purpose systems also apply in this case, which are mainly concerned with the cost and time of developing a new system. Moreover, presently, sufficient information about APP based

IS & R systems is not available. Hence, a first-time attempt to implement a special purpose hardware may contribute to many undesirable effects. On the other hand a modest approach of simulating the proposed system, utilizing existing facilities at Brunel University, seems more practicable. It was initially intended to examine the system requirements and to evaluate the operation of APP in IS & R applications. Finally with a well defined system specification, which is derived from the above experiment, a dedicated stand-alone on-line information storage and retrieval system could be implemented. In the light of the above discussions, a hybrid computer simulation for the implementation of on-line retrieval systems was chosen. This proposed simulation consists of associative memory hardware with a specially developed software to accomplish other functions.

The block diagram of Fig 6.5 shows some important components of the proposed system. Here a tele-type writer and graphic terminal are used as interactive terminals for on-line IS&R system. The tele-type accepts both control and user data (profile). The set of disc units are used as a back up store for both input index and output files. The associative memory array is used as a parallel search unit. The line printer provides hard-copy printouts for the output files. The PDP 11/40 processor[177] and core memory are utilised to simulate the control structure and input/output buffer of the IS&R system. The allocation of core memory of PDP 11/40 system is shown in Fig.6.5. The RT11[180] system and Keyboard monitor are permanently kept in the memory to respond to any general RT11 system control. The IS&R control program co-ordinates the simulated system. From the description of on-the-fly search technique it could be seen that a direct link between the disc and APP is required to maintain a steady flow of data between them. But

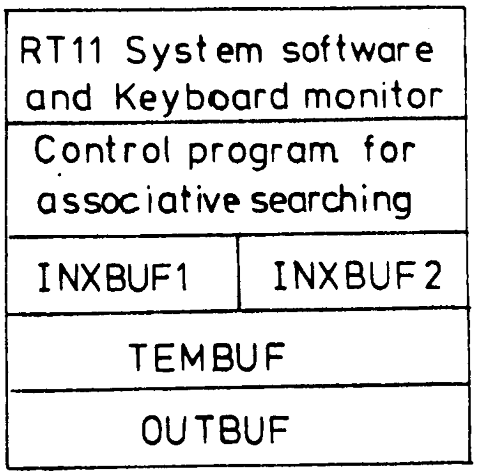

RT11 System software and Keyboard monitor
Control program for associative searching
INXBUF1
INXBUF2
TEMBUF
OUTBUF

FIG. 6. 6. Memory allocation

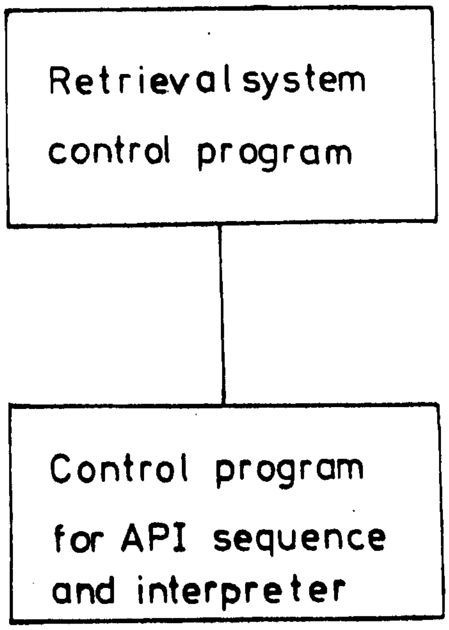

Retrieval system control program
Control program for API sequence and interpreter

FIG. 6.7. Hierarchy of control program

presently at Brunel University hardware facilities to enable such direct transfer of data between the disc and the APP do not exist. On the other hand, all data transfer operations are carried out through an intermediate buffer area, which is specially allocated in the core memory. To reduce the waiting time in between the transfer of two blocks of data from the disc unit a double buffer scheme$^{180}$ is adopted. In this arrangement while data from one buffer area are being transferred to the APP; the other buffer area is simultaneously loaded from the disc unit, using data-break techniques. The buffer areas INXBUF1 and INXBUF2 are allocated for this purpose. Similarly, other buffers are set aside for storing profile,output file and associative memory maps (See section 5.5.2.). The sequence of associative memory maps are dumped in a separate output file to provide debugging and feed-back information, to improve performance of the system. The function of 'TEMBUF' (temporary buffer) is to provide an intermediate storage between index and output file transfer. The character strings which are passed on to the search unit are simultaneously stored in a first-in-first-out type temporary buffer. When a record satisfies the search-criterion the content of this temporary buffer is transferred to the output file, otherwise it is cleared.

## 6.5 Software and Control Structure:

It has been seen from earlier descriptions that the operation of an on-line retrieval system comprises of three major sequences.

During the initial sequence of operation the description of input files, such as index and user profile, are specified.

In the next sequence of operation, the search operation is carried out on the index file and the matching records are stored in an output file.

During the final sequence of the operation the output file is made available to the user. The other output file containing the memory maps could also be referred for debugging operation.

The input and output operations of the system involve conventional file storage and transfer function. Hence, a control program, which includes conventional file transfer operations as well, satisfies the requirements of the proposed system.

The searching and match resolving operations of the system are carried out in the associative parallel processer. Thus algorithms to perform these operations, along with the initial loading of the associative memory, involve associative processing. In the following paragraph, the underlying philosophy for implementation of the algorithms, which uses associative processing are discussed.

At Brunel University research is being currently carried out to develop a suitable machine oriented to higher-level language, for the existing BO-VRL APP system. Unfortunately any intermediated languages, excepting simple API sets, are not yet available. Thus, at the time of writing this report the only choice open was to use simple Associative Processing Instructions, as described in Appendix B. This simple instruction set does not include any control structures, such as unconditional and conditional transfers of control. Hence, all these necessary control structures, to implement algorithms containing API's, would be embodied in a special control program. This control program would assist to maintain the sequential flow of API. This stream of API's, as they occur, are individually interpreted. The interpreter consists of a set of routines which are used to execute an API by transferring proper sets of information across the interface to enable the hardware to execute an API.

From the above discussions, it is realised that two sets of control programs should be incorporated within the proposed simulation. The hierarchy of these control programs are shown in Fig 6.7. These include:

1) System control:- This provides all input/output and system control operations.
2) APP interpreter and API sequence control:- The function of this control is to provide an appropriate system control for the algorithm, which uses API's and to interpret APIs so that they could be executed by the APP simulator.

In section 6.6. algorithms for the proposed simulations are discussed.

## 6.6. Algorithms:

In the previous section the data structures used in an on-line retrieval system are discussed. In Chapter 5 a byte-oriented VRL-APP suitable for the parallel search unit of a retrieval system is both specified and described. In this section operational requirements of an on-line retrieval system together with their respective algorithms are discussed.

In an on-line retrieval system, three major operations are involved; these are:

1) input

2) output

and 3) search

1) input operation:- During this operation a user specifies his profile or he can enter his profile directly from the console typewriter. The user is also allowed to select a file from the file-set of the data base as an index file (Document file).

2) output operation:- At the completion of a search operation, two output files are produced; these are:

   a) Output file

   b) Associative memory maps

a) output file:- This file consists of all documents which have been selected as an outcome of a successful search operation.

b) Associative memory maps:- Whenever an operation is carried out on the associative memory, a memory map (see Sec. 5.5.2.) is produced. This file contains a sequence of such memory maps. The contents of this file, thus provides a very useful feed-back information to verify, debug and improve the search algorithms.

During the output operation the above mentioned files could be transferred to any desired device.

From the foregoing discussion it has been seen that the input and output are similar to conventional file-transfer operations. Hence no further descriptions of these operations are given. In the following subsection an algorithm for the search operation is described.

## 6.6.1. Algorithm for search operation:

The flow chart of Fig. 6.8 shows the basic sequence of operations during a search mode of IS&R system. A brief description of these follows:-

### 6.6.1.1. Initialisation:

During this phase of the operation, the following initialisation steps are carried out.

| | | |
|---|---|---|
| Step 1: | - | clear associative memory array and match hit counter and other buffer areas. |
| Step 2: | - | open user profile for loading it in associative memory. |
| Step 3: | - | load user profile in associative memory |
| Step 4: | - | open index file for serial transfer to the parallel search unit. |

The transfer of a character string from an index file to a parallel search unit is carried out serially; one character at a time. When this transfer operation proceeds, a hardware or software trap looks for the occurrence of four special symbols. These are:

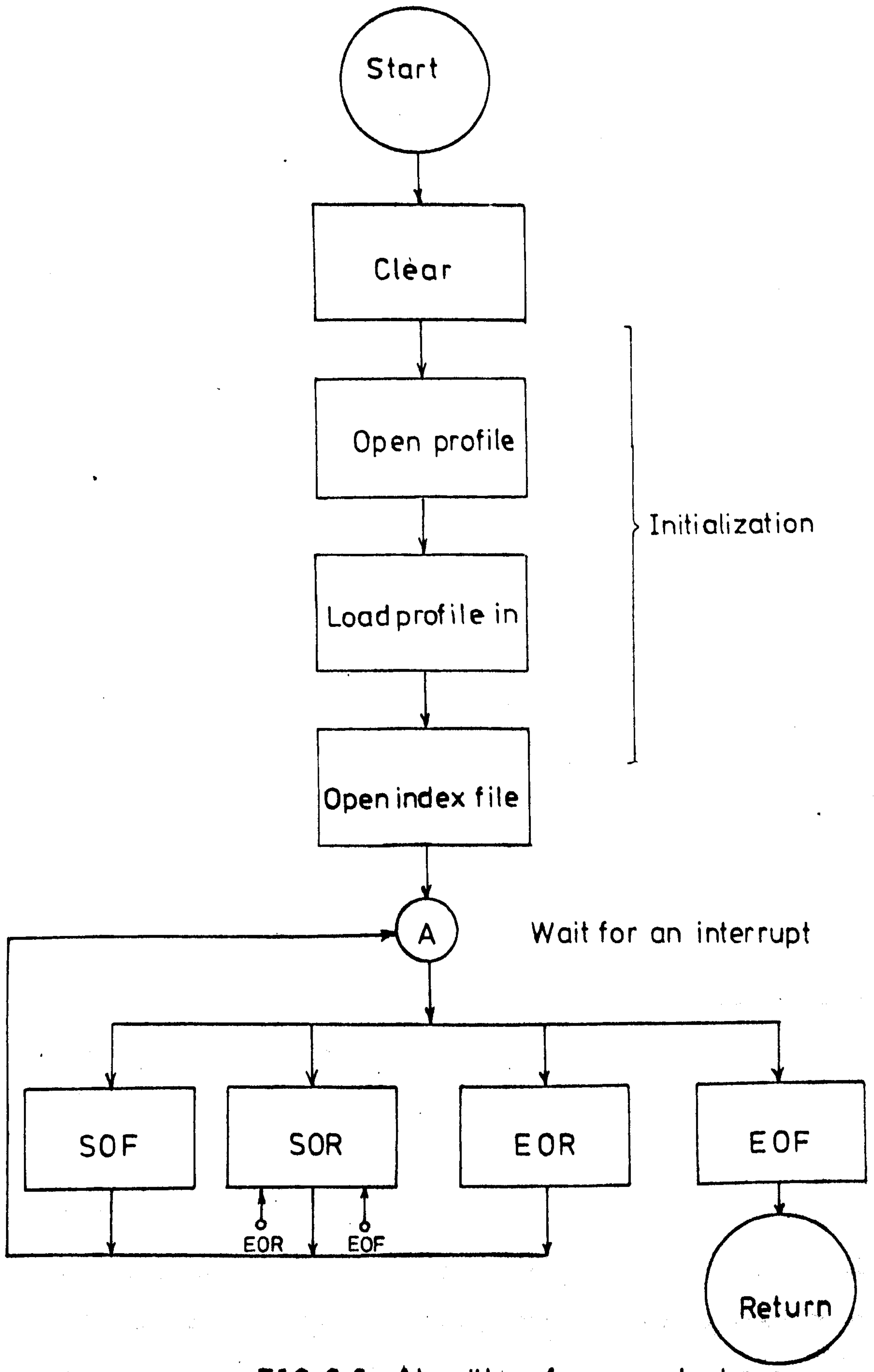


FIG. 6.8. Algorithm for search phase

i) SOF (start-of-file)

ii) SOR (start-of-record)

iii) EOR (End-of-record)

iv) EOF (End-of-file)

As they appear on the transfer line, the current routine is interrupted according to a predefined priority. The control is then transferred to the interrupting routine. After step 4 of the initialisation routine, the program waits for the occurrence of a special symbol to transfer the control to one of these four subroutines. Each of these sub-routines can operate independently, and continues to do so until, either it completes the job assigned to it, or it is interrupted by other higher priority symbols.

The selection of a document in the proposed associative retrieval system is carried out by a two part algorithm. During the first part of the algorithm, the 'compare' subroutine (see Section 6.6.1.3.) is used to mark the occurrences of a Key in the currently scanned record. The next part of the algorithm, 'Document Hit' (see Section 6.6.1.4.) is called at the end-of the scanning of a record. This verifies the validity of the current record by evaluating the search criterion.

In the following sections the operational steps of the subroutines SOF, SOR, EOR and EOF are described.

## 6.6.1.2. Start of File:

During this operation, the identification of the index file is transferred to the output file. The flow-chart of the algorithm is shown in the Fig. 6.9

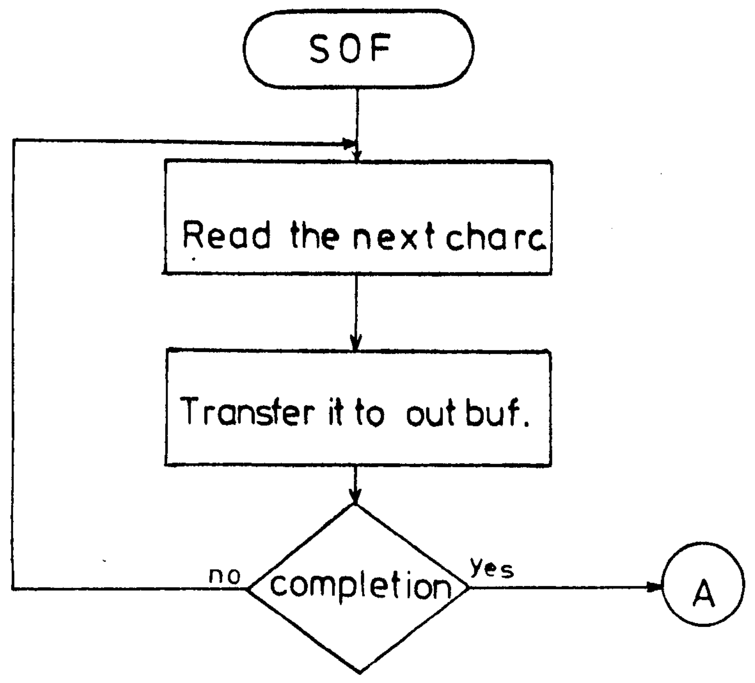


FIG. 6.9. Flow chart for SOF algorithm

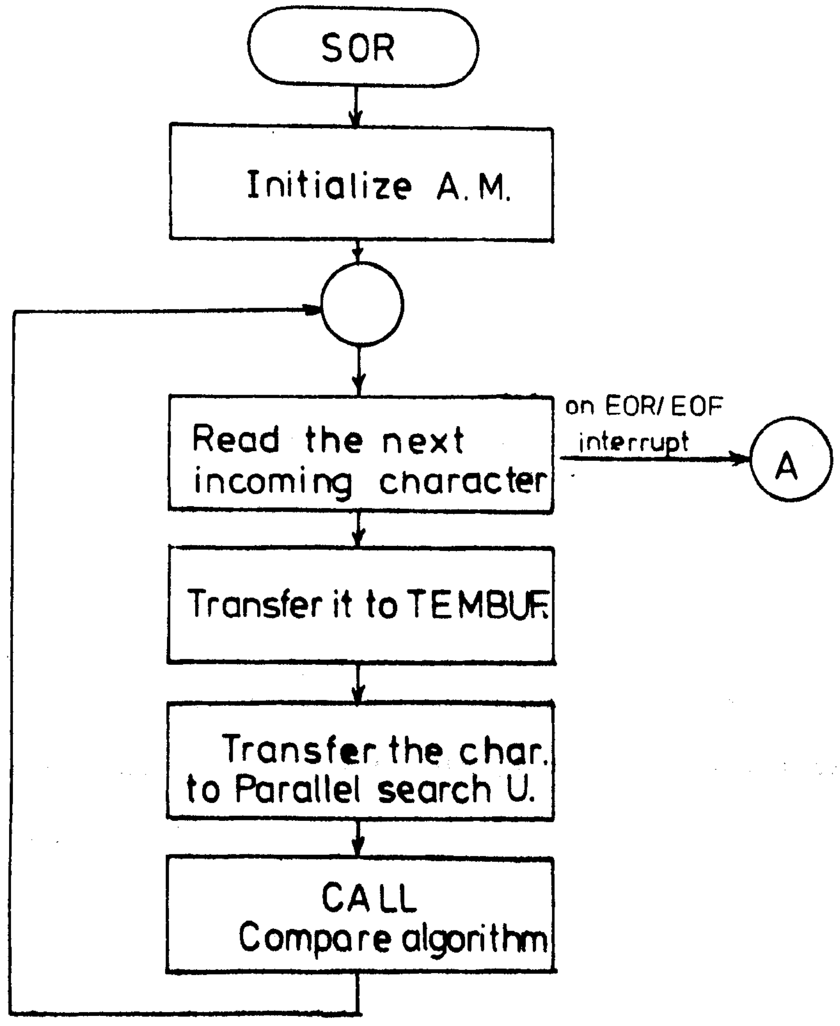


FIG. 6.10. Flow chart for SOR algorithm

### 6.6.1.3. Start of record;

The flow-chart for this subroutine is shown in Fig.6.10. In this subroutine the search operation on the incoming document is carried out by a 'compare' routine. The function of the compare routine is to mark the presence of any desired Keys in the current document. The marking of the presence of a Key is done by writing CB4 = 1 in the corresponding Key delimiter. This operation continues until an end-of-record or end-of-file is detected, and the control is then transferred to appropriate subroutine.

### 6.6.1.4. End-of-record

This routine is entered on detection of an end-of-record mark to verify the validity of the record just compared. The flow-chart for this algorithm is shown in Fig. 6.11. As described in section 6.6.1.3., at the completion of the start of record subroutine, the presence of the Keys in a document are marked. In the end-of-record subroutine a special routine, called 'Document Hit', is entered. The function of the document Hit routine is to verify whether or not the search criterion is satisfied by the current document. If a document satisfies the search criterion, a 'Document Hit' flag is set.

The other functions of the end-of-record routine on a successful 'document hit' operation, are to transfer the document to the output file and to increment the match hit counter.

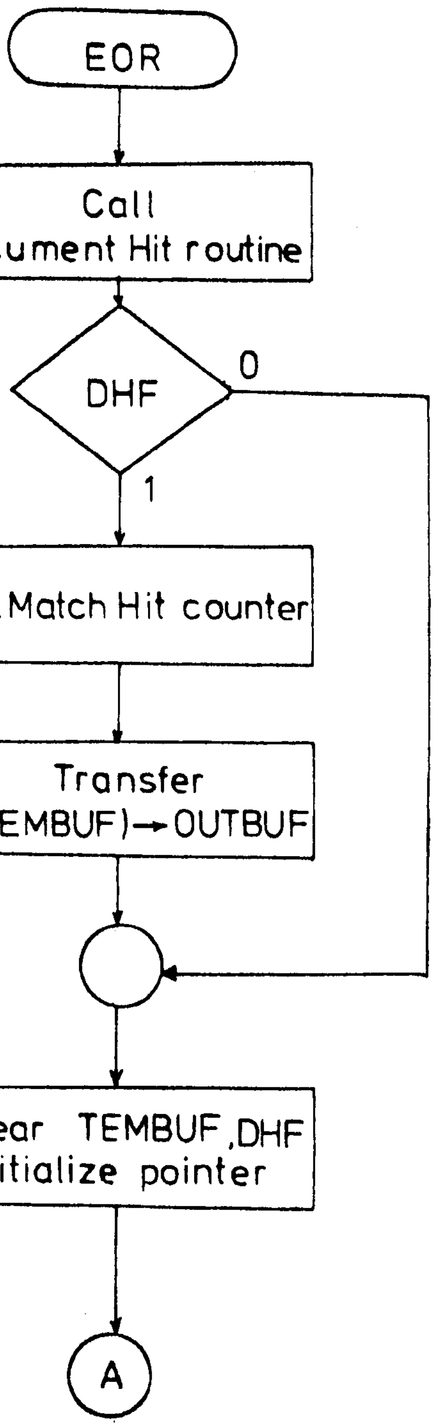


FIG. 6.11. Flow chart for EOR algorithm

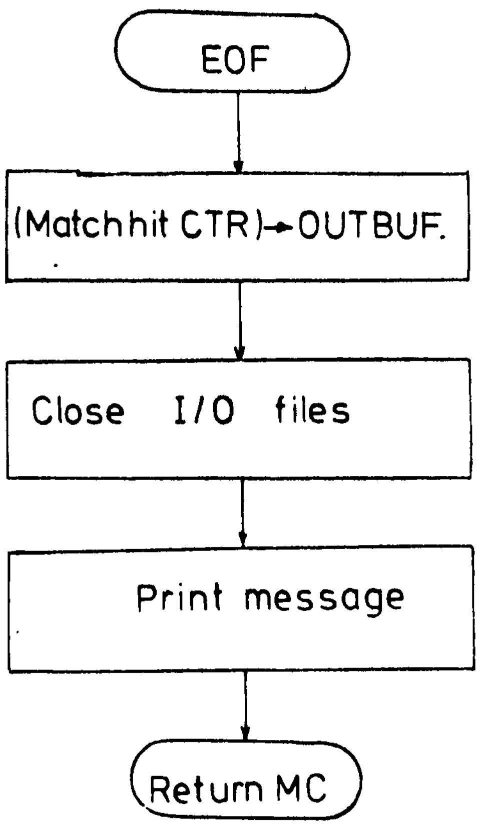


FIG. 6.12. Flow chart for EOF algorithm

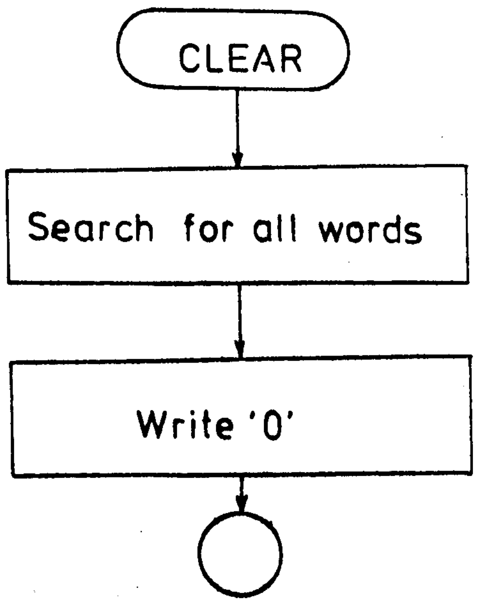


FIG.6.13. Clear routine

### 6.6.1.5. End-of-file.

Detection of End-of-file mark indicates that the compare operation on index file has been completed. The End-of-file routine is called to terminate the current search operation and it also provides some important statistics of the terminated search operation. Figure 6.12 represents the flow-chart of the end-of-file routine.

## 6.6.2. Details of the Algorithm using Associative Processing Instructions

In this section, the algorithms which include API are first described with a flow-chart. Then the corresponding API's are listed. The algorithms, which use API's, are 'clear AM' 'Load AM','compare' character string and 'Document Hit'. These 'compare' character-string and 'Document-Hit' algorithms may vary with different search equations of the user profile. Here, to illustrate these algorithms, a simple example of Boolean 'OR' operation of all Keys in the user profile is chosen.

### 6.6.2.1. Clear (Associative Memory)

The flow-chart of Fig 6.13 shows the clear operation.

Algorithm:-

```
    Step 1:    Clear all memory word
    Step 2:    Return
```

The API used for performing this operation is given below:

```
API CL:  W  @00000000 0000 * XXXX 0000 0 0S0 N
```

## 6.6.2.2. Load: (Associative Memory)

Fig. 6.14 shows the flow-chart for the Load associative memory operation. The loading is terminated either when the user profile is exhausted or an overflow of associative memory has occurred. In the last case an overflow message is first printed and the present search request is aborted. The Keys, which are specified for complement operation (NOT KEY 1 = $\overline{\text{KEY 1}}$;), are represented as % KEY 1 -# in the user profile. And corresponding Key delimiters are stored in the associative memory with their control bit 3 (CB3) set to 1, for example the AM map is: % KEY 1 $\overset{3}{\#}$

Algorithm:

Step 1: Isolate the first word row. (By writing CB1=1 on the first word row).

Step 2: Read the first character from the user profile.

Step 3: Load the first character and CB1 = 1 in first word row.

Step 4: Read the next character from the user profile. If end of profile is encountered go to Step 12.

Step 5: Check for complement sign. If complement sign has occured go to Step 7.

Step 6: Get the last occupied word-row in AM. Clear all CB1 (writer CB1=0). Write the character and CB1=1 on the right neighbouring word row. Go to Step 9.

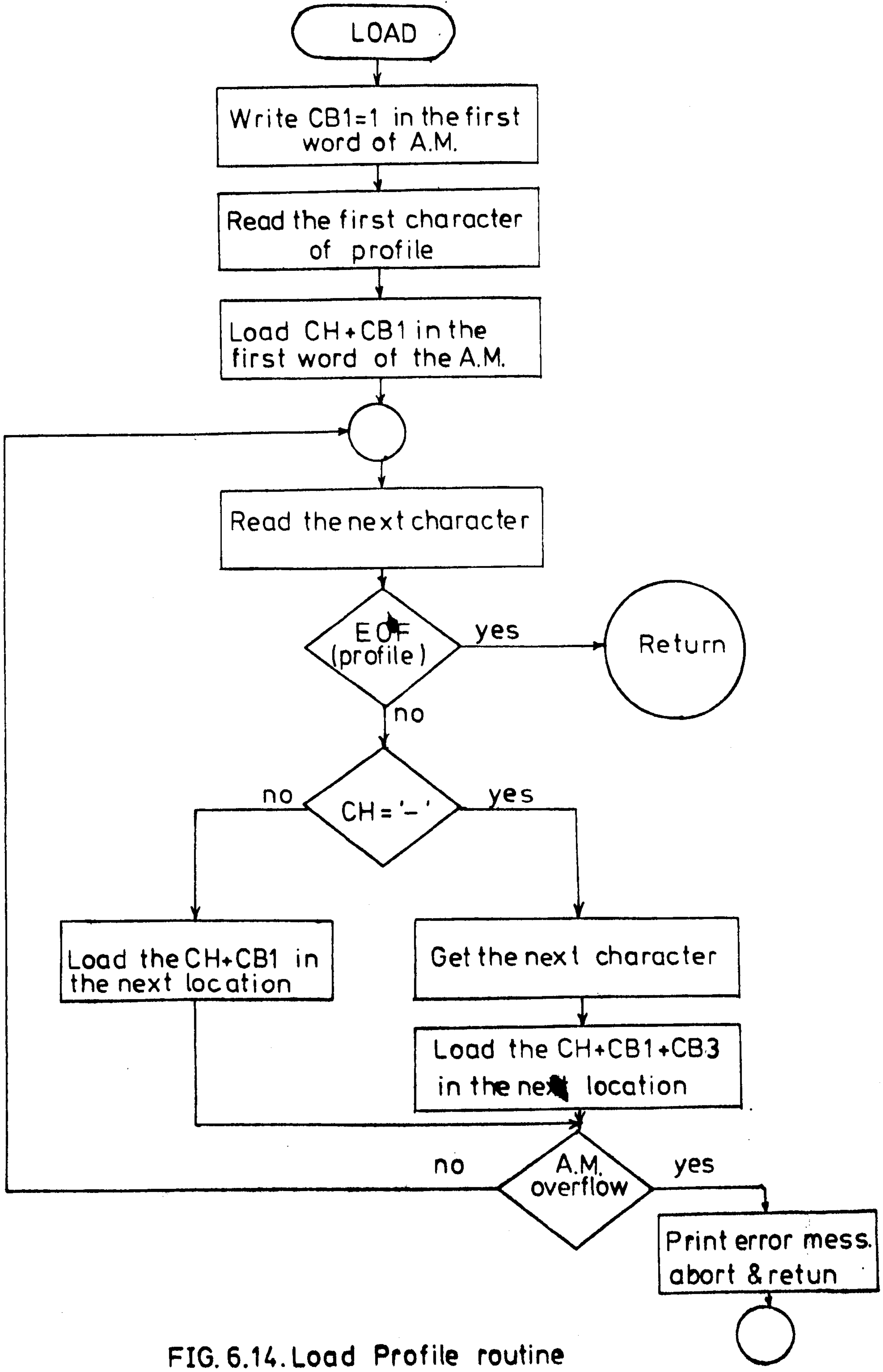

LOAD
Write CB1=1 in the first word of A.M.
Read the first character of profile
Load CH+CB1 in the first word of the A.M.
Read the next character
EOF (profile)
yes
Return
no
CH='–'
no
yes
Load the CH+CB1 in the next location
Get the next character
Load the CH+CB1+CB3 in the next location
A.M. overflow
no
yes
Print error mess. abort & retun

FIG. 6.14. Load Profile routine

Step 7: Get the next character.

Step 8: Get the last occupied word-row in the AM.
Clear all CB1
Write the character, CB1=1 and CB3=1 on the right neighbouring word row.

Step 9: Check for associative memory over-flow.
If overflow is set; go to Step 11.

Step 10: Go to Step 4.

Step 11: Print associative memory over flow message.
Abort the present search request.

Step 12: Return.

The API's used for operation of the steps 1,3,6,8 of the load algorithm are listed below.

Step 1:-

```
APIL1:- W  *  1XXX  *  XXXX 0000 0 0S0 N
        ;write CB1=1 to all word rows.

API L2:-  W  *  0XXX  *  1XXX 0000 0  00D N
          ;write CB1=0 to all but the first word-row
```

Step 3:-

```
API L3:-  W ch 1XXXX  *  1XXXX 1010  0  0S0 N
          ;write ch + CB1=1 in the first word row.
```

Step 6:-

```
API L4:-  W ch 1XXX  *  1XXX 1010  0  00D N
          ;write ch + CB1=1 in the right neighbouring word row
```

Step 8:-

```
API L5:-  W ch 1X1X  *  1XXXX 1010  0  00D N
          ; write ch + CB1 & CB3 in the right neighbouring word row.
```

### 6.6.2.3 Start-of-record:-

The initialisation of the associative memory, prior to the compare algorithm, is done by writing zero to the control bits CB1, CB2 and CB4 of all word rows. The API for this operation is given below.

```
APIS:  W  *  00X0  *  XXXX 0000 0 0S0 N; write CB1,2,4 = 0 in all word rows.
```

### 6.6.2.4 Compare Algorithm:-

The main function of the 'compare' algorithm is to find the presence of a Key within the record currently under examination. This is performed by comparing the incoming character string with the Keys stored in the associative memory. If a desired Key is found in the current document, the presence of this Key is marked by writing CB4 = 1 in the corresponding

Key delimiter. This function of the compare algorithm is carried out in three different steps. During the first step, the first character of all keys in the associative memory are enabled by writing CB1 = 1. Then all characters, which are marked by CB1 = 1 are compared with incoming characters. At the end of this operation the information on the control bit 1 of the matching characters is transferred to the next character of the Keys. Otherwise the control bit 1's of the character sequence are cleared. Proceeding in this manner, when the control bit 1 hits a key delimiter symbol, it indicates the occurrence of that Key in the current record. This information is stored by writing CB4 = 1 in the matching Key delimiter. The flow-chart for the compare algorithm is shown in Fig 6.15.

In this paragraph the compare algorithm is illustrated with an example. It is assumed that a user wants to locate all documents containing either KEY 1 or KEY 2. The contents of the associative memory corresponding to this user's profile is shown in Fig 6.16. It is also assumed that a record containing the character string of Fig 6.17 is under the read head of the disc unit. It has been stated earlier that three instructions are required to process a single character from input index file (record). The contents of the associative memory during each steps of the compare algorithms are shown in Fig 6.18 and API's required for this algorithm are listed below.

Step (a) W * 1XXX % XXXX 0000 0 00D N

;search for Key prelimiter (%); write CB1 = 1 in right neighbours of the matching word. This enables beginning of each Keywords in the profile to be a candidate for taking part in the matching operation with the incoming character.

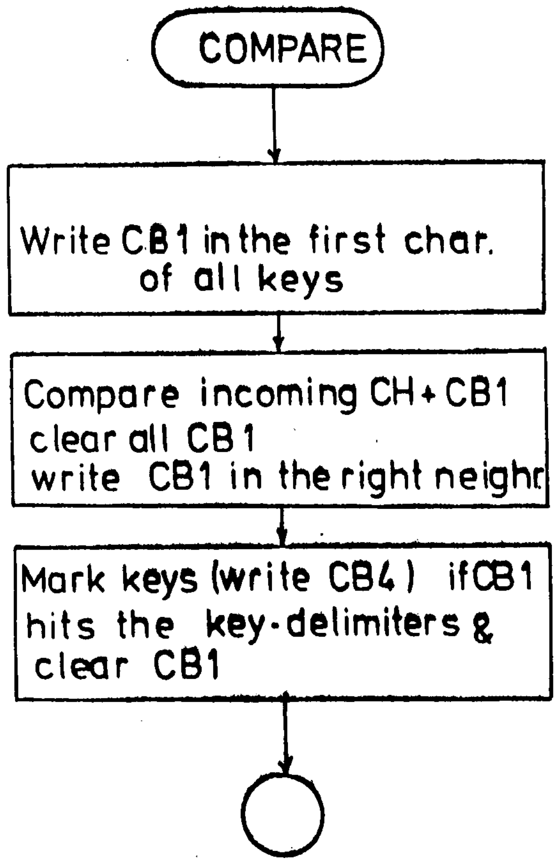

COMPARE
Write CB1 in the first char. of all keys
Compare incoming CH + CB1
clear all CB1
write CB1 in the right neighr
Mark keys (write CB4) if CB1 hits the key-delimiters & clear CB1

FIG. 6.15. Flow chart for Compare algorithm

%KEY 1#%KEY 2#

FIG.6.16 Contents of A.M.

(SOR)······KEY 1,KEY 5,·· –···(EOR)

FIG. 6.17. Character string under read head.

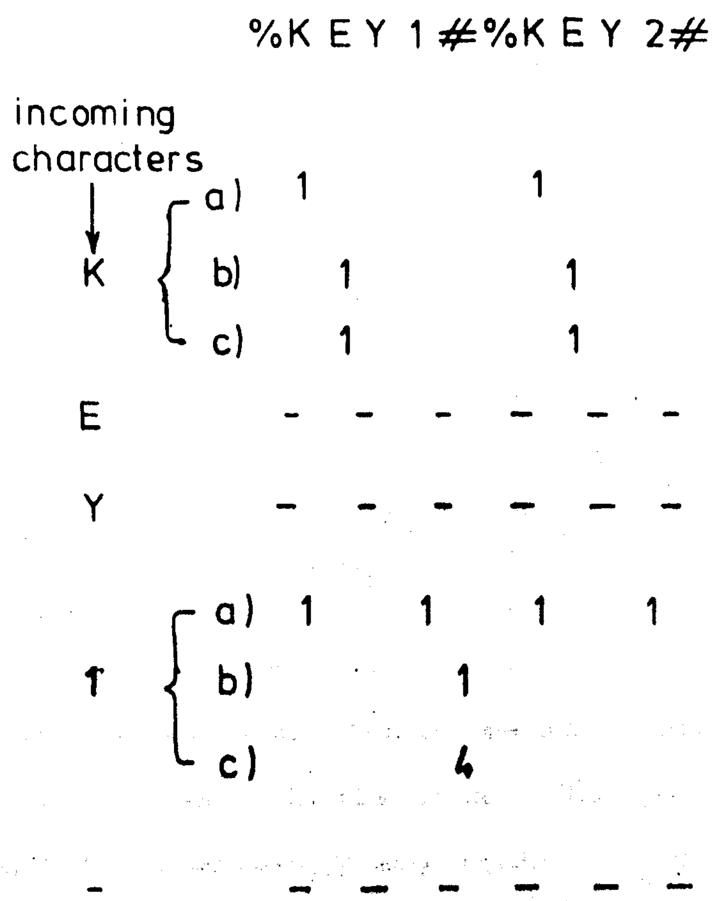


FIG. 6.18 Contents of A.M. during Compare operation.

(b) W * 1XXXX ch 1XXX 1010 0 00D N

; Search the incoming character (ch) and CB1=1; clear all CB1; write CB1=1 in the right neighbour of the matching word-rows. This permits a Key (string of characters) to compared sequentially.

(c) W * 0XX1 ≠ 1XXX 0000 0S0 N

; search for a Key delimiter ≠ with CB1=1, Write CB1=0, CB4 =1 Write CB1=0, CB4=1 in the matched word. This establishes the presence of a Keyword in the record.

At the end of the operation occurrence of 'KEY 1' in the current record is marked by writing CB4=1 in the complex symbol of the delimiter of KEY 1.

Similar operations can be parallelly carried out on all Keys of the profile. At the end of a record, the occurrence of relevent Keys are marked by CB4, and Document Hit sub-routine is called.

## 6.6.2.5 Document Hit Algorithm

The function of the Document Hit algorithms are to varify the validity of record under consideration. When the record satisfies the desired search criterion the 'Document Hit' flag is set. There are various possible search equations; and correspondingly many Document Hit algorithms. Some of these Document Hit algorithms are described in this sub-section.

1. LOGICAL 'OR' Operation:-

Let us consider a profile containing a number of Keys, say Key 1, Key 2, Key 3, Key 4. The state of the associative memory at the

```
%KEY1#%KEY2#%KEY3#%KEY4#
     4              4
```

FIG. 6.19 Content of A.M. at the beginning of document hit algorithm.

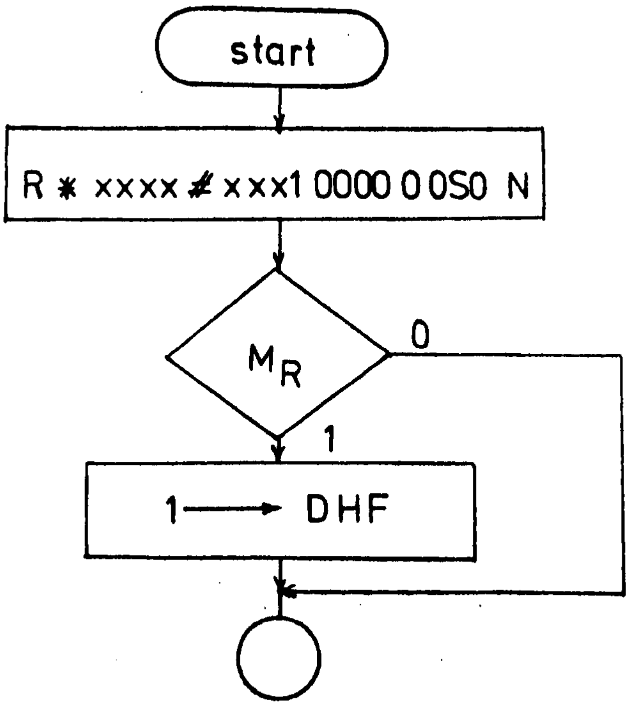


FIG.6.20 DOC. HIT Logical OR

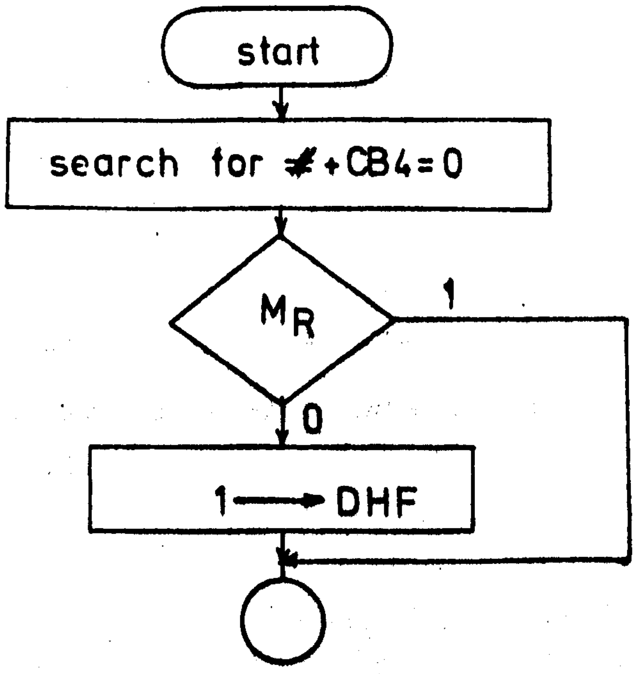


FIG.6.21 Logical AND

beginning of the Document Hit algorithm is shown in Fig. 6.19
In this particular case the search equation is

DHF = KEY 1 + KEY 2 + KEY 3 + KEY 4.

Here the Document Hit algorithm (Fig. 6.20) looks for any occurrence of Key delimiter with CB4=1. This is done by performing a dummy 'Read' instruction as shown below.

R * XXXX #XXX1 0000 0 0S0 N;

The execution of this instruction provides a match reply MR output, indicating the presence of a Key in the document. As in this case, MR = 1; the document hit flag is set.

(2) Logical "AND' operation:-

Considering the previous example, the content of associative memory is shown in Fig. 6.21 Here the search equation is

DHF = KEY 1. KEY 2. KEY 3. KEY 4.

The corresponding algorithm is given in Fig.6.21 Here the algorithm looks for absence of any of the Keys. The steps of the algorithm are explained below.

Step 1. Search with CB4 = 0; Read
Step 2. If match reply MR=0; set Document Hit flag
Step 3. Return.

```
% K 1 ≠ % K 2 ≠
      3         3
      4
```

FIG. 6.22

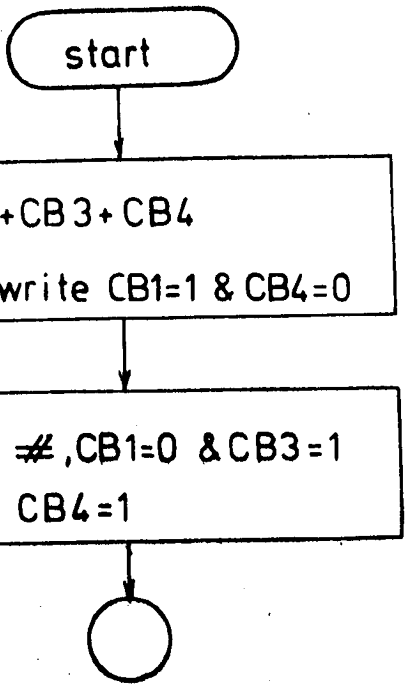


FIG.6.23 Complement negated keys

```
                 %K 1 ≠%K 2 ≠
                      3       3
Initial state         4


                      1
   step1.             3       3


                      1
   step2.             3       3
                              4
```

FIG. 6.24

API:-

Step 1. R * XXXX ≠ XXX0 0000 0 0S0 N;

To interrogate match reply.

Step 2. If MR = 0; set DHF

Step 3. Return

(3) Logical 'NOT' Operation:

The Fig.6.22 shows the contents of the associative memory at the end of compare operation. Here occurrences of the Keys K1 and K2 are to be negated. The algorithm is shown in Fig 6.23 and corresponding steps are explained.

Step 1. Search for occurrence of Key delimiter with negation. Write CB1 = 1 and CB4 = 0, on the match word.

Step 2. Search for non-occurrence of negated Keys; mark them by writing CB4 = 1

The corresponding API's are given below

```
API1:-    W * 1XX0 ≠ XX11  0000 0 0S0 N
API2:-    W * XXX1 ≠ 0X1X  0000 0 0S0 N
```

At the end of this algorithm occurrence of all negated Keys are complemented, (Fig.6.24) These could be now treated as simple Bolean variables for further logical 'OR' or 'AND' operations.

```
( %K1  ≠%K 2≠) ( %K 1 ≠%K 2≠) ( %K 3≠%K 4≠ )
            3         3
```

FIG. 6.25 Contents of A.M.

```
              ( %K1  ≠%K 2≠) ( %K 1 ≠%K 2≠) ( %K 3≠%K 4≠ )
Initial                   3         3
state                4              4                 4

                                    1
API 1                     3         3
                     4                                4

                                    1
API 2                     3         3
                     4    4                           4

                                    1
API 3                     3         3
                     4    4          4 4 4 4 4           4

API 4         MR = 1
```

FIG. 6.26 .Contents of A.M. during document hit (s-o-p) algorithm.

(4) Document Hit Algorithm for generalised Boolean equations:

Two generalised Boolean equations of search Keys are considered here. These are

a) Sum of product terms

b) Product of sum terms

In these Boolean equations restrictions on number of appearance of a Key in either true of negated forms are not imposed.

a) Sum of Product terms:

A typical Document Hit equation for Keys is given below.

$$DHF = K1.\ \overline{K2}. + \overline{K1}.\ K2. + K3\ K4.$$

This search equation is stored in associative memory as shown in Fig 6.25 In the Fig6.25 terms within the parentheses are product terms and logical summation are to be carried out with the terms delimited by brackets. The Fig.6.26 also shows the contents of the associative memory at different phases of Document Hit algorithm. The Document Hit algorithm for this logical equation is shown in the flow-chart of Fig. 6.27 and the corresponding steps are explained below.

Algorithm:-

Step 1. Complement the negated Keys

Step 2. Mark the product terms which are not present in the record.

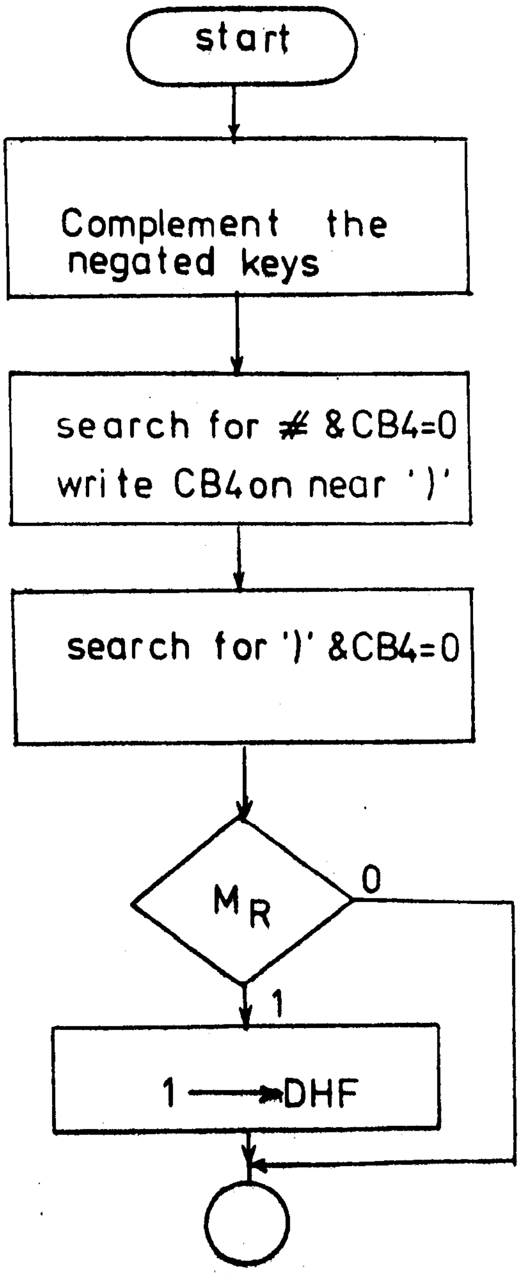


FIG. 6.27 Flow chart for sum of product terms

Step 3. Check for the presence of any product term.
If none present go to Step 5.

Step 4. Success, set Document Hit flag.

Step 5. Return.

The API's used for this algorithm are given below.

```
Step 1:     a)    W  *  1XX0  ≠  XX11  0000  0  0S0 N
            b)    W  *  XXX1  ≠  0X1X  0000  0  0S0 N

Step 2:     W  )  XXXX ≠ XXX0  0000  0  00D  G

Step 3:     R  *  XXXX  )  XXX0  0000  0  0S0 N
            If MR= 0, go to Step 5.

Step 4:     Set DHF = 1

Step 5:     Return
```

b) <u>Product of sum:-</u>

A search equation involving product of sum terms is given below.

$DHF = (K1 + \overline{K2}).\ (\overline{K1} + K2).\ (K3 + K4)$

The corresponding profile is given in Fig 6.28 here the terms included in brackets are sum terms, and the product of these sum terms are

```
( %K 1 ≠ %K 2 ≠ ) ( %K 1 ≠ %K 2 ≠ ) ( %K 3 ≠ %K 4 ≠ )
              3           3
      4       4           4      4          4
```

FIG.6.28

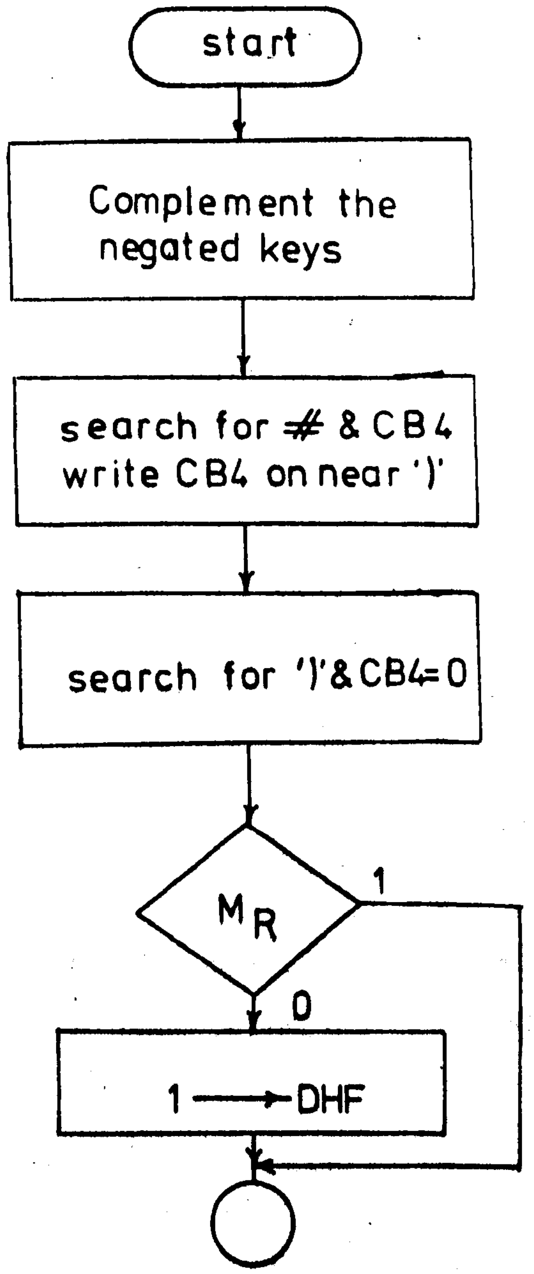

start
Complement the negated keys
search for ≠ & CB4 write CB4 on near ')'
search for ')' & CB4=0
M_R
1
0
1 ⟶ DHF

FIG. 6.29 Flow chart for product of sum terms

delimited by brackets.

The flow-chart of the algorithm is shown in Fig 6.29. The Fig. 6.30 shows the contents of the associative memory at different steps of the algorithm.

<u>Algorithm</u>:-

Step 1: Complement the negated Keys

Step 2: Mark the presence of sum terms.

Step 3: Check for the presence of all sum terms.
If not, go to Step 5.

Step 4: Success, set Document Hit flag

Step 5: Return

The API's used for this algorithm are given below:

```
Step 1:    a)   W  *  1XX0  ≠XX11  0000  0  0S0 N
           b)   W  *  XXX1  ≠0X1X  0000  0  0S0 N

Step 2:    W  )  XXXX  ≠XXX1  0000 0 00D G

Step 3:    R  *  XXXX  )  XXX0  0000  0  0S0  N
           If MR = 1 go to Step 5.
```

```
          ( %K 1 ≠%K2 ≠ ) ( %K1 ≠%K 2 ≠ )  ( %K 3 ≠°oK 4 ≠ )
Initial ⎧              3           3
state   ⎩       4      4           4      4           4

        ⎧              1           1
API 1   ⎨              3           3
        ⎩       4                         4           4

        ⎧              1           1
API 2   ⎨              3           3
        ⎩       4                         4           4

        ⎧              1           1
API 3   ⎨              3           3
        ⎩       4 4 4 4 4 4               4 4         4 4 4 4 4 4

API4            MR = 0
```

FIG. 6.30 Contents of A.M. during document hit (product-of- sum terms) algorithm

Step 4: Set DHF = 1

Step 5: Return

(5) Threshold Searches:-

A generalised profile for threshold search is shown in Fig. 6.31 where Ki's are the Keys and Wi's are their respective weights.

In the threshold search a threshold value is initially stored in the threshold register. And the weight Wi, corresponding to occurrence of a Key Ki, is subtracted from the contents of the threshold register when thisprocess produces a zero or negative result in the threshold register, the record is considered to be satisfied the search equation.

(a) M out of n:-

In this type of Document Hit algorithm, equal weights are assigned to all Key. It is generally normalised to 'One'; and hence Wi could be omitted from the profile. The threshold value, that is, 'm' is stored in the threshold register. And 'One' is subtracted from this register for occurrence of each Key. The algorithm for this Document Hit operation is shown in flow-chart of Fig 6.32 and explained below:

Step 1: Transfer threshold value to threshold register

Step 2: Complement the negated Keys

Step 3: Resolve the left-most Key. If no match reply occurs go to Step 8.

%K 1 #$W_1$%K 2 #$W_2$ · · · · · · %K N #$W_n$

FIG. 6.31

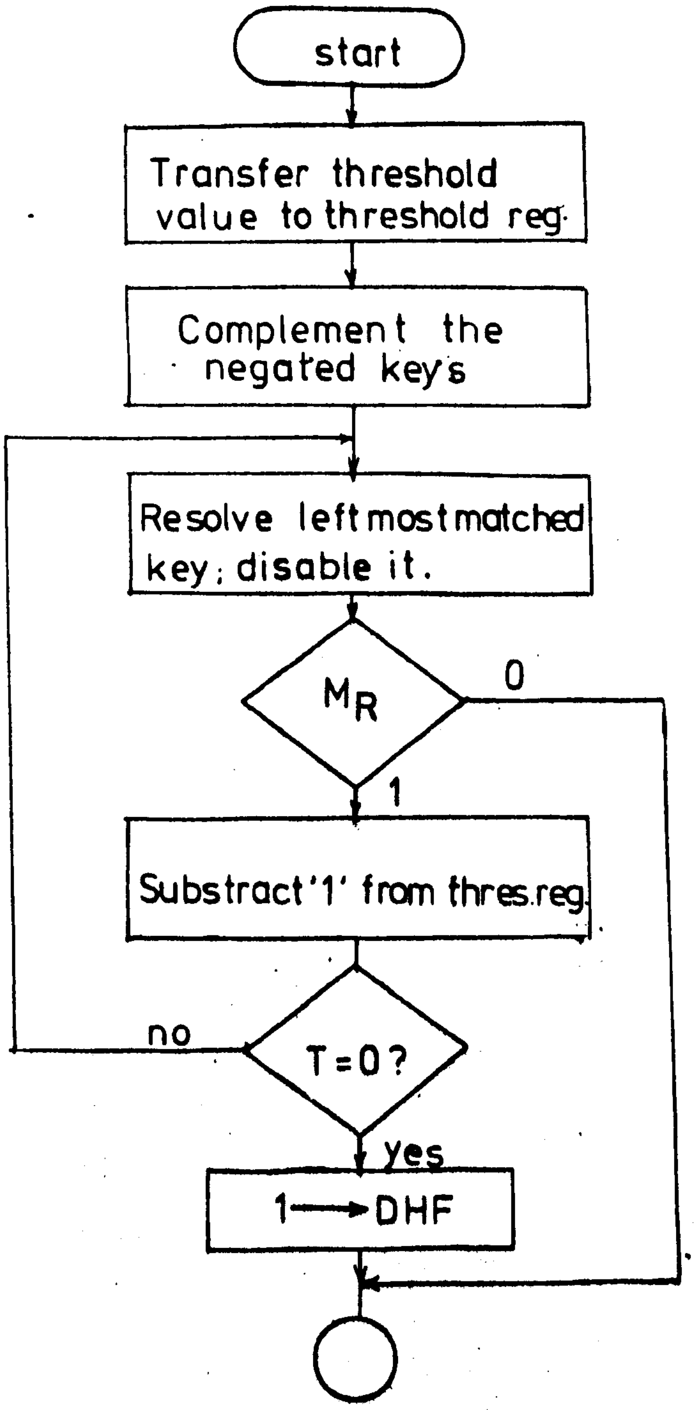

start
Transfer threshold value to threshold reg.
Complement the negated key's
Resolve leftmost matched key; disable it.
$M_R$
0
1
Substract '1' from thres.reg.
T=0?
no
yes
1 ⟶ DHF

FIG. 6.32 'm' out of 'n'.

Step 4: Subtract '1' from the threshold register

Step 5: Check if contents of threshold register is zero or negative. If true to to Step 7.

Step 6: Repeat steps 3 to 5.

Step 7: Set 'Document Hit' flag to 1

Step 8: Return

The API's used in this algorithm are given below:

```
Step 2:   a)   W  *  1XX0 ≠ XX11  0000  0  0S0  N  )  Complement the
                                                  )  Negated Keys.
          b)   W  *  XXX1 ≠ 0X1X  0000  0  0S0  N  )

Step 3:   W  *  X1XX ≠ X0X1  0010  0  00D  T;  resolve the left
          most matched Keys.
          If MR = 0; go to Step 8.

Step 4:   T = T-1
Step 5:   T  0        if true, to to Step 7.
Step 6:   Go to Step 3.
Step 7:   Set DHF= 1
Step 8:   Return
```

(b) Threshold Function (variation of emphasis on Keys or Boolean function)

The algorithm for this Document Hit operation is similar to previous algorithm. However, in this case some additional steps are to be included to read the weight of a Key from the associative memory. Here, it is assumed that the weights of a Key are stored in one word-row. The algorithm for this Document Hit operation is shown in Fig.6.33 and explained below:-

Steps of algorithm:-

| | |
|---|---|
| Step 1: | Transfer the threshold value to the threshold register. |
| Step 2: | Complement the negated Keys |
| Step 3: | Shift the validity bit (CB4) of the Keys to the right neighbour. |
| Step 4: | Resolve the left most Wi. If no match reply; to to Step 10 |
| Step 5: | Read the resolved word; and clear its validity bit (CB4) |
| Step 6: | Substract Wi from T |
| Step 7: | If T is zero or negative go to Step 9. |
| Step 8: | Repeat steps 4 to 7 |
| Step 9: | Set Document Hit flag to 1 |
| Step 10: | Return. |

The API's used for this algorithm are given below:-

```
Step 2     a)   W  *  1XX0 ≢ XX11  0000  0  0S0 N  )  Complement negated
                                                   )  Keys
           b)   W  *  XXX1 ≢ 0X1X  0000  0  0S0 N  )

Step 3:    W  *  XXX1 ≢ XXX1  1010  0  00D  N; shift validity bit
           on to the word containing the weights.
```

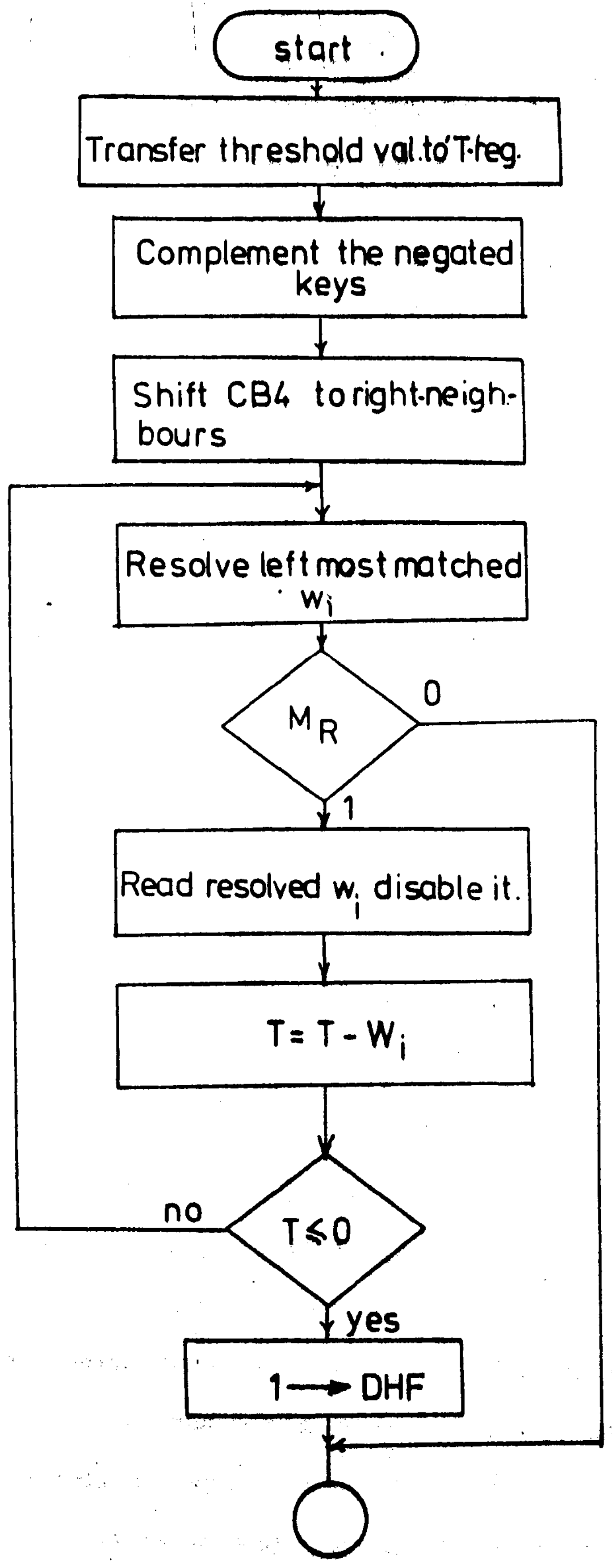

start
Transfer threshold val.to T-reg.
Complement the negated keys
Shift CB4 to right-neighbours
Resolve left most matched $w_i$
$M_R$
0
1
Read resolved $w_i$ disable it.
$T = T - W_i$
no
$T \leq 0$
yes
$1 \longrightarrow$ DHF

FIG. 6.33 Threshold search.

Step 4: W * X1XX * X0X1 0000 0 00D T; resolve the left most weights.
If MR = 0; go to Step 10.

Step 5: R * XXX0 * X1X1 0000 0 0S0 N; Read the resolved weight.

Step 6: T = T-(DOR); (DOR) = Wi

Step 7: Test T 0 ; if true go to Step 9

Step 8: Go to Step 4.

Step 9: Set DHF = 1

Step 10: Return

(6) <u>Interactive Approach:</u>-

In an interactive approach the user wants to limit the number of retrieved documents to a threshold value. When total number of documents retrieved exceed this threshold value, he can specify a new set of index and profile to initiate a new search operation. This can be done by either manual interruption or automatic transfer of control. In the latter case, at the detection of 'end-of-file' symbol match hit counter is compared with the threshold value. When it is greater than the threshold value, the output file is assumed to be the new index file and the next more precisely defined profile is loaded in the associative memory. This process is repeated until either less than the desired number of documents are retrieved or all user profiles have been processed.

## 6.7. Implementation:-

The on-line retrieval system was implemented by using the APP emulation as described in Chapter 5. The programming of this emulation was carried out using the assembler code of the PDP/11/40. The control programs for the on-line retrieval simulation were also developed using this assembler language.

A top-down approach was adopted to develop the software for the system. In this approach each part of the program is portioned into a number of hierarchical levels. The first level of control programs are general. And these controls only establish the flow of control. In the subsequent levels, functions of various routines are carried out. Hence, the complexity of programs increases at each level.

The flow of controls, within an algorithm using API, are carried out by a second level of control. This interpreter accepts an API, as pointed by an API pointer, and executes this API. At the completion of an API, the interpreter monitors the feed-back signal from the hardware to enable further sequencing of APIs. In addition, the interpreter was used to read the content of the associative memory to produce a memory map. This facility is initially incorporated to varify the validity of algorithms, and could be easily removed if desired. The detail operation of the control unit simulator for the APP is given in Chapter 5.

In the following discussion simple operational steps for the on-line retrieval system is described. As stated earlier, the on-line retrieval system performs three basic operations: Input, output and search. Transfer of control to a particular mode of operations is done by the mode control command as shown in Fig 6.34. Initially the system waits for a mode control command from the console teletypewriter. When a control command is entered at the console terminal, the control program is switched to the corresponding routines. At the completion of a mode of operation the control is transferred back to the initial waiting state. A brief description of each mode of operation is given below.

1. Input(I):- when the input mode (I) is entered the control is transferred to the input specification program. In this mode of operation the user is requested to define his index and profile.

2. Output (O):- In this mode of operation the user can specify and desired device, where the the output files are to be transferred.

3. Search (S):- During this mode of operation, On-the-Fly search of the index file, against the user profile is carried out according to the algorithm described in the section. 6.6 This also provides a number of different options, where the user can select a particular searching criterion. On successful operation, the program terminates by showing the total number of retrieved records and also produces two output files. One of these output file contains the bibliographic information of the matched documents. And the other provides a continuous record of associative memory maps. These memory maps help to debug and improve algorithms used in the system.

4. Exit (E):- This mode terminates current session of the on-line retrieval system simulation program and the control is transferred to the Key board minitor program of the RT 11 operating system.

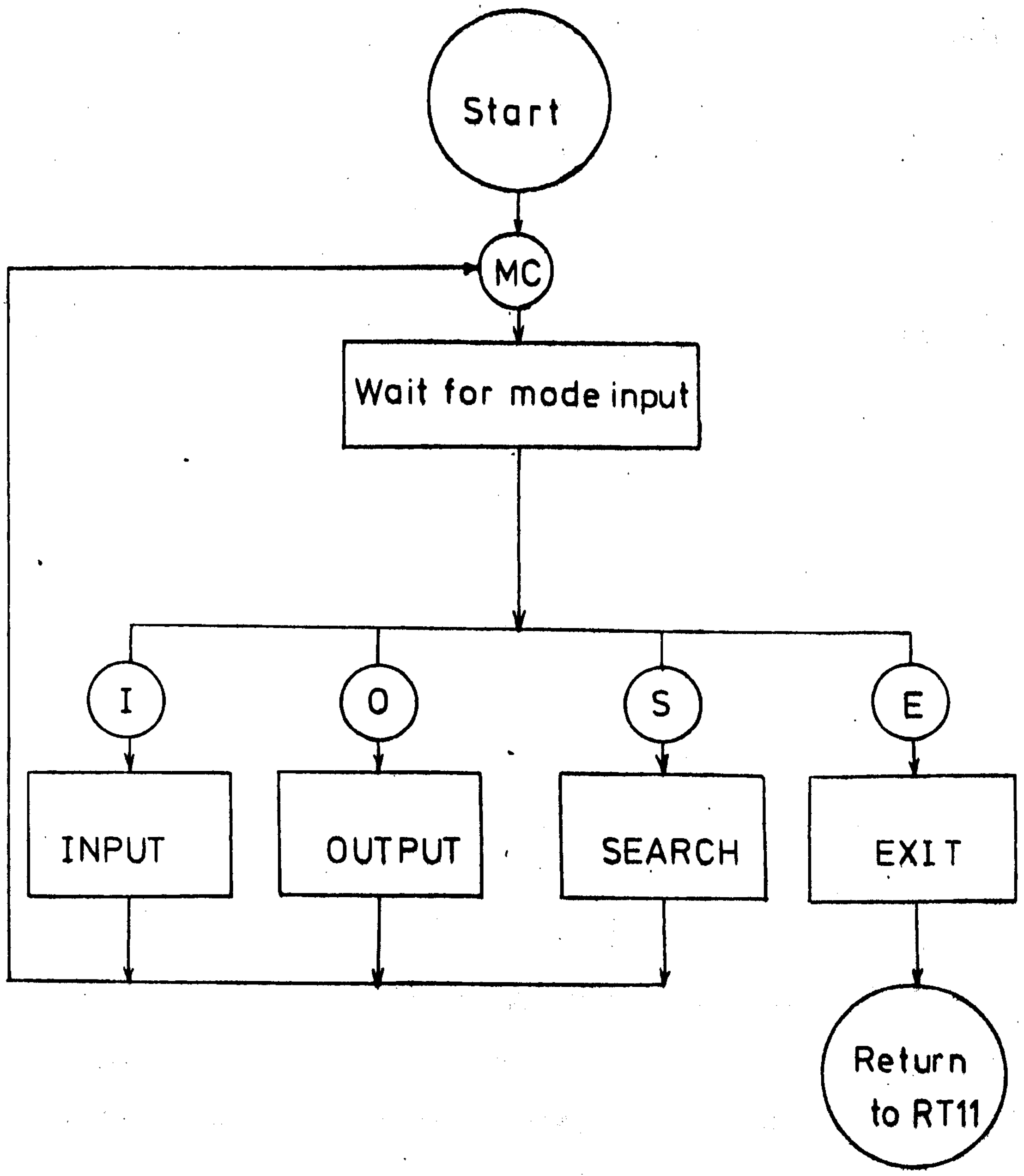


FIG. 6.34 Control program for the associative retrieval system simulation.

## 6.8 Discussions:-

The purpose of the present exercise was to carry-out a feasibility study of using an APP in an on-line retrieval application. The first phase of this work was to develop an on-line retrieval system, based on structure free data-base, utilising current resources available in 'APG' at Brunel University. This inter-active system was then used as a vehicle to develop and improve algorithms for such a system. Finally it was assumed that this application study would provide a basis for an evaluation of APP based systems when compared to its conventional counter part.

In this chapter underlying design philosophy and implementation of such a system are described. The simulation performs on-the-fly search of Keys, present in an user profile, against a bibliographic data-base. The current investigation was restricted to the search-part of the problem. The problem associated with creation of a new data-base and consequent maintenance of the data-base were beyond the scope of this work. The implemented system was initially constructed as a one user terminal. But it could be easily extended to accept many user terminals, working on either time-sharing or batch processing mode. The batch processing mode would be particularly useful to provide SDI and current-awareness services. In this simulation study a simple search equation (logical simmulation of Keys) as described in the section 6.6 is implemented. The algorithms for other proposed search criteria are also included. In the light of present experience, it has been seen that APP can provide a very flexible on-line retrieval system. The only difficulty of using an APP system is the unavailability of hardware. However, a project for implementation of micro-APP[182] on a single chip has been currently undertaken by researchers at

Brunel University. When a low-cost APP system is readily available, it can be used to replace a major portion of the software of the current simulation. On the other hand a completely dedicated stand-alone, on-line retrieval system, based on APP, can be devised. In such a APP based system, some inconsistencies in the user Keyword can be easily absorbed. Further increase in the softness of the system can be achieved by allowing for a set of fuzzy matching algorithms. Thus it is expected that an APP based system may lead to a very efficient and flexible on-line retrieval operation.

# CHAPTER 7.

## CONCLUSIONS

In an attempt to propose a cost-effective searching mechanism suitable for an on-line bibliographic retrieval system a survey of the conventional searching techniques has been carried out. This survey shows that the complexities of internal file - and data-structure are associated with the improvement of the response time of the system. These complexities lead to a degradation of the cost-effectiveness of the system. The usefulness of the two-level hierarchy of data-base has been observed. It has been shown that among all data-structures employed in an inverted file, the performance of the cellular serial file-structure is optimal.

The various possible alternatives of associative retrieval systems are studies. The 'ON-THE-FLY' searching techniques using a BO-VRL-APP system has been selected as a cost-effective searching mechanism.

A survey of associative parallel processor has been done. This includes architecture, operations, hardware, software and applications of an APP system.

It has been stated earlier that a formal specification for the BO-VRL-APP system has to be derived before the implementation of an associative retrieval system. A simulation of the BO-VRL-AOO system has been developed. This consists of a combination of hardware associative memory array and the software of a general purpose mini-computer.

The hardware emulation of the associative unit was implemented by 'Nand Gates' and was available at Brunel University. This comprises

i) AMA (Associative Memory Array)
ii) BCL (Bit Control Logic)
iii) WCL (Word Control Logic)
iv) Data routing registers

A software system has been developed to simulate

i) Micro-order generation logic
ii) Control Unit
iii) Program store
iv) I/O facilities

of the BO-VRL-APP system.

A two-way communication link for transferring data and control signals between the Associative memory hardware and the PDP 11 system has been established.

A provision for bulk initialization, loading and reading of the associative memory has been included.

To monitor the status of the associative memory hardware the memory map has been designed. The contents of the entire associative memory array along with the data and control signals transferred to and from the hardware are displayed by the memory map.

The facilities for specifying and monitoring an Associative Processing Instruction and a sequence of Dynamic-Micro-Orders have been incorporated. The API's are automatically converted by a special software subroutine to provide a desired combination of data and static-Micro-orders.

Execution of an API has been facilitated by transferring the specified SMO's and DMO's to the hardware. The verification of proper execution of the API was done by comparing successive memory maps.

The above experimental steps have been repeated for the entire set of proposed API's. Thus, this has not only tested the feasibility of the proposed API set, but also proved the logical operation of the BO-VRL-APP system. As the result of these experiments, a formal set of precise and unambigous API's has been specified.

The software facility has been extended to accept an API from the tele-type, where upon the desired sequence of DMO's are generated and the instruction is executed.

The software simulation has been further improved to accept a number of API's. A buffer has been allocated to store up to sixteen API's which are then executed sequentially. Two 'switch Register' options have been included; these allow

1) A hard copy of memory map to be printed out at the completion of each API.
2) Repeating the execution of the set of API's stored in the instruction buffer.

A main achievement of this experiment was the establishment of a well defined logical structure and unambiguous specification of API set for a BO-VRL-APP system. On the basis of this result a Micro-APP has been proposed. Currently an ATCP contract has been undertaken to implement the proposed Micro-APP. This will be developed as a joint venture between Brunel University and Plessey and will employ Schottky $I^2L$ technique.

The 'ON-THE-FLY' searching technique utilizing a BO-VRL-APP as a 'parallel search Unit' is chosen for the implementation of the proposed associative retrieval system.

The simplicity of flexibility of the data-structure employed in the index and profile have also been demonstrated.

A program to acquirethe desired fields from the inspec magnetic tape service has been developed. This enables the creation of a realistic data-base of chosen format on a disc Unit.

The simulation of the associative retrieval system has been developed. This has been done by adding two-control programs to the BO-VRL-APP simulation, these are

1) System control
2) API sequences and interpreter

The system control co-ordinates the input/output and search operations of the simulated retrieval system. The API's interpreter has been used to implement associative algorithms.

A two-part algorithm has been developed for the associative searching mechanism. This allows the selection of the desired records. The two-parts of the search algorithm are

1) Compare Algorithm

2) Document Hit Algorithm

The compare algorithm has been developed to mark the presence or absence of a Keyword in a record.

The document-hit algorithm has been developed to establish the validity of a record by evaluating the search - criterion. A number of various search criterion has been considered; these include

1) logical 'AND', 'OR' and NOT operation
2) Sum-of-product and product-of-sum terms.
3) Threshold search including 'm' out of 'n' and variation of emphasis of individual Keyword.
4) Interactive searching strategy to control the mode and precision of retrieval operation.

The facility to provide the statistics of the retrieval has also been included. The output file containing the sequences of memory maps has been made available to monitor the associative activities. This also provides a facility to debug and improve the algorithms.

The algorithms for performing the proposed search-strategies have been developed. Although the experimental varifications to justify the claims do not exist, the algorithms have been extensively checked and are expected to work satisfactorily.

The instruction counts involved in the compare and document hit algorithms indicate that for an average record of fifty characters long, the time required to perform document hit algorithm is insignificant compared to the compare algorithm. The compare algorithm shows that three instructions for each character will be required to perform an exact match operation. These instructions include back-tracking, matching a selected substring and marking the presence of a Keyword. Assuming an average API cycle time of 100ns, this indicates that a data-rate up to 3 M bytes/sec can be supported by the system.

The present investigation establishes the feasibility and provides a provisional specification of an associative retrieval system. It has been also seen that an efficient and flexible retrieval system can be supported by a simple and low-cost hardware. Thus it is envisaged that using the results of this investigation as a basis, a stand alone associative retrieval system can be developed. Alternatively a card containing the basic 'Parallel Search Unit' may be introduced between the main storage and a DMA channel. This would enable a sophisticated selection criterion to be evaluated for filtering out the desired records. Similar system could find applications in Content - Addressable-File-Storage (CAFS) systems which are now manufactured by ICL. This could lead to more flexible and cheaper solution to the CAFS systems.

## 7.1. Criticism of Work:

A thorough survey of most commonly employed conventional file-searching technique is done. This also includes inverted files. The Table 2.6 compares the performances of the various data-structures used in an inverted file. Although the comparison of the performances as shown in this table is not rigorous, it provides a qualititive basis for assessment of relative merits of the file-structures. Fig2.2 6 which summarizes the performances, clearly shows that the cellular serial file is better.

In order to select a provisional specification for a cost-effective associative searching mechanism a survey of associative parallel processing systems is carried out. Lots of publications by many workers on APP systems and its applications in IS&R system have been reported. In Chapter Three it was not attempted to cover these fields thoroughly. Instead the scope of the discussions was purposefully limited only to point out the basic idea of associative processing. Although the discussion was not rigorous, it was sufficient enough to justify the choice of the proposed associative searching mechanism.

Phase I:- This phase of work was essential to provide a basis for the implementation of the proposed associative searching mechanism. The experimental investigation was thoroughly carried out to derive a formal specification of the BO-VRL-APP system. The experimental BO-VRL-APP system was used by Reynolds and Ofulue for carrying two specific application studies. Finally on basis of this result an ACTP contract has been undertaken to implement a L.S.I. version of Micro-APP.

Phase II:- Unlike Phase I, the result of this phase of work was not obtained. The experiments of Phase II could not be performed because the simulation of the BO-VRL-APP suitable for practical application was not ready. The author was also pressed for returning to India.

However, from the experience gained during the Phase I best effort has been made to justify and predict the results of this phase of work. These may justify the indication that the BO-VRL-APP can easily support an 'ON-THE-FLY' searching technique, which can be used as a cost-effective associative searching mechanism.

## 7.2. Future Work:-

1. From the present investigation, it has been realised that the capabilities of an associative retrieval system in terms of flexibility, efficiency and speed cannot be fully appreciated unless a proto-type system incorporating a hardware APP is developed. This would also allow further improvement of an associative retrieval system. Hence in the opinion of the author further research should be carried out in order to achieve these goals.

2) The present investigation has been primarily carried out on the exact matching of the Keywords, but it is understood that for increasing flexibility, the problem of differences and inconsistencies in the context of the Keywords should be included. Hence it is suggested that further research should be carried out to incorporate 'whole', 'fragment', substring and universal character matching scheme within the system.

It is also envisaged that a set of fuzzy Keywords matching scheme, such as:

a) Transcription error
b) Transposition error
c) Omission error
d) Insertion error

and the combinations of them can be easily supported by an associative retrieval system. Whereas in a conventional information retrieval system these are extremely difficult to achieve. Hence further research should be carried out to incorporate these flexibilities.

3) The present investigation has been limited to a feasibility study of an associative retrieval system where the data-base has been restricted to reasonable size. However, as the size of the data-base grows, it becomes increasingly difficult to scan the entire data-base within an acceptable time. In such cases, it is suggested that further research should be carried out to investigate a two-level hierarchy of associative addressing. This concept of two-level hierarchy is similar to a cellular serial inverted file-structure. Hence this would include the advantage of a cellular serial file-structure. Moreover, the scanning at each level would be equally benefitted by the simplicity and flexibility of the 'ON-THE-FLY' searching technique.

4) It is considered that on-line facilities of dictionary consultation for the profile formulation would be useful. Finally it is suggested that the research should be carried out for developing a multi-terminal on-line associative retrieval system which would include all above mentioned facilities.

## BIBLIOGRAPHY

1. SHARP, J.R., "Some Fundamentals of Information Retrieval" Deutsch, 1975.

2. MEADOW, C.T., "The Analysis of Information Systems - A Programmer Introduction to information Retrieval" Wiley, 1967.

3. LANCASTER, F.W., "Information Retrieval Systems", Eiley 1968.

4. LYNCH, M.F., "Computer-based Informational Services in Science and Technology; Principles and Techniques". Peter Peregrinus, 1974.

5. VICKERY, B.C., "On Retrieval system Theory", Butterworths 1965.

6. MEETHAM, R., "Information Retrieval - the Essential Technology", Aldus 1969.

7. KENT, A., "Information Analysis and Retrieval". Wiley, 1971.

8. BECKER, L., and HAYES, R.M., "Information Storage and Retrieval: Tools, Element, Theories.

9. SALTON, G., - "The SMART Retrieval System - Experiment in Automatic Document processing". Prentice-Hall 1971.

10. HENLEY, J.P., "Computer-Based Library and Information Systems", Macdonald, 1972.

11. LEFKOVITZ, D., "File structures for on-line systems", Spartan Book, 1969.

12. KNUTH, D.E., "The Art of Computer Programming". Vol. 3., Wesley 1973.

13. HELLERMAN, H., "Digital Computer System Principles", McGraw-Hill 1967, pp. 114-159.

14. GEAR, C.W.., "Computer Organization and Programming" McGraw-Hill, 1976, pp 376-409

15. DIJKSTRA, E.W., "A discipline of Programming" Prentic-Hall 1976.

16. NEWELL, A., et Al, :- A command structure for complex information processing, AFIPS, Vo. 13, p.119, 1958.

17. NEWELL, A. TONGE, F.M., :- An Introduction to information processing language V, CACM, Vol.3 p 205, 1960.

18. DEWEY, MELVIL, "Deway Decimal Classification and Relative Index", Forest Press, N.Y. 1959.

19. MILLS, J., "Guide to the use of the VDC". British standard institution, 1963.

20. VICKER, B.C., "Faceted Classification: a Guide to construction and use of special schemes" ASLIB, 1960

21. RANGANATHAN, S.R., "Colon Classification", Rutgers, New Brunswick, N.J. 1964.

22. RANGANATHAN, S.R., "Classified Catalogue Code with Additional Rules for a Dictionary Catalogue Code" Asia Publishing House, 1964.

23. JASTER, J.J., MURRAY, B.R., TAUBE, M., "The state of the Art of Co-ordinate indexing". Documentation Inc. Feb 1962.

24. WADINGTON, J.P., "Unit concept co-ordinate Indexing", American Documentation, 9, No.2, pp 107-113.

25 FARRARDANE, J., DATTA, MRS. S., POULTON, R.K. "Report on Research on Information Retrieval by Relational Indexing" The City University, 1966.

26. LANCASTER, F.W., "On the Need for Role Indicators in Post-colordinated systems", American Documentation, 19, No.1 pp 42-46, 1968.

27. , "Thesaurus of ASTIA Descriptions". Dec. 1962.

28. PAPIER, L., "Reliability of Scientist in supplying titles: Implications for Permutation Indexing", ASLIB proc.15. No.11, pp. 333-337 Nov. 1963.

29. RESNICK, A., "Relative effectiveness of Document Titles and Abstracts for Determining Relevance of Documents", Science, 134, No 3484, pp 1004-1005, OCT 1961.

30. I.B.M. - "General Information Manual, Keyword-in-context Indexing", 1962.

31. LUHN, H.P., "Potentialities of Auto-Encoding of Scientific Literature", Research Report RC-101, IBM, May 1959.

32. MARON, M.E., "Automatic Indexing: an Experimental Inquiry", Journal of ACM, 8, No. 3. pp 404-417, Jul. 1961.

33. BAXENDALE, P.B., "Machine-made Index for Technical Literature - an Experiment", I.B.M. Journal of Res. and Dev. 2, No.4, pp 354-361, Oct., 1958.

34. STONE, H.S., "Introduction to computer Organization and Data structures" pp. 263-292, McGraw-Hill 1972.

35. CLIMENSON, W.D., "File organization and search techniques." Annual Review of Info. sc. and tech. Vol 1, pp 107-135 Wiley, 1966.

36. DODD, G.G., "Elements of Data Management systems", Computer Survey, Vol. 1, No.2, pp. 117-133, Jun 1969.

37. PATTERSON, G.W., "Theory and techniques for the design of electronic digital computers". 1946.

38. LANDAUER, W.I., "Balanced tree and its utilization in information retrieval" IEEE Trans EC12, pp 863-871, 1963.

39. , "Introduction to IBM/360 direct access storage devices and organisation methods". C 20-1649. IBM Comp. 1966

40. MAURER, W.D., LEWIS, T.G., "HASH Table methods", Computing Surveys Vol. 7, No.1., pp 5-19 Mar. 1975.

41. BOWDEN, K.F., JONES, D.M., STANDEVEN, J., FORTH, L., SLOMAN, M. "A low-cost content Addressable Memory using conventional Memory Elements", IEE Conf. on Computer Systems and Tech, (IEE Pub. No. 121), pp 195-200, 1974.

42. CAGAN, C., "Data Management Systems", Melville 1973.

43. BEZTISS, A.T., :- Data Structives, Theory and practice, Academic press, London 1971.

44. WILKES, M.V., :- Lists and why they are useful, Computer Journal 7, p 231, 1964

45. JOHNSON, L.R., "An Indirect chaining Method for Addressing Secondary Keys", Comm. ACM, pp 218-222, May 1961.

46. BURKS, A.W., GOLDSTINE, H.H., VON NEWMAN, J., :- Preliminary Discussion of the Logical Design of an Electronic Computing instrument, Collected works, Vol. 5., p 34. Pergamon, 1961.

47. PARHAMI, B., - Associative Memories and processor: An Overview and selected Bibliography, Proc. IEEE, Vol.61 p 722, June 1973.

48. HENLAN, A.G., - Content Addressable and Associative Memory Systems - A Survey. IEEE Trans. Vol. EC - 15, p 505 1966.

49. THURBER, K.J. AND WALD, L.D., "Associative and Parallel processors" Computing survey, Vol. 7 No.4, pp 214 - 255., Dec 1975.

50. FENG, T., - An overview of parallel processing systems, Wescan Technical Paper., Vol. 16, Session 1., 1972.

51. LEA, R.M., :- Information processing with an associative parallel Processor, IEEE Computer Vol. 8, No.11, p 25, Nov 1975.

52. LEA, R.M., - An associative parallel processing system for the memory structure of a symbol processing machine, Brunel University, Tech, Memo No, C/SP/014, 1972.

53. LEA, R.M., Information Processing with an Associative Parallel Processor, Brunel University Tech Memo No. C/SR/021, Feb 1975.

54. LEA, R.M., WRIGHT, J.S., - A novel memory concept for information processing, Datafair Research Papers, Vol. II, p.413, 1973

55. LEWIN, D.W. :- Highly parallel processing systems, theory and design of digital systems, p.306-334 Nelson & Sons Ltd., London 1972.

56. HOBBS, L.C., et. al :- Parallel processing Systems technologies and applications, Spartan Books, N.Y. 1970.

57. WRIGHT, J.S., - Design philosophy for a symbol processing Machine, PhD. Thesis, University of Southampton 1972.

58. WRIGHT, J.S., :- System design of a Symbol Processing Machine, Internal circulation, Brunel University, Dept of Elect. Engg., Uxbridge, Oct 1969.

59. EWING, R.G., DAVIES, P.M., - An associative Processor, Proc. AFIPS (FJCC), Vol. 26, p 147, 1964.

60. DAVIES, P.M., :- Design for an Associative Computer, Proc. IEEE Pacific Computer Conf. 1963.

61. LEWIN, D.W. - Whither Data-processing, The Radio and Electronic Engineer, Vol. 45, No. 10, P 627, Oct 1975.

62. RUDOLPH, J.A., FULMER, L.C., MEILANDER, W.C., - The coming of Age of the associative process, Electronics P 91. 15 Feb 1971.

63. DUGAN, J.A., GREEN, R.S., MINKER, J., SHINDLE, W.E., - A study of the utility of associative memory processors, Proc. ACM National Meeting, P 347, 1966.

64. FULLER, R.H., Associative parallel processing, Proc. AFIPS (SJCC) Vol. 30, p 471, 1967.

65. MULLERY, A.P., et Al,:- ADAM -A problem-oriented symbol Processor, AFIPS, Vol. 23, p 367, 1963.

66. McKEEVER, B.T.,:- The associative memory structure, Proc. F.J.C.C. Vol. 27, p 371, 1965.

67. ROGERS, T.W. - Data Bases; Their impact on Computer Hardware, IEE Conf. Computer systems and technology (IEE publication No 121), p 150, 1974

68. RUDOLF, J.A., "A production inplementation of an associative array processor - STARAN", AFIPS Conf Proc. Vol. 41 pp 229-241, 1972.

69. BATCHER, K.E., "STARAN/RADCAP hardware architecture" Proc. Sagamore Comput. Conf. on Parallel Processing pp 147-152, 1973.

70. BATCHER, K.E., "STARAN Parallel Processor System Hardware", AFIPS Conf. Proc. Vol. 43, pp 405-410, 1974.

71. WEINBERGER, A., "The hybrid associative memory concept", Computer Design, pp 77-85, Jan 1971.

72. LEE, C.Y.:- Inter-communicating cells, basis for a distributed Logic Computer, Proc. Fall Joint Computer Conf. AFIPS, Vol. 22, p.130, 1962.

73. GAINES, R.S., LEE, C.Y.:- An inproved cell memory, IEEE trans.,Vol. EC-14, p 72, 1965.

74. LEE, C.Y., PAUL, M.C., :- A content-addressable distributed Logic memory with Applications to information retrieval, Proc. IEEE Vol. 51, P 924, 1963.

75. STURMAN, J.N., :- A Iteratively structured Digital computer, PhD Thesis, Cornell University, 1966

76. STURMAN, J.N.:- An Iteratively structured General Purpose Digital Computer, IEEE trans on Computers, Vol. C 17 p 2., 1968.

77. STURMAN, J.N., :- A synchronous operation of an Iteratively structured G.P. D.C., IEEE trans. on computer, Vol. C 17 p 10, 1968.

78. LIPOVSKI, G.J.,:- The architecture of a large distributed logic associative processor, Co-ordinated Science Lab, R-24 July 1969.

79. LIPOVSKI, G.J., :- An architecture of a large associative processor, AFIPS Proc. SJCC, p 385, 1970.

80. CRANE, B.A., GITHENS, J.S., :- Bulk Processing in distributed logic memory, IEEE trans , Vol. EC-14, p 186, 1965.

81. KISYLIA, A.P., :- An associative processor for information retrieval, Co-ordinated science Lab (Illinois University) Report R-390 (AD 675310), Aug. 1968.

82. AGRAWAL, D.P., ;- Simultaneous complex search in associative memories, Proc. Computer System & Technology Conf. No.121 180 - 181, London, 1974.

83. BEAVAN, P.A., LEWIN, D.W., - An associative parallel processing system for non-numerical computation, The Computer Journal Vol. 15., p343, No.4 1973.

84. WRIGHT, J.S., LEWIN, D.W.:- A draft specification for a symbol processor, IEE Conf. Computer Science & Technology (IEE Publication No 55) p 282, 1969.

85 SEEBER, R.R., LINDQUIST, A.B.:- Associative memory with ordered retrieval, I.B.M. Jorunal of R & D, Vol. 6, p 126 1962.

86. KAPLAN, A., "A search memory subsystem for a general purpose computer", FJCC, AFIPS Conf Proc. pp. 193-200, 1963.

87. FOSTER, C.C., STOCKTON, "Counting Responders in an associative memory", IEE Trans on Computer, Vol. C-20, pp 1580-1583, Dec 1971.

88. LEWIN, M.H. - Retrieval of ordered lists from a content addressed memory, RCA Review, vol. 23 p 215, 1962.

89. SLADE, A.E., McMAHON, H.O., :- A cryoton Catalog memory system. Proc. E.J.C.C. 10 p 115, 1956.

90. CROWE, J.W., "Trapped-flux super conductive memory", I.B.M. Res. and Dev. Vol. 1 pp 294-303, Oct. 1957.

91. SLADE, A.E., "A cryotron memory Cell" Proc. IRE, Vol 50, pp 81-82, Jan 1962.

92. SLADE, A.E., SMALLMAN, G.B., "Thin-film cryotron catalog memory system", Proc. EJCC, Vol. 10, pp 81-82 Jan 1962.

93. DAVIES, P.M., "A super-conductive associative memory", Proc SJCC (AFIPS), Vol. 21, pp. 79-88, May 1962.

94. ROSIN, R.F., An organisation of an associative cryogenic computer, Proc SJCC, Vol. 21, P 203 1962.

95. NEWHOUSE, V.L., FRUIN, R.E., :- A cryogenic data addressed memory, Proc. S.J.C.C., Vol. 21, p 89 1962

96. KISEDA, J.R., PETERSEN, H.E., SEELBACK, W.C., TEIG, M. :- A magnetic associative memory, I.B.M. Journal of R & D, Vol. 5., p 106 1961

97. SHEAD, C.J. - The associative memory - A versatile circuit element G.E.C. Jorunal of Science and Technology, Vol. 40 p 119 No. 3. 1973.

98. APICELLA, A., FRANKS, J., "BILOC _ A high-speed NDRO One Core-per-bit Associative element". Int. Conf. on Magnetics. April 1965.

99. CHOW, W.F.:- Plated wire conten-addressable memories with bit-steering technique, IEEE Trans. EC - 101. 16, p 642, 1967

100. EWING, R.C., DAVIES, P.M., "An associative processor", Proc. FJCC, Vol 25, pp 147 - 149, 1964.

101. FULLER, R.H., "Content-addressable memory systems" Disser. Absts. Vol. 24. p 1960,611 pp. Nov 1963.

102. TUTTLE, G.T., "How to quiz a whole memory at once" Electronics, Vol 36. pp 43 - 46, Nov. 1963.

103. LEA, R.M. - Low-cost high-speed associative memory, Brunel University Tech. Memo. No CE/R/023 1975 - IEEEE - TSSC, Vol SC-10 No. 3, p 179, June 1975.

104. LEA, R.M., - A Design for a low-cost High speed MOS Associative memory, Brunel University Tech. Memo No. CE/R/022, 1975. - Radio and Electronic Engineer, Vol. 45, No.4, p 177 April 1975.

105. LEA. R.M., Towards a Low-cost cell design for High-speed MOS Associative Memories, Datafair Research Papers, Vol. 11, p 418, 1973.

106. LEA, R.M. - A Design for a High Speed MOS Associative Memory Electronics Letters, Vol. 8, p 391, 27th July 1972.

107 IGRARASHI, R., YAITA, T., :- An integrated MOS Transistor Associative Memory with 100 nsec cycle time, Proc S,J,C,C, Vol. 30, p 499, 1967.

108. HERLEIN, R.F., THOMPSON, A.V., :- An integrated associative memory element, I.S.S.C.C., Digest of Tech papers, p 42 Feb 1969.

109. BIDWEL, A.W., PRICER, W.D. :- A high speed associative memory, I.S.C.C. Digest of Tech. Papers p 78, Feb 1967.

110. HOFF, M.E., :- Designing a L.S.I. memory system that out-performs coves-economically. Design Electronics, p 33, April/May 1971.

111. FREDKIN, E., :- Tric Memory, Vomn ACM, Vol. 3, p 490, 1960

112. BARTLETT, J., MUDGE, J., SPRINGER, J.,:- Associative memory chips: fast voratile and here, Electronics, p 96, Aug 17, 1970.

113. ASPINALL, D., KINNEMENT, D.J., EDWARDS, D.B.G.:- An integrated associative memory matrix I.F.I.P. Congress (Edinburgh) p D 86, Aug 1968.

114. FELDMAN, J.D., FULMER, L.C. - RADCAP - An operational parallel processing facility. Proc. AFIPS (NCC), Vol 43, p 7, 1974.

115. De FIORE, C.R., VITO, A.A., BAUER, L., - Toward the development of a higher order language for RADCAP, Proc of the 1972 Sagamore Computer Conf. p 99 1972.

116. DAVIS, E.W. - STARAN Parallel processor system software, Proc AFIPS (NCC), Vol. 43, p 17 1974.

117. LINDE, R.R., GATE, R., PENG, T., - Associative processor application to real-time data management Proc. A.F.I.P.S. (NCC) Vol. 42, p. 187, 1973.

118. PATTERSON, W.W., - Some thoughts on associative processing languages AFIPS Vol. 43, p. 23 1974.

119. FINDLER, N.V., "On a computer language which simulates Associative memory and parallel processing", Cybernetica, Vol. 10., No. 4 pp 229 - 254, 1967.

120. DODD, G.G., - APL - A Lanugage for associative data handling in PL/1, Proc AFIPS (FJCC), Vol. 29, p 677, 1966.

121. THURBER, K.J., MYRNA, J.W.:- System Design of a cellular APL computer, IEEE trans, Computers, Vol. C - 19, No.4, p 291, 1970.

122. SAVITT, D.A., LOVE, H.H., TROOP, R.E. - ASP - A new concept in language and machine organisation, Proc, AFIPS (SJCC) Vol. 30, p 87 1967.

123. FELDMAN, J.A., ROVNER, P.D., - An Algol-based associative language Commun. A.C.M. Vol. 12, No.8 p 439 Aug. 1969.

FELDMAN, J.A. - Aspects of associative processing MIT Lincoln Labs. Tech note 1965 - 13 1965.

125. ASH, W.L., SIBLEY, F.H. - TRAMP: A relational memory with an associative base, AD 672206

126. ABRAHAM, P.W., et al, The LISP2 Programming Language and system AFIPS, Vol. 29, p 661, 1966.

127. DONNELLY, R.K. : - VAPP (Simulation of a virtual associative parallel processor) WR/2/75, Internal circulation. Brunel University, Elect. Engg. 1975.

128. DONNELLY, R.K., :- IFPL: An Inyermediate level field processing Language for associative parallel processors, Brunel University tech. memo No. C/R/028, July 1975.

129. DONNELLY, R.K., :- SNAPP; A simulation of a variable record-length parallel processor. Internal circulation, Brunel University Elect. Engg. June 1975.

130. FINDLER, N.V., - On a computer language which simulates Associative memory and parallel processing, cybernetica, Vol. X p 229 1967.

131. DONNELLY, R.K., :- A survey of string processing techniques, Internal circulation, Brunel University, November 1972.

132. KODIN, Y.Y., :- Logical analysis of associative memory structures, Cybernatics (U.S.A.), Vol. 6, No. 4, p 522, July/Aug 1970.

133. GREEN B.F., Computer languages for symbol manipulation, IEEE trans, Vol. EC 10, p 729 1961.

134. BOBROW, D.G. RAPHAEL, B., :- A comparison of List Processing languages, Communication ACM, Vol. 7., p 231. 1964.

135. FARBER, D.J., et al., :- The SNOBEL 3 Programming Language, Bell System, tech. Journal p 895, July/Aug 1966.

136. FARBER, D.J., FRISWOLD, R.E., POLONSKY, I.P., - SNOBOL, string manipulation Lanugage, JACM Vol. 11 p 21, Feb 1964,

137. THURBER, K.J., BERG, R.O., - Applications, of Associative processors, Computer Design, p 103, NOv. 1971.

138. ELLIS, A.B.E. - The Associative Memory and its applications, Marconi Review, P 42, First Quarter 1972,

139 MOULDER, R., - An implementation of a data management system on an associative processor, AFIPS Vol. 42, p 171. 1973

140. De FIORE, C.R., BERRA, P.B., :- A Quantitative analysis of the utilisation of associative memories in Data Management, IEEE trans. on Computer Vol. C - 23, p 121, Feb 1974.

141. De FIORE, C.R., BERRA, P.B.:- A Data management system utilising an associative memory, National computer Conf., p 187, 1973.

142. De FIORE, C.R., - An analysis of associative Processing methods in data management AD 750 147

143. CRANE, B.A. - Path finding with associative memory, IEEE Trans. Comp. Vol. C - 17 p 691 July 1968.

144. LIPOVSKI, G.J.:- On Data Structures in Associative memories, SIGPLAN Notices, Vol. 6, No. 2, p 346, Feb 1971.

145. POPOVA, G.M., PRANGISHVILI:- Associative Parallel processor for grouped processing of Data, Automation & Remote Control (U.S.A.) Vol. 33., No. 1., Part 2, p 152, Jan 1972.

146. "Pattern recognition by using an associative memory" Electronic Computer, Vol EC-15 pp 944-947, Dec 1966.

147. FINDLER, N.V., McKENZIE, W.R., - On a new tool in articicial intelligence research, an associative memory parallel processing language AMPPL - II, Proc. Int. Joint. Conf. on Artificial Intelligence p 259, May 1969.

148. FULLER, R.H., BIRD, "An associative parallel processor for picture processing". Proc. FJCC, pp 105 - 116 1965.

149. DUFF, M.J.B., WATSON, D.M., FOUNTAIN, T.J. AND SHARE, G.K. - A cellular logic array for image processing, Patter Recognition 5 p 229, 1973.

150. FULLER, R.H., BIRD, R.M.:- An associative parallel Processor with application to picture processing. Proc. F.J.C.C. Vol. 27 p 105 1965.

151. LEA, R.M. "An associative parallel processor for efficient and flexible file-searching", Proc IEEE Int, symp.on tech. for SDI pp 73-78, Sept 1976.

152. DONNELLY, R.K., "Some thoughts on using an associative processor for Information Retrieval", Tech. Memo C/R/042 Brunel University August 1976.

153. STOTNICK, D.L., "Logic per track Devices" Advances in Computers Vol. 10, Academic Press pp 291-196, 1970.

154. PARHAMI, BEHROOZ:- A highly parallel computing system for information retrieval -RAPID AFIPS Proc. FJCC, p 13, 1972.

155. COULOURIS, G.F., EVANS, J.M., MITCHELL, R.W., "Towards content- addressing in Data-Bases", Computer Journal, 15 pp. 95 - 98 1973.

156. NOE, J.D. NOE, "MIRF (multiple instantaneous response file) "Electronics, Vol. 35 pp 31 - 36 May 1963.

157. FENG, T., - Large scale information processing systems, U.S. Dept. of commerce Report (AD 708725), studies of associative memory systems Vol. 5, May 1970.

158. GOLDBERG, J., AND M. W. GREEN, "Large files for information retrieval based on simultaneous interrogation of all items", Large-capacity memory technique for computing systems, MacMillan, pp 63-67 1962.

159. PETERSON, H.E., Content addressing and information retrieval" IFIPS, Aug 1962.

160. MINNICK, R.C., Magnetic comparaters and code converters", Proc. symp-application of switching theory in space technology pp 193 - 204 Feb 1962.

161. BROWN, J.R. "A semi permanent associative memory and code converter. Conf. on Non-linear Magnetics, Nov. 1961

162. EDDEY, E.E., "The use of associative processors in radar tracking and correlation", Nat. Aerospace Electronics Conf. Proc. pp 39 - 42, 1970.

163. GITHENS, J.A. "An associative, highly parallel Computer for radar data processing", Parallel processor systems, technologies and applications, Ed. Hobbs et all, Spartan, pp 71-86 1970.

164. COSTANZO, A., GARRETT, J., "Application of an associative processor to an interceptor radar system," Nat. Aerospace Electronics Conf. Proc. pp 107-112 1969.

165. THURBER, K.J., "An associative processor for air traffic control", SJCC, AFIPS Conf. Proc. pp 49-59 1971.

166. MORENOFF, E., et al. "4-Way Parallel Processor Partition of an Atmospheric Primitive Equation Predition model"., Proc. AFIPS SJCC pp 39-48 1971.

167. LINDQUIST, A.B., SEEBER, R.R., COMEAN, L.W., :- A time sharing system using an associative memory. Proc IEEE Vol. 54, p 1774, 1966.

168. WALD, L.D., ANDERSON, G.A., "Associative memory for multiple processor control", Final Report NAS 12 - 2087, Sept 1971.

169. BERG, R.O., JOHNSON, M.D., "An assoicative memory for Executive Control Functions in an advanced Avionics computer system", Proc. of IEEE int. comp, group Conf. p 336 - 342, Jun 1970.

170. ERWIN, J.D., JENSEN, E.D., "Interrupt processing with queued content addressable memories". Proc. AFIPS FJCC, pp 621-627 1972.

171. DYKE, J.G., LEA, R.M., - An associative parallel processor for cost-effective local editing applications, Brunel University Tech Memo No C/R/025 1975.

172. DONNELLY, R.K., LEA, R.M., - The application of an associative parallel processor to data-compression for conventional file storage systems, Brunel University Tech Memo No C/N/025. 1975.

173. LEA, R.M., "Nand-gate Implementation for Associative memory", Digital processes, 2, pp 83-88, 1976.

174. REYNOLDS, W.T., "Local text editing", Masters Thesis, Brunel University, October 1976.

175. OFULUE, J.N.,"A survey of List and String Processing languages" Brunel University Tech Memo No C/SR/049, Jan 1976.

176. NICOLAOU, N.P., LEA, R.M., "Implementation of an experimental research vehicle for FRL APP investigation", Brunel University, Tech. Memo. No. C/R/027, June 1975.

177. PDP - 11 Processor Handbook, Digital Equipment Corp, Massachusetts 1975.

178. VT - 11 Graphic Display Processor, D.E.C. 1974.

179. PDP - 11 Peripherals Handbook, D.E.C. 1975.

180. RT - 11 F/B System Reference Manual, D.E.C. 1976.

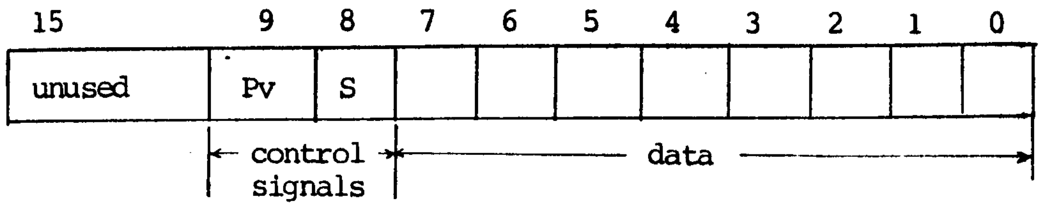
181. INSPEC Tape Service Manual, Inst. of Elec. Engg. London.

182. LEA, R.M. "Micro-APP: a building block for low-cost high-speed associative parallel processing" The Radio and Electronic Engr. Vol. 47. No. 3, pp 91-99 March 1977.

## APPENDIX A

### Description of information transferred through interface.

The input data highway to the associative memory as described in Section 3.1. is redrawn.



Input highway

The low-order eight bits (0-7) of DR11-C contain data information, and two bits (8,9) carry control signals. The control signal 'S' resets the interface logic and 'Pv' enables data to be stored in either SMO or DMO register.

The complete loading of SMO register requires seven interface transfer cycles. Two transfer cycles are required to load DMO register at each time slots. The contents of each transfer cycle along with the transfer sequence number is given below. The glossary of symbols describes individual signals.

## A.1 Input Transfer.

### A.1.1. STATIC MICRO-ORDER

Transfer Sequence Number

Contents of DR 11-C

1.

|  |  |  |  |  |  |  |
|---|---|---|---|---|---|---|
|  |  |  | symbol bit 8 |  |  |  |
| 0 | 0 | Symbol bit 7 | $D_{AW}$ | $D_{BW}$ | $D_{AS}$ | $D_{BS}$ |

2.

|  |  |  |  |
|---|---|---|---|
| 0 | 0 | Symbol bit 5 | symbol bit 6 |

3.

|  |  |  |  |
|---|---|---|---|
| 0 | 0 | Symbol bit 3 | symbol bit 4 |

4.

|  |  |  |  |
|---|---|---|---|
| 0 | 0 | Symbol bit 1 | Symbol bit 2 |

5.

|  |  |  |  |
|---|---|---|---|
| 0 | 0 | Control bit 2 | Control bit 1 |

6.

|  |  |  |  |
|---|---|---|---|
| 0 | 0 | Control bit 4 | Control bit 3 |

7.

|  |  |  |  |  |  |  |  |  |  |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 |  |  | RU | B | A | RN | ST | LN |

### A.1.2. Dynamic Micro-order

1.

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | $O_{2R}$ | $ST_C$ | $\emptyset_{xc}$ | $\emptyset_{yc}$ | $O_{2L}$ | $ST_s$ | $\emptyset_{xs}$ | $\emptyset_{ys}$ |

2.

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | TM | RW | MW | MM | GR | TG | $O_2$ | $O_1$ |

## A.2. Out put transfer

1.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| OVA | OVB | MR | Symbol bit 5 | Symbol bit 6 | Symbol bit 7 | $D_{OA}$ | $D_{OB}$ |

2.

| | | | | | | |
|---|---|---|---|---|---|---|
| OVA | OVB | MR | Symbol bit 1 | Symbol bit 2 | Symbol bit 3 | Symbol bit 4 |

3.

| | | | | | | |
|---|---|---|---|---|---|---|
| OVA | OVB | MR | Control bit 4 | Control bit 3 | Control bit 2 | Control bit 1 |

## A.3. Glossary of symbols/notations

A.3.1.

| Signal name Control signals | Function/Description |
|---|---|
| S | 1; reset interface<br>0; inactive |
| Pv | 1; Enable DMO register input to load data from input highway.<br>0; Enable SMO register input to load data from input highway. |

## A.3.2. Static Micro Order

### A.3.2.1. Data Search and write

$D_{AS}$, $D_{BS}$ Search Data

| $D_{AS}$ | $D_{BS}$ | |
|---|---|---|
| 0 | 0 | don't care |
| 0 | 1 | search zero |
| 1 | 0 | search one |
| 1 | 1 | illegal |

$D_{AW}$, $D_{BW}$ write data

| $D_{AW}$ | $D_{BW}$ | |
|---|---|---|
| 0 | 0 | stand by |
| 0 | 1 | write one |
| 1 | 0 | write zero |
| 1 | 1 | illegal |

### A.3.2.2. Direction Specification

| | | |
|---|---|---|
| LN | 1; | enable left neighbour |
| RN | 1; | enable right neighbour |
| ST | 1; | enable straight through |

### A.3.2.3. Run specification

| Type of Run | Code | R1 | R2 | A | B | RV |
|---|---|---|---|---|---|---|
| No run | N | 0 | 0 | 0 | 0 | 1 |
| Top run | T | 0 | 1 | 1 | 0 | 1 |
| Bottom run | B | 1 | 0 | 0 | 1 | 1 |
| Group run | G | 1 | 1 | X | X | 0 |

## A.3.3. Dynamic micro-order signals

### A.3.3.1. Bit selection logic

| | | |
|---|---|---|
| $ST_C$ | 1; | Enable data $D_B$ Control bit |
| $ST_S$ | 1; | Enable data $D_B$ symbol bit |
| $\emptyset_{xc}$ | 1; | Enable time phase X control bit |
| $\emptyset_{xs}$ | 1; | Enable time phase X symbol bit |
| $\emptyset_{yc}$ | 1; | Enable time phase Y control bit |
| $\emptyset_{ys}$ | 1; | Enable time phase Y symbol bit |

### A.3.3.2. Strobe tag

| | | |
|---|---|---|
| TG | 1; | Strobe tag register 1 |
| GR | 1; | Strobe tag register 2 |

### A.3.3.3. Word logic signals

| | | |
|---|---|---|
| MW | 1; | Multiwrite |
| MM | 1; | Compare mismatch |
| $O_1$ | 1; | Strobe 1 |
| $O_2$ | 1; | Strobe 2 |
| $O_{2R}$ | 1; | Enable Run (Right) |
| $O_{2L}$ | 1; | Enable Run (Left) |
| TM | 1; | Enable top of memory |
| R/W | 1; | Strobe read register |

### A.3.4. Output signals

$D_{OA}$, $D_{OB}$ data output

| $D_{OA}$ | $D_{OB}$ | |
|---|---|---|
| 0 | 0 | multiple response |
| 0 | 1 | zero output |
| 1 | 0 | one output |
| 1 | 1 | no output |

| | |
|---|---|
| MR | Match reply |
| OVA | Overflow at A |
| OVB | Overflow at B |

# APPENDIX B

## Specification for the Instruction Set of the BO-VRL-APP

### B.1.1. BO-VRL-APP Input

The only input to the BO-VRL-APP comprises a 59-bit Associative Processing Instruction (API) which is defined below in BNF notation.

⟨API⟩ :: = ⟨FN⟩ ⟨DA⟩ ⟨DM⟩

| | | | |
|---|---|---|---|
| Function | ⟨FN⟩ | :: = | ⟨OP code⟩⟨data⟩=⟨RW ⟨Ch.spec.2⟩ ⟨ CB.spec.2⟩ |
| Domain Addresses | ⟨DA⟩ | :: = | ⟨Ch.spec.1⟩ ⟨CB.spec.1⟩ |
| Domain Modifier | ⟨DM⟩ | :: = | ⟨CO⟩ ⟨C⟩ ⟨PR⟩ ⟨RN⟩ |

| | | | |
|---|---|---|---|
| Clear Options | ⟨CO⟩ | :: = | ⟨MW⟩ ⟨CC⟩ ⟨CB⟩ ⟨MM⟩ |
| Propagate tags | ⟨PR⟩ | :: = | ⟨U⟩ ⟨S⟩ ⟨D⟩ |
| Run tags | ⟨RN⟩ | :: = | ⟨R1⟩ ⟨R2⟩ |

⟨Ch.spec.1/2⟩ :: = ⟨T⟩⟨T⟩⟨T⟩⟨T⟩⟨T⟩⟨T⟩ ⟨T⟩⟨T⟩

⟨CB.spec.1/2⟩ :: = ⟨T⟩⟨T⟩⟨T⟩⟨T⟩

⟨T⟩ :: = 0/1/X

| | | | |
|---|---|---|---|
| Read/Write | ⟨RW⟩ | :: = | 0/1 |
| Multi-Write | ⟨MW⟩ | :: = | 0/1 |
| Clear Character | ⟨CC⟩ | :: = | 0/1 |
| Clear Control-Bits | ⟨CB⟩ | :: = | 0/1 |
| Match/Mismatch | ⟨MM⟩ | :: = | 0/1 |
| Complement tags | ⟨C⟩ | :: = | 0/1 |

| | | |
|---|---|---|
| Propagate Up | ⟨U⟩ | :: = 0/1 |
| Straight-through | ⟨S⟩ | :: = 0/1 |
| Propagate Down | ⟨D⟩ | :: = 0/1 |
| Run bit R1 | ⟨R1⟩ | :: = 0/1 |
| Run bit R2 | ⟨R2⟩ | :: = 0/1 |

## B.1.2. BO-VRL-APP Output

The only output from the BO-VRL-APP comprises a 15-bit Output Word, which is defined below in BNF notation.

⟨Output Word⟩ :: = ⟨Ch.Field⟩ ⟨CB.Field⟩ ⟨ST⟩

⟨Ch. Field⟩ :: = ⟨B⟩ ⟨B⟩ ⟨B⟩ ⟨B⟩ ⟨B⟩ ⟨B⟩ ⟨B⟩ ⟨B⟩

⟨Cb. Field⟩ :: = ⟨B⟩ ⟨B⟩ ⟨B⟩ ⟨B⟩

⟨B⟩ :: = 0/1

Status ⟨ST⟩ :: = ⟨MR⟩ ⟨OVA⟩ ⟨OVB⟩

| | | |
|---|---|---|
| Match Reply | ⟨MR⟩ | :: = 0/1 |
| Overflow at A | ⟨OVA⟩ | :: = 0/1 |
| Overflow at B | ⟨OVB⟩ | :: = 0/1 |

## B.2. API Execution

⟨API⟩ :: = ⟨FN⟩ ⟨DA⟩ ⟨DM⟩

The API is executed in an automatic sequence of three beats.

### B.2.1. Beat 1. Domain Address

```
Reset TR1 and TR2
SEARCH <Ch.spec.1> <CB.spec.1>
For all matching word-rows set tags in TR1 and TR2
Set MR in the DOR
```

### B.2.2. Beat 2. Domain Modification

```
If Group Run is specified
  *Complement TR1 tags
  For all TR1 tags propagate Up and/or Down
  Start Group Runs from TR1 tags
  Reset TR2
  SEARCH <Ch.spec.2> <CB.Spec.2>
  For all matching word-rows set tags in TR2
  Stop Group Runs at TR2 tags
  Activate selected word-row groups
If Group Run is not specified
  WRITE 0 to perform the specified clear option
  *Complement TR1 tags
  *For all TR1 tags propagate Up and/or Down
  *Start Top or Bottom Run
  Activate selected word-row(s) or word-row group
  Set OVA and OVB in the DOR
```

## B.2.3. Beat 3. Function Execution

If WRITE is specified

Update all activated word-rows with ⟨Ch.spec.2⟩ ⟨CB.spec.2⟩ (Or ⟨xxxxxxxx⟩ ⟨xxx1⟩ if Group Run is specified.

If READ is specified

Update the activated word-row(s) with ⟨xxxxxxxx⟩ ⟨CB.spec.2⟩ Transfer the contents of the activated word row(s) to the DOR and then to the store/buffer address specified by ⟨Ch.spec.2⟩

* These operations are performed only if they are specified by the Domain Modifier.

## B.3. API Description

⟨API⟩ :: = ⟨FN⟩ ⟨DA⟩ ⟨ DM⟩

The constituent parts of the API are described in detail in the following sub-sections.

### B.3.1. Function

|  |  |  |  |
|---|---|---|---|
| Function | ⟨FN⟩ | :: = | ⟨RW⟩ ⟨Ch.spec.2⟩ ⟨CB.spec.2⟩ |
| Read/Write | ⟨RW⟩ | :: = | 0/1 |

B.3.1.1. WRITE: RW = 0 causes all activated word-rows to be updated by ⟨Ch.spec.2⟩ ⟨CB.spec.2⟩

NB. (1) where T = x the corresponding bit-column is masked.

(2) if Group Run is specified all activated word-rows are updated by ⟨xxxxxxxx⟩ ⟨xxx1⟩

B.3.1.2. READ: RW = 1 causes

(1) the ⟨CB.Field⟩ of the activated word-row(s) to be updated by ⟨CB.spec.2⟩

NB. Where T = x the corresponding bit-column is masked.

followed by

(2) the ⟨Ch.Field⟩ and ⟨CB.Field⟩ of the activated word-row(s) are transferred to the DOR and then to the host store location specified by the 16-bit ⟨Ch.spec.2.⟩

NB. The READ function is not permitted when a Group Run is specified.

## B.3.2. Domain Address

Domain Address ⟨DA⟩ :: = ⟨Ch.spec.1⟩ ⟨CB.spec.1⟩

During Beat 1 the AMA is searched for ⟨Ch.spec.1⟩ ⟨CB.spec.1⟩ and, for all matching word-rows, tags are set in TR1 and TR2.

NB. (1) Where T = x the corresponding bit-columns are masked.

(2) TR1 and TR2 are reset before the search operation is performed.

(3) MR is set if at least one tag is set in TR1 after the search operation.

## B.3.3. Domain Modifier

Domain Modifier ⟨DM⟩ :: = ⟨CO⟩ ⟨C⟩ ⟨PR⟩ ⟨RN⟩

The Domain Modifier provides programmer-control over the mapping between the content of TR1 and TR2 and the word-rows which are activated for function execution.

Four modification options are provided:

| | | |
|---|---|---|
| (1) | ⟨CO⟩ | Clear Options |
| (2) | ⟨C⟩ | Complement tags |
| (3) | ⟨PR⟩ | Propagate tags |
| (4) | ⟨RN⟩ | Run Tags |

### B.3.3.1. Clear Options

Clear Options ⟨CO⟩ :: = ⟨MW⟩ ⟨CC⟩ ⟨CB⟩ ⟨MM⟩

Multi-write ⟨MW⟩ :: = 0/1 Clear Control-Bits ⟨CB⟩ :: = 0/1

Clear-character ⟨CC⟩ :: = 0/1 Match/Mismatch ⟨MM⟩ :: = 0/1

The clear options operate during Beat 2 (unless a Group Run is specified) to reset selected bits in ⟨Ch.Field⟩ and ⟨CB.Field⟩ of activated word-rows.

Bit-column selection:

The bit-column selected for the clear operation are those specified by T = 1 in ⟨Ch.spec.1⟩ and/or ⟨CB.spec.1⟩ as indicated in the table below.

| | | Fields enabled for the clear operation | |
|---|---|---|---|
| ⟨CC⟩ | ⟨CB⟩ | ⟨Ch.Field⟩ | ⟨CB.Field⟩ |
| 0 | 0 | - | - |
| 0 | 1 | - | E |
| 1 | 0 | E | - |
| 1 | 1 | E | E |

E = Enabled

Word-row activation:

The word-rows activated for the clear operation are those specified by the logical content of TR1 as indicated in the table below.

| | | Logical content of TR1 causing word-row activation | |
|---|---|---|---|
| ⟨MW⟩ | ⟨MM⟩ | 0 | 1 |
| 0 | 0 | - | a |
| 0 | 1 | a | - |
| 1 | 0 | a | a |
| 1 | 1 | a | a |

a = activation

## B.3.3.2. Complement tags

Complement tags ⟨C⟩ :: = 0/1

⟨C⟩ selects the true (C=0) or the complement (C=1) outputs of TR1

## B.3.3.3. Propagate tags

Propagate tags ⟨PR⟩ :: = ⟨U⟩ ⟨S⟩ ⟨D⟩

⟨PR⟩ allows a single tag (or its complement if C=1) in TR1 to activate adjacent word-rows.

| | | |
|---|---|---|
| Propagate Up | ⟨U⟩ | :: = 0/1 |
| Propagate Straight-through | ⟨S⟩ | :: = 0/1 |
| Propagate Down | ⟨D⟩ | :: = 0/1 |

NB. All 8 propagate modes are allowable.

The following table indicates which word-rows will be activated for each propagation mode when a tag is set (in TR1) in word-row n and C = 0.

| Propagation Mode | | | Activated Word-row | | |
|---|---|---|---|---|---|
| ⟨U⟩ | ⟨S⟩ | ⟨D⟩ | n-1 | n | n+1 |
| 0 | 0 | 0 | - | - | - |
| 0 | 0 | 1 | - | - | a |
| 0 | 1 | 0 | - | a | - |
| 0 | 1 | 1 | - | a | a |
| 1 | 0 | 0 | a | - | - |
| 1 | 0 | 1 | a | - | a |
| 1 | 1 | 0 | a | a | - |
| 1 | 1 | 1 | a | a | a |

a = activation

N.B. (1) The selected propagation mode operates on all word-rows in parallel and applies to true or complemented tags according to the value of ⟨C⟩

(2) The overflow bits ⟨OVA⟩ and ⟨OVB⟩ are set in the DOR if the selected propagation mode causes propagation out of the A and B ends of the WCL unit.

## B.3.3.4. Run Tags

| | | | |
|---|---|---|---|
| Run Tags | ⟨RN⟩ | :: = | ⟨R1⟩ ⟨R2⟩ |
| Run bit R1 | ⟨R1⟩ | :: = 0/1 | |
| Run bit R2 | ⟨R2⟩ | :: = 0/1 | |

⟨RN⟩ allows a single tag in Tag Register 1 or 2 to activate an adjacent group of word-rows.

There are three different types of run, which are selected according to the values of ⟨R1⟩ and ⟨R2⟩

| Run Mode | | Run Type |
|---|---|---|
| ⟨R1⟩ | ⟨R2⟩ | |
| 0 | 0 | No run |
| 0 | 1 | Top run |
| 1 | 0 | Bottom run |
| 1 | 1 | Group run |

The direction of the run, and hence the location of the 'Top' and 'Bottom' is determined by the selected propagation mode, as indicated below.

| Run Direction | | word-row locations | |
|---|---|---|---|
| | | A------n------- | B |
| Up | U = 1 | Bottom | Top |
| Down | D = 1 | Top | Bottom |

The run logic for Up and DOWN is implemented separately such that a Up-run and a DOWN-run may proceed in parallel.

The overflow bits ⟨OVA⟩ and ⟨OVB⟩ are set in the DOR if the selected run type would cause word-rows to be activated 'beyond' the A and B ends of the WCL unit.

(a) <u>Top Run</u>

The Top Run activated all word-rows from (and including) the top word-row to (and including) the first word-row which has been tagged in TR2, as indicated below.

| Propagation Mode | | | Activated Word Rows |
|---|---|---|---|
| | | | Contents of Tag Register TR2 |
| ⟨U⟩ | ⟨S⟩ | ⟨D⟩ | A 00000001000000010000000100000 B |
| 0 | 0 | 0 | |
| 0 | 0 | 1 | aaaaaaa |
| 0 | 1 | 0 | a a a |
| 0 | 1 | 1 | aaaaaaa a a |
| 1 | 0 | 0 | aaaaaa |
| 1 | 0 | 1 | aaaaaaa aaaaaa |
| 1 | 1 | 0 | a a aaaaaa |
| 1 | 1 | 1 | aaaaaaa a aaaaaa |

a = activation

NB. The complement option is inhibited for micro-instructions including a Top Run.

(b) Bottom Run

The Bottom Run activates all word-rows from (but not including) the first, word-row which has been tagged (or not tagged if C = 1) in TR1 to (and 'beyond') the bottom word-row. If S = 1 the first word row is also activated, as indicated below.

Assuming C = 0

| Propagation Mode | | | Activated Word-Rows |
|---|---|---|---|
| | | | Contents of Tag Register TR1 |
| ⟨U⟩ | ⟨S⟩ | ⟨D⟩ | A 000000100000001000000010000 0 B |
| 0 | 0 | 0 | |
| 0 | 0 | 1 | aaaaaaaaaaaaaaaaaaaaa |
| 0 | 1 | 0 | a a a |
| 0 | 1 | 1 | aaaaaaaaaaaaaaaaaaaaaa |
| 1 | 0 | 0 | aaaaaaaaaaaaaaaaaaaaa |
| 1 | 0 | 1 | aaaaaaaaaaaaaaaaaaaaaaaaaaa |
| 1 | 1 | 0 | aaaaaaaaaaaaaaaaaaaaaa |
| 1 | 1 | 1 | aaaaaaaaaaaaaaaaaaaaaaaaaaaa |

a = activation

NB. (1) For C = 1, the above table remains valid if the contents of TR1 are inverted.

(2) The overflow bit ⟨OVA⟩ will be set, in the above example, for propagation modes ⟨100⟩ ⟨101⟩ ⟨110⟩ and⟨111⟩

(3) The overflow bit ⟨OVB⟩ will be set, in the above example, for propagation modes, ⟨001⟩ ⟨011⟩ ⟨101⟩ and⟨111⟩

(c) Group Run

When a Group Run is specified, Beat 2 is modified such that

(1) Tag Register TR2 is reset

(2) Clear options are inhibited

(3) A second search operation is initiated in which the AMA is searched for ⟨Ch.spec.2⟩ ⟨CB.spec.2 ⟩ and, for all matching word-rows, tags are set in Tag Register TR2.

The group Run activates all word-rows from those word-rows having a tag set (or reset if C = 1) in Tag Register TR1 to (and including) the first occurrences of word-rows which have been tagged in Tag Register TR2. The following table indicates which word-rows will be activated when a Group Run is specified

Assuming C = 0

| Propagation Mode |  |  | Activated Word-Rows |
|---|---|---|---|
| 〈U〉 | 〈S〉 | 〈D〉 | Contents of Tag Registers TR1 and TR2<br>A B<br>TR1 00000010000000100000000100000000<br>TR2 0000000000100000000010000001000 |
| 0 | 0 | 0 |  |
| 0 | 0 | 1 | aaaa aaaaa aaaaa |
| 0 | 1 | 0 | a a a |
| 0 | 1 | 1 | aaaaa aaaaaa aaaaaa |
| 1 | 0 | 0 | aaaaaa aaa aa |
| 1 | 0 | 1 | aaaaaa aaaaaa aaaaaaa aaaaa |
| 1 | 1 | 0 | aaaaaaa aaaa aaa |
| 1 | 1 | 1 | aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa |

a = activation

NB. (1) For C = 1, the above table remains valid if the contents of the Tag Register TR1 are inverted.

(2) The overflow bit 〈OVA〉 will be set, in the above example for propagation modes 〈100〉 〈101〉 and 〈111〉

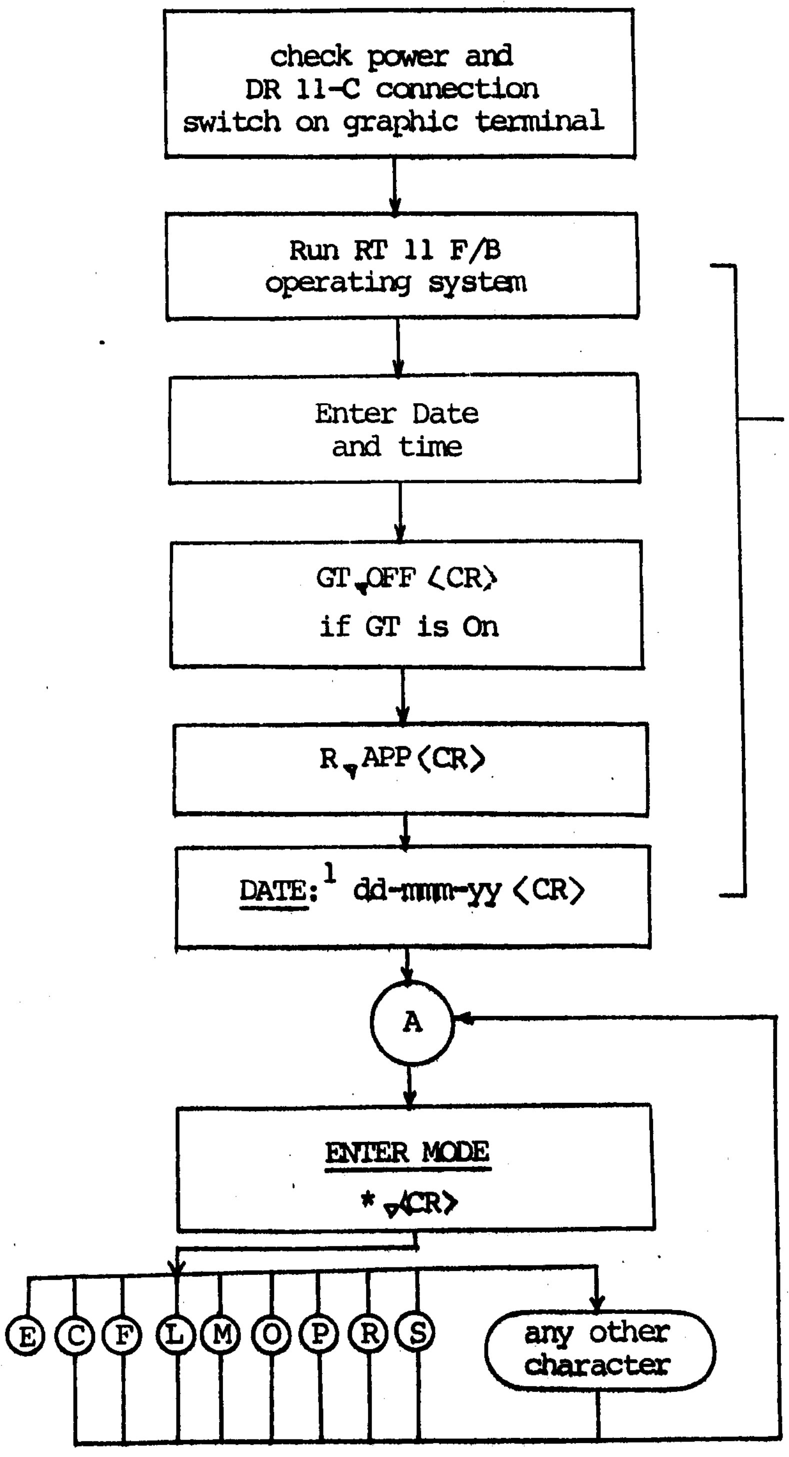
When a Group Run is specified Beat 3 is modified such that the function execution is restricted to a WRITE operation involving only Control Bit 4. Thus the function is automatically executed as if it were expressed as follows.

〈FN〉 :: = 〈RW〉〈Ch.spec.2〉〈CB.spec.2〉

〈FN〉 :: = 〈0〉〈xxxxxxxx〉〈xxx1〉

## APPENDIX C.

Flow Chart



See RT 11 system reference manual

▿ = Space

<CR> = Carriage return

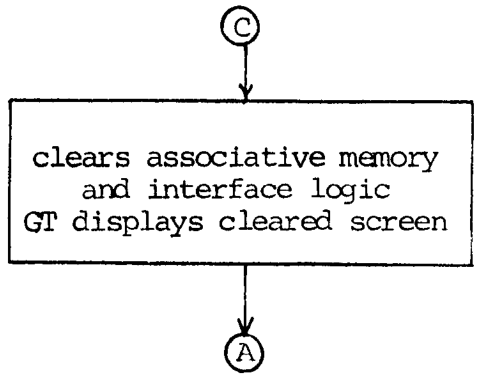
[1] the outputs of the machine are underlined.

*= C,E,F,L,M,O,P,R,S

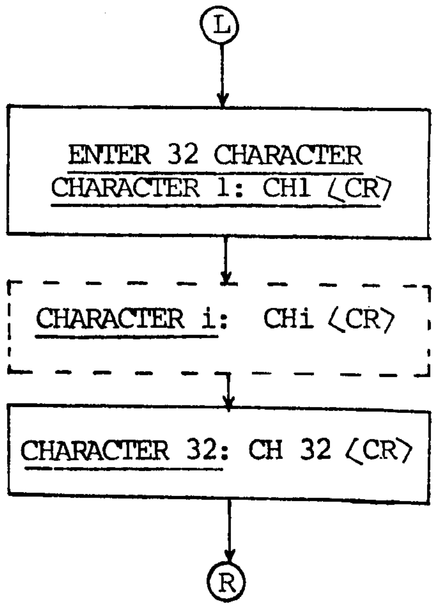
Any other characters including default will be ignored and a fresh request will be issued.

The program is transferred to the desired mode. At the completion of operation (except 'E') the control is transferred back to the node A.

## C.1. Clear



## C.2. Load

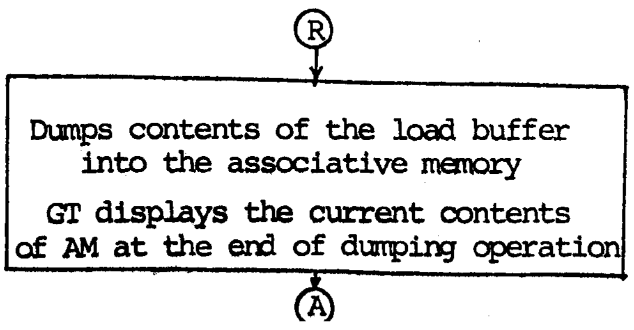


Stores currently entered character CHi in the ith location of the Load buffer.
Where $CH_i$ = Any ASCII character except @

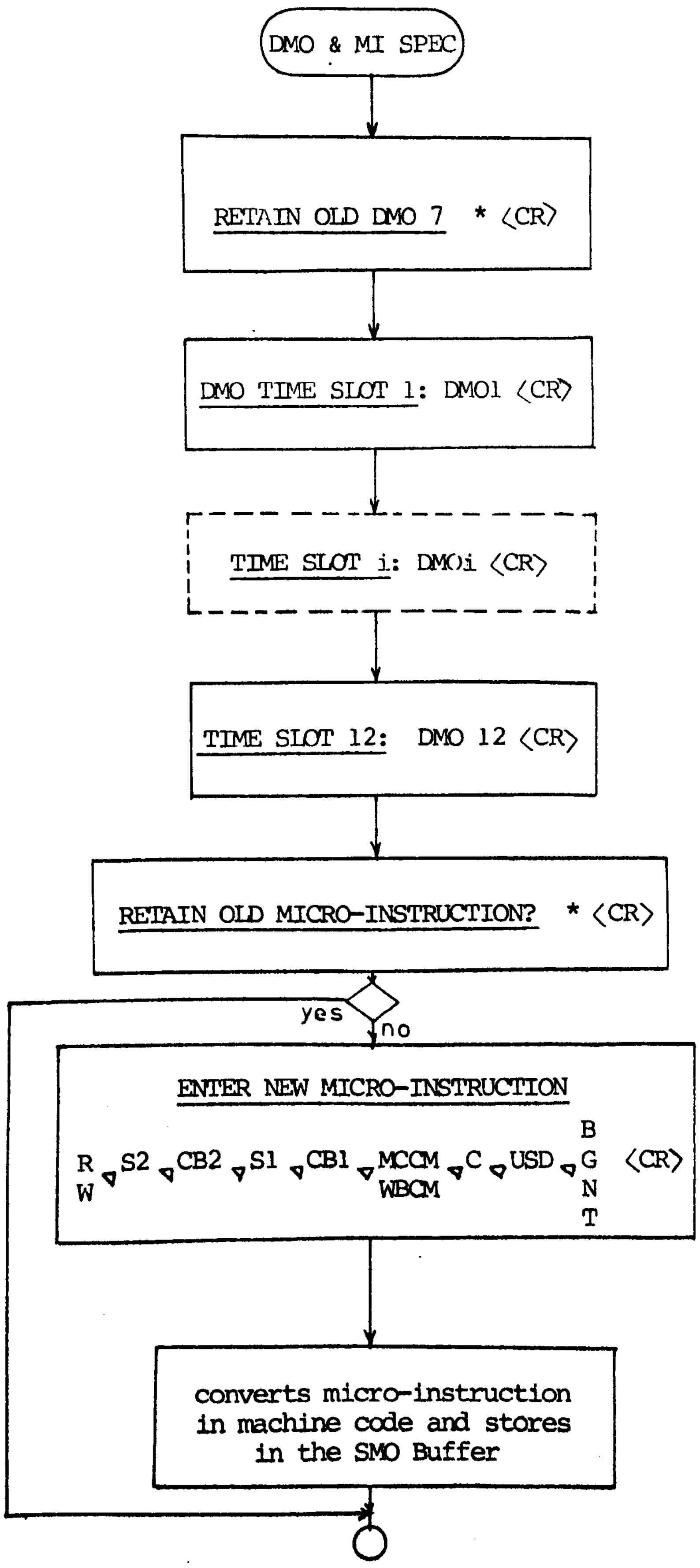
or = @ $b_8 b_7 \ldots b_2 b_1$
;bi = 1 or Ø

For a default value of Chi, old contents of ith location is unaltered.

## C.3 Reload


R
Dumps contents of the load buffer into the associative memory
GT displays the current contents of AM at the end of dumping operation
A

## C.4. Dynamic micro-order and Micro instruction specification


DMO & MI SPEC
RETAIN OLD DMO ? * ⟨CR⟩
DMO TIME SLOT 1: DMO1 ⟨CR⟩
TIME SLOT i: DMOi ⟨CR⟩
TIME SLOT 12: DMO 12 ⟨CR⟩
RETAIN OLD MICRO-INSTRUCTION? * ⟨CR⟩
yes
no
ENTER NEW MICRO-INSTRUCTION
R/W ▿ S2 ▿ CB2 ▿ S1 ▿ CB1 ▿ MCCM/WBCM ▿ C ▿ USD ▿ B/G/N/T ⟨CR⟩
converts micro-instruction in machine code and stores in the SMO Buffer

* = Y; yes
  = N or any other character including default; modification request.

DMOi = $N_{16}N_{15} \cdots N_2N_1$

where Ni = 1 or Ø
for default DMOi is maintained

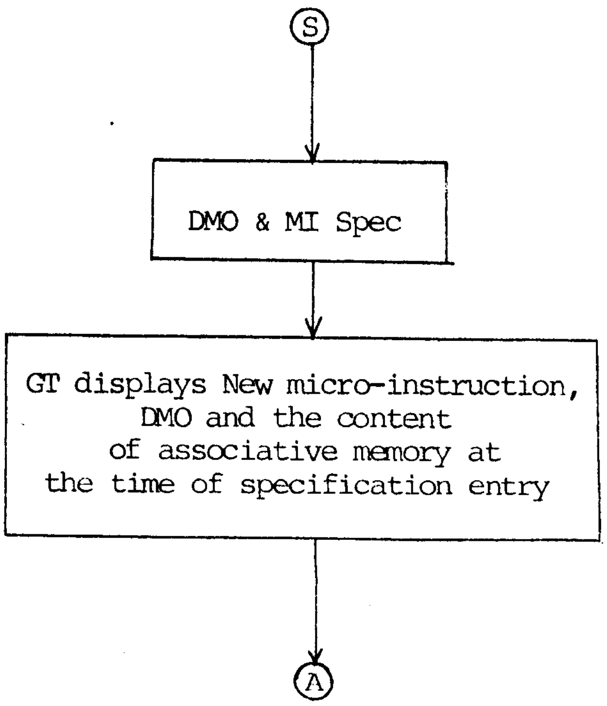
These DMO's are stored in DMO buffer. For further details of content of DMO refer to C.13

* = y; Yes

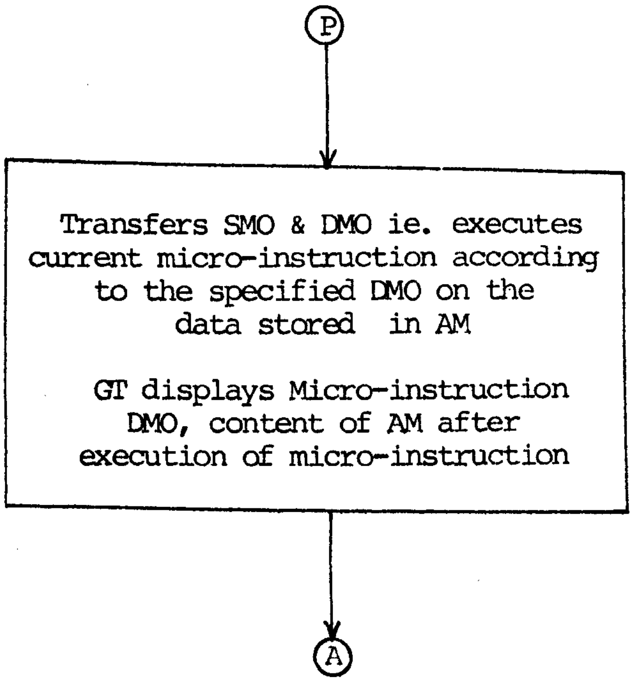
  = N or any other character including default; No

refer Appendix A for explanation

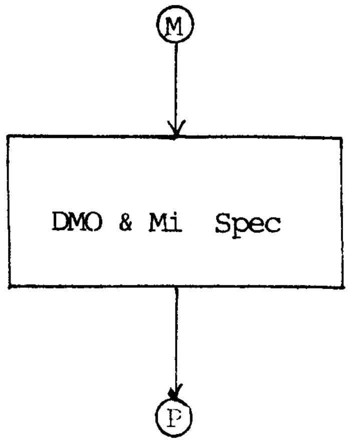
## C.5. Specify


S
DMO & MI Spec
GT displays New micro-instruction, DMO and the content of associative memory at the time of specification entry
A

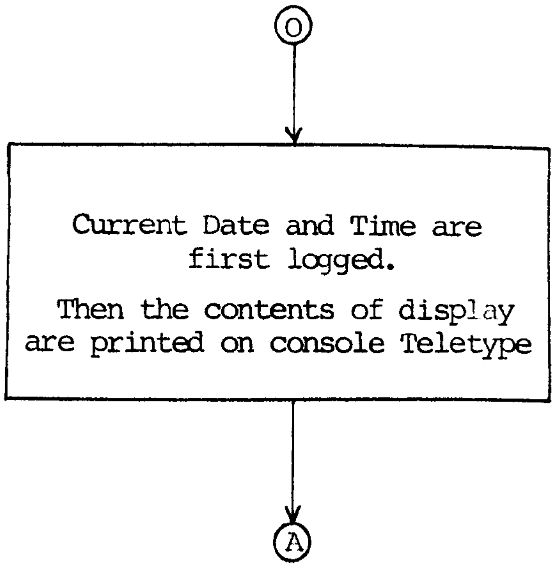
## C.6. Process


P
Transfers SMO & DMO ie. executes current micro-instruction according to the specified DMO on the data stored in AM
GT displays Micro-instruction DMO, content of AM after execution of micro-instruction
A

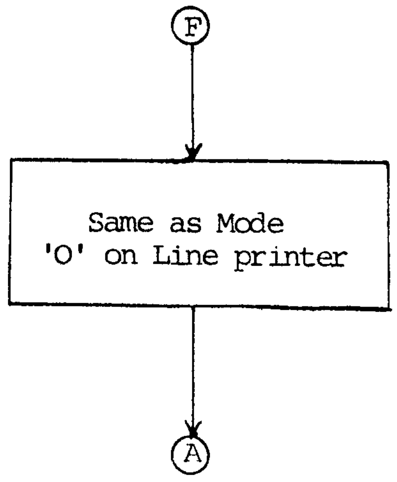
## C.7. Micro Instruction Specify & Execute



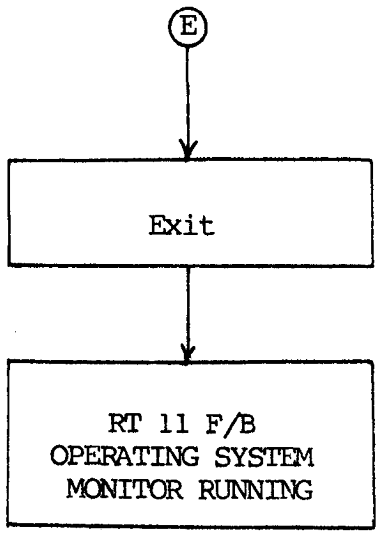
## C.8. Output (Hard copy on console TTY)


O
Current Date and Time are first logged.
Then the contents of display are printed on console Teletype
A

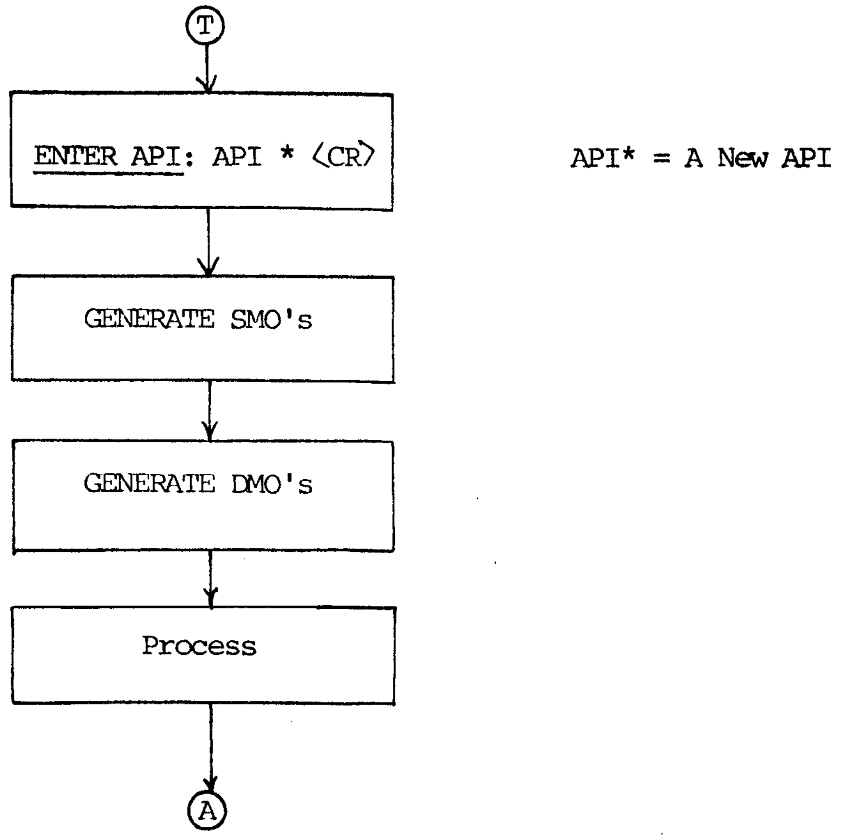
## C.9. Fast output (Hard copy on LP)



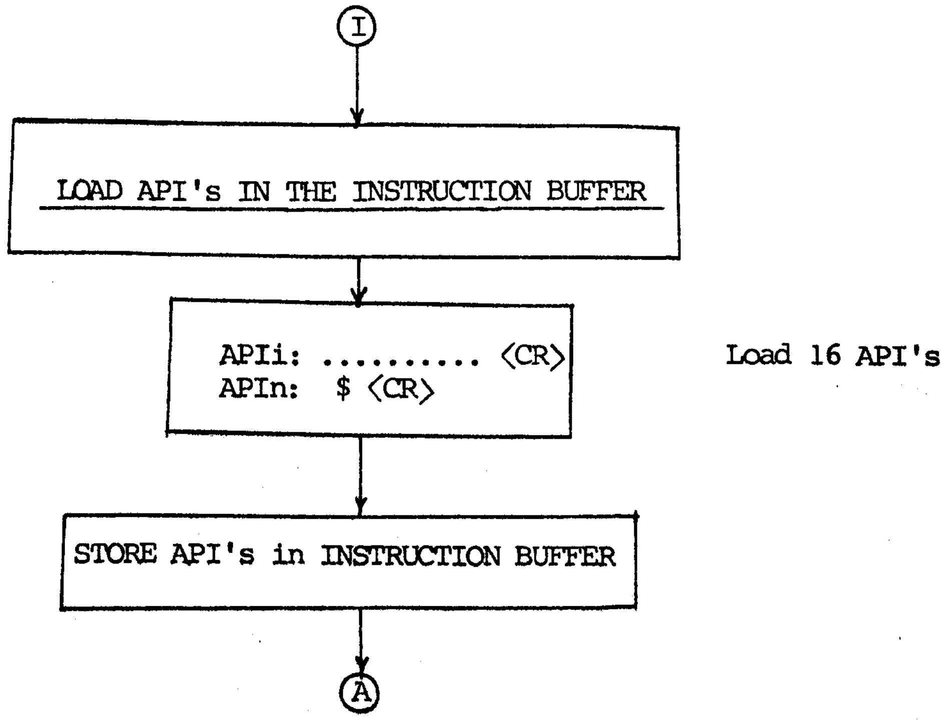
## C.10 Exit



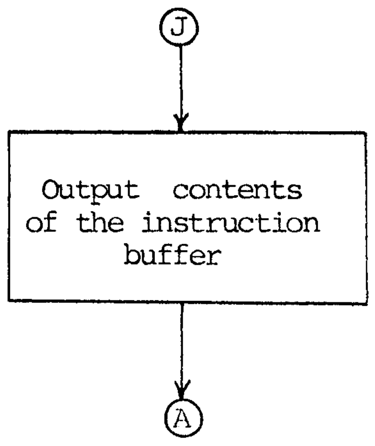
## C.11 Translate API and Execute



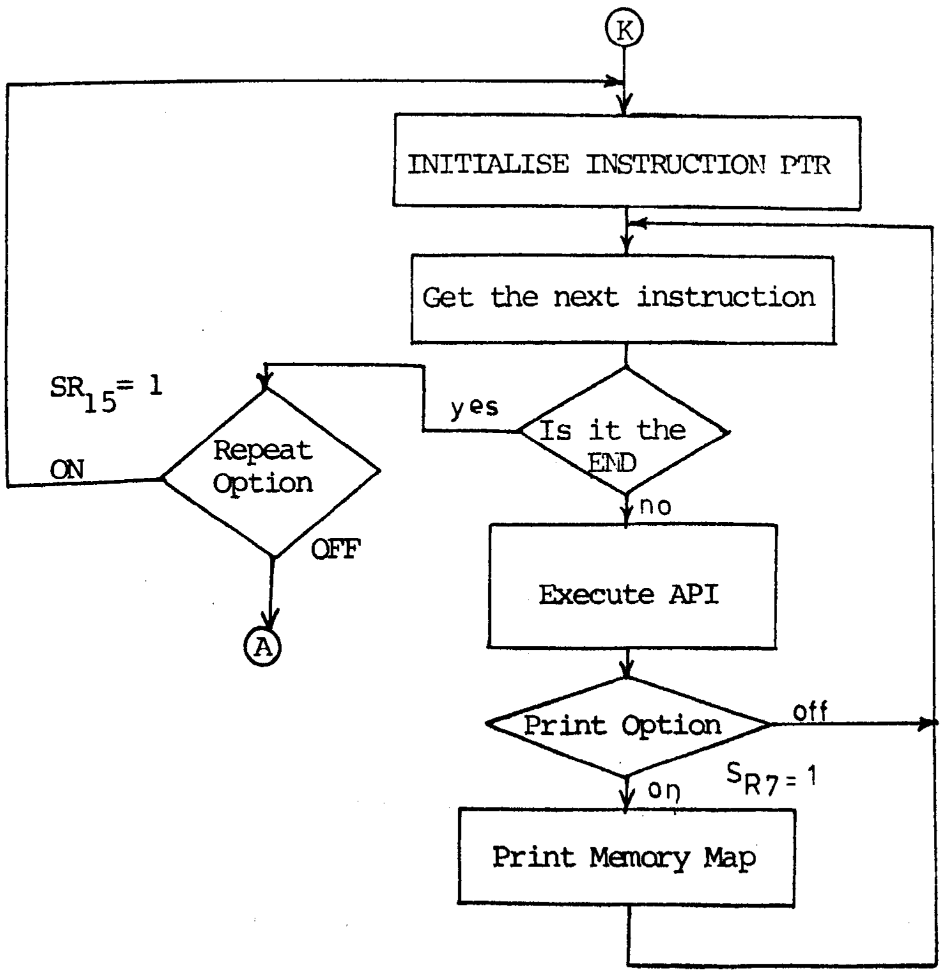
## C.12 Instruction buffer loading


I
LOAD API's IN THE INSTRUCTION BUFFER
APIi: .......... <CR>
APIn: $ <CR>
Load 16 API's
STORE API's in INSTRUCTION BUFFER
A

## C.13 J: Verify Contents of Instruction Buffer

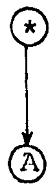


## C.14 Execute the instructions stored in BUFFER



$SR_i$ = ith bit of the switch reg.

## C.15 Any other character



## C.16 NOTE:-

(1) For explanation of RT-11 system command refer 'RT-11 system reference manual'.

(2) The following notes are applicable to all Key board operations.

(a) The monitor echos all character typed; lower case characters are converted to upper case.

(b) CTRLU ( U) and Rubout perform line deletion and character deletion respectively.

(c) A carriage return, line feed, CTRLZ or CTRLC must be struck before characters on the current line are to be made available to the program. The users are requested to use only carriage return CR as terminating character.

(d) ALTMODE (octal codes 175 & 176) are converted to escapes (octal 33)

## C.17 The contents of DMOi

| $N_{16}$ | $N_{15}$ | $N_{14}$ | $N_{13}$ | $N_{12}$ | $N_{11}$ | $N_{10}$ | $N_9$ | $N_8$ | $N_7$ | $N_6$ | $N_5$ | $N_4$ | $N_3$ | $N_2$ | $N_1$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $O_{2R}$ | $ST_c$ | $\emptyset_{xc}$ | $\emptyset_{yc}$ | $O_{2L}$ | $ST_s$ | $\emptyset_{xs}$ | $\emptyset_{ys}$ | TM | RW | MW | MM | GR | TG | $O_2$ | $O_1$ |

For explanation of signals see glossary of signals in Appendix A.2.2.

# APPENDIX D

## Field Acquisition from on Inspec data-base

### D.1. The Database Structure

The INSPEC data bases consist of abstracts of journal articles, technical reports, patents, conference proceedings, books and theses, classified and indexed, with bibliographic citations included.

The database used in the present investigation was INSPEC-1 on a 9-track, 800 b.p.i. magtape. The record format conforms broadly to the ISO standard 2709 for bibliographic information exchange on magtape and with ANSI-239.

Each record contains such data as the title, abstract, authors, full bibliographic references, indexing and cross-references with all items carrying hierarchial classification codes, subject headings and free-index terms. The tape is an 8-bit EBC DIC, IBM code using the reduced character set for computer line printer output, this character set allows only the upper-case alphabet. See section D.2.6.

### D.2.1. The File Layout

The file leader follows immediately on the beginning-of-tape marker without intervening tapemark. There is no beginning of file mark and the file is terminated by two immediately consecutive tape marks.

Records are unblocked and each logical record starts at a physical block boundary and may extend over more than one block. The maximum physical block size is $2000_{10}$ characters with continuation blocks of less than $20_{10}$ characters being filled out to this minimum ($20_{10}$) length. The fill characters are indeterminate, however, these characters will not be accessed by the acquisition program and therefore cause no error.

The maximum logical record size is $6,200_{10}$ characters. In each Inspec file the first record will be a leader record which describes the contents of the file. Each subsequent logical record holds information for a different bibliographic item. The items are sequenced in ascending accession number order but can be regarded as randomly ordered in any other respect.

## D.2.2. The Record Layout

The layout of the records is based on the USA and British standards for bibliographic communication which in turn are based on the Library of Congress MARC format.

Each record is divided into three parts:-

a) Fixed length leader

b) Variable length directory

c) Variable length data fields

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Leader | Directory | FT | Control No. | FT | Data Field 1 | FT | Data Field N | FT | RT |

D.2.2.1. The Leader

In accordance with bibliographic standards the leader contains 24 bytes of 6-bit characters (ASCII) or 8-bit (EBC.DIC) in the following format;-

| Record Length | Status | Type of Record | Not Used | Indicator Count | Delimiter Count | Base Address of Data | Not Used | Entry map |
|---|---|---|---|---|---|---|---|---|
| 0 - 4 | 5 | 6 | 7 - 9 | 10 | 11 | 12-16 | 17-19 | 20-23 |

D.2.2.2. The Directory

The directory is a variable length field consisting of a field terminator character and a variable number of fixed length entries. It contains one entry only for each data field present in the record, and these entries are recorded in ascending numeric sequence according to the tag field. If a record does not contain a specific data field, the entry is entirely omitted.

D.2.2.2.1. The layout of an entry.

| | | |
|---|---|---|
| Tag | 0<br>2 | The first three decimal characters in a directory entry uniquely define the type of field addressed by the entry. |
| Length of Field | 3<br>6 | The number of characters in the data field specified by this entry including the data field indicator and terminator characters. |
| Address of Field | 7<br>11 | The position of the first character of the data field relative to the base address of the data. See section (2.2.1.) (leader layout). |

The first three characters in a directory entry constitute a data field which uniquely defines the type of field addressed by character 7 to 11 of the directory entry. The data fields contained in a record are specified by broad category and sub-category. The format of the tag field in a directory is as follows:-

| | |
|---|---|
| Main Category | Sub-Category |
| ch.0 | ch.1-2 |

Character 0 indicates the broad category of the data field as follows:-

0 Control fields

1 'subject delineation' : title, abstract, classification, indexing.

2 Personal names

3 Identifying codes

4 Volume, issue, part

5 Locations

6 Number of pages etc.

7 Organisations

8 Dates

9 File description

Characters 1 and 2 identify the sub-category within main category.

All tags are numeric and are arranged, within the directory, in ascending numeric sequence. The list of tags are given in the following table D.1.

TABLE D.1

TAG LIST DESCRIPTIONS

| TAG List | | Data Field |
|---|---|---|
| Main Category | 00 | (Control fields) |
| 001 | | Control number |
| 010 | | Record type |
| Main Category | 1 | (Subject delineation) |
| 100 | | Title of record |
| 110 | | Text of abstract |
| 120* | | Sectional classification codes |
| 121* | | Unified classification codes |
| 130* | | Subject index headings |
| 131* | | Free-indexing terms |
| 132 | | Treatment codes |
| 150 | | Title of corresponding higher level publication (2) from which this item has been taken (if relevant) |
| 151 | | Title of cover-to-cover translation journal |
| 160 | | Language |
| 170 | | Title of conference |

| | | |
|---|---|---|
| <u>Main Category</u> | 2 | (Personal names) |
| 200* | | Author(s) |
| 210* | | Editor(s) |
| 220* | | Translator(s) |
| <u>Main Category</u> | 3 | (Identifying codes) |
| 300* | | Abstract number(s) (appearing in INSPEC abstracts journals) |
| 310 | | CODEN |
| 311 | | CODEN of cover-to-cover translation |
| 320 | | Standard Book Number |
| 330 | | Report number |
| 340 | | U.S. Government Clearing House number |
| 350 | | Contract number |
| 360 | | Patent number |
| 370 | | Original patent application number |
| <u>Main Category</u> | 4 | (Volume and issue) |
| 400 | | Volume and issue number |
| 401 | | Volume and issue number of cover-to-cover translation |
| 450 | | Part number |
| <u>Main Category</u> | 5 | (Locations) |
| 500 | | Location of conference |
| 510 | | Place of publication |
| 520 | | Country of patent |
| 530 | | Country of original patent application |

| | | |
|---|---|---|
| Main Category | 6 | (Number of pages etc.) |
| 600 | | Number of pages of level 1** record |
| 610 | | Number of pages of level 2** record |
| 620 | | Inclusive page numbers |
| 621 | | Inclusive page numbers of level 2** cover-to-cover translation |
| 630 | | Number of references |
| 640 | | Description of unconventional medium |

| | | |
|---|---|---|
| Main Category | 7 | (Organisations) |
| 700* | | Author affiliation |
| 710* | | Editor affiliation |
| 730* | | Assignees |
| 740 | | Publisher |
| 750 | | Organisation issuing report |
| 760 | | Sponsoring organisation |
| 770 | | Availability |

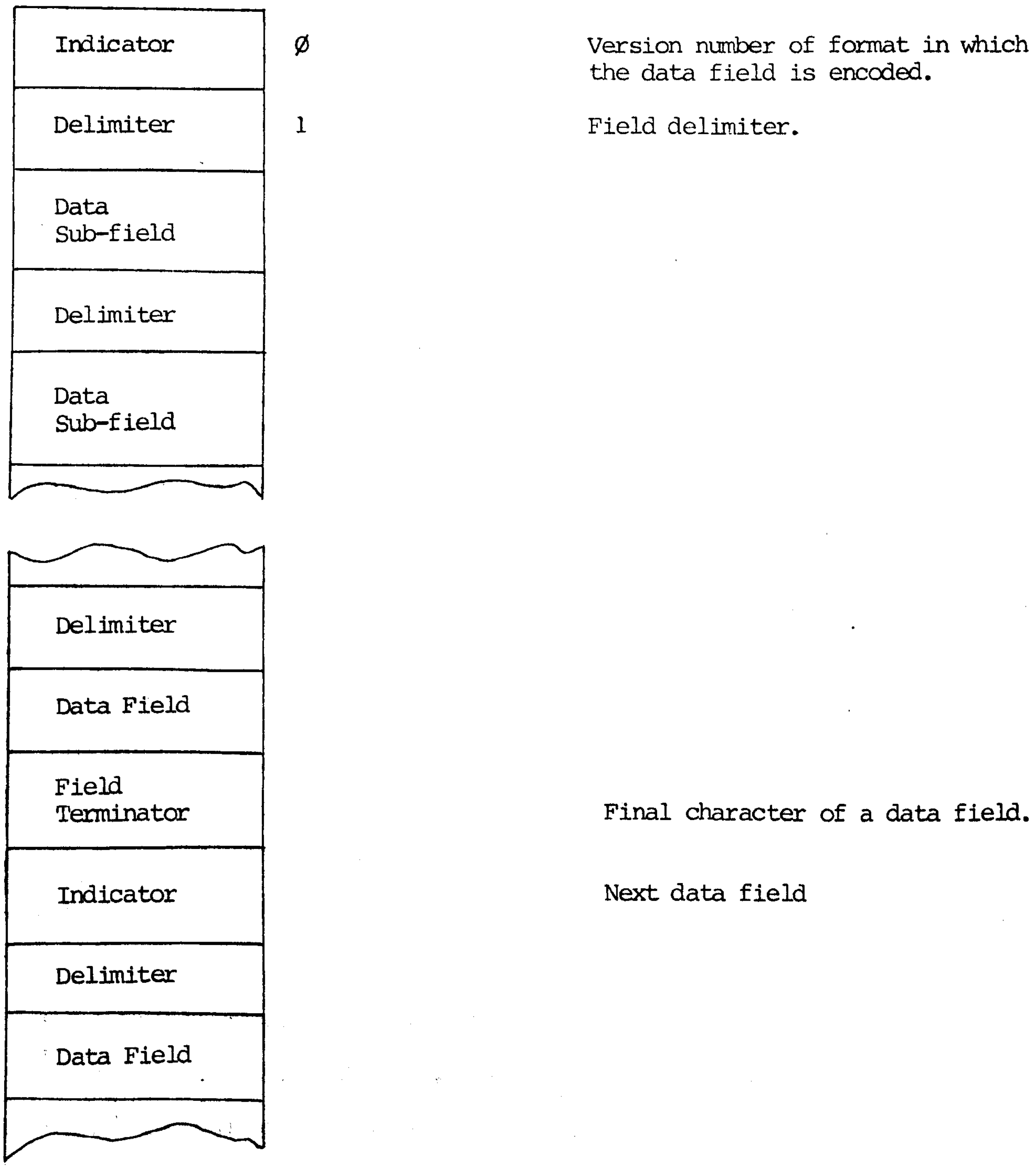
| | | |
|---|---|---|
| Main Category | 8 | (Dates) |
| 800 | | Inclusive dates of conference |
| 810 | | Date of publication |
| 811 | | Date of publication of cover-to-cover translation |
| 820 | | Date filed or submitted |
| 830 | | Priority date |

| | | | |
|---|---|---|---|
| Main Category | 9 | (File description) | |
| 900 | | Identification ) | These fields can only appear in the file header record |
| 910 | | Destination ) | |
| 920 | | Date written ) | |
| 930 | | Selection criteria ) | |

** See Ref.181

## D.2.2.3. The data fields

The data area of a record is made up of a variable number of variable length fields, each field has the following format:-



The possible data fields are detailed in Table D.1, with their corresponding directory entry tags.

## D.2.3. The Inspec 1 character set and coding

The database uses an abridged character-set for computer line-printer output. This character set allows only upper-case alphabetic characters and does not include shift codes in the data fields. Certain common characters are translated into other symbols and many rarely used mathematical symbols are replaced by a delete code to indicate their position.

Certain characters are modified in the translation program, to suit the output devices available and for program requirements, to various low frequency printing or non-printing characters unlikely to cause confusion by their position. These characters correspond to the Inspec function codes.

| INSPEC function code | | | Ascii code used. | |
|---|---|---|---|---|
| Record terminator | ≢ | ($117_8$EBC DIC) | $377_8$ (ASCII) | non-printing |
| Field terminator | ≠ | ($340_8$EBC DIC) | $176_8$ (ASCII) | ~ |
| Subfield delimiter | $ | ($133_8$EBC DIC) | $044_8$ (ASCII) | $ |
| Tapemark | ✓ | ($177_8$EBC DIC) | $100_8$ (ASCII) | @ |
| Deleted character | / | ($156_8$EBC DIC) | $136_8$ (ASCII) | ↑ |

The magtape character set is encoded in 8-bit IBM EBCDIC The abridged INSPEC character set is translated to 7-bit standard ASCII set. Translation is done after removal of the parity bit.

TABLE D.2

| | INSPEC Character | IBM code in Octal | ASCII Code in Octal |
|---|---|---|---|
| | . | 113 | 056 |
| | ) | 114 | 051 |
| | | 115 | 133 |
| | | 116 | 074 |
| Record | | 117 | 377 non-printing |
| Terminator | + | 120 | 053 |
| Subfield | $ | 133 | 044 $ |
| Delimiter | * | 134 | 052 |
| | | 135 | 135 |
| | ; | 136 | 073 |
| | - | 140 | 055 |
| | / | 141 | 057 |
| | , | 153 | 054 |
| | ( | 154 | 050 |
| Deleted | | 156 | 136 ↑ |
| Character | space | 172 | 040 |
| | = | 173 | 075 |
| | ' | 174 | 047 |
| | : | 175 | 072 |
| | | 176 | 076 |
| Typemark | | 177 | 100 @ |
| | ? | 300 | 077 |
| | A | 301 | 101 |
| | B | 302 | 102 |
| | C | 303 | 103 |
| | D | 304 | 104 |
| | E | 305 | 105 |
| | F | 306 | 106 |
| | G | 307 | 107 |
| | H | 310 | 110 |
| | I | 311 | 111 |
| | ! | 320 | 041 |
| | J | 321 | 112 |
| | K | 322 | 113 |
| | L | 223 | 114 |
| | M | 224 | 115 |
| | N | 325 | 116 |
| | O | 326 | 117 |
| | P | 327 | 120 |
| | Q | 330 | 121 |
| | R | 331 | 122 |
| Field | | 340 | 176 ~ |
| Terminator | S | 342 | 123 |
| | T | 343 | 124 |
| | U | 344 | 125 |
| | V | 345 | 126 |
| | W | 346 | 127 |
| | X | 347 | 130 |
| | Y | 350 | 131 |
| | Z | 351 | 132 |
| | Ø | 360 | 060 |
| | 1 | 361 | 061 |
| | 2 | 362 | 062 |
| | 3 | 363 | 063 |
| | 4 | 364 | 064 |
| | 5 | 365 | 065 |
| | 6 | 366 | 066 |
| | 7 | 367 | 067 |
| | 8 | 370 | 070 |
| | 9 | 371 | 071 |

In the field acquisition program there is no storage or output of the acquired fields in the original 8-bit coding. All input data is translated immediately on entry and subsequently handled, displayed and output in the ASCII equivalents shown in Table D.2.

## D.3. The Program Environment.

The field acquisition program is implemented in Macro-11 assembler under the RT11 F/B operating system on a DIGITAL PDP 11/40. It forms part of a suite of programs for the study of associative retrieval system using a simulated associative processor applied to an Inspec-1 data base.

## D.4. IO Concepts

Implemented under the RT11 operating system the database block structure is non-standard. As the block size is variable the input philosophy of the program is to attempt a read of the maximum block size 2,000 characters; return is then achieved on recognition of a physical block boundary.

All records start at a physical block boundary and overflow into continuation blocks as necessary; therefore, the input of a record entails reading blocks sequentially to a core buffer DS4BUF of maximum possible block length. The buffer is flushed to nulls previous to each input of a block from the data-set to overcome the problem of the buffer being only partially filled by a short block.

Each block is translated and transferred, character-serially, to the next free location in the core buffer BUFASC, the length of which is equal to the maximum possible record length ($6{,}200_{10}$ characters). The blocks are read, translated and loaded to BUFASC until a record terminator character is recognised. BUFASC is flushed to nulls before starting the load, to allow for the variable record size.

The minimum unit of data transfer under RT11 is the $256_{10}$ word block; therefore, the output datasets are buffered and characters loaded serially to the buffers until a complete block of data is available, when it is automatically output to the dataset. A partially filled buffer may be output by direct access to the output routines whenever necessary as the unfilled portion of the buffer will in all cases contain nulls.

## D.5. The Input and Translation of the Records

### D.5.1. Block Loading and record Translation

The flow-chart of the block loading and translation program is given in Fig.D.1

A block of record is read from the magtape data-base and temporarily stored in the input buffer area. The block of data (characters) available in the input buffer is accessed serially. The parity bit is stripped off from the characters and the character code is used to look up its 7 bit ASCII translated ASCII characters are loaded sequentially at the next free location of a buffer area (BUFASC).

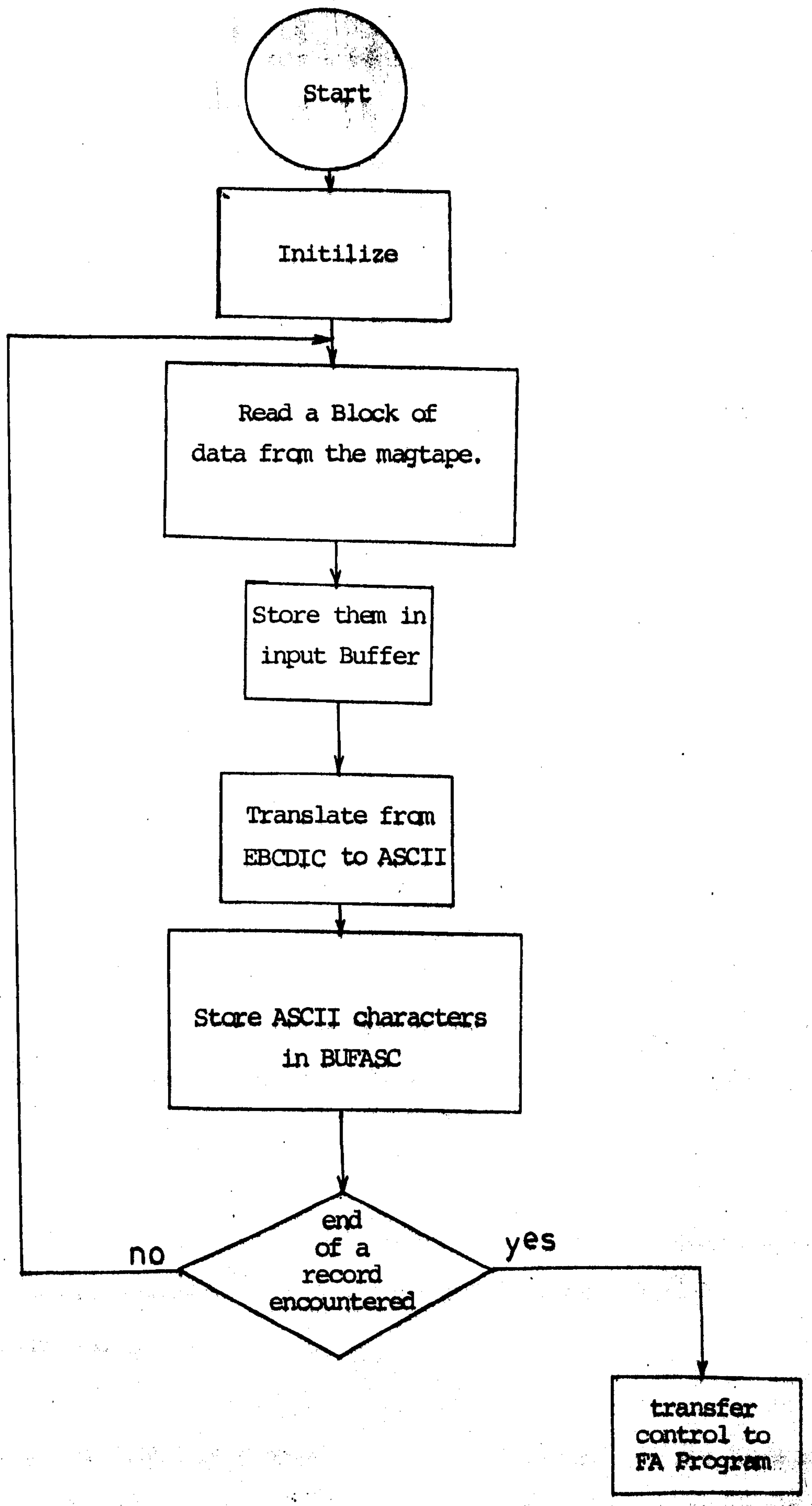

Start
Initilize
Read a Block of data from the magtape.
Store them in input Buffer
Translate from EBCDIC to ASCII
Store ASCII characters in BUFASC
end of a record encountered
no
yes
transfer control to FA Program

Fig.D.1. Flow-Chart for the Input and Translation program.

This procedure is terminated on the recognition of the 'record terminator' character. The complete translated record is then available in BUFASC, and the control is transferred to the field acquisition program.

## D.6. Data Field Acquisition and Output

### D.6.1. General record-data access

A translated record is available in the buffer BUFASC, the record length and the base address of the data fields are acquired from the record's leader and converted to binary values from 5-character ASCII strings using the utility subroutine BYTES. See Section (D.6.5.).

### D.6.2. Tag Matching

The start of the directory entries is found and the first tag key in the desired tag list LSTTG accessed. Both the tags in the directory and the list are 3 character ASCII decimal strings, these are converted to binary values before comparison.

All tags in the directory and in the desired tag list are unique and in ascending numeric sequence. The entry ØØØ ASCII in the desired tag list terminates the search list.

A desired tag from LSTTG is checked sequentially against all the directory tags in a record until matched or less than the directory tag compared in which case the desired field does not occur in the record. The next tag key in LSTTG is then checked against the remainder of the directory starting from the tag entry that numerically exceeded the preceding tag key. When the tag list is exhausted the next record is input from the file.

On a match the data field length, a 4 character ASCII decimal string and the offset to the start address of the field, a 5 character ASCII decimal string, are read from the directory entry, and converted to binary values and used to access the data field.

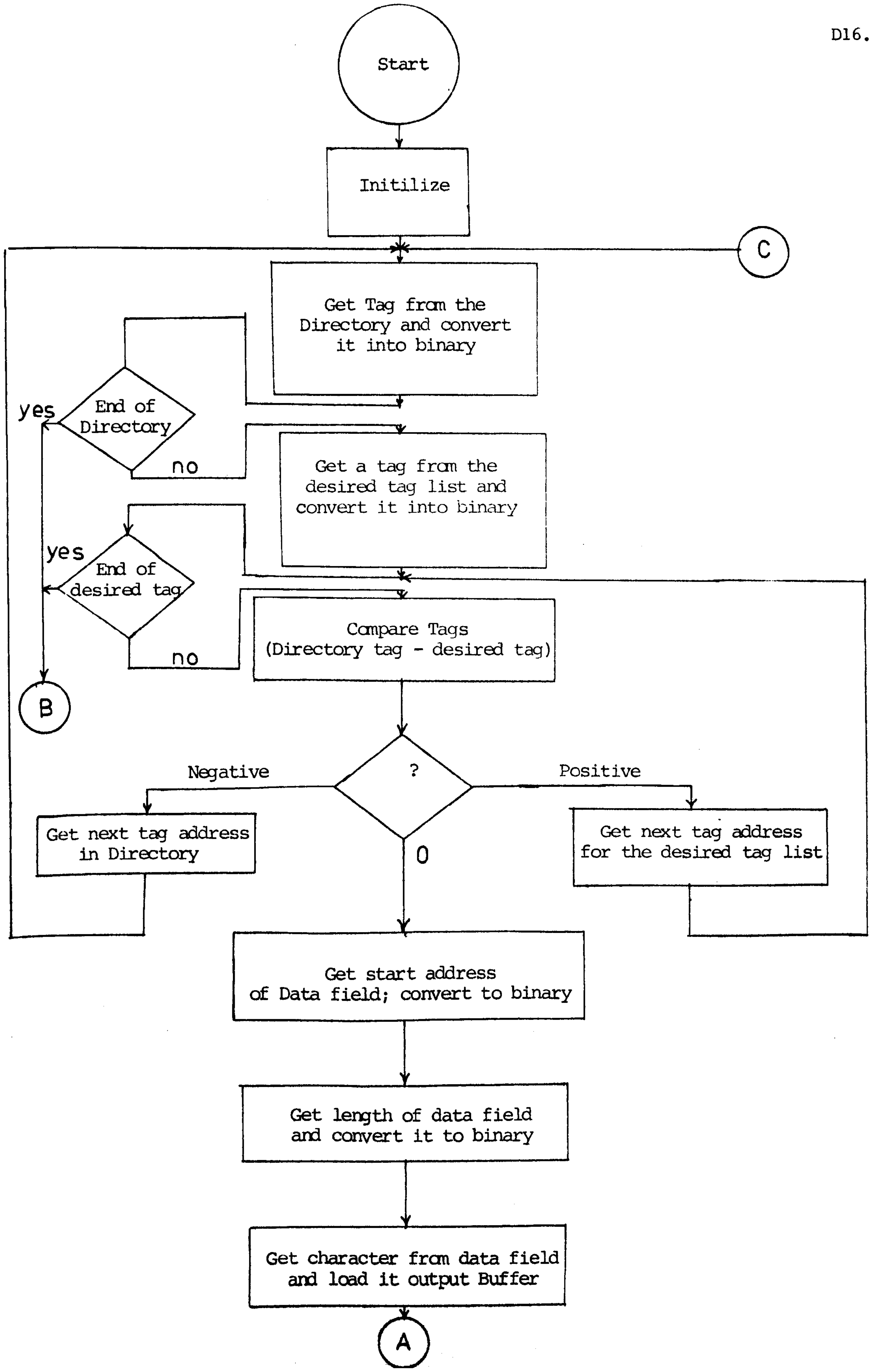
## D.6.3. Data field output

The data is loaded into the output buffer DS6BUF with a start-of-record mark at the beginning and into the hard copy buffer DS5BUF starting at the next available location in each.

When either output buffer is filled its contents are written to the disc Unit. The buffer is then flushed to nulls (Ø) to ensure correct output when a buffer is only partially filled,as may occur upon the input record being exhausted. The output buffers are $256_{10}$words in length, the standard RT11 block size; the output is done block serially.

## D.6.4. Input record termination.

When the input file is exhausted a flag is set and any remaining untransferred data in the output buffer is stored in the disc unit with an end-of-record mark.


Start
Initilize
C
Get Tag from the Directory and convert it into binary
End of Directory
yes
no
Get a tag from the desired tag list and convert it into binary
End of desired tag
yes
no
B
Compare Tags (Directory tag - desired tag)
?
Negative
Positive
0
Get next tag address in Directory
Get next tag address for the desired tag list
Get start address of Data field; convert to binary
Get length of data field and convert it to binary
Get character from data field and load it output Buffer
A

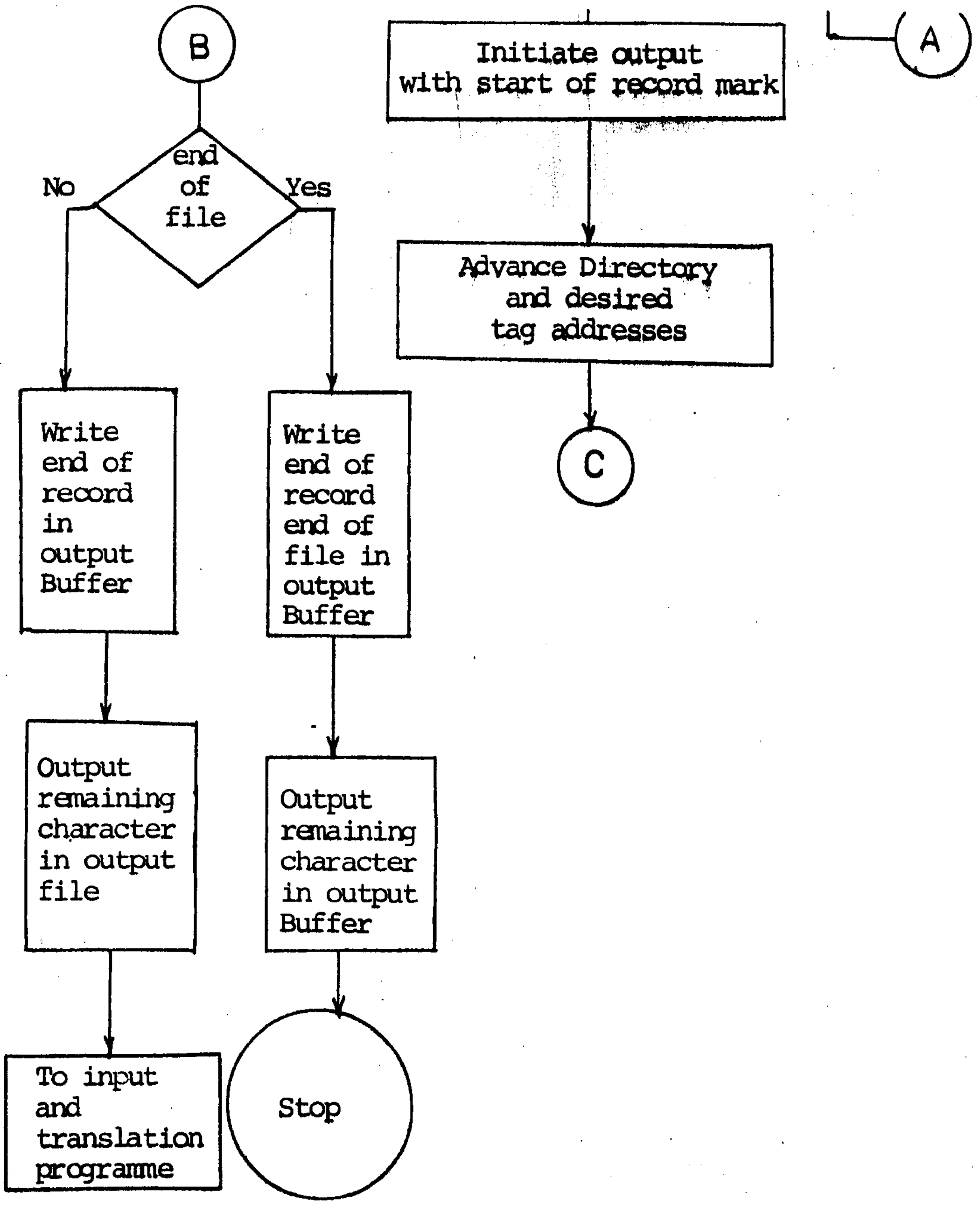

B
end of file
No
Yes
Write end of record in output Buffer
Write end of record end of file in output Buffer
Output remaining character in output file
Output remaining character in output Buffer
To input and translation programme
Stop
Initiate output with start of record mark
A
Advance Directory and desired tag addresses
C

Fig. D.2 Flow-Chart for the field acquisition and output program.

## D.6.5. The acquisitionProgram

The flow chart for the field acquisition program is given in Fig.D.2

On the entry a complete translated record is available in BUFASC.

NOTE:- 1) The subroutine BYTES advances the string pointer, past the last digit of the ASCII decimal string.

2) All tags are arranged in ascending numeric order.

## D.6.6. Block output

The desired data-fields are acquired by the field acquisition program. The acquired data-fields are loaded in the output data file on the disc unit.

This process is continued for the entire data-base. When a file terminator is encountered an end-of-file mark is stored in the output file.

## S U P P L E M E N T

**DYNAMIC-MICRO-ORDER SPECIFICATIONS FOR THE BYTE-ORIENTED VARIABLE RECORD LENGTH ASSOCIATIVE PARALLEL PROCESSOR.**

This supplement is to be read in conjunction with the thesis entitled "AN INVESTIGATION TO STUDY THE FEASIBILITY OF ON-LINE BIBLIOGRAPHIC INFORMATION RETRIEVAL SYSTEM USING AN APP". The supplement provides a complete specification of dynamic micro-order sequences for all valid Associative Processing Instructions ( API's -- refer APPENDIX B).

The sequences of dynamic micro-orders for all propagation options are identical; hence these are not repeated.

In WRITE instruction the dynamic micro-order sequences for all 'RUN' options excepting the 'GROUP RUN' are also identical. Hence two sets of dynamic micro-order sequences valid for

1) TOP, BOTTOM and NO RUN's
2) GROUP RUN

are given.

In APPENDIX B, it has been mentioned that all sixteen clear options is valid from the point of view of API definition. However, in some cases no meaningful operation takes place. These are marked by '*'.

**LEGEND.**

SMO = STATIC MICRO-ORDER
DMO = DYNAMIC MICRO -ORDER

For detail definitions of micro-orders APPENDIX A and B may be referred.

**NOTE :-**

$\emptyset_x$ and $\emptyset_y$ are listed as $0_x$ and $0_y$ .

```
WRITE INSTRUCTIONS ( R/W = 0 )
For TOP, BOTTOM and NO RUN options.
Sixteen CLEAR options for COMPLEMENT option C= 0



S        R MCCM C             R MCCM C             R MCCM C             R MCCM C *
M        W WCBM               W WCBM               W WCBM               W WCBM
O
         0 0000 0             0 0010 0             0 0011 0             0 0001 0

D     SOO SOOTRMMGTOO      SOO SOOTRMMGTOO      SOO SOOTRMMGTOO      SOO SOOTRMMGTOO
M     TXY TXYMWWMRG21      TXY TXYMWWMRG21      TXY TXYMWWMRG21      TXY TXYMWWMRG21
O     CCC SSS              CCC SSS              CCC SSS              CCC SSS

T1 0110011000000000   0110011000000000   0110011000000000   0110011000000000
T2 0110011000001100   0110011000001100   0110011000001100   0110011000001100
T3 0110011000000000   0110011000000000   0110011000000000   0110011000000000
T4 0000000000000000   0010000000000000   0010000000010000   0000000000010000
T5 0000000000000001   0010000000000001   0010000000010001   0000000000010001
T6 0000000000000000   0010000000000000   0010000000010000   0000000000010000
T7 0101010100000000   0101010100000000   0101010100000000   0101010100000000
T8 0101010100000010   0101010100000010   0101010100000010   0101010100000010
T9 0101010100000000   0101010100000000   0101010100000000   0101010100000000


S        R MCCM C *           R MCCM C             R MCCM C             R MCCM C
M        W WCBM               W WCBM               W WCBM               W WCBM
O
         0 1000 0             0 1010 0             0 1011 0             0 1001 0

D     SOO SOOTRMMGTOO      SOO SOOTRMMGTOO      SOO SOOTRMMGTOO      SOO SOOTRMMGTOO
M     TXY TXYMWWMRG21      TXY TXYMWWMRG21      TXY TXYMWWMRG21      TXY TXYMWWMRG21
O     CCC SSS              CCC SSS              CCC SSS              CCC SSS

T1 0110011000000000   0110011000000000   0110011000000000   0110011000000000
T2 0110011000001100   0110011000001100   0110011000001100   0110011000001100
T3 0110011000000000   0110011000000000   0110011000000000   0110011000000000
T4 0000000000000000   0010000000000000   0010000000010000   0000000000100000
T5 0000000000100001   0010000000100001   0010000000110001   0000000001100001
T6 0000000000000000   0010000000000000   0010000000010000   0000000000100000
T7 0101010100000000   0101010100000000   0101010100000000   0101010100000000
T8 0101010100000010   0101010100000010   0101010100000010   0101010100000010
T9 0101010100000000   0101010100000000   0101010100000000   0101010100000000
```

```
S        R MCCM C            R MCCM C            R MCCM C            R MCCM C
M        W WCBM              W WCBM              W WCBM              W WCBM
O
         0 0100 0            0 0110 0            0 0111 0            0 0101 0

D    SOO SOOTRMMGTOO     SOO SOOTRMMGTOO     SOO SOOTRMMGTOO     SOO SOOTRMMGTOO
M    TXY TXYMWWMRG21     TXY TXYMWWMRG21     TXY TXYMWWMRG21     TXY TXYMWWMRG21
O    CCC SSS             CCC SSS             CCC SSS             CCC SSS

T1 0110011000000000    0110011000000000    0110011000000000    0110011000000000
T2 0110011000001100    0110011000001100    0110011000001100    0110011000001100
T3 0110011000000000    0110011000000000    0110011000000000    0110011000000000
T4 0000001000000000    0010001000000000    0010001000010000    0000001000010000
T5 0000001000000001    0010001000000001    0010001000010001    0000001000010001
T6 0000001000000000    0010001000000000    0010001000010000    0000001000010000
T7 0101010100000000    0101010100000000    0101010100000000    0101010100000000
T8 0101010100000010    0101010100000010    0101010100000010    0101010100000010
T9 0101010100000000    0101010100000000    0101010100000000    0101010100000000


S        R MCCM C            R MCCM C            R MCCM C            R MCCM C
M        W WCBM              W WCBM              W WCBM              W WCBM
O
         0 1100 0            0 1110 0            0 1111 0            0 1101 0

D    SOO SOOTRMMGTOO     SOO SOOTRMMGTOO     SOO SOOTRMMGTOO     SOO SOOTRMMGTOO
M    TXY TXYMWWMRG21     TXY TXYMWWMRG21     TXY TXYMWWMRG21     TXY TXYMWWMRG21
O    CCC SSS             CCC SSS             CCC SSS             CCC SSS

T1 0110011000000000    0110011000000000    0110011000000000    0110011000000000
T2 0110011000001100    0110011000001100    0110011000001100    0110011000001100
T3 0110011000000000    0110011000000000    0110011000000000    0110011000000000
T4 0000001000000000    0010001000000000    0010001000010000    0000001000100000
T5 0000001000100001    0010001000100001    0010001000110001    0000001001100001
T6 0000001000000000    0010001000000000    0010001000010000    0000001000100000
T7 0101010100000000    0101010100000000    0101010100000000    0101010100000000
T8 0101010100000010    0101010100000010    0101010100000010    0101010100000010
T9 0101010100000000    0101010100000000    0101010100000000    0101010100000000
```

```
Sixteen CLEAR options for COMPLEMENET option C= 1.



S        R MCCM C            R MCCM C            R MCCM C            R MCCM C *
M        W WCBM              W WCBM              W WCBM              W WCBM
O
         0 0000 1            0 0010 1            0 0011 1            0 0001 1

D    SOO SOOTRMMGTOO     SOO SOOTRMMGTOO     SOO SOOTRMMGTOO     SOO SOOTRMMGTOO
M    TXY TXYMWWMRG21     TXY TXYMWWMRG21     TXY TXYMWWMRG21     TXY TXYMWWMRG21
O    CCC SSS             CCC SSS             CCC SSS             CCC SSS

T1 0110011000000000    0110011000000000    0110011000000000    0110011000000000
T2 0110011000001100    0110011000001100    0110011000001100    0110011000001100
T3 0110011000000000    0110011000000000    0110011000000000    0110011000000000
T4 0000000000000000    0010000000000000    0010000000010000    0000000000010000
T5 0000000000000001    0010000000000001    0010000000010001    0000000000010001
T6 0000000000000000    0010000000000000    0010000000010000    0000000000010000
T7 0101010100010000    0101010100010000    0101010100010000    0101010100010000
T8 0101010100010010    0101010100010010    0101010100010010    0101010100010010
T9 0101010100010000    0101010100010000    0101010100010000    0101010100010000


S        R MCCM C *          R MCCM C            R MCCM C            R MCCM C
M        W WCBM              W WCBM              W WCBM              W WCBM
O
         0 1000 1            0 1010 1            0 1011 1            0 1001 1

D    SOO SOOTRMMGTOO     SOO SOOTRMMGTOO     SOO SOOTRMMGTOO     SOO SOOTRMMGTOO
M    TXY TXYMWWMRG21     TXY TXYMWWMRG21     TXY TXYMWWMRG21     TXY TXYMWWMRG21
O    CCC SSS             CCC SSS             CCC SSS             CCC SSS

T1 0110011000000000    0110011000000000    0110011000000000    0110011000000000
T2 0110011000001100    0110011000001100    0110011000001100    0110011000001100
T3 0110011000000000    0110011000000000    0110011000000000    0110011000000000
T4 0000000000000000    0010000000000000    0010000000010000    0000000000100000
T5 0000000000100001    0010000000100001    0010000000110001    0000000001100001
T6 0000000000000000    0010000000000000    0010000000010000    0000000000100000
T7 0101010100010000    0101010100010000    0101010100010000    0101010100100000
T8 0101010100010010    0101010100010010    0101010100010010    0101010100100010
T9 0101010100010000    0101010100010000    0101010100010000    0101010100100000
```

```
S       R MCCM C          R MCCM C          R MCCM C          R MCCM C
M       W WCBM            W WCBM            W WCBM            W WCBM
O
        0 0100 1          0 0110 1          0 0111 1          0 0101 1

D   SOO SOOTRMMGTOO   SOO SOOTRMMGTOO   SOO SOOTRMMGTOO   SOO SOOTRMMGTOO
M   TXY TXYMWWMRG21   TXY TXYMWWMRG21   TXY TXYMWWMRG21   TXY TXYMWWMRG21
O   CCC SSS           CCC SSS           CCC SSS           CCC SSS

T1 0110011000000000  0110011000000000  0110011000000000  0110011000000000
T2 0110011000001100  0110011000001100  0110011000001100  0110011000001100
T3 0110011000000000  0110011000000000  0110011000000000  0110011000000000
T4 0000001000000000  0010001000000000  0010001000010000  0000001000010000
T5 0000001000000001  0010001000000001  0010001000010001  0000001000010001
T6 0000001000000000  0010001000000000  0010001000010000  0000001000010000
T7 0101010100010000  0101010100010000  0101010100010000  0101010100010000
T8 0101010100010010  0101010100010010  0101010100010010  0101010100010010
T9 0101010100010000  0101010100010000  0101010100010000  0101010100010000
```

```
S       R MCCM C          R MCCM C          R MCCM C          R MCCM C
M       W WCBM            W WCBM            W WCBM            W WCBM
O
        0 1100 1          0 1110 1          0 1111 1          0 1101 1

D   SOO SOOTRMMGTOO   SOO SOOTRMMGTOO   SOO SOOTRMMGTOO   SOO SOOTRMMGTOO
M   TXY TXYMWWMRG21   TXY TXYMWWMRG21   TXY TXYMWWMRG21   TXY TXYMWWMRG21
O   CCC SSS           CCC SSS           CCC SSS           CCC SSS

T1 0110011000000000  0110011000000000  0110011000000000  0110011000000000
T2 0110011000001100  0110011000001100  0110011000001100  0110011000001100
T3 0110011000000000  0110011000000000  0110011000000000  0110011000000000
T4 0000001000000000  0010001000000000  0010001000010000  0000001000100000
T5 0000001000100001  0010001000100001  0010001000110001  0000001001100001
T6 0000001000000000  0010001000000000  0010001000010000  0000001000100000
T7 0101010100010000  0101010100010000  0101010100010000  0101010100100000
T8 0101010100010010  0101010100010010  0101010100010010  0101010100100010
T9 0101010100010000  0101010100010000  0101010100010000  0101010100100000
```

```
GROUP RUN :---




S        R MCCM C          R MCCM C
M        W WCBM            W WCBM
O
         0 0000 0          0 0000 1


D    SOO SOOTRMMGTOO   SOO SOOTRMMGTOO
M    TXY TXYMWWMRG21   TXY TXYMWWMRG21
O    CCC SSS           CCC SSS

T1  0110011000000000  0110011000000000
T2  0110011000001100  0110011000001100
T3  0110011000000000  0110011000000000
T4  0101010100000000  0101010100000000
T5  0101010100001000  0101010100001001
T6  0101010100000000  0101010100000000
T7  0000000000000000  0000000000010000
T8  0000000000000010  0000000000010010
T9  0000000000000000  0000000000010000
```

```
READ INSTRUCTION ( R/W = 1 ).
Sixteen CLEAR options for COMPLEMENT option C = 0.




S       R MCCM C            R MCCM C            R MCCM C            R MCCM C *
M       W WCBM              W WCBM              W WCBM              W WCBM
O
        1 0000 0            1 0010 0            1 0011 0            1 0001 0

D    SOO SOOTRMMGTOO     SOO SOOTRMMGTOO     SOO SOOTRMMGTOO     SOO SOOTRMMGTOO
M    TXY TXYMWWMRG21     TXY TXYMWWMRG21     TXY TXYMWWMRG21     TXY TXYMWWMRG21
O    CCC SSS             CCC SSS             CCC SSS             CCC SSS

T1 0110011000000000   0110011000000000   0110011000000000   0110011000000000
T2 0110011000001100   0110011000001100   0110011000001100   0110011000001100
T3 0110011000000000   0110011000000000   0110011000000000   0110011000000000
T4 0000000000000000   0010000000000000   0010000000010000   0000000000010000
T5 0000000000000001   0010000000000001   0010000000010001   0000000000010001
T6 0000000000000000   0010000000000000   0010000000010000   0000000000010000
T7 0101000000000010   0101000000000010   0101000000000010   0101000000000010
T8 0101000001000010   0101000001000010   0101000001000010   0101000001000010
T9 0101000000000010   0101000000000010   0101000000000010   0101000000000010


S       R MCCM C *          R MCCM C            R MCCM C            R MCCM C
M       W WCBM              W WCBM              W WCBM              W WCBM
O
        1 1000 0            1 1010 0            1 1011 0            1 1001 0

D    SOO SOOTRMMGTOO     SOO SOOTRMMGTOO     SOO SOOTRMMGTOO     SOO SOOTRMMGTOO
M    TXY TXYMWWMRG21     TXY TXYMWWMRG21     TXY TXYMWWMRG21     TXY TXYMWWMRG21
O    CCC SSS             CCC SSS             CCC SSS             CCC SSS

T1 0110011000000000   0110011000000000   0110011000000000   0110011000000000
T2 0110011000001100   0110011000001100   0110011000001100   0110011000001100
T3 0110011000000000   0110011000000000   0110011000000000   0110011000000000
T4 0000000000000000   0010000000000000   0010000000010000   0000000000100000
T5 0000000000100001   0010000000100001   0010000000110001   0000000001100001
T6 0000000000000000   0010000000000000   0010000000010000   0000000000100000
T7 0101000000000010   0101000000000010   0101000000000010   0101000000000010
T8 0101000001000010   0101000001000010   0101000001000010   0101000001000010
T9 0101000000000010   0101000000000010   0101000000000010   0101000000000010
```

```
S         R MCCM C             R MCCM C             R MCCM C             R MCCM C
M         W WCBM               W WCBM               W WCBM               W WCBM
O
          1 0100 0             1 0110 0             1 0111 0             1 0101 0

D     SOO SOOTRMMGTOO      SOO SOOTRMMGTOO      SOO SOOTRMMGTOO      SOO SOOTRMMGTOO
M     TXY TXYMWWMRG21      TXY TXYMWWMRG21      TXY TXYMWWMRG21      TXY TXYMWWMRG21
O     CCC SSS              CCC SSS              CCC SSS              CCC SSS

T1 0110011000000000   0110011000000000   0110011000000000   0110011000000000
T2 0110011000001100   0110011000001100   0110011000001100   0110011000001100
T3 0110011000000000   0110011000000000   0110011000000000   0110011000000000
T4 0000001000000000   0010001000000000   0010001000010000   0000001000010000
T5 0000001000000001   0010001000000001   0010001000010001   0000001000010001
T6 0000001000000000   0010001000000000   0010001000010000   0000001000010000
T7 0101000000000010   0101000000000010   0101000000000010   0101000000000010
T8 0101000001000010   0101000001000010   0101000001000010   0101000001000010
T9 0101000000000010   0101000000000010   0101000000000010   0101000000000010
```

```
S         R MCCM C             R MCCM C             R MCCM C             R MCCM C
M         W WCBM               W WCBM               W WCBM               W WCBM
O
          1 1100 0             1 1110 0             1 1111 0             1 1101 0

D     SOO SOOTRMMGTOO      SOO SOOTRMMGTOO      SOO SOOTRMMGTOO      SOO SOOTRMMGTOO
M     TXY TXYMWWMRG21      TXY TXYMWWMRG21      TXY TXYMWWMRG21      TXY TXYMWWMRG21
O     CCC SSS              CCC SSS              CCC SSS              CCC SSS

T1 0110011000000000   0110011000000000   0110011000000000   0110011000000000
T2 0110011000001100   0110011000001100   0110011000001100   0110011000001100
T3 0110011000000000   0110011000000000   0110011000000000   0110011000000000
T4 0000001000000000   0010001000000000   0010001000010000   0000001000100000
T5 0000001000100001   0010001000100001   0010001000110001   0000001001100001
T6 0000001000000000   0010001000000000   0010001000010000   0000001000100000
T7 0101000000000010   0101000000000010   0101000000000010   0101000000000010
T8 0101000001000010   0101000001000010   0101000001000010   0101000001000010
T9 0101000000000010   0101000000000010   0101000000000010   0101000000000010
```

Sixteen CLEAR options for COMPLEMENT option C = 1.

```
S         R MCCM C            R MCCM C            R MCCM C            R MCCM C *
M         W WCBM              W WCBM              W WCBM              W WCBM
O
          1 0000 1            1 0010 1            1 0011 1            1 0001 1

D     SOO SOOTRMMGTOO     SOO SOOTRMMGTOO     SOO SOOTRMMGTOO     SOO SOOTRMMGTOO
M     TXY TXYMWWMRG21     TXY TXYMWWMRG21     TXY TXYMWWMRG21     TXY TXYMWWMRG21
O     CCC SSS             CCC SSS             CCC SSS             CCC SSS

T1   0110011000000000    0110011000000000    0110011000000000    0110011000000000
T2   0110011000001100    0110011000001100    0110011000001100    0110011000001100
T3   0110011000000000    0110011000000000    0110011000000000    0110011000000000
T4   0000000000000000    0010000000000000    0010000000010000    0000000000010000
T5   0000000000000001    0010000000000001    0010000000010001    0000000000010001
T6   0000000000000000    0010000000000000    0010000000010000    0000000000010000
T7   0101000000010010    0101000000010010    0101000000010010    0101000000010010
T8   0101000001010010    0101000001010010    0101000001010010    0101000001010010
T9   0101000000010010    0101000000010010    0101000000010010    0101000000010010


S         R MCCM C *          R MCCM C            R MCCM C            R MCCM C
M         W WCBM              W WCBM              W WCBM              W WCBM
O
          1 1000 1            1 1010 1            1 1011 1            1 1001 1

D     SOO SOOTRMMGTOO     SOO SOOTRMMGTOO     SOO SOOTRMMGTOO     SOO SOOTRMMGTOO
M     TXY TXYMWWMRG21     TXY TXYMWWMRG21     TXY TXYMWWMRG21     TXY TXYMWWMRG21
O     CCC SSS             CCC SSS             CCC SSS             CCC SSS

T1   0110011000000000    0110011000000000    0110011000000000    0110011000000000
T2   0110011000001100    0110011000001100    0110011000001100    0110011000001100
T3   0110011000000000    0110011000000000    0110011000000000    0110011000000000
T4   0000000000000000    0010000000000000    0010000000010000    0000000000100000
T5   0000000000100001    0010000000100001    0010000000110001    0000000001100001
T6   0000000000000000    0010000000000000    0010000000010000    0000000000100000
T7   0101000000010010    0101000000010010    0101000000010010    0101000000100010
T8   0101000001010010    0101000001010010    0101000001010010    0101000010100010
T9   0101000000010010    0101000000010010    0101000000010010    0101000000100010
```

```
S       R MCCM C            R MCCM C            R MCCM C            R MCCM C
M       W WCBM              W WCBM              W WCBM              W WCBM
O
        1 0100 1            1 0110 1            1 0111 1            1 0101 1

D    SOO SOOTRMMGTOO     SOO SOOTRMMGTOO     SOO SOOTRMMGTOO     SOO SOOTRMMGTOO
M    TXY TXYMWWMRG21     TXY TXYMWWMRG21     TXY TXYMWWMRG21     TXY TXYMWWMRG21
O    CCC SSS             CCC SSS             CCC SSS             CCC SSS

T1 0110011000000000  0110011000000000  0110011000000000  0110011000000000
T2 0110011000001100  0110011000001100  0110011000001100  0110011000001100
T3 0110011000000000  0110011000000000  0110011000000000  0110011000000000
T4 0000001000000000  0010001000000000  0010001000010000  0000001000010000
T5 0000001000000001  0010001000000001  0010001000010001  0000001000010001
T6 0000001000000000  0010001000000000  0010001000010000  0000001000010000
T7 0101000000010010  0101000000010010  0101000000010010  0101000000010010
T8 0101000001010010  0101000001010010  0101000001010010  0101000001010010
T9 0101000000010010  0101000000010010  0101000000010010  0101000000010010


S       R MCCM C            R MCCM C            R MCCM C            R MCCM C
M       W WCBM              W WCBM              W WCBM              W WCBM
O
        1 1100 1            1 1110 1            1 1111 1            1 1101 1

D    SOO SOOTRMMGTOO     SOO SOOTRMMGTOO     SOO SOOTRMMGTOO     SOO SOOTRMMGTOO
M    TXY TXYMWWMRG21     TXY TXYMWWMRG21     TXY TXYMWWMRG21     TXY TXYMWWMRG21
O    CCC SSS             CCC SSS             CCC SSS             CCC SSS

T1 0110011000000000  0110011000000000  0110011000000000  0110011000000000
T2 0110011000001100  0110011000001100  0110011000001100  0110011000001100
T3 0110011000000000  0110011000000000  0110011000000000  0110011000000000
T4 0000001000000000  0010001000000000  0010001000010000  0000001000100000
T5 0000001000100001  0010001000100001  0010001000110001  0000001001100001
T6 0000001000000000  0010001000000000  0010001000010000  0000001000100000
T7 0101000000010010  0101000000010010  0101000000010010  0101000000100010
T8 0101000001010010  0101000001010010  0101000001010010  0101000010100010
T9 0101000000010010  0101000000010010  0101000000010010  0101000000100010
```