A highly parallel computing system for information retrieval*

by BEHROOZ PARHAMI

University of California
Los Angeles, California

INTRODUCTION

The tremendous expansion in the volume of recorded knowledge and the desirability of more sophisticated retrieval techniques have resulted in a need for automated information retrieval systems. However, the high cost, in programming and running time, implied by such systems has prevented their widespread use. This high cost stems from a mismatch between the problem to be solved and the conventional architecture of digital computers, optimized for performing serial operations on fixed-size arrays of data.

It is evident that programming and processing costs can be reduced substantially through the use of special-purpose computers, with parallel-processing capabilities, optimized for non-arithmetic computations. This is true because the most common and time-consuming operations encountered in information retrieval applications (e.g., searching and sorting) can make efficient use of parallelism.

In this paper, a special-purpose highly parallel system is proposed for information retrieval applications. The proposed system is called RAPID, Rotating Associative Processor for Information Dissemination, since it is similar in function to a conventional byte-serial associative processor and uses a rotating memory device. RAPID consists of an array processor used in conjunction with a head-per-track disk or drum memory (or any other circulating memory). The array processor consists of a large number of identical cells controlled by a central unit and essentially acts as a filter between the large circulating memory and a central computer. In other words, the capabilities of the array processor are used to search and mark the file. The relevant parts of the file are then selectively processed by the central computer.

PARALLELISM AND INFORMATION RETRIEVAL

Information retrieval may be defined as selective recall of stored knowledge. Here, we do not consider information retrieval systems in their full generality but restrict ourselves to reference and document retrieval systems. Reference (document) retrieval is defined as the selection of a set of references (documents) from a larger collection according to known criteria.

The processing functions required for information retrieval are performed in three phases:

1. Translating the user query into a set of search specifications described in machine language.
2. Searching a large data base and selecting records that satisfy the search criteria.
3. Preparing the output; e.g., formatting the records, extracting the required information, and so on.

Of these three phases, the second one is by far the most difficult and time-consuming; the first one is straightforward and the third one is done only for a small set of records.

The search phase is time-consuming mainly because of the large volumes of information involved since the processing functions performed are very simple. This suggests that the search time may be reduced by using array processors. Array processing is particularly attractive since the search operations can be performed as sequences of very simple primitive operations. Hence, the structure of each processing cell can be made very simple which in turn makes large arrays of cells economically feasible.

Associative memories and processors constitute a special class of array processors, with a large number of small processing elements, which can perform simple pattern matching operations. Because of these desirable characteristics, several proposals have been made for

* This research was supported by the U.S. Office of Naval Research, Mathematical and Information Sciences Division, Contract No. N00014-69-A-0200-4027, NR 048-129.

P2

using associative devices in information retrieval applications.

Before proceeding to review several attempts in this direction, it is appropriate to summarize some properties of an ideal information retrieval system to provide a basis for evaluating different proposals.

P1. *Storage medium:* Large-capacity storage is used which has modular growth and low cost per bit.
P2. *Record format:* Variable-length records are allowed for flexibility and storage efficiency.
P3. *Search speed:* Fast access to a record is possible. The whole data base can be searched in a short time.
P4. *Search types:* Equal-to, greater-than, less-than, and other common search modes are permitted.
P5. *Logical search:* Combination of search results is possible; e.g., Boolean and threshold functions of simple search results.

Some proposals[1–3] consider using conventional associative memories with fixed word-lengths and, hence, do not satisfy P2. While these proposals may be adequate for small special-purpose systems, they provide no acceptable solution for large information retrieval systems. With the present technology, it is obviously not practical to have a large enough associative memory which can store all of the desired information[1,2] without violating P1. Using small associative memories in conjunction with secondary storage[3] results in considerable amounts of time spent for loading and unloading the associative memory, violating P3.

Somewhat more flexible systems can be obtained by using better data organizations. In the distributed-logic memory,[4,5] data is organized as a single string of symbols divided into substrings of arbitrary lengths by delimiters. Each symbol and its associated control bits are stored in, and processed by, a cell which can communicate with its two neighbors and with a central control unit. In the association-storing processor,[6] the basic unit of data is a triple consisting of an ordered pair of items (each of which may be an elementary item or a triple) and a link which specifies the association between the items. Very complex data structures can be represented conveniently with this method. Even though these two systems provide flexible record formats, they do not satisfy P1.

It is evident that with the present technology, an information retrieval system which satisfies both P1 and P3 is impractical. Hence, trading speed for cost through the use of circulating memory devices seems to provide the only acceptable solution. Delay-line associative devices that have been proposed[7,8] are not suitable for large information retrieval systems because of their fixed word-lengths and small capacities. The use of head-per-track disk or drum memories as the storage medium appears to be very promising because such devices provide a balanced compromise between P1 and P3. An early proposal of this type is the associative file processor[9] which is a highly specialized system. Slotnick[10] points out, in more general terms, the usefulness of logic-per-track devices. Parker[11] specializes Slotnick's ideas and proposes a logic-per-track system for information retrieval applications.

DESIGN PHILOSOPHY OF RAPID

The design of RAPID was motivated by the distributed-logic memory of Lee[4,5] and the logic-per-track device of Slotnick.[10] RAPID provides certain basic pattern matching capabilities which can be combined to obtain more complicated ones. Strings, which are stored on a rotating memory, are read into the cell storage one symbol at a time, processed, and stored back (Figure 1). Processing strings one symbol at a time allows efficient handling of variable-length records and reduces the required hardware for the cells.

Figure 2 shows the organization of data on the rotating memory. Each record is a string of symbols from an alphabet X, which will not be specified here. It is assumed that members of X are represented by binary vectors of length N. Obviously, each symbol must have some control storage associated with it to store the search results temporarily. One control bit has proven to be sufficient for most applications even though some

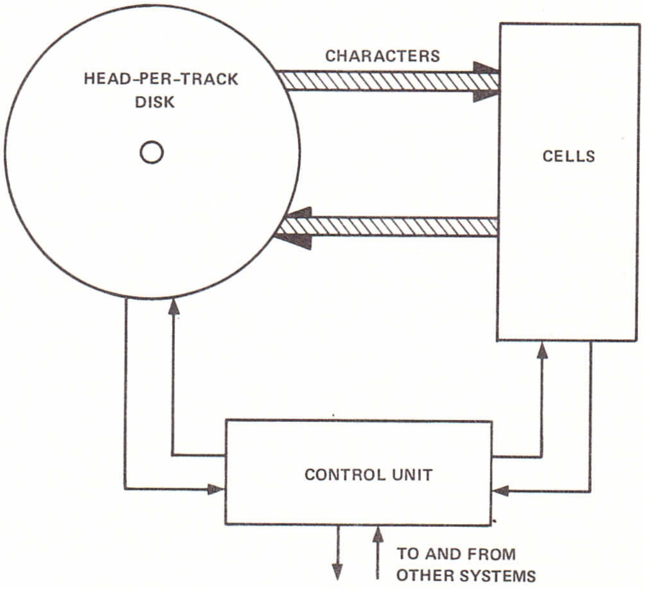

Figure 1—Overall organization of RAPID

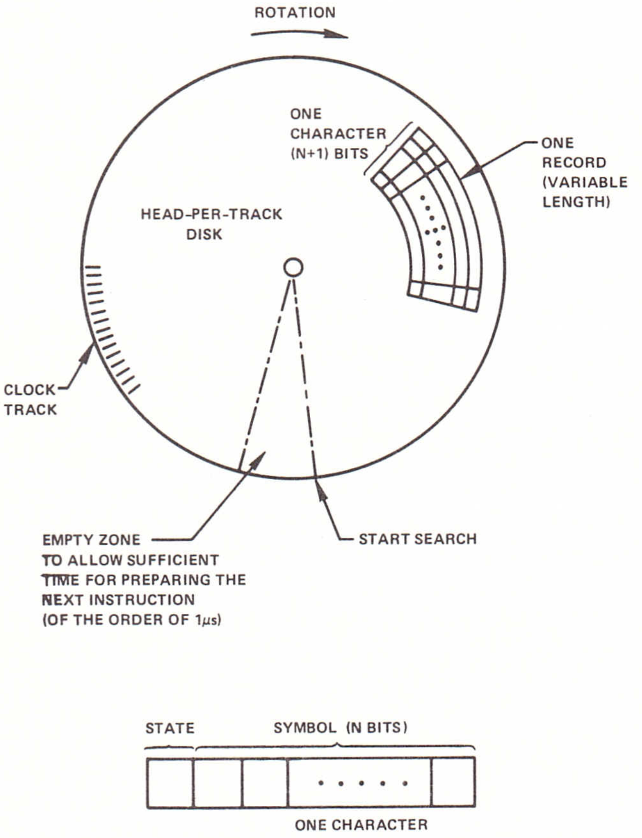
ROTATION
ONE CHARACTER (N+1) BITS
ONE RECORD (VARIABLE LENGTH)
HEAD-PER-TRACK DISK
CLOCK TRACK
EMPTY ZONE TO ALLOW SUFFICIENT TIME FOR PREPARING THE NEXT INSTRUCTION (OF THE ORDER OF 1μs)
START SEARCH
STATE
SYMBOL (N BITS)
ONE CHARACTER

Figure 2—Storage of characters and records

operations may be performed faster with a larger control field. Control information for a symbol will be called its state, $q \in \{0, 1\}$. A symbol x and its state q constitute a character, (q, x).

One of the members of X is a don't-care symbol, δ, which satisfies any search criterion. As an example for the utility of δ, consider an author whose middle name is not known or who does not have one. Then, one can use δ as his middle initial in order to make the author field uniform for all records. We will use the encoding $11 \ldots 1$ for δ in our implementation. In practice, it will become necessary to have other special symbols to delimit records, fields, and so on. The choice of such symbols does not affect the design and is left to the user. It should be emphasized, at this point, that RAPID by itself is only capable of simple pattern matching operations. Appropriate record formats are needed in order to make it useful for a particular information retrieval application. One such format will be given in this paper for general-purpose information retrieval applications.

The idea of associating a state with each symbol is taken from Lee's distributed-logic memory.[4,5] In fact, RAPID is very similar to the distributed-logic memory in principle but differs from it in the following:

1. Only one-way communication exists between neighboring characters in RAPID. This is necessitated because of the use of a cyclic memory but results in little loss in power or flexibility.
2. The use of a cheaper and slower memory makes RAPID more economical but increases the search cycle from microseconds to miliseconds.
3. Besides match for equality, other types of comparisons such as less-than and greater-than are mechanized in RAPID.
4. Basic arithmetic capability is provided in RAPID. It allows for threshold combinations of search functions as well as conventional Boolean combinations.

With the above data organization, the problem of searching for particular sets of records will reduce to that of locating substrings which satisfy certain criteria. Search for successive symbols of a string is performed one symbol per disk or drum revolution. There are at least two reasons for this design choice:

1. At any time, all the cells will be performing identical functions (looking for the same symbol). This reduces the hardware complexity of each cell since the amount of local control is minimized and fewer input and output leads are required.
2. The alternative approach of processing a few symbols at a time fails in the case of overlapping strings. Suppose one tries to process k symbols at a time ($k > 1$) by providing local control for each cell in the form of a counter. Then, if the i-th symbol in the input string is matched, the cell proceeds to match the $(i + 1)$-st symbol. Hence, if one is looking for the pattern ABCA in the string . . .DCABCABCADA. . . , only one of the two patterns will be found. Also, the pattern BCAD will not be found in the above example.

THE CONTROL UNIT

Figure 3 shows a block diagram of RAPID which is a synchronous system operating on the disk clock tracks. The phase signal generator sequences the operations by generating eight phase signals. PHA, PHB, PHC, and PHZ are generated once every disk revolution while PH1, PH2, PH3, and PH4 are generated once every bit time (Figure 4). During PHA, the cell control register (CCR), input symbol register (ISR), and address

P2

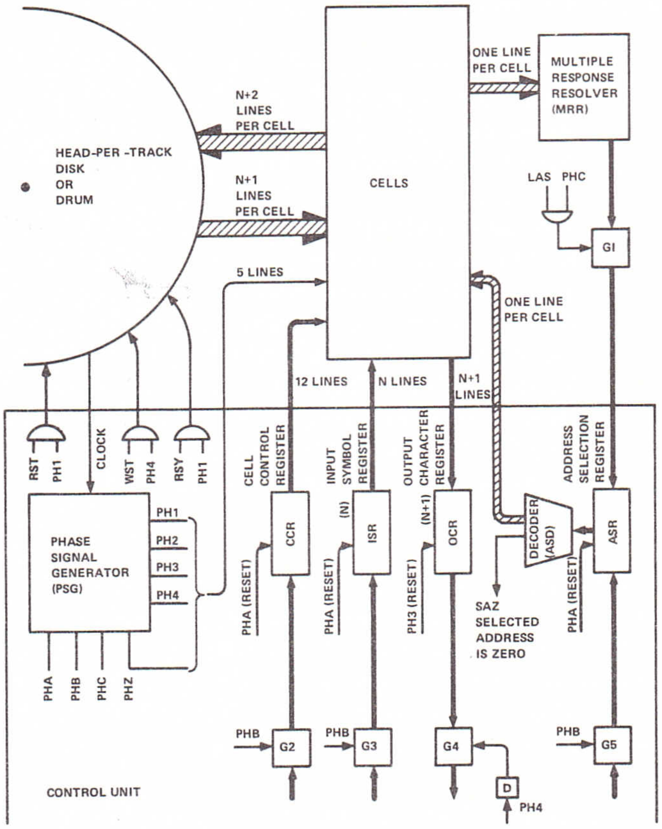

Figure 3—Block diagram of RAPID

selection register (ASR) are cleared. During PHB and PHC, these registers are loaded. Then the execution of the instruction in CCR starts. During PH3, the output character register is reset. It is loaded during PH4 and is unloaded, through G4, after a certain delay.

Most parts of the control unit, namely the instruction sequencing section and the auxiliary registers which are used to load CCR, ISR, and ASR or unload OCR, are not shown in Figure 3. It should be noted, however, that these parts process instructions at the same time that the cells are performing their functions such that the next instruction and its associated data are ready before the next PHB signal. The system can also be controlled by a general-purpose computer which is interrupted during PHB to load the auxiliary registers with the next instruction and associated data.

The arrangement of records on disk is shown in Figure 2. The $N+1$ bits of a character are stored on parallel tracks while the characters of a record are stored serially. One or more clock tracks supply the timing pulses for the system. The empty zone is provided to allow sufficient time for loading the control registers for the next search cycle.

Figure 5 shows the cell control register (CCR) which holds the instruction to be executed for one disk revolution. The function of various fields in this register will now be described.

Read field

This field consists of two bits, RST and RSY. RST commands the cells to read the state bit into the current state flip-flop, CSF. RSY commands the cells to read the symbol bits into the current symbol register, CSR.

Write field

This is similar to the read field and consists of WST and WSY. WST commands that the condition bit, CON (see description of condition field), replace the current state. WSY is a command to replace the current symbol by the contents of current symbol register, CSR, if CON=1.

Address selection field

This field contains two bits, LAS and RAS. If the LAS bit of this field is set, the address selection register

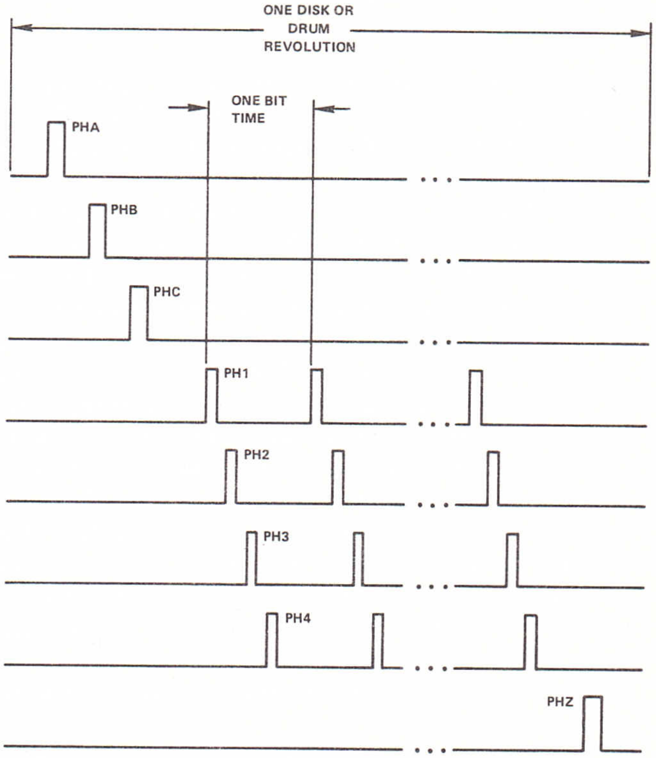

Figure 4—Timing signals

(ASR) is loaded from the multiple response resolver (MRR). MRR outputs the address of the first cell with its ASF on. If the RAS bit is set, the accumulated state flip-flop, ASF, in the cells will be reset. The function of ASF will be described with the cell design. The address selection field allows the sequential readout of the tracks which contain information pertinent to a search request.

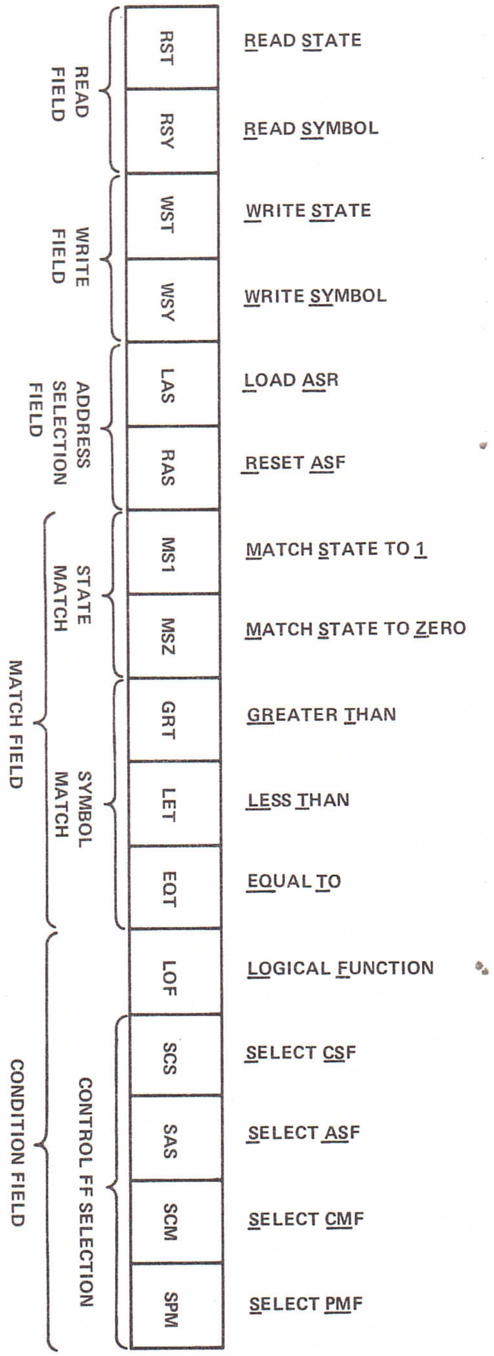

Figure 5—The cell control register (CCR)

TABLE I—The Match Condition for the State Part of a Character

MS1	MSZ	Match
0	0	never
0	1	if $q = 0$
1	0	if $q = 1$
1	1	always

Match field

This field consists of two subfields; the state match subfield, and the symbol match subfield. These subfields specify the conditions that the state and symbol of a character must meet. If both conditions are satisfied for a particular character, the current match flip-flop (CMF) of the corresponding cell is set. The state match subfield consists of MS1 and MSZ. The conditions for all combinations of these two bits are given in Table I. The symbol match subfield consists of three bits; GRT, LET, and EQT. All the symbols in the cells are simultaneously compared to the 1's complement of the contents of ISR. Table II gives the conditions for all combinations of the three signals. S is the symbol in a cell and $\bar{Y}$ is the 1's complement of the contents of ISR.

Condition field

This field specifies how the condition bit, CON, is to be computed from the contents of the following four flip-flops in a cell: current state flip-flop, CSF; accumulated state flip-flop, ASF; current match flip-flop, CMF; and previous match flip-flop, PMF. LOF specifies the logical function to be performed (AND if LOF=1, OR if LOF=0). The other four bits in this field specify a subset W of the set of four control flip-flops on which the logical function is to be performed. For example, if SCS=1, then CSF $\in W$.

TABLE II—The Match Condition for the Symbol Part of a Character

GRT	LET	EQT	Match
0	0	0	never
0	0	1	if $S = \bar{Y}$ or $S = \delta$
0	1	0	if $S < \bar{Y}$ or $S = \delta$
0	1	1	if $S \leq \bar{Y}$ or $S = \delta$
1	0	0	if $S > \bar{Y}$ or $S = \delta$
1	0	1	if $S \geq \bar{Y}$ or $S = \delta$
1	1	0	if $S \neq \bar{Y}$ or $S = \delta$
1	1	1	always

P2

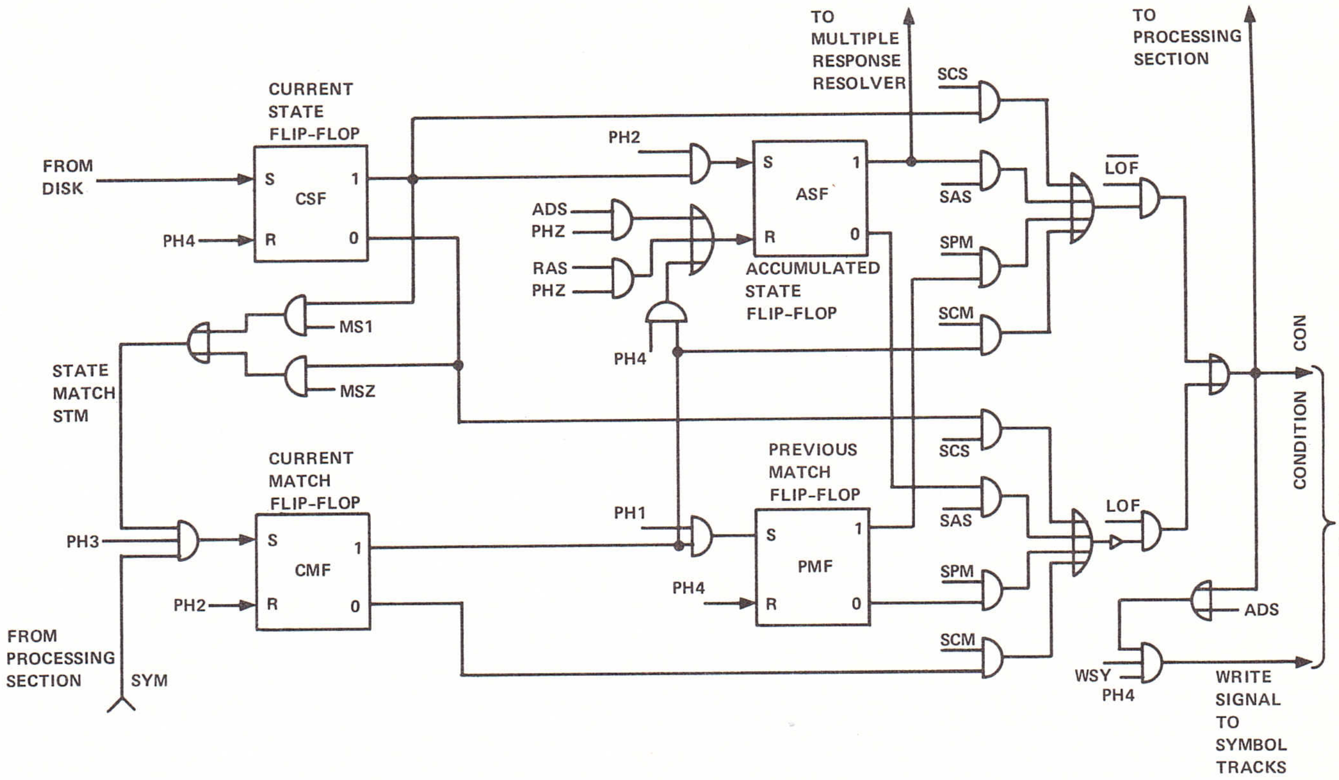

Figure 6—Control section of a cell

As will be seen later, the cell design is such that by appropriate combinations of bits in CCR, other functions besides simple comparison can be performed.

THE CELL DESIGN

Each cell consists of two sections; the control section, and the processing section. Roughly speaking, the control section processes the state part of a character while the processing section operates on the symbol part.

The control section (Figure 6) contains four flip-flops: current state flip-flop, CSF; accumulated state flip-flop, ASF; current match flip-flop, CMF; and previous match flip-flop, PMF. CSF contains the state of the character read most recently from the disk. ASF contains the logical OR of the states of characters read since it was reset. This flip-flop serves two purposes: finding out which tracks contain at least one character with a set state (reset by ADS during PHZ) and propagating the state information until a specified character is encountered (reset by RAS during PHZ and by CMF during PH4). CMF contains (after PH3) the result of current match. It is set if both the state and symbol of the current character meet the match specifications. Finally, PMF contains the match result for the previous character.

The condition signal, CON, is a logical function of the contents of control flip-flops. The four signals SCS, SAS, SCM, and SPM select a subset of these flip-flops and the logical function signal, LOF, indicates whether the contents of selected flip-flops should be ANDed (LOF=1) or ORed (LOF=0) together to form CON. The value of CON will replace the state of current character if the write state signal, WST, is activated.

The address selection signal, ADS, is activated by the address selection decoder. This signal allows conventional read and write operations to be performed on selected tracks of the disk. It is also possible, through the multiple response resolver, to read out sequentially the contents of tracks whose corresponding ASF's are set.

The processing section, shown in Figure 7, contains an N-bit adder with inputs from ISR and the current symbol register, CSR. During PH1, a symbol is read into CSR. During PH2, contents of CSR are added to contents of ISR with the result stored back in CSR. Overflow indication is stored in the overflow flip-flop, OFF. Before the addition takes place, the don't-care flip-flop, DCF, is set if CSR contains the special don't-care symbol δ. From the results of addition, it is decided whether the symbol satisfies the search specification (SYM=1 if it does, SYM=0 if it does not).

The adder in each cell allows us to add the contents of ISR to the current symbol or to compare the symbol to the 1's complement of the contents of ISR. If we denote the current symbol by S, the contents of ISR by Y, and its 1's complement by $\bar{Y}$, then:

$$S=\bar{Y} \text{ iff } S+Y+1=2^N$$

$$S>\bar{Y} \text{ iff } S+Y+1>2^N$$

$$S<\bar{Y} \text{ iff } S+Y+1<2^N$$

N is the length of the binary vector representation of S and Y. Hence if we denote the result of addition in CSR by Z and the overflow by OFF, we have:

$$S=\bar{Y} \text{ iff } Z=0$$

$$S>\bar{Y} \text{ iff } Z\neq 0 \text{ and OFF}=1$$

$$S<\bar{Y} \text{ iff OFF}=0$$

Note that the carry signal into the adder is activated if any one of the signals GRT, LET, or EQT is active. The above equations are used in the design of the circuit which computes the symbol match result, SYM (upper right corner of Figure 7). The result of symbol match is ANDed with the result of state match (STM) during PH3 to set the current match flip-flop.

Finally, during PH4, the contents of CSR can be written onto the disk or put on the output bus. Since the address selection line, ADS, is active for at most one cell, no conflict on the output bus will arise.

EXAMPLES OF APPLICATIONS

We first give a set of 12 instructions for RAPID. These instructions perform tasks that have been found to be useful in information retrieval applications. Each instruction, when executed by RAPID, will load CCR with a sequence of patterns. These sequences of patterns are also given. We restrict our attention to search

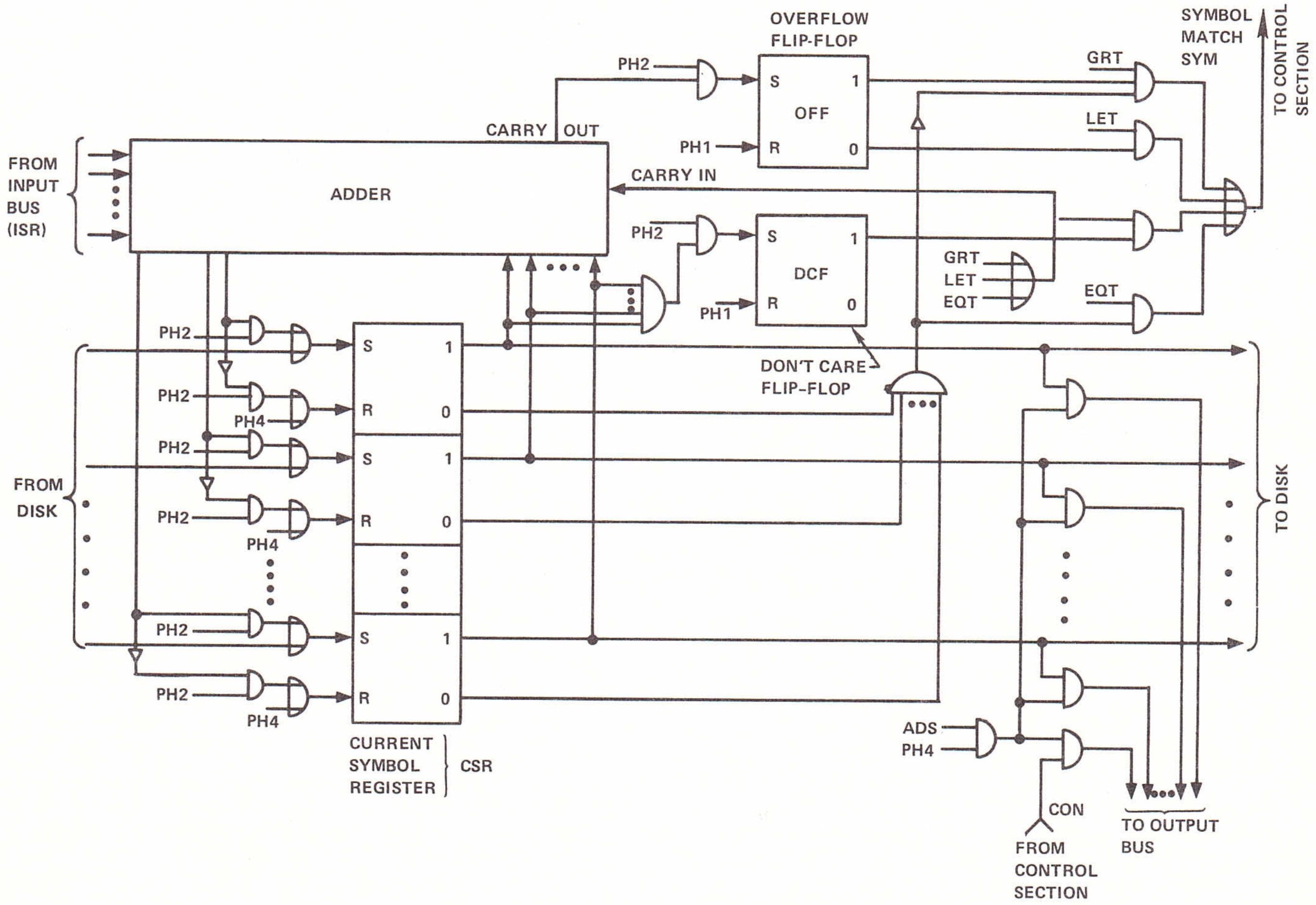

Figure 7—Processing section of a cell

P2

instructions only. Input and output instructions must also be provided to complete the set.

1. **search and set** s: Find all occurrences of the symbol s and set their states.
2. **search for** $s_1s_2 \ldots s_n$: Find all the occurrences of the string $s_1s_2 \ldots s_n$ and set the state of the symbols which immediately follow s_n.
3. **search for marked** $s_1s_2 \ldots s_n$: Same as the previous instruction except that for a string to qualify, the state of its first symbol must be set.
4. **search for marked** ψ s: Search for symbols whose states are set and have the relation ψ with s. Then, set the state of the following symbol. Possible relations are $<$, $\leq$, $>$, $\geq$, and $\neq$.
5. **propagate to** s: If the state of a symbol is set, reset it and set the state of the first s following it.
6. **propagate** i: If the state of a symbol is set, reset it and set the state of the i-th symbol to its right.
7. **expand to** s: If the state of a symbol is set, set the state of all symbols following it up to and including the first occurrence of s.
8. **expand** i: If the state of a symbol is set, set the state of the first i symbols following it.
9. **contract** i: If the state of a symbol is reset, reset the state of the first i symbols following it.
10. **expand** i **or to** s: If the state of a symbol is set, perform 7 if an s appears within the next i symbols; otherwise, perform 8.
11. **add** s: Add the numerical value of s to the numerical value of any symbol whose state is set.
12. **replace by** s: If the state of a symbol is set, replace the symbol by s.

The microprograms for these instructions are given

TABLE III—Microprograms for RAPID Instructions

Number	Instruction		Repetition	Contents of ISR	Contents of CCR															
					Read Field		Write Field		Address Selection		Match Field					Condition Field				
											State		Symbol			Logic	FF Selection			
					RST	RSY	WST	WSY	LAS	RAS	MS1	MSZ	GRT	LET	EQT	LOF	SCS	SAS	SCM	SPM
1	search and set s		1	$\bar{s}$		1	1	0	0		1	1	0	0	1		0	0	1	0
2	search for $s_1s_2 \ldots s_n$		1	$\bar{s}_1$		1	1	0	0		1	1	0	0	1		0	0	0	1
			j=2 to n	$\bar{s}_j$	1	1	1	0	0		1	0	0	0	1		0	0	0	1
3	search for marked $s_1s_2 \ldots s_n$		j=1 to n	$\bar{s}_j$	1	1	1	0	0		1	0	0	0	1		0	0	0	1
4	search for marked ψs	<	1	$\bar{s}$	1	1	1	0	0		1	0	0	1	0		0	0	0	1
		≤	1	$\bar{s}$	1	1	1	0	0		1	0	0	1	1		0	0	0	1
		>	1	$\bar{s}$	1	1	1	0	0		1	0	1	0	0		0	0	0	1
		≥	1	$\bar{s}$	1	1	1	0	0		1	0	1	0	1		0	0	0	1
		≠	1	$\bar{s}$	1	1	1	0	0		1	0	1	1	0		0	0	0	1
5	propagate to s		1	$\bar{s}$	1	1	1	0	0	1	1	1	0	0	1	1	0	1	1	0
6	propagate i		i		1		1	0	0		1	0	1	1	1		0	0	0	1
7	expand to s		1	$\bar{s}$	1	1	1	0	0	1	1	1	0	0	1		0	1	0	0
8	expand i		i		1		1	0	0		1	0	1	1	1	0	1	0	0	1
9	contract i		i		1		1	0	0		1	0	1	1	1	1	1	0	0	1
10	expand i or to s		i	$\bar{s}$	1	1	1	0	0		1	1	1	1	0	1	1	0	1	0
					1		1	0	0		1	0	1	1	1	0	1	0	0	1
11	add s		1	s	1	1		1	0								1	0	0	0
12	replace by s		1	s	1	0		1	0								1	0	0	0

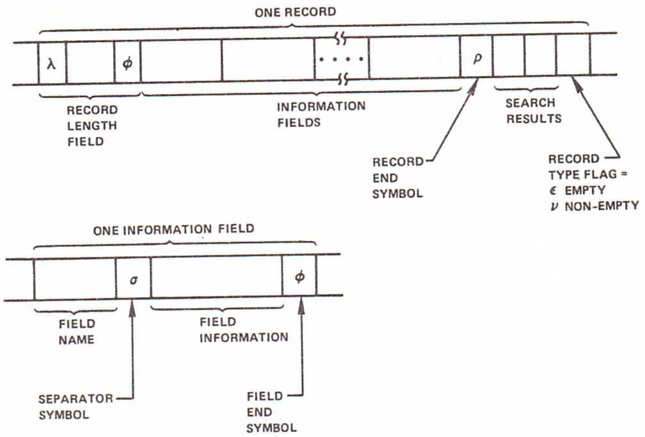

Figure 8—Data storage format

in Table III. A blank entry in this table constitutes a don't-care condition. The entries in the repetition column specify the number of times the given patterns should be repeated. As can be seen from Table III, this set of instructions does not exploit all the capabilities of RAPID since some of the bits in the CCR assume only one value (0 or 1) for all the instructions.

To illustrate the applications of RAPID, we first choose a format for the records (Figure 8). The record length field must have a fixed length in order to allow symbol by symbol comparison of the record length to a given number. The information fields can be of arbitrary lengths. The flag field contains three characters; two for holding the results of searches, and one which contains a record type flag. The Greek letters used on Figure 8 are reserved symbols and should not be used except for the purposes given in Table IV.

As mentioned earlier, a special symbol, δ, is used as a don't-care symbol. It is also helpful to have a reserved symbol, τ, which can be used as temporary substitute for other symbols during a search operation. Let us now consider two simple examples to show the utility of the given instruction set.

Example 1. Assuming that the record length is specified by one symbol, the following program marks all the empty records whose lengths are not less than s. This is useful when entering a new record of length s to find which tracks contain empty records that are large enough.

```
search for $\lambda$
search for marked $\geq s$
propagate to $\rho$
propagate 3
search for marked $\epsilon$
```

If the record length is specified by two characters, we note that $t_1t_2 \geq s_1s_2$ iff $t_1 > s_1$ or $t_1 = s_1$ and $t_2 \geq s_2$. Hence, we write the following program:

```
search for $\lambda$
search for marked $> s_1$
propagate 1
replace by $\tau$
search for $\lambda$
search for marked $s_1$
search for marked $\geq s_2$
replace by $\tau$
search and set $\tau$
replace by $\phi$
propagate to $\rho$
propagate 3
search for marked $\epsilon$
```

Example 2. The following program marks all non-empty records which contain in their title field, designated by TI, a word having "magnet" as its first six characters and having 3 to 10 non-blank characters after that. β designates the "blank" character.

```
search for $\phi$TI$\sigma$
expand to $\phi$
search for marked magnet
expand 10 or to $\beta$
contract 3
propagate to $\rho$
propagate 3
search for marked $\nu$
```

It is important to note that the record format given here serves only as illustration. Because of its generality and flexibility, this format is not very efficient in terms of storage overhead and processing speed. For any given application, one can probably design a format which is more efficient for the types of queries involved.

CONCLUSION

In this paper, we have described a special-purpose highly parallel system for information retrieval applica-

TABLE IV—List of Reserved Symbols

λ	Indicates start of *length* field.
ρ	Indicates end of a *record.*
σ	*Separates* name and information subfields in a field.
ϕ	Indicates end of a *field.*
ϵ	Designates the end of an *empty* record.
ν	Designates the end of a *non-empty* record.
δ	Is the *don't-care* symbol.
τ	Is used as *temporary* substitute for other symbols.

P2

tions. This system must be evaluated with respect to the properties of an ideal information retrieval system summarized earlier. It is apparent that RAPID satisfies P2, P4 and P5. The extent to which P1 and P3 are satisfied by RAPID is difficult to estimate at the present.

With respect to P1, the storage medium used has a low cost per bit. However, the cost for cells must also be considered. Because of the large number of identical cells required, economical implementation with LSI is possible. Figures 6 and 7 show that each cell has one N-bit adder, $N+6$ flip-flops, $6N+39$ gates, and $4N+23$ input and output pins. For a symbol length of $N=8$ bits, each cell will require no more than 250 gates and 60 input and output pins. The number of input and output pins can be reduced considerably at the expense of more sophisticated gating circuits (i.e., sharing input and output connections).

With respect to P3, the search speed depends on the number of symbols matched. If we assume that on the average 50 symbols are matched, the matching phase will take about 70 disk revolutions (to allow for overhead such as propagation of state information and performance of logical operations on the search results). Hence, the search time for marking the tracks which contain relevant information is of the order of a few seconds.

Some important considerations such as input and output of data and fault-tolerance in RAPID have not been explored in detail and constitute possible areas for future research. The interested reader may consult Reference 12 for some thoughts on these topics.

ACKNOWLEDGMENTS

The author gratefully acknowledges the guidance and encouragement given by Dr. W. W. Chu in the course of this study. Thanks are also due to Messrs. P. Chang, D. Patterson, and R. Weeks for stimulating discussions.

REFERENCES

1 J GOLDBERG M W GREEN
Large files for information retrieval based on simultaneous interrogation of all items
Large-capacity Memory Techniques for Computing Systems New York Macmillan pp 63-67 1962

2 S S YAU C C YANG
A cryogenic associative memory system for information retrieval
Proceedings of the National Electronics Conference pp 764-769 October 1966

3 J A DUGAN R S GREEN J MINKER W E SHINDLE
A study of the utility of associative memory processors
Proceedings of the ACM National Conference pp 347-360 August 1966

4 C Y LEE
Intercommunicating cells, basis for a distributed-logic computer
Proceedings of the FJCC pp 130-136 1962

5 C Y LEE M C PAULL
A content-addressable distributed-logic memory with applications to information retrieval
Proceedings of the IEEE Vol 51 pp 924-932 June 1963

6 D A SAVITT H H LOVE R E TROOP
ASP; a new concept in language and machine organization
Proceedings of the SJCC pp 87-102 1967

7 W A CROFUT M R SOTTILE
Design techniques of a delay line content-addressed memory
IEEE Transactions on Electronic Computers Vol EC-15 pp 529-534 August 1966

8 P T RUX
A glass delay line content-addressable memory system
IEEE Transactions on Computers Vol C-18 pp 512-520 June 1969

9 R H FULLER R M BIRD R M WORTHY
Study of associative processing techniques
Defense Documentation Center AD-621516 August 1965

10 D L SLOTNICK
Logic per track devices
Advances in Computers Vol 10 pp 291-296 New York Academic Press 1970

11 J L PARKER
A logic-per-track retrieval system
Proceedings of the IFIPS Conference pp TA-4-146 to TA-4-150 1971

12 B PARHAMI
RAPID; a rotating associative processor for information dissemination
Technical Report UCLA-ENG-7213 University of California at Los Angeles February 1972