RAPID SEARCH OF TEXTUAL DATA BASES BY INTELLIGENT PERIPHERALS

E. W. Grundke and H. S. Heaps
School of Computer Science
Technical University of Nova Scotia
Halifax, N. S. B3J 2X4

ABSTRACT

Although there are techniques to search textual data bases, there is still a need for rapid linear searches. Likewise, in searches that use inverted files there is a need for rapid processing of document lists. Computers designed for data base applications may pass such tasks to "intelligent" peripheral devices in order to reduce the burden on the central processor. An intelligent disc interface being designed at the Technical University of Nova Scotia will allow rapid searches to be made by use of a term comparator consisting of two, or more, pipelined stages. Each successive stage performs inexact comparisons at the effective data rate until a final microprocessor stage can perform the last comparison. High cost-effectiveness is achieved because of the low cost of each stage. The system is best suited for use in mini or micro computer environments.

LA RECHERCHE RAPIDE DANS DES BASES DE DONNEES TEXTUELLES AU MOYEN D'EQUIPEMENTS PERIPHERIQUES INTELLIGENTS

RESUME

Même s'il existe déjà des techniques pour interroger des bases de données textuelles, il existe toujours un besoin pour effectuer rapidement des recherches linéaires. Parallèlement, dans les recherches utilisant des fichiers inversés, il existe un besoin pour traiter rapidement les listes de documents. Les ordinateurs conçus pour la gestion de bases de données peuvent maintenant remettre de telles tâches à des équipements périphériques intelligents afin de réduire la charge du processeur central. Un interface intelligent sur disque est présentement en phase de conception à la Technical University de Nouvelle-Ecosse. Cet interface permettra d'effectuer rapidement des recherches en utilisant un comparateur de termes consistant en deux ou plusieurs passes canalisées. Chaque passe successive effectue des comparaisons inexactes au taux réel de transmission des données jusqu'à une dernière passe où un microprocesseur peut effectuer la dernière comparaison. On atteint aussi un haut niveau de coût-efficacité à cause des coûts réduits à chaque passe. Le système est spécialement adapté aux environnements de mini et de microordinateurs.

RAPID SEARCH OF TEXTUAL DATA BASES BY INTELLIGENT PERIPHERALS

E. W. Grundke and H. S. Heaps
School of Computer Science
Technical University of Nova Scotia
Halifax, N. S. B3J 2X4

1. INTRODUCTION

In order to allow rapid retrieval of records from textual data bases in response to queries that involve specified relations between terms it is common to use inverted files in which each searchable data base term is associated with a set of pointers to records that contain the term. Such a scheme eliminates the need to examine all records, but introduces considerable overhead with regard to file storage space. A further disadvantage is that all possible forms of query term must be anticipated at the time of creation of the inverted file, and this leads to impractically large storage requirements in the instance that queries may contain not only complete words but also fragments of words. A further disadvantage of an inverted file structure is the amount of processing required for its creation and for any subsequent updating. Also, there are many forms of query that cannot be processed by reference only to the information within an inverted file, and so there is a need to consider search procedures that may be used with linear files.

The traditional way to automate the task of information retrieval is to use a general purpose computer under control of software that constitutes a retrieval program. In the instance of a linear file the software is used mainly to direct extractions and comparisons of character strings. It is characteristic of software processes that not only must the data be processed but also the program instructions must be repeatedly read from memory. Thus the hardware must process two inputs, the data stream and the instruction stream. The time required to process the instruction stream may be regarded as an overhead that degrades the speed of processing the data stream. Since most of the data stream is irrelevant to any given query it is particularly desirable to process irrelevant portions as rapidly as possible.

A number of processor architectures have been investigated with a view to improving search efficiencies (see reviews by Hollaar (1979), Salton (1980), and Hsiao (1980)). Since the rate of flow of irrelevant data is nowhere greater than at the initial storage device, considerable attention has been given to the development of "intelligent" storage devices that are able to recognize data base items that may be relevant to a current query. Workers in industry (Goodyear, 1975, Meilander, 1980 and Bird, 1979), government agencies (Roberts, 1977), and numerous universities have considered the design of special "back-end" hardware in the form of subsystems that operate on the data stream close to the data base and hence furthest from the user.

The present paper is concerned with the design of an intelligent peripheral interface, or hardware processor acting as a filter as shown in Fig. 1, for use in an environment in which a typical search is for up to 20 specified character strings in records of a data base of approximately 100 million characters per disc drive may be scanned at a transfer rate of the order of 1 million characters per second. Thus the entire data base may be read in approximately 2 minutes. It is supposed that any use of a stored program, as shown by the software processor in Fig. 1, will be on a mini or micro computer. An important design goal is to avoid any undue increase in cost of the disc drive.

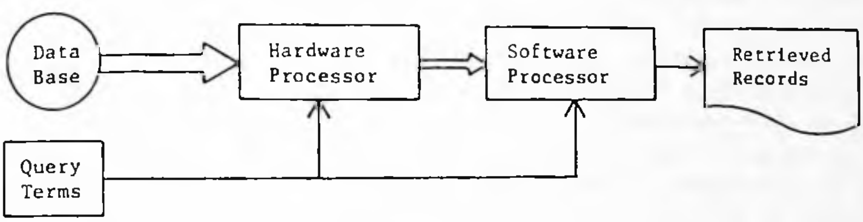

Fig. 1: Search Using Hardware Processor and Software Processor

2. THE RETRIEVAL PROCESS

Consider a sequential document data base of records of bibliographic material such as author names, titles, abstracts, and keywords. A simple form of query might consist of a set of query terms connected by logic relations such as AND, OR, NOT, ADJACENT, etc. Each record consists of a set of fields, each of which is associated with a particular attribute such as AUTHORS, TITLE, KEYWORDS, ABSTRACT, etc. Each query term is prescribed as pertaining to a particular attribute, and will be searched for only in fields of that attribute. The set of query terms may arise either from a single query or from a set of different queries that are batched in order to be processed by a single search.

Within each record of the sequential data base there must be some means of identifying the beginning and end of each variable-length field. One method is to include a directory at the beginning of each record in order to specify the beginning of each field relative to the beginning of the record. Another method is to place a tag of several characters at the beginning of each field in order to identify its attribute type.

Before the data base records are processed the query terms may be grouped according to attribute and stored in separate tables. This allows terms of each data base field to be compared only with query terms of the same attribute.

A query syntax table may be stored to indicate the logical structure of each question. During the subsequent search an entry is placed in the syntax table whenever a term in the data base is found to match one of the query terms. After all the characters of any record have been processed the query syntax table is examined to determine whether the logic of any query is satisfied. If it is satisfied then an appropriate action must be performed. For example, the entire record might be copied to an output file. Then, the query syntax table may be initialized for use in processing the next record.

3. THE SOFTWARE PROCESSOR

As shown in Fig. 1 the hardware processor receives a data stream from the sequential data base and outputs a reduced set of records at a sufficiently low rate that they may be searched by software running on a dedicated microprocessor. The required efficiency of the hardware processor, as measured by its reduction of the data rate presented to the software processor, is thus dependent on the speed of execution of the software. It is therefore relevant to discuss the structure, and execution speed, of sequential search software. Since the disc is used as a sequential source of data no consideration will be given to fast algorithms such as those of Boyer and Moore (1977) or Knuth, Morris and Pratt (1977).

In the following discussion it is supposed that each record is divided into fields, and that each field begins with a 2-character tag in which the first character is $ and the second character is an attribute descriptor such as T for title, K for keywords, etc. It is supposed that query terms are grouped by attribute. The j-th character of the i-th query term of attribute corresponding to descriptor d is denoted by term(d,i,j), the length of the query term is denoted by length(d,i), and the number of query terms of attribute d is denoted by n(d).

Each query term may be specified as having a mode(d,i) with regard to truncation. Thus a mode of "non-truncation" signifies that the term is to be searched for as a character string that is both preceded and followed by a delimiter such as a blank, a period, or the character $ that marks the beginning of a new field. A mode of "right truncation" signifies that the term is to be searched for as a character string preceded by a delimiter without regard to the presence or absence of a following blank. A mode of "left truncation" signifies that the term is to be searched for as a character string followed by a delimiter but preceded by any character. A mode of "double truncation" signifies that the term may be preceded and followed by any character. It may be noted that one of the advantages of a sequential search, in comparison to one that uses an inverted file, is its ability to deal with query terms specified in left and double truncation modes.

It is supposed that no query terms contain delimiting characters. Thus, if query terms may have the form of phrases that contain blanks then non-blank delimiters, such as commas or slashes, must be used.

The structure of programs for sequential search has been discussed in detail by Heaps (1978). A typical algorithm is shown in Fig. 2. Successive characters of the data stream are read and designated as char(m). The value of m is fixed at 1 until a $ character is found, in which instance the next character is read as an attribute descriptor. Further characters are read to form

char(1), char(2), ..., char(m)

until a delimiting character char(m+1) is encountered. Query terms that could be contained in the character string of length m are then compared with it, and whenever a match is found an appropriate entry is made in the query syntax table. After comparison with all possible query terms a further sequence of characters is read from the data stream and the process is continued, with resetting of d whenever an attribute descriptor is encountered, until an end of record character, EOR, is encountered.

```
READ char(1);
WHILE char(1)≠EOF DO
   BEGIN
   WHILE char(1)≠$ DO READ char(1);
      (*Next character is attribute descriptor*)
   REPEAT (*Process fields until EOR*)
      READ char(1);
      d:=char(1);
         (*Next characters form data base term*)
      REPEAT (*Process terms until $ or EOR*)
         m:=1;
         REPEAT READ char(m); m:=m+1
         UNTIL char(m)=(delimiter OR $ OR EOR*)
            (*Has read term plus subsequent character*)
         m:=m-1;
         FOR i:=1 TO n(d) DO
         CASE length(d,i) OF
         m:    IF char(1:length(d,i))=term(d,i,1:length(d,i))
               THEN place entry in syntax table;
         <m:   CASE mode(d,i) OF
               right truncation:
                  IF char(1:length(d,i))=term(d,i,1:length(d,i))
                  THEN place entry in syntax table;
               left truncation:
                  IF char(m-length(d,i)+1:m)
                     =term(d,i,1:length(d,i))
                  THEN place entry in syntax table;
               double truncation:
                  FOR j=1 TO m-length(d,i)+1 DO
                     IF char(j:j+length(d,i)-1)
                        =term(d,i,1:length(d,i))
                     THEN place entry in syntax table
               END (*of case mode*)
         END (*of case length*)
      UNTIL char(m+1)=($ OR EOR)
   UNTIL char(m+1)=EOR;
   Process, and initialize, syntax table;
   READ char(1)
   END (*while char(1)≠EOF*)
```

Fig. 2: Sequential Search Algorithm for Software Implementation

After any record has been read the syntax table is examined by software to determine whether the record satisfies any of the queries. If it does then appropriate action is taken. The syntax table is then initialized, the next record is processed, and the procedure is continued until an end-of-file character, EOF, is read.

To simplify the discussion it will be supposed that queries are restricted to have the form of "OR parameters" connected by AND or NOT (=AND NOT) logic. Two examples of such queries are as follows in which tn denotes a query term together with a truncation specification:

(t1 OR t2 OR t3) AND (t4) AND (t5 OR t6)
(t1 OR t2) NOT (t3 OR t4 OR t5) AND (t6) NOT (t7 OR t8)

Such a form of query syntax allows query terms and the syntax table to be stored in very simple tables. For each attribute there may be created a term table whose entries have the form

term, length, mode, question number, parameter position.

A term that appears in several questions, or several parameters in the same question, will occur in several different entries. The parameter position is a bit string chosen as 100 ... 0 or 0100 ... 0, etc. according as the term occurs in the first or second, etc. parameter. Such term tables may be processed very simply by the search algorithm of Fig. 2.

The syntax table may consist of entries that are indexed by question number. Each entry has the form of two bit strings:

parameter descriptor, parameter indicator.

Each question parameter is represented by a 1 in the parameter descriptor unless the parameter is empty or preceded by NOT. Thus the above two questions correspond to parameter descriptors 11100 ... 0 and 10100 ... 0 respectively. At the beginning of the search through any record all the parameter indicators are set to 0. During execution of the algorithm of Fig. 2 the operation "Place entry in syntax table" is implemented by using the question number to index the syntax table, and then performing the logic operation

parameter indicator:=(parameter indicator)OR(parameter position)

Similarly, the operation "Process syntax table" is implemented as follows:

```
FOR each entry in syntax table DO
    x:=(parameter descriptor)EXC.OR(parameter indicator);
    IF x=0 THEN save record with appropriate labels;
    parameter indicator:=0
```

The number of records that contain query terms is likely to be a very small percentage of the total number of records. Thus the time required to place entries in, and process, the syntax table is a very small proportion of the total execution time. Therefore, there is little advantage in using hardware for the steps that involve the syntax table. Most of the time required for execution of the algorithm of Fig. 2 is in execution of the comparisons within the CASE statements. For a given microprocessor the average time may be estimated in terms of the statistics of the lengths of the terms in the data base and the distribution of query terms with respect to length and truncation specifications.

4. CONVENTIONAL STRING COMPARATORS

Three main categories of string comparator designs (Hsiao, 1980) are summarized in Fig. 3. In the parallel comparator scheme (Stellhorn, 1974) of Fig. 3a the characters in the serial stream from the data base are clocked into a shift register SR that constitutes a "window" to a set of m adjacent characters. Its characters are fed in parallel to a set of s comparators, each of which contains a query term of up to m characters and which places an entry in the syntax table whenever a match is found. Thus ms comparisons of characters are performed simultaneously. An associative array (Bird, 1979) may also be used for parallel comparisons. The recent design by Burkowski (1982) may be regarded as a sophisticated comparator.

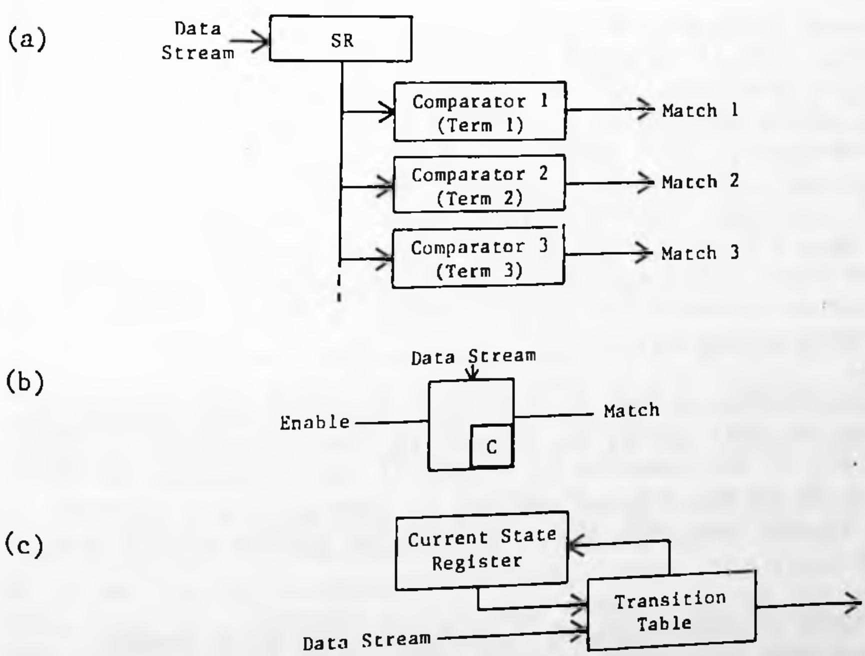

Fig. 3: Three Main Categories of String Comparator

In the cellular comparator scheme (Copeland, 1978) of Fig. 3b each comparator has the form of a cell that stores a single character. Each cell receives identical input and has an enable line. A cell outputs a "match" signal only if its enable line is ON and the input character is the same as its stored character. Cells are cascaded, with match lines of some connected to enable lines of others, so that the presence of a query term in the input stream causes a sequence of match/enable signals to propagate through a network of cells and place an entry in the query syntax table. A sequence of such cells could be used to implement each comparator of Fig. 3a. The cell concept provides a powerful tool provided the network of cells may be configured to reflect the structure of the query terms.

The third scheme (Aho and Corasick, 1975 and Roberts, 1977) shown in Fig. 3c uses a finite state machine that has a well-defined state transition for each input character. The transitions are determined by comparisons between incoming characters of the data stream and the various characters contained in the query terms. Because of the complexity of configuring a hardware realization this scheme is not used in the present approach.

5. STRING COMPARATOR HARDWARE FOR PARTIAL FILTERING

Cellular comparators have the disadvantage that query-dependent interconnections are needed between the cells. However, a generalization of the cellular method, as shown in Fig. 4, provides an excellent means of reduction of the data stream rate by partial filtering. It is supposed that there are m different characters in the data base. There are n memories M1 to Mn that each contain m addressable bits, the addresses being codes for the m different characters. Each memory has a set of address lines A, a single output data line D, and a single read enable line $\overline{RE}$. If $\overline{RE}$ is set to 0 the 1-bit content of the memory bit at address A is placed on the data line D. If $\overline{RE}$ is set to 1 the memory is not enabled and the data line D is set to 1. As described later, each memory functions as a comparator able to detect a match with any of several prescribed characters.

The input buffer of Fig. 4 is a shift register that receives characters from the data stream at successive time intervals Δ. The output data lines of the memories M1 to M(n-1) are connected to the enable lines of M2 to Mn, but are subject to time delays Δ through being clocked by the same unit that controls the shifts within the shift register.

The purpose of memory M0 is to translate the data stream codes for characters into codes used to address the memory units. For example, 7-bit ASCII codes in the data stream might be used as addresses to 6-bit words of M0 that contain 6-bit codes for A to Z, 0 to 9, and blank. The words in M0 addressed by the ASCII codes for comma, period, etc. could contain the 6-bit code for blank if it was required that the search treat such characters as equivalent to the blank. Thus, memory MO is set before the search in accordance with the data base conventions and requirements of the particular search with regard to character significance. It may also be set differently for fields of different attributes.

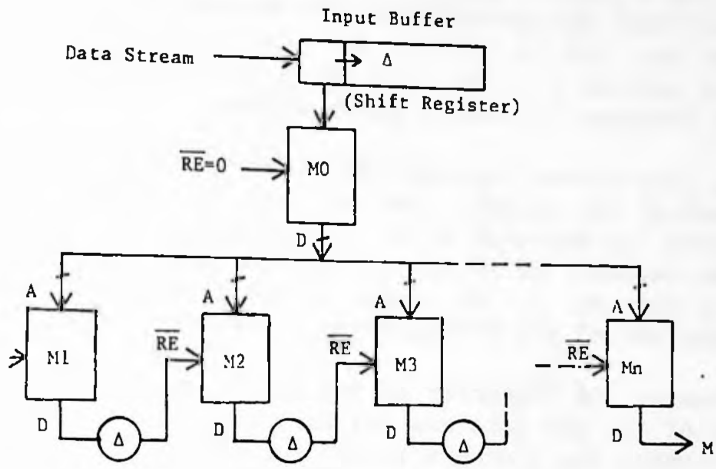

Fig. 4: Comparator Using n Memory Chips, Each of m 1-bit Words, and a Memory Chip MO for Character Encoding

Suppose the comparator of Fig. 4 is used to search for the term DISC preceded and followed by a blank. In M1, M2, M3, M4, M5, M6 the bits at address ƀ, D,I,S,C,ƀ respectively are set to 0 but all other bits are set to 1. The occurrence of a blank in the input buffer causes M1 to set 0 on its D line. After time Δ the $\overline{RE}$ line of M2 is enabled, and if the character D has then entered the input buffer it causes M1 to set 0 on its D line. Thus the only way that the output M may be set to 1 is through the data stream containing the sequence of successive characters ƀDISCƀ.

It may be noted that the contents of memories MO to Mn are set by software prior to commencement of the search. However, the data stream is then processed entirely by hardware designed to operate at the rate at which data enters the input buffer.

The configuration of Fig. 4 may be modified as shown in Fig. 5 in order to search simultaneously for r query terms. The memories M1 to Mn contain m addressable 2-bit words. The first bit of each data line is connected to the read enable line of the next memory. The second bit of each data line is connected to a read enable line of an r-bit register R1 to Rm. The m-bit data lines from each register are input to an AND gate whose output is used to trigger appropriate action such as setting the query syntax table or allowing the current record to enter the data stream to the software processor.

Suppose the comparator is used to search simultaneously for query terms

ƀINTELLIGENTƀ ƀDISCƀ ƀINTERFAC ƀINFORMATIONƀ ƀRETRIEV

Before commencement of the search the first bits of the words of M1 are set to 0 only at address ƀ, the first bits of words of M2 are set to 0 only at addresses I,D,R (=second characters of search terms), and etc. for M3 to M13. The only second bits set to 0 are in words at address ƀ in M6, V in M8, C in M9, and ƀ in M13. These bits indicate the end of query terms.

The bits within registers R1 to R13 are also set before commencement of the search. The register R6 is set as 10111 to indicate that the addressƀ in M6 corresponds to the end of term No. 2. The register R8 is set to 11110 to indicate the end of the 8-character term No. 5. The register R13 is set to 01101 to indicate the end of the 13-character terms No. 1 and No. 4.

Whenever the character string ƀDISCƀ enters the input buffer the output of the AND gate has the form x0xxx (where each x may be 0 or 1) in which the 0 occurs in the second bit because ƀDISCƀ is query term No. 2. Similarly, whenever ƀRETRIEV enters the input buffer the output of the AND gate has the form xxxx0.

The comparator of Fig. 5 is designated as a partial filter because it may lead to indications of false matches as illustrated below:

(i) Occurrence of false combinations of fragments. Thus ƀINFOƀ and ƀRITEƀ both produce an output x0xxx that is the same as for ƀDISCƀ.

(ii) Ambiguous indications of query terms. For example, both query terms ƀINTELLIGENTƀ and ƀINFORMATIONƀ produce an output of 0xx0x.

Since the purpose of the hardware processor is to reduce the data rate of information fed to the software processor the indications of false matches are unimportant provided they occur in a sufficiently small fraction of the records. However, the number of false matches may be reduced substantially by the addition of one, or more, parallel processors to check simple properties such as initial character, final character, or parity of each term.

6. STRING COMPARATOR HARDWARE USING DATA COMPRESSION

The filtering process may be examined using concepts of information theory. If data base characters are represented by 7-bit ASCII codes then 128 different characters may be present. However, when matching the data stream to a set of query terms any character not present in the query may be replaced by a special "mismatch" character. This has the effect of reducing the information content of the data stream, and hence allows it to be coded in fewer bits. Furthermore, not all possible pairs of adjacent characters will be present in the set of query terms, and those not present may again be replaced by a mismatch symbol with a consequent further reduction in the information content of the data stream. Many larger fragments of terms may similarly be eliminated from the data stream.

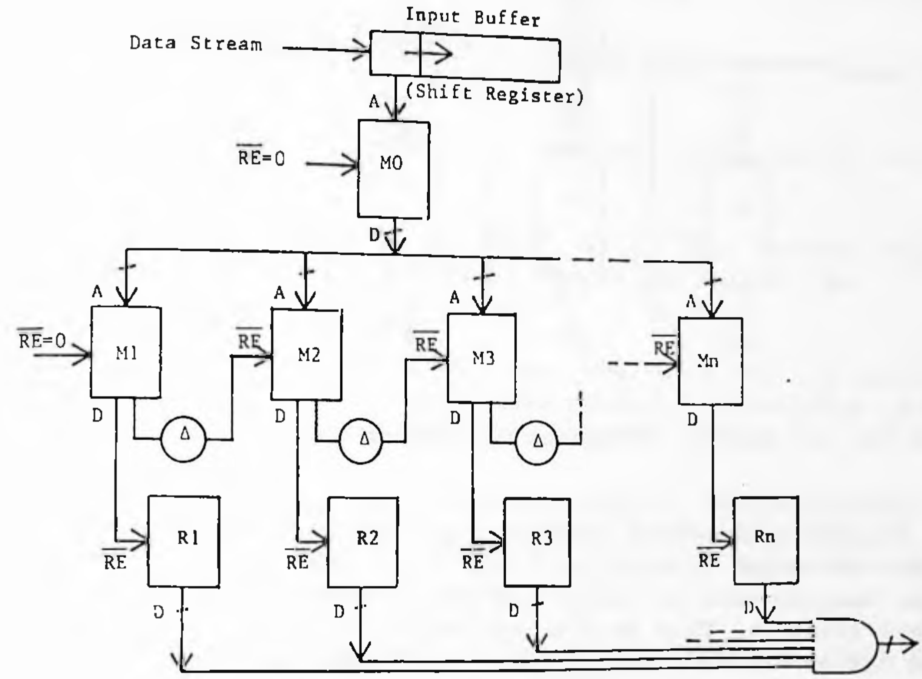

Fig. 5: Comparator for Multiple Query Terms

If there are no more than 32 query terms all of length 16 characters then no more than 256 pairs of adjacent characters, or bigrams, are needed to represent the set of query terms. In fact, the number of different bigrams will be significantly less than 256 because of repetitions. It is known (Heaps, 1978, Table 7.14) that for large collections of subject words only 20 different bigrams account for about 28% of the total number, and an additional 35 account for a further 25%. The number of different 4-grams needed to represent the 32 query terms will certainly not exceed 128.

The above remarks suggest a fragment comparator scheme as shown in Fig. 6. The memory M1 functions as M0 in Fig. 4 except that characters not present in any query term are assigned a mismatch code. Successive pairs of characters, thus modified, are input to the shift register SR2 to form a bigram that is used to address memory M2. Each word of M2 contains a code for the bigram if it is one used within a query term, and contains a mismatch code otherwise. The output of M2 is fed to a shift register SR4 that is used to address memory M4 and hence place the code for a 4-gram into shift register SR8. Each word of M2, M4, and M8 also contains a bit M for indication of a match.

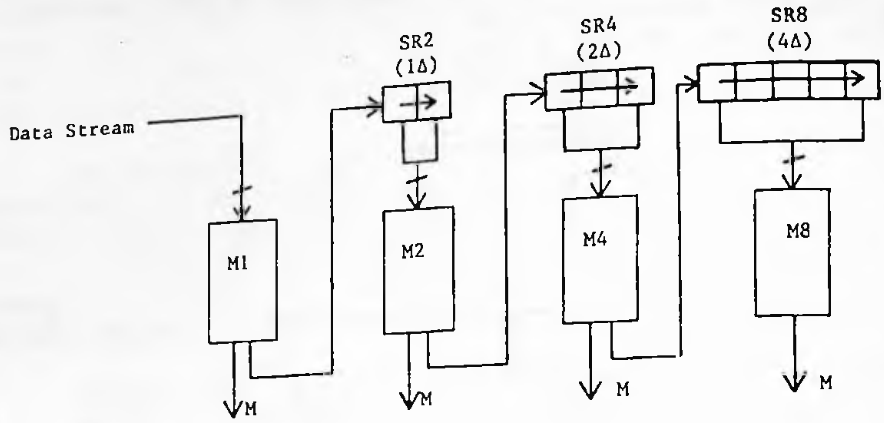

Fig. 6: Fragment Comparator

The multiple string comparator of Fig. 6 allows a search only for terms of length 1, 2, 3, 4, 6, 7, 8, 12, 14, or 15. Query terms of other lengths must be decomposed appropriately into fragments of the above lengths. This is a rather minor disadvantage and must be weighed against the advantages that fewer memories are required and any output data stream copied from SR8, SR4, SR2, M1 has a reduced data rate because of the coding in addition to the filtering.

REFERENCES

AHO, A. V. and CORASICK, M. J. "Efficient String Matching: An Aid to Bibliographic Search", Comm. A.C.M., vol. 18, No. 6, (June 1975), pp. 333-340.

BOYER, R. S. and MOORE, J. S. "A Fast String Searching Algorithm", Comm. A.C.M., Vol. 20, No. 10, (October 1977), pp. 762-772.

BIRD, R. M. "The Associative File Processor, A Special Purpose Hard-Wave System for Text Search and Retrieval", Proc. IEEE Aerospace and Electronics Conference NAECON 1979, Dayton, Ohio, pp. 433-439.

BURKOWSKI, F. J. "A Hardware Hashing Scheme in the Design of a Multi-term String Comparator", IEEE Transactions on Computers, Vol. C-31, No. 9, (September 1982), pp. 825-834.

COPELAND, G. P. "String Storage and Searching for Data Base Applications: Implementation on the INDY Backend Kernel", Proceedings Workshop on Computer Architecture Non-Numeric Processing, (1978), pp. 8-17.

GOODYEAR AEROSPACE COPRORATION. STARAN Reference Manual, Rev. 2, GER-15636B, Akron, Ohio, 1975.

HEAPS, H. S. Information Retrieval: Computational and Theoretical Aspects, Academic Press, New York, 1978.

HOLLAAR, L. A. "Text Retrieval Computers", Computer, Vol. 12, No. 3, (March 1979), pp. 40-50.

HSIAO, D. K. "Data Base Computers", Advances in Computers, Vol. 19, (1980), pp. 1-65.

KNUTH, D. E., MORRIS, J. H. and PRATT, V. R. "Fast Pattern Matching in Strings", SIAM Journal of Computing, Vol. 6, No. 2, (June 1977), pp. 323-350.

MEILANDER, N. C. "High-Speed Text Retrieval with MPP. A Parallel Processor", Communicating Information. Proceedings 43rd ASIS Annual Meeting, Vol. 17, (October 1980), pp. 303-305.

ROBERTS, D. C. (Editor). A Computer System for Text Retrieval: Design Concept Development, Office for Research and Development, Central Intelligence Agency, Washington, D.C., Technical Report RD-77-10011, 1977.

SALTON, G. "Automatic Information Retrieval", Computer, Vol. 13, No. 9, (September 1980), pp. 41-56.

STELLHORN, W. H. "A Processor for Direct Scanning of Text", Proceedings Workshop on Computer Architecture Non-Numeric Proceeding (1974).