A Design of a Fast Cellular Associative Memory for Ordered Retrieval

C. V. RAMAMOORTHY, FELLOW, IEEE, JAMES L. TURNER, AND BENJAMIN W. WAH, STUDENT MEMBER, IEEE

Abstract—**In this paper, we design some simple schemes for a variety of searches, each of which may be performed in one complete memory cycle using bit-memory logic primarily. The searches we study include the basic equality search, the threshold searches (both greater than and less than searches), and most importantly, the greatest value and the least value searches. For each kind of search, we present both the algorithm suitable for our needs and the logic circuit of the memory cell required by the algorithm. Based on the basic search schemes, an algorithm for ordered retrieval is developed. A comparison for ordered retrieval schemes is then made between the proposed scheme and the previous algorithms. It is found that this algorithm outperforms all the other algorithms compared, particularly in the resolution of multiple responses. Finally, issues relating to LSI implementation, manufacturing defects, modular expansion, and extension to associative sequential memories are discussed.**

Index Terms—**Basic search, content-addressable memory, equality-threshold search, maximum value search, minimum value search, multiple match resolution, ordered retrieval.**

I. Introduction

CONTENT-ADDRESSABLE memories (CAM's), alternatively known as associative memories (AM's), have received much attention in the literature since they were first described in 1956 [14]. The distinguishing feature of such memories is that stored words are accessed by matching some portion of their contents to a search word and selecting the first one that matches rather than accessing the data using its physical location in the memory as in standard random access memories (RAM's). It can be readily seen that CAM's must depend upon a high degree of parallelism in their search schemes in order to compete in memory access times with RAM's. Large speed improvements can be gained from this parallelism and this makes CAM's attractive to a wide variety of applications. A good survey of the current technology in CAM's can be found in [12], [3], and [5].

With the advent of large scale integration (LSI) technology, it becomes feasible to economically implement fast search algorithms in CAM's by incorporating much of the control logic into the memory plane. Several search algorithms for CAM's have been developed in the past two decades, [1], [4], [13], [10]. Some algorithms, such as [13] and [17] have been based upon distributed logic design, but few have incorporated a high percentage of their search logic in the memory cell. An exception to this is found in a design by Kautz [7] for a special purpose sorting array. His design is oriented towards ordering, rather than searching, of the memory, but does include associative capabilities as a byproduct.

The trend in associative memory design is toward distributed logic. Previous designs have placed control logic outside the storage logic. This control logic includes comparison logic, propagation logic, multiple response resolution logic, arithmetic logic, etc. In a distributed logic design, the control logic and the storage logic are designed together. The controls are brought into the cells as part of the storage itself. The cells become more complex and have more control functions associated with them, but it also results in more homogeneous and modular design. In this paper, we propose the basic design of such a memory and present some searching and sorting schemes and the implementation of some basic searches using distributed cellular logic which is considerably faster than any of the previous sorting methods. The capabilities of the cells are actually a subset of the capabilities of Kautz's augmented CAM array [8]. The searches that we will examine include the basic equality search, the threshold searches (both greater than and less than searches) and most importantly, the greatest value search and the least value search.

The following conventions are used throughout the paper:

Manuscript received January 18, 1977; revised September 9, 1977. This research was supported by National Science Foundation Grant MCS72-03734-A02.

C. V. Ramamoorthy and B. W. Wah are with the Computer Science Division, Department of Electrical Engineering and Computer Sciences and the Electronics Research Laboratory, University of California, Berkeley, CA 94720.

J. L. Turner is with the National Semiconductor Corporation, Santa Clara, CA 95051.

0018-9340/78/0900-0800$00.75

B_i the value of the ith word of memory,
$B_{i,j}$ the value of the jth bit of the ith word of memory,
C a priority circuit which is used to sequence responses in W_2, W_3, or W_4,
D a circuit used to detect responses in W_2, W_3, or W_4,
$E_{i,j}$ the equality enable signal for the jth bit of the ith word in the equality–inequality search mode and the least value search mode,
$E_{i,n+1}$ signal which can be gated to set (or reset) any one of the word control registers W_2, W_3, W_4,
$F_{i,j}$ the enable signal for the jth bit of the ith word in the greatest value search,
$F_{i,n+1}$ signal which can be gated to set (or reset) any one of the word control registers W_2, W_3, W_4,
G the associative memory search mode command (equality–inequality mode or the least value mode),
i an index for a word in the memory, $1 \le i \le m$,
I_j the value of the jth bit of the input/output register I,
j an index for a bit in the word, $1 \le j \le n$,
k a variable index, $1 \le k \le n$,
L_i the less than state signal for the ith word of memory; a signal which can be gated to set (or reset) any one of the word control registers W_2, W_3, W_4,
LSB Least Significant Bit,
m the number of words in the CAM,
M_j the value of the jth bit of the mask register M, (used in the minimum value search, the equality search and the threshold searches),
M_j^* the value of the jth bit of the mask register M^* (used in value search the greatest),
MSB Most Significant Bit,
MZ the set of all bit positions with $M_j = 0$.
n the number of bits in a word of the CAM,
P_j the synchronization bus signal for the jth bit-slice in the least value search,
Q_j the default-detection bus signal for the jth bit-slice in the least value search,
r an index in the word control logic, $1 \le r \le 4$,
R_j the signal for the jth bit-slice shared by the equality–inequality search and the least value search,
S the value of the search register S,
S_j the value of the jth bit of the search register S,
T_j the search-default feedback bus signal for the jth bit-slice in the greatest value search,
U_j the synchronization bus signal for the jth bit-slice in the greatest value search,
V_j the default-detection bus signal for the jth bit-slice in the greatest value search,
$w_{i,r}$ the ith flip-flop of the word control register W_r,
W_1 the word flags register with m flip-flops,
$W_2 - W_4$ results stores or temporary stores in the word control logic,
$\cup$ abbreviation for logical OR operation,
$\in$ abbreviation for "an element of,"
$\forall$ abbreviation for "for all,"
$\exists$ abbreviation for "there exists."

II. Basic Associative Memory Organization

The associative memory organization shown in Fig. 1 is used to implement the search schemes to be presented. A *bit-slice* is a vertical slice through the memory as arranged in Fig. 1. The jth bit-slice is made up of the jth bit of every word in the memory. The search operations are parallel by word and serial by bit-slice. A *minor cycle* refers to the time needed to perform an operation on a single bit-slice and a *major cycle* refers to the time needed to complete an operation on all bit-slices of the memory. Hence, a major cycle for the present AM organization is composed of n minor cycles where n is the number of bits in a word. It is shown later that some searches will require a longer minor cycle than others, thereby lengthening the major cycle as well. A "*basic*" *operation* is an operation which may be performed in a single major cycle.

III. Definition of Search Operations

In each of the following search definitions, the set of words involved in the search are those where $w_{i,1} = 1$ and $i \in \{1, 2, \cdots, m\}$. The result of the search partitions this set of words into two sets, the set that satisfies the search condition and the set that does not. Let B_i be the content of the ith word in the memory, S be the content of the search register, and M be the content of the mask register. That is,

$$B_i = \sum_{j=1}^{n} 2^{n-j} \cdot B_{i,j}, \qquad S = \sum_{j=1}^{n} 2^{n-j} \cdot S_j$$

and

$$M = \sum_{j=1}^{n} 2^{n-j} \cdot M_j.$$

The search is performed only on that part of the search word which is not masked. In other words, only those S_j bits for which the corresponding M_j bits are 0's are included in the matching (comparison) process. Let MZ be this set of bit positions. Other bit positions with $M_j = 1$ are bypassed. (Note that $j = 1$ for MSB and $j = n$ for LSB.) We define the various searches as follows:

A. Equivalence Searches

1) *Equality Search:* $B_{i,j} = S_j$, $\forall j \in \{1, 2, \cdots, n\}$.
2) *Inequality Search:* $\exists k \in MZ$ such that $B_{i,k} \neq S_k$.

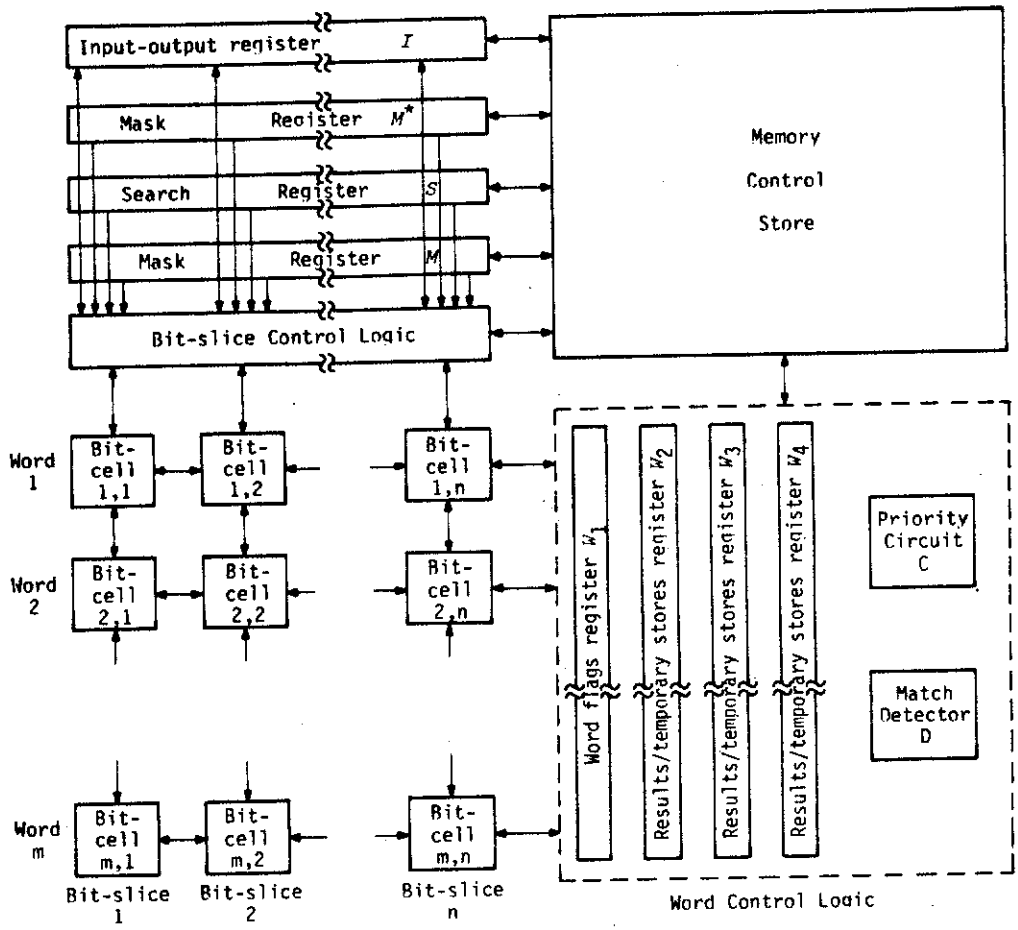


Fig. 1. Cellular logic associative memory block diagram.

3) *Similarity Search (Masked Equality Search):* $B_{i,j} = S_j$ $\forall j \in MZ$.

4) *Proximity Search:* There is *exactly one* $k \in MZ$ such that $B_{i,k} \neq S_k$.

Note: The similarity search is also known as masked-equality search. It differs from the equality search in that the mask is effectively not used in the latter while it is used in the former search. In most cases, this distinction is so insignificant that the "equality search" is used to mean both the equality and the masked-equality searches. Unless specified otherwise, we will assume that all searches are masked.

B. Threshold Searches

1) *Greater-Than Search:* $B_i > S$.
2) *Less-Than Search:* $B_i < S$.
3) *Greater-Than-or-Equal-To Search:* $B_i \geq S$.
4) *Less-Than-or-Equal-To Search:* $B_i \leq S$.

C. Double-Limits Searches

1) *Between-Limits Search:* Let X and Y be the limits such that $X > Y$. Then B_i is
   a) $< X$ and $> Y$,
   b) $< X$ and $\geq Y$,
   c) $\leq X$ and $> Y$,
   d) $\leq X$ and $\geq Y$.

2) *Outside-Limits Searches:* Let X and Y be the limits such that $X < Y$. Then B_i is
   a) $< X$ or $> Y$,
   b) $< X$ or $\geq Y$,
   c) $\leq X$ or $> Y$,
   d) $\leq X$ or $\geq Y$.

D. Extremum Searches

1) *Least Value Search:* $B_i \leq B_k$, $\forall k \neq i$ and $k \in \{1, 2, \cdots, m\}$.

2) *Greatest Value Search:* $B_i \geq B_k$, $\forall k \neq i$ and $k \in \{1, 2, \cdots, m\}$.

E. Adjacency Searches

1) *Nearest-Above Search:* $\exists k \in \{1, 2, \cdots, m\}$ such that $B_i > B_k > S$.

2) *Nearest-Below Search:* $\exists k \in \{1, 2, \cdots, m\}$ such that $B_i < B_k < S$.

There are other nonsearch operations that can be performed in associative memories. These include word addition, field addition, summation, counting, shifting, complementing, logical sum, logical product, etc. Devices that incorporate nonsearch operations may be referred to as associative processors [3]. We will not investigate further on nonsearch operations in this paper.

IV. Algorithms and Implementations of Basic Searches

We define a *basic search* as one which can be completed in exactly one major cycle, assuming multiple response resolution as an operation separate from search operations. This definition applies only to the configuration of the CAM in Fig. 1. A *multiple response* is the situation when more than one word satisfies the given search condition. The multiple response resolution resolves this situation by means of a priority circuit [2] or other schemes, e.g., [9], [6], [16], and outputs all responders one at a time. Among the searches listed in the previous section, not all of them can be economically implemented as basic searches. Therefore, we choose to implement those searches which are most frequently used as basic searches while the rest can be performed in a series of the basic searches. As an example, the between-the-limits search ($Y \leq B_i < X$) can be generated by performing a less-than search ($< X$) followed by a greater-than-and-equal-to search ($\geq Y$) on the responders of the first search. In the implementation to be presented, the basic searches are the equality search, the similarity (masked equality) search, the four threshold searches, and the two extremum searches. Each of these basic searches can be performed alone or many combinations of them can be performed simultaneously. These searches are grouped into three groups called Mode A, Mode B, and Mode C operations. The Mode groupings are as follows.

Mode A: The equality search, the similarity search and the four threshold searches.

Mode B: The least value search.

Mode C: The greatest value search.

Searches in Mode A can be performed simultaneously. Furthermore, Mode A or Mode B operations can be performed simultaneously with Mode C operations. We will assume that positive logic is used throughout our designs.

A. Mode A: Equality-Threshold Search Mode

In this mode, the CAM is partitioned according to the magnitude of the search word S into three sets, namely, words which are equal to S, words which are less than S, and words which are greater than S. The result of this search mode is stored in two of the word control registers, W_2 and W_3, and the interpretation is given in the algorithm to follow. This search mode is characterized by the signal $G = 0$, which gates the contents of the search register S to the search bus. That is, $R_j = S_j \forall j \in \{1, 2, \cdots, n\}$. The basic searches performed in this mode are the equality searches and the four threshold searches (namely, $> S$, $< S$, $\geq S$, and $\leq S$). $M_j = 0$ means that S_j is not masked while $M_j = 1$ means S_j is masked.

The three query states are shown in the following table.

M_j	S_j	Query State
0	0	0
0	1	1
1	0	d
1	1	d

d = don't care

Algorithm A—Mode A Search Operation

1)

a) Initialization:

$S \leftarrow$ Search word, $M \leftarrow$ Mask, $G = 0$, $j = 0$,

$w_{i,1} = 1$, $w_{i,2} = 0$, $w_{i,3} = 0$, $\forall i \in \{1, 2, \cdots, m\}$.

b) Data Path Setting:

Gate $E_{i,n+1}$ to $w_{i,3}$, L_i to $w_{i,2}$, and $w_{i,1}$ to $E_{i,1}$,

$\forall i \in \{1, 2, \cdots, m\}$.

These data paths and the control signal G are held until the completion of the major cycle.

2) *Let* $j \leftarrow j + 1$.

3) *Compute*

a) $E_{i,j+1} = E_{i,j} \cdot (M_j + B_{i,j} \cdot R_j + \bar{B}_{i,j} \cdot \bar{R}_j)$, $\forall i \in \{1, 2, \cdots, m\}$, simultaneously.

b) $d_{i,j} = E_{i,j} \cdot \bar{B}_{i,j} \cdot (M_j + B_{i,j} \cdot R_j + \bar{B}_{i,j} \cdot \bar{R}_j)$, $\forall i \in \{1, 2, \cdots, m\}$, simultaneously.[1]

c) $L_i = \bigcup_{k=1}^{j} d_{i,k}$ (wired-OR), $\forall i \in \{1, 2, \cdots, m\}$, simultaneously.

4) *Is* $j = n$?

a) Yes—Proceed to step 5).

b) No—Proceed to step 2).

5) *Result Interpretation:*

$w_{i,2}$ $(= L_i)$	$w_{i,3}$ $(= E_{i,n+1})$	*Interpretation*
0	0	B_i is greater than search word,
0	1	B_i matches search word,
1	0	B_i is less than search word,
1	1	[does not occur].

In this search algorithm, the minor cycle is composed of step 3) alone while the major cycle is composed of steps 2) 4). The result of this search mode is handled by the match detector D in the word control logic section. Any multiple responses will be resolved by the priority circuit C. The

[1] $d_{i,j}$ will be sensitive to $\bar{B}_{i,j}$ and only to the first bit mismatch between B_i and S. A simpler design using $d_{i,j} = E_{i,j+1} \cdot \bar{B}_{i,j}$ can be used. In this case, $d_{i,j}$ will be sensitive to all mismatches between B_i and S. Since L_i is obtained by wired-ORing $d_{i,j}$'s, the final output voltage of the wired-OR will depend on the number of mismatches. It will be more appropriate to eliminate this dependence by only taking the first mismatch as what is done here. We must confess that the exact design is highly technology-dependent. The above point was pointed out by one of the referees.

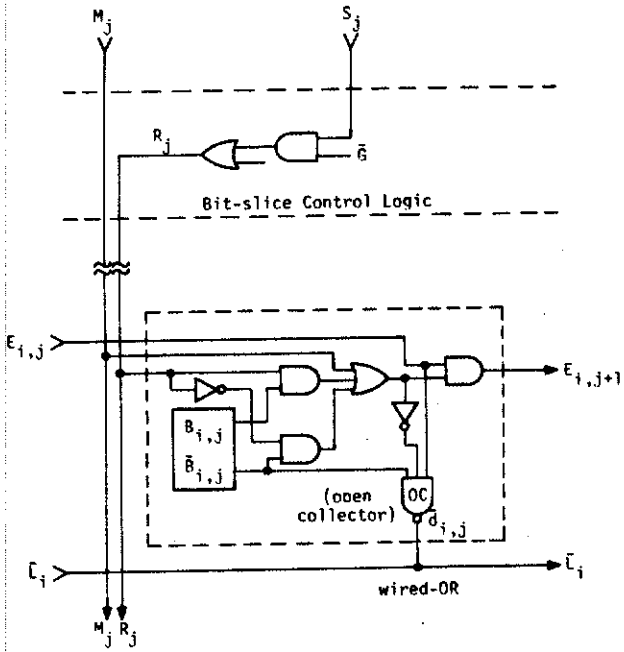


Fig. 2. Bit-cell with equality, greater-than, and less-than capability for Mode A operation.

bit-cell logic needed to implement this equality-inequality search mode is shown in Fig. 2. The delay in each minor cycle is one gate delay. The following example shows the state of L_i and $E_{i,j}$ for a Mode A search of 6 words, each 5 bits long.

Example 1: "Mode A" Search Operation

Search Word—S	10110,
Mask Word—M	00100,
Effective Search Word—S'	10d10 (d = don't care).

	i	B_i	State of $(L_i, E_{i,j+1})$ Lines at the End of the Minor Cycle $j=0$	1	2	3	4	5
Memory Words	1	10111	01	01	01	01	01	00
	2	11000	01	01	00	00	00	00
	3	10010	01	01	01	01	01	01
	4	10110	01	01	01	01	01	01
	5	10101	01	01	01	01	10	10
	6	01101	01	10	10	10	10	10

For interpretation of L_i, $E_{i,6}$, see step 5) of Algorithm A, for Mode A search operations.

B. Mode B: Least Value Search Mode

In this mode, the search register is no longer needed because no search word is used. However, the minor cycle is more complicated than that in Mode A. It now consists of a comparison phase and a default phase. Consider the jth minor cycle. In the comparison phase, one of the three conditions is to be detected: 1) that the bit-slice is masked, 2) that the bit-slice is not masked and at least one enabled bit-cell contains a "0," and 3) that the bit-slice is not masked and all enabled bit-cells contain a "1." In the first case, all enable signals to this bit-slice are passed on to the next bit-slice on the right. In the second case, those enabled bit-cells containing a "0" will pass its enable signal to the next bit-cells on the right. In both cases, the minor cycle is complete. The third case, however, is called the default case and the default phase is entered. The default condition is detected in the default-detection bus and the default signal Q_j is fed back to the bit-slice via R_j. R_j is connected to the default feedback circuitry ($R_j = P_j \cdot \bar{Q}_j \cdot G$) when this search mode is activated by setting $G = 1$. P_j is a synchronization signal and it also serves as the search signal in the comparison phase. After the default phase, all enabled bit-cells pass their enable signals to the next bit-cells on the right, thus completing the minor cycle. The result of Mode B can be stored in either W_2 or W_3 because Mode A does not operate simultaneously with Mode B.

The implementation of the Mode B search in each bit-cell is shown in Fig. 3. Note that this implementation shares much of the circuitry with that for Mode A and that at the beginning of the jth minor cycle, $R_j = 0$.

Algorithm B—Mode B Search Operation—The Least Value Search Algorithm

1)

a) Initialization:

$G = 1, j = 0, w_{i,1} = 1, w_{i,2} = 0 \quad \forall i \in \{1, 2, \cdots, m\}$.

b) Data Path Setting:

Gate $E_{i,n+1}$ to $w_{i,2}$, $w_{i,1}$ to $E_{i,1} \quad \forall i \in \{1, 2, \cdots, m\}$.

The data paths and the control signal G are held until the completion of the major cycle.

2) Let $j \leftarrow j + 1$.

3) Minor Cycle:

a) Comparison Phase: Compute

i) $E_{i,j+1} = E_{i,j} \cdot (M_j + B_{i,j} \cdot R_j + \bar{B}_{i,j} \cdot \bar{R}_j) \quad \forall i \in \{1, 2, \cdots, m\}$ simultaneously.

ii) $p_{i,j}(t) = E_{i,j}(t-2)$ (delay element used to synchronize the feedback of Q_j via R_j) $\forall i \in \{1, 2, \cdots, m\}$ simultaneously.

iii) $q_{i,j} = E_{i,j} \cdot \bar{B}_{i,j} \; \forall i \in \{1, 2, \cdots, m\}$ simultaneously.
$q_{i,j} = 1$ means that $B_{i,j}$ is enabled and equals 0.

iv) $P_j = \bigcup_{i=1}^{m} p_{i,j}$ (wired-OR).

v) $Q_j = \bigcup_{i=1}^{m} q_{i,j}$ (wired-OR),
$Q_j = 1$ means at least one enabled bit in the jth column is 0.

vi) $R_j = P_j \cdot \bar{Q}_j \cdot G$.

b) Is $R_j = 1$?

i) Yes—Default detected, proceed to step 3c).

ii) No—Default inhibited, proceed to step 4).

c) Default Phase: Compute $E_{i,j+1} = E_{i,j}$.

4) Is $j = n$?

a) Yes—Proceed to step 5).

b) No—Proceed to step 2).

5) Read out the words indicated by $w_{i,2} = 1$.

Example 2 shows an example search of Mode B operation on 5 words, each 10 bits long.

Example 2: "Mode B" Search Operation

a) WORDS in which the least value is to be retrieved:

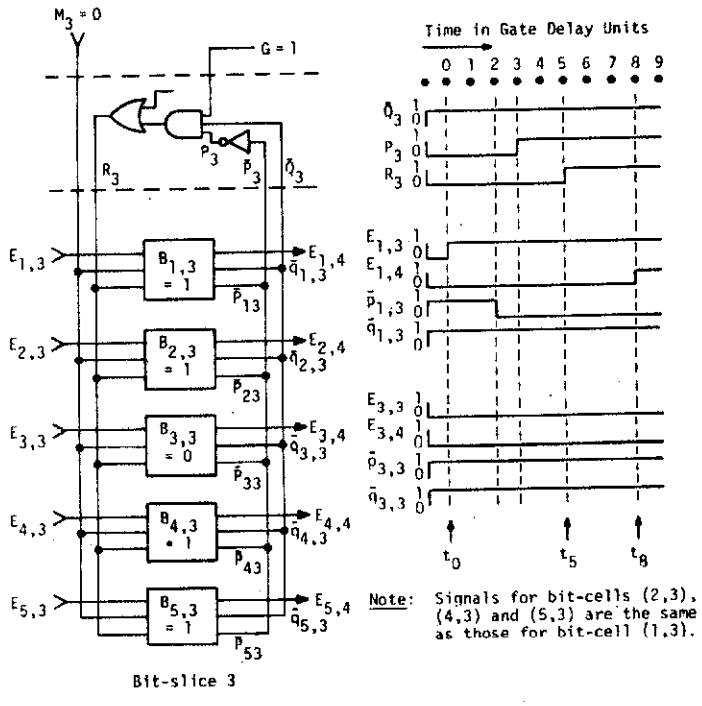
Word Number (i)	Bit Positions (j) 1	2	3	4	5	6	7	8	9	10	Order of Retrieval
1	0	0	1	1	0	0	1	0	0	1	3
2	0	0	1	0	1	0	0	1	0	1	1
3	1	0	0	1	1	0	0	0	0	1	5
4	0	0	1	0	1	0	1	0	1	1	2
5	0	0	1	1	0	0	1	0	1	1	4

b) STATES of all enable lines ($E_{i,j+1}$) at the end of the major cycle:

Word Number (i)	(j) 0	1	2	3	4	5	6	7	8	9	10
1	1	1	1	1	0	0	0	0	0	0	0
2	1	1	1	1	1	1	1	1	1	1	1
3	1	0	0	0	0	0	0	0	0	0	0
4	1	1	1	1	1	1	1	0	0	0	0
5	1	1	1	1	0	0	0	0	0	0	0

Note: Minor cycles 3, 5, 8, and 10 go through the default phase.

c) Timing diagram for bit-slice 3 in the major cycle:



Bit-slice 3

t_0: Starting of minor cycle for bit-slice 3;
t_5: Default condition detected;
t_8: End of minor cycle for bit-slice 3;
FROM t_0 TO t_5: Comparison Phase of Minor Cycle;
FROM t_5 TO t_8: Default Phase of Minor Cycle.

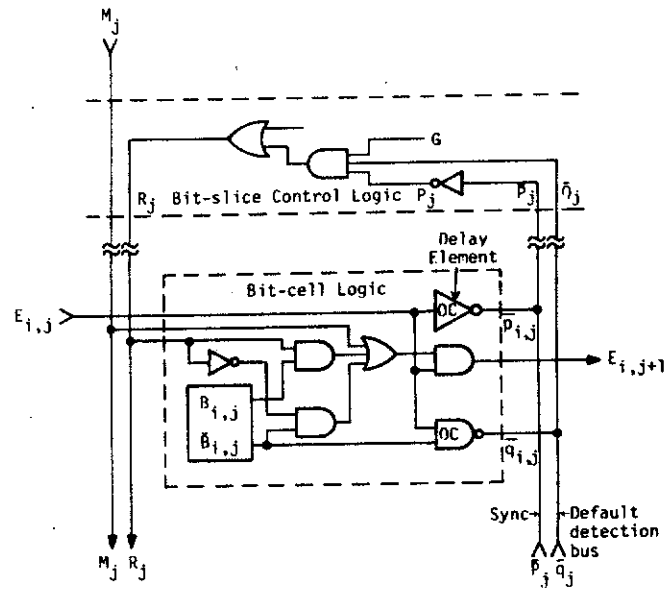
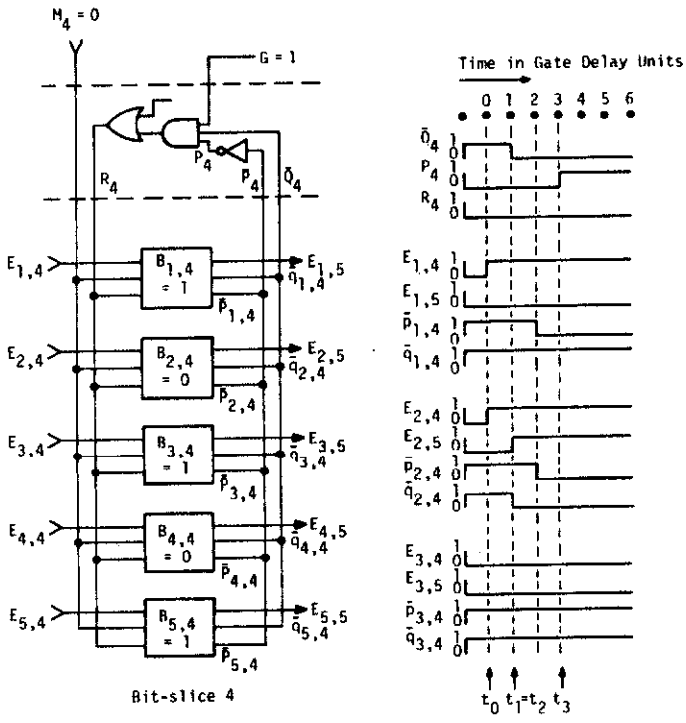


Fig. 3. Bit-cell with least value search logic for Mode B operation.

d) Timing diagram for bit-slice 4 in the major cycle:



Bit-slice 4

t_0: Starting of minor cycle for bit-slice 4;
t_1: End of minor cycle for bit-slice 4;
t_2: Default-inhibit signal becomes stable;
t_3: Bit-slice control logic in stable state.

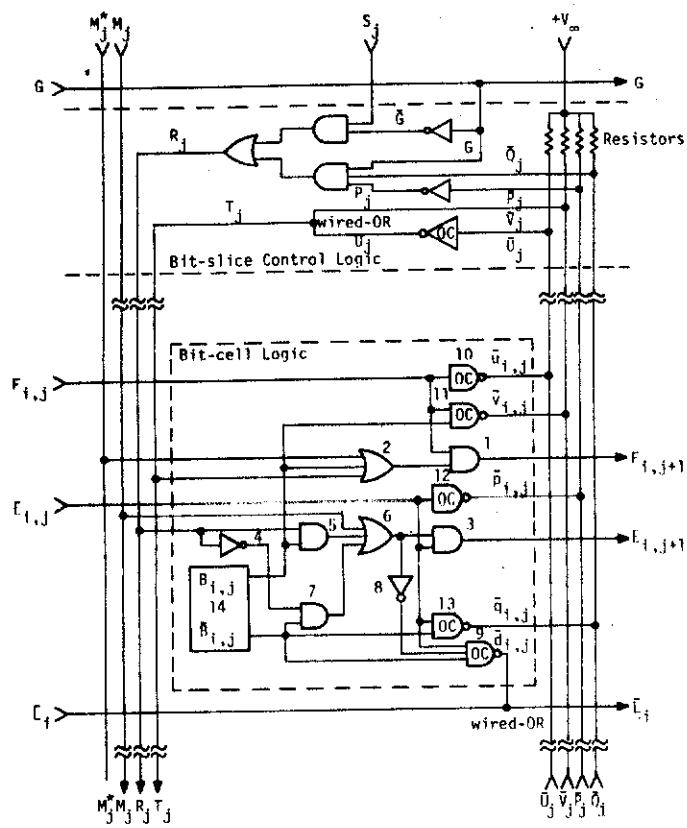


Fig. 4. Bit-cell for simultaneous ascending retrieval and descending retrieval or equality-threshold searches.

C. Mode C: Greatest Value Search Mode

In the implementation of the least value search scheme, the speed for searching is traded for less hardware in each bit-cell by sharing much of the logic with the equality–inequality search. Had it not been required for the latter search, the comparison time for the least value search could be shortened by looking only at the content of the bit-cell, and the default time could be shortened by looking at the feedback signal. Since the least value and the greatest value searches are analogous to each other, we shall demonstrate the speed-up design for the greatest value search. The implementation of the new design is illustrated in Fig. 4 which shows the complete design for each bit-cell. With this implementation, Mode C operations can be executed simultaneously with either Mode A or Mode B operations. Note that $T_j = 0$ at the beginning of the jth minor cycle.

Algorithm C—Mode C Search Operation—The Greatest Value Search Algorithm

1)

a) Initialization:

$$j = 0,\ w_{i,1} = 1,\ w_{i,4} = 0,\ \forall i \in \{1, 2, \cdots, m\}.$$

b) Data Path Setting:

Gate $F_{i,n+1}$ to $w_{i,4}$, and $w_{i,1}$ to $F_{i,1}, \forall i \in \{1, 2, \cdots, m\}$. The data paths are held until the completion of the major cycle.

2) Let $j \leftarrow j + 1$.

3) Minor Cycle:

a) Comparison Phase: Compute

i) $F_{i,j+1} = F_{i,j} \cdot (M_j^* + B_{i,j} + T_j)\ \forall i \in \{1, 2, \cdots, m\}$ simultaneously.

ii) $u_{i,j}(t) = F_{i,j}(t-1)$ (delay element used to synchronize the feedback of V_j via T_j) $\forall i \in \{1, 2, \cdots, m\}$ simultaneously.

iii) $v_{i,j} = F_{i,j} \cdot B_{i,j}\ \forall i \in \{1, 2, \cdots, m\}$ simultaneously. $v_{i,j} = 1$ means that $B_{i,j}$ is enabled and equals 1.

iv) $U_j = \bigcup_{i=1}^{m} u_{i,j}$ (wired-OR).

v) $V_j = \bigcup_{i=1}^{m} v_{i,j}$ (wired-OR). $V_j = 1$ means at least one enabled bit in the jth column is 1.

vi) $T_j = U_j \bar{V}_j$ (wired-AND).

b) Is $T_j = 1$?

i) Yes—Default detected, proceed to step 3c).

ii) No—Default inhibited, proceed to step 4).

c) Default Phase: Compute $F_{i,j+1} = F_{i,j}$.

4) Is $j = n$?

a) Yes—Proceed to step 5).

b) No—Proceed to step 2).

5) Read out the words indicated by $w_{i,4} = 1$.

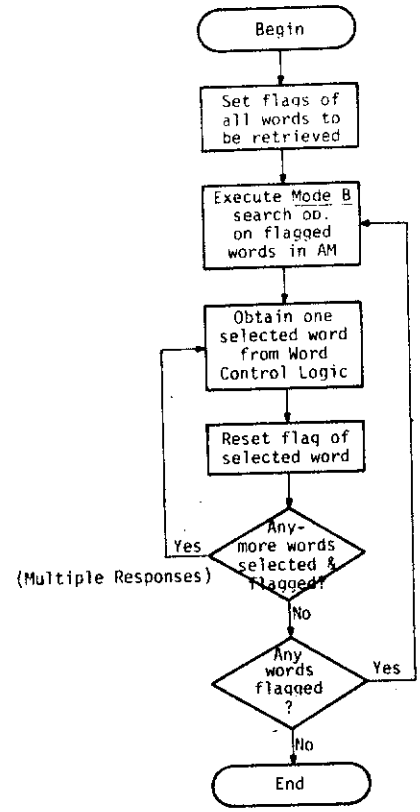


Fig. 5. Flow chart for ascending order retrieval.

V. Ordered Retrieval

A. Ascending Order Retrieval

The ascending order retrieval of a set of data can be achieved by performing the least value search repeatedly until all the data are retrieved. With the CAM organization that we have presented, a microprogram in the Memory Control Store provides an economical and efficient implementation of such a retrieval algorithm. A flow chart for such ascending order retrieval algorithm is shown in Fig. 5.

B. Descending Order Retrieval

The algorithm described for ascending order retrieval can be modified for descending order retrieval by substituting the greatest value search for the least value search. That is, Mode C search operation is executed in the CAM instead of Mode B search operation. Hence, the algorithm for descending order retrieval is to perform the greatest value search repeatedly until all the data are retrieved.

VI. Some Speed-Up Techniques

The design shown in this paper will have one gate delay per minor cycle in each of the Mode A search operations. The delay in Mode B operations ranges from 3 to 8 gate delays per minor cycle, while for Mode C operations, it ranges from 1 to 4 gate delays per minor cycle.[2] We now consider several techniques that can be used to reduce the search times. The four areas that bear investigation are lookahead techniques, external examination of retrieval process, implementation of additional basic operations, and modifications to the scheme involving greater parallelism in the search.

In the first area, lookahead logic can be added to each word in the memory. The algorithms we have described previously are all bit-serial and word-parallel in nature in which the enable signal for each word propagates from bit to bit and operations for each word are performed in parallel. The speed of a search operation is therefore proportional to n where n is the number of bits in each word. We can increase the speed by adding some lookahead logic to each word. Each word is segmented into contiguous groups of bits of equal size k, and a lookahead circuit is added to each group (assuming k is a factor of n). Each lookahead circuit operates

[2] In the case of the least value search, the maximum and the minimum delays are actually shorter. The output from gate 6 of each bit-slice can be assumed to be settled before the major cycle starts (see Fig. 4). This means that M_j lines are enabled ahead long enough for the outputs of gate 6 to settle. In this case, the minimum time to pass through each bit-slice is 1 gate delay. The maximum time to pass through each bit-slice is also shorter than 8 (the maximum gate delay count). When default occurs, $B_{i,j} = 1$ for all enabled words. Therefore, output from gate 7 is 0 and the feedback through R_j never has to go through gate 4. Hence, the maximum delay through a bit-slice is 7 gate delays.

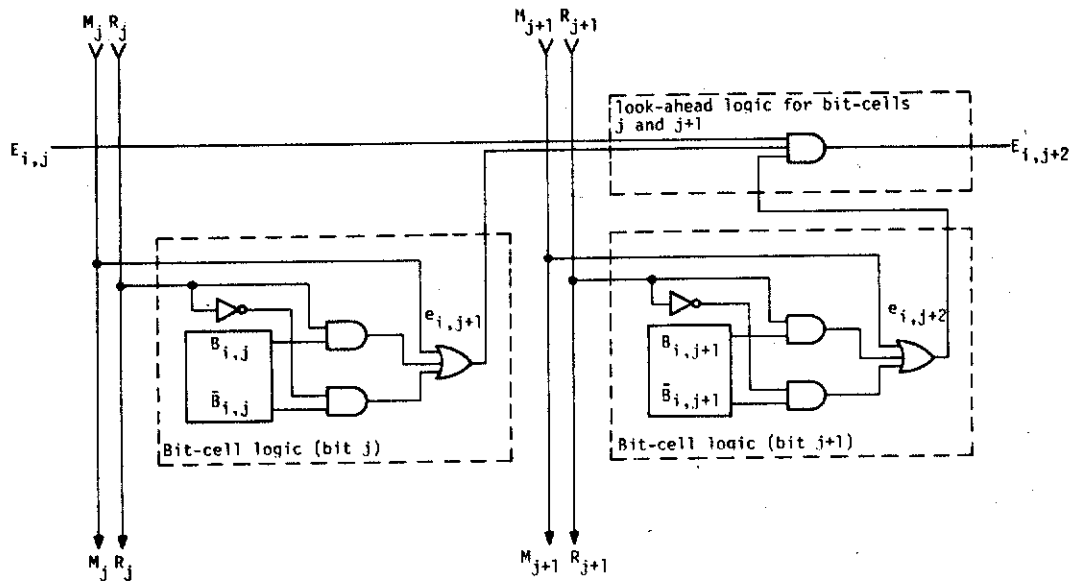


Fig. 6. Bit-cells j and $j+1$ of word i with equality search and lookahead logic.

on all the bits in its group in parallel and passes the result onto the next group when it has finished. The speed of an equality search operation using this lookahead circuit will be proportional to n/k but for extremum searches, no improvement is found. This type of lookahead is essentially single-leveled or cascaded. This means that the signals still have to propagate from group to group instead of from bit to bit, and the lookahead circuits exist in a single level above the storage circuits of each word. The cellular property of the design is preserved because each group, instead of a bit in a word, can now be regarded as a cell. We will not investigate other types of lookahead circuits, e.g., tree-lookahead circuits, because they do not preserve the cellular property. We now illustrate the construction of these lookahead circuits for the equality and the Mode B and Mode C searches.

An examination of the equality search operation shows that each of the $E_{i,j+1}$ signals propagates from bit slice j to bit slice $j+1$ in one gate delay where j ranges from 1 to n. Improvement can be achieved by grouping bits in each word and performing the comparisons in parallel. An example is shown in Fig. 6 where the necessary lookahead logic for grouping bits j and $j+1$ of word i is shown. Comparison in each bit is done in parallel. The results of comparison, $e_{i,j+1}$ and $e_{i,j+2}$, are ANDed together with $E_{i,j}$ to form $E_{i,j+2}$. The propagation time for these two bits is 1 gate delay instead of 2 in the usual bit serial operation. The speed of the equality search will therefore be proportional to $n/2$ gate delays.

For Mode B and Mode C operations, lookahead will require more hardware. The existence of default cases has caused the increased complexity. Previously, without lookahead, default is detected for a bit-slice when certain conditions exist on all enabled words in that bit-slice. These conditions include 1) all enabled words have 1's in this bit-slice for the least value search and 2) all enabled words have 0's in this bit slice for the greatest value search. The number of default feedback lines is 1 for each search mode. With added lookahead circuit to each word for a group of k bit-slices, the number of default feedback lines will be 2^k. These 2^k lines can be shared by both the least value search and the greatest value search. Consider a particular group; the following operations are to be carried out: a) The bits of each word in this group are decoded into 2^k lines. b) The corresponding lines from each word of this group are wired-ORed together to form default feedback lines 0 to $2^k - 1$; a particular feedback line p will be 1 when there exists an enabled word in this group whose decoded value equals p. c) In the group-slice control logic, if it is a Mode B operation, it will scan from feedback lines 0 to $2^k - 1$ until the first line with a 1 is found; similarly if it is a Mode C operation, it will scan from feedback lines $2^k - 1$ to 0; this line will represent the minimum/maximum of all these enabled words in this group. d) This line is encoded into k search bus signals to be fed back to each word in this group. e) In a particular word, the enabled line for the next group is enabled if the current group of this word is enabled and the value of this part of the word equals the search bus signal, i.e., it equals the minimum/maximum value found by the group slice control logic. However, there are some disadvantages of using lookahead on Mode B and Mode C operations. The extensive amount of decoding requires an order of 2^k gates for each group in each word, each with a fan-in of k. For each group-slice, there are 2^k default feedback buses running across all the words and this can cause difficulty in integrated circuit implementation. The biggest difficulty, however, lies in the implementation of the scanning algorithm in the group-slice control logic. The algorithm of scanning across a set of lines until the first 1 is found is essentially a multiple match resolution problem. If a tree-type multiple match resolution circuit is used, e.g., [1], a maximum delay of $\log_2 2^k = k$ will be observed. That is, the overall speed of a group of bit slices, with or without lookahead, is of the order of k. Unless a faster multiple match resolution circuit is used, and the cost of hardware is sufficiently low, lookahead for Mode B and Mode C searches is not beneficial.

An examination of the example illustrated in the previous section points out another possible source of improvement, this time in the algorithm itself. In many cases the number of words still enabled at the end of a minor cycle rapidly drops to one within a few minor cycles. At this point the completion of the major cycle is a formality since the greatest (or the least) valued word must be the only remaining enabled word. Unfortunately the detection of this condition, the only-one-respondant-left condition, is too complex to be performed at the end of every minor cycle, and would require extensive external wiring and logic.

We have implemented some of the search operations defined in Section III as basic operations. Some other useful searches may be performed by combining two or more basic searches and possibly some nonsearch operations. An example is the between-the-limits searches, which is generated by performing a less-than search followed by a greater-than search on words selected by the first search. In fact, all the searches described in Section III, except for the proximity search, can be performed as a basic search or as a combination of basic searches designed in this paper. Speed improvements can of course be gained by implementing these search operations as basic searches, but the amount of logic circuits may be extensive. In most other cases, the more complicated searches, such as the case of ordered retrieval, are implemented as a combination of simple searches.

One modification to our ordered retrieval technique that yield positive results without compromising our cellular logic approach is to increase the parallelism of the algorithm itself. This can be done by simultaneously performing the greatest value search and the least value search on the same set of enabled words. The associative sort is complete when both searches select the same word, an easily detectable condition, or when no words are still enabled at the beginning of a major cycle, also an easily detectable condition. A small additional amount of external manipulation of the sorted file block is required by the nonassociative processor controlling the sort to concatenate the two halves of the sorted block since one will be in the reverse of the desired order, but it is felt that this is a small price to pay for a speed-up factor of greater than 2. This technique is shown in Example 3 that follows.

The speed-up involved in this approach is greater than a factor of 2. To understand why it is greater than a factor of 2 instead of exactly equal to 2, we must consider the properties of the fields to be searched. Assuming an even distribution, there is on the average one more bit with the value "1" in the higher valued half of a sorted file than in the lower valued half of the same file. This can be verified in Example 3a). The greatest value search has a shorter minor cycle time for bit positions with a value of "1" in the word with the greatest value than for bit positions with a "0" in the word with the greatest value. Likewise, the least value search has a shorter minor cycle time for bit positions with a value of "0" in the word with the least value than for bit positions with a "1" in the word with the least value. This provides for an average major cycle time five gate delays shorter than if all words were to be selected in an ordered retrieval by either search alone (assuming the delay for each minor cycle of both Mode B and Mode C search operations ranges from 3 to 7 gate delays). The design for this technique has been indicated in Fig. 4.

Example 3: "Mode B" and "Mode C" Parallel Operation.

a) WORDS to be retrieved:

Word Number	Bit Positions 1	2	3	4	5	6	7	8	9	10	A = Number of 1's per Word	Ascending Order of Retrieval
1	0	0	1	0	1	0	1	1	0	1	5	12
2	1	1	0	0	1	0	0	1	0	0	4	20
3	0	0	0	0	1	1	1	1	0	1	5	6
4	1	1	1	1	1	1	0	0	0	0	6	30
5	0	0	1	0	0	0	0	0	0	0	1	11
6	1	1	1	0	0	0	0	1	0	0	4	23
7	0	0	0	0	0	0	0	0	0	0	0	1
8	0	1	0	1	1	1	1	1	0	1	7	14
9	1	1	0	1	0	1	0	0	1	0	5	21
10	1	1	1	1	0	0	1	1	0	0	6	28
11	1	1	0	1	1	1	1	1	1	1	9	22
12	0	0	0	1	1	1	1	0	1	1	6	10
13	0	0	0	0	0	1	1	1	1	1	5	4
14	0	1	1	0	1	0	1	1	1	1	7	15
15	1	1	1	1	1	1	0	1	1	0	8	31
16	1	1	1	0	1	0	0	0	0	0	4	26
17	1	1	1	1	1	0	0	0	0	0	5	29
18	1	1	1	0	0	0	1	0	0	1	5	24
19	0	0	0	0	0	0	1	0	0	1	2	2
20	1	0	1	0	0	0	0	0	1	0	3	19
21	0	0	1	1	0	1	1	0	1	1	6	13
22	0	0	0	1	0	1	1	1	1	1	6	7
23	1	1	1	1	1	1	1	1	1	1	10	32
24	0	0	0	0	0	0	1	1	1	1	4	33
25	0	1	1	1	1	1	0	1	1	0	7	16
26	0	0	0	1	1	0	1	0	1	1	5	8
27	1	1	1	0	0	1	0	1	0	0	5	25
28	0	0	0	1	1	1	0	1	1	0	5	9

Example 3a (continued):

Word Number	Bit Positions 1	2	3	4	5	6	7	8	9	10	A = Number of 1's per Word	Ascending Order of Retrieval
29	1	1	1	1	0	0	0	0	1	0	5	27
30	1	0	0	1	0	1	0	0	0	0	3	18
31	0	0	0	0	1	1	0	0	1	1	4	5
32	1	0	0	0	0	0	1	0	0	1	3	17

Number of 1's in memory: 160

Number of bits in memory: 320

Number of 1's per word in the smaller half of the ordered list = 4.69.

Number of 1's per word in the larger half of the ordered list = 5.31.

b) ORDER of retrieval in parallel operation:

Let L_B and L_C be the lists of words retrieved by Mode B and Mode C search operations, respectively. Both lists are ordered with respect to time, in Gate Delay Units, at which they are retrieved, and neglecting overhead time between major cycles. Assume that for the least value search, the gate delays for each minor cycle range from 1 to 7 (see footnote 2) and that for the greatest value search, they range from 1 to 4.

L_B	Time[a]	L_C	L_B	Time[a]	L_C	L_B	Time[a]	L_C
Start	0	Start	—	173	27	5	368	—
7	10	23	3	180	—	—	385	32
—	26	15	—	198	18	—	404	25
19	32	—	22	226	6	1	408	—
—	48	4	—	239	11	—	423	14
24	66	—	—	264	9	—	442	8
—	73	17	26	266	—	21	454	—
—	95	10	—	292	2	End of Retrieval		
13	106	—	28	306	—			
—	120	29	—	323	20			
31	140	—	12	352	—			
—	148	16	—	354	30			

[a] Time in Gate Delay Units.

Throughput = 454 gate delays/(32 × 10) bits = 1.42 gate delays/bit.

VIII. Issues and Limitations

We have presented a design of an associative memory that can be used for fast ordered retrieval. From Example 3, neglecting the overhead in loading and unloading the memory, the sorting speed is 1.42 gate delays per bit. This design is therefore very attractive and can be used in many places where fast searching and sorting is required. However, there exists many issues that need to be carefully considered and resolved before successful operations can result. We discuss five of these issues here, namely, LSI implementation, manufacturing defects, modular expansion, multiple match resolution and extension to sequential associative memories. We do not contend that they exhaust all the issues in this design. New issues may come up during the implementation phase and will have to be resolved by the designer.

A. LSI Implementation

In Fig. 4, a complete design has been shown. Each bit cell requires 13 gates. There will be extra logic associated with the registers and the controls. Consider a 32-bit word and a 64-word memory. This design will need over 26 000 gates for the logic in the bit-cells only, excluding all other register and control logic. Therefore, the memory size that can be effectively implemented on an LSI chip is very limited. Furthermore, the pins on the LSI package will also limit the word size. In order to maintain fast response and high throughput, parallel reading and writing of bits of a word in the memory is necessary. The major portion of the pins of an LSI package is usually taken up by those used for parallel reading and writing. For a 32-bit word memory, the pin requirement is 32 plus a few for control and selection. On the other hand, the pin limitation will put a maximum word size that can be implemented. It becomes obvious that modular expansion is necessary in order for this design to be practical. The issue of modular expansion will be discussed later.

B. Manufacturing Defects

After the LSI chip has been manufactured, tests are made to determine whether any cells are faulty. A faulty cell can be determined by injecting certain test patterns into the memory. If the number of defects are small and their locations can be determined up to the locality of certain gates in the cell, then these faults can be bypassed by utilizing some spare bit-slices designed into the memory. The difficulty in recovering an error in a faulty cell of the CAM is that the error may not only affect the word itself, but it may also affect other words because the value of the faulty bit is available to other words via the feedback circuitry. Therefore, it may be necessary to remove the current bit-slice or the current and all bit-slices to the right from operation when an error occurs in a cell. We have assumed that only stuck-at

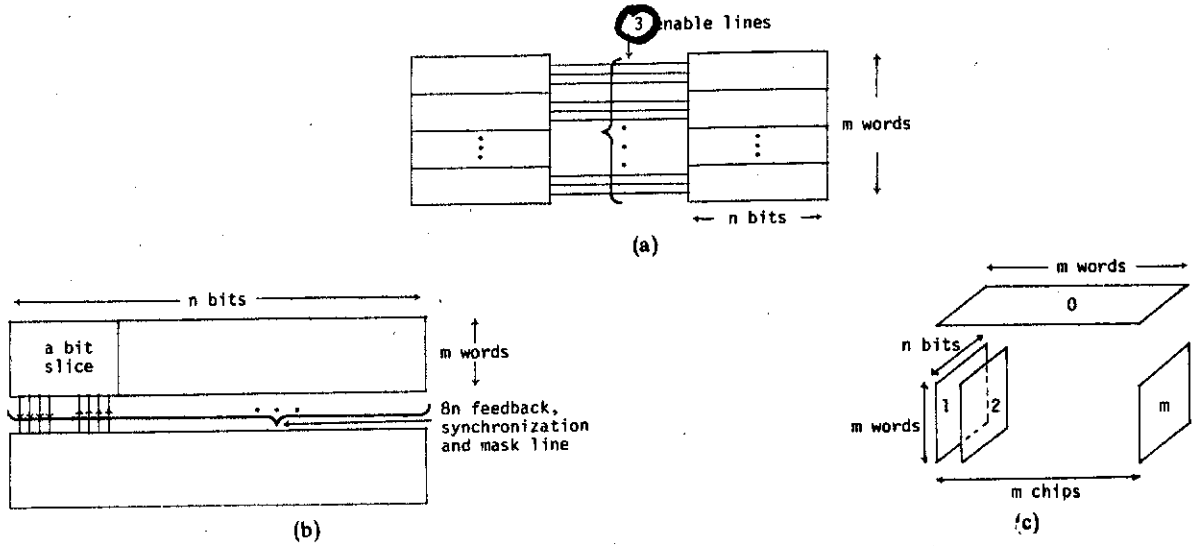


Fig. 7. (a) Word size extension. (b) Memory size extension. (c) Three-dimensional associative memory for memory size extension.

faults can occur in the gates of memory cells and bit-slice control logic. Faults occurring in registers and control store are not considered since the logic there is only a small fraction of all the logic on the chip. By assuming that the jth bit of the ith word is faulty, we can identify two types of faults, one in which the jth bit-slice has to be removed from operation and one in which all the remaining bit-slices are rendered useless. Referring to Fig. 4, for faults that occur in gates 10–13 and the bit-slice control logic, they only affect the feedback values but they do not affect the enable lines so long as the mask bits are 1, that is, the bit-slice is masked off. This can be done by setting a 1 permanently in the jth bit of the mask registers and shifting the external pin connection to the chip by 1 bit. For faults that occur in gates 1–9 and the storage cell 14, they affect the enable lines for the next bit-slice. If an enable line has a faulty value of 1, that is, the remaining bits of this word are enabled regardless of whether the current word or bit-slice are masked off, it may cause a faulty feedback to other bit-slices on the right. So unless all the remaining bit-slices are masked off, the fault that occurs in cell (i, j) will propagate to these bit-slices. If an enable line stucks at 0, it will not affect the remaining bit-slices, and only the ith word needs to be disabled from operation. A finer recovery procedure can be developed if we can identify which of the enable lines are faulty.

From the above discussion, we see that recovery from manufacturing defects are not easy and only a small fraction of the faults are recoverable.

C. Modular Expansion

Our philosophy of the design in this paper is that we want to distribute the logic into the storage cells. In order for all the distributed logic to perform coherently, extra communication lines are needed to transfer enable and feedback signals from bit to bit. The number of these communication lines are usually large and this will eliminate the possibility of modular expansion which is easy in the case of RAM's. Consider our design in Fig. 4; each cell has 3 enable lines to communicate with the cell on its right; each bit-slice has 8 lines which are used for feedback, synchronization and mask. These lines run across all words in the bit-slice (these exclude lines needed to read and write data into each bit). Suppose a memory chip of m words by n bits is available. To extend the word size of this memory, we can put 2 memory chips together side by side as shown in Fig. 7(a). However, our design will need $3m$ enable lines to pass enable signals from the chip on the left to the one on the right. This will not be feasible even for a small m. To extend the memory size, we can put 2 chips one over the other as shown in Fig. 7(b). This design will need $8n$ feedback lines to pass the feedback, synchronization and mask signals between the two chips. Even for a small value of n, the number of interconnections is very large. The requirement for a large number of interconnections in modular expansion has forced us to put as much logic as possible on one chip. Due to the limitation of technology, the memory size that can be implemented on a chip is very small; therefore other schemes for modular expansion are necessary in order for our design to be practical. In Fig. 7(c), we show a scheme that allows us to extend the memory size by increasing the dimensions of the memory. A batch of m memory chips are put together in parallel. There will be an extra dimension and it is composed of a single memory chip running across all the m parallel chips. A flow chart for an ascending order retrieval algorithm of m^2 words is shown in Fig. 8. The time needed to orderly retrieve m^2 words will take $m^2 + m$ units of load time (time to store a word into the memory) and $m^2 + m$ units of search time (a search time includes the time to execute a Mode C operation and to read it out into the I/O register). In a single memory chip which can accomodate m^2 words, the time needed for this memory system will be m^2 units of load time and m^2 units of search time. Therefore the degradation in performance is minimal when m is large. For a memory size larger than m^2 words, extra dimensions are needed.

We conclude that our scheme on memory size expansion has minimal degradation on performance. The difficulty

TABLE I
A COMPARISON TABLE FOR ORDERED RETRIEVAL SCHEMES

Scheme	Speed - R(n,k) (Cycles per Retrieval of an n Bit Tag)	Memory Size Dependency	Relative Complexity of Hardware Required	Best Class of Problem
Frei and Goldberg	For n = 5 Best Case ($k = 2^5$): R = 2k Worst Case (k = 1): R = 7k	Length of tag field (n) only	Basic CAM	High density of responders
Seeber and Lindquist	$t = 2^n$ $R(n,k) = \frac{1}{k}((t+k-1) + \frac{t(t-1)^k}{t^k} - \frac{2t}{t^k}\sum_{i=1}^{n}(k(t-2^{i-1})^{k-1} + 2^{-i}(t-2^{i-1})^k))$	Length of tag field (n) only	Complex cryogenic logic at each bit (18 gates)	Density dependent
Lewin	$\frac{2k-1}{k}$ Exact	Independent	A registers plus 9 gates per bit slice	Independent
Miller	Best case: $\frac{k+1}{k}$ Worst case: $\frac{2k-1}{k}$	Independent	Basic CAM plus some additional control and storage	Multiple response resolution only
Foster	1 cycle per retrieval	External logic uses $3(2^{m-1})-1$ gates for 2^m words of memory	Tree Circuit external to CAM	Multiple response resolution only
Proposed	1/2 cycle per retrieval	Increases as* $\lceil \log_2 m \rceil$ for m words of memory	~13 gates per bit cell	Independent

*$\lceil \log_2 m \rceil$ is the size of a tag that must be used to uniquely identify each word for a memory size of m. This differs from the other schemes which do not use a specialized tag for ordered retrieval.

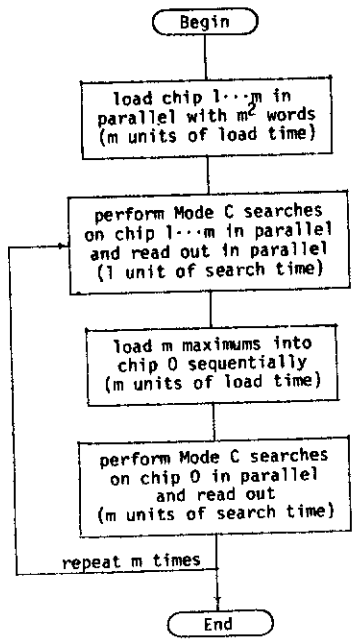


Fig. 8. Flow chart for ascending order retrieval of m^2 words in a three-dimensional associative memory [see Fig. 7(c)].

still exists in word size expansion. The limitation is due to the pin requirements. However, we can trade performance for word size by loading bits of a word in groups instead of all in parallel. However, the degradation in performance due to this loading scheme is more pronounced than our memory size expansion scheme.

D. Multiple Match Resolution

The most useful application in our design is in the multiple response resolution. A tag field can be included in each word. Each tag is a distinguishable number. The size of each tag must be at least $\lceil \log_2 m \rceil$ for a memory size m. When there are multiple responses, each of the tags serves as a number for the ordered retrieval scheme. The words used in the ordered retrieval are those that respond. Only the bit-slices containing the tag are used in the search. The first cycle can retrieve 2 words, the one with the maximum tag, and the one with the minimum tag. Subsequent searches give 2 responses each time. The speed of this resolution scheme is 1/2 memory cycle per word and is independent of the memory size. A comparison is made between our proposed scheme and the other published schemes in Table I.

There are two disadvantages in using tags for multiple match resolution. First, there are irregularities in implementation. Because each tag has a distinguishable value and if each tag is hardwired into the memory, it will involve a different design for each word and it will also be difficult to overcome the problem of manufacturing defects when a cell in the tag is bad. This problem can be solved by loading the

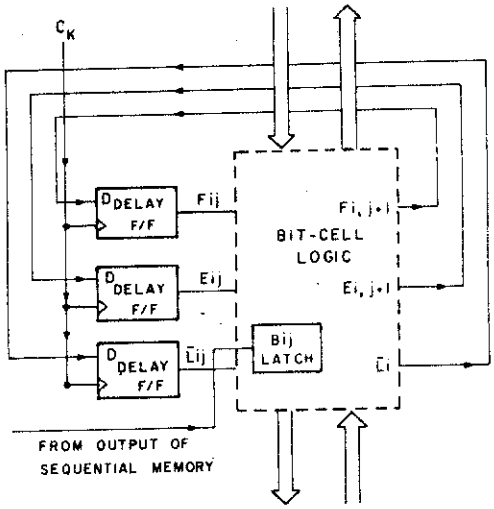


Fig. 9. Associative sequential memory.

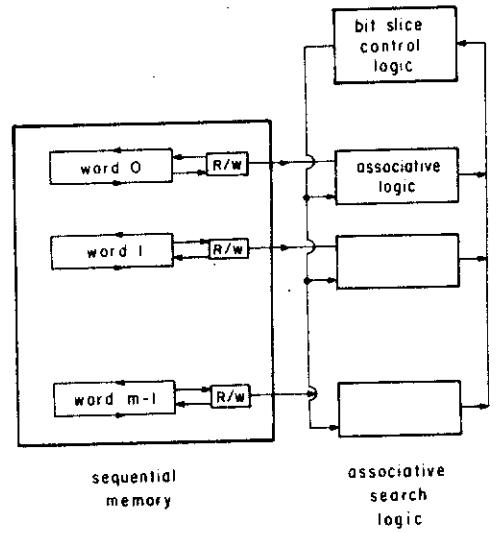


Fig. 10. Associative logic for associative sequential memory.

tags from a PROM when the memory is first used. Second, when a cell in the tag becomes bad during operation, e.g., stuck at 0, there is a probability that two of the words in the memory will have identical tags.

We can also perform the multiple match resolution without using special fields as tags. This can be done by treating the contents of each word or part of the word as a tag itself. It requires all words under consideration in the memory to be different in order for unique responses to result.

E. Extension to Sequential Memories

Our design presented in this paper can be extended to the design of associative sequential memories which is made up of multiple loops of circulating bits shifting in synchronism. There is a read/write head for each loop so that one bit from each loop can be read or modified in one clock period. This can be extended to include multiple heads for each loop. Examples of such sequential memories include charge-coupled device memory, bubble memory, and fixed head disk.

Since only one bit is available from each loop at any time, we can design the associative logic outside the sequential memory as shown in Fig. 9. In this design, m words are stored in the memory, with one word occupying each loop. During a clock period, a bit-slice of these m words is shifted out from the memory. This bit-slice is then processed by the associative logic and the enable signals are stored in temporary flip flops. Note that in the design presented earlier, the enable signals propagate from the MSB to the LSB and the data are stored in flip-flops. In the case of a sequential memory, the enable signals can be stored in temporary flip-flops. As the bit-slice is shifted out, MSB first, the bit-slice, together with the stored enable signals, generate a new set of enable signals which are stored back into the temporary flip-flops. The exact design is shown in Fig. 10.

There are two advantages in this design. First, the additional logic for each word is very small and therefore the cost increase is minimal. Second, when the memory size is extended, only 8 lines due to the associative logic are needed to be connected between adjacent modules. Therefore, the memory size can be modularly expanded. Moreover, the amount of bit-slice control logic is small, so we can design a memory with n modules, each with its own associative and bit-slice control logic. During normal operations, each module can perform independent associative search operations. When it is necessary to perform associative search operations on 2 or more adjacent modules, all except one of the bit-slice control logic for these modules are switched out of the system and the feedback lines are connected together to form a large block of associative memory. This dynamic reconfiguration capability is useful in applications where the nature and size requirements may change dynamically. However, there are two limitations with this design. First, the words must be organized as described here because our design can only process one bit-slice at a time. Second, it is limited to memory types in which these logic can be easily implemented in LSI technology, e.g., CCD memory. In bubble memory, the associative logic have to be implemented on a separate chip and the amount of interconnections between the memory and the associative logic may become prohibitively large.

IX. Comparisons with Other Methods of Ordered Retrieval

We have presented in this paper several of the search schemes, namely, the equality search, the inequality searches, the threshold searches, and the extremum searches. The other searches defined in Section III, except for proximity searches which we have not implemented, can be implemented as a combination of basic searches. Using the implementation in this paper, we computed the maximum and the minimum of the search time in each of the searches.

Search Type	Minimum Number of Gate Delays	Maximum Number of Gate Delays
Equality Search	$n+5$	$n+5$
Inequality Search	7	$n+5$
Similarity Search	$n+5$	$n+5$
Greater-than Search	$n+5$	$n+5$
Less-than Search	7	$n+5$
Greater-than-or-equal-to Search	$n+5$	$n+5$
Less-than-or-equal-to Search	$n+5$	$n+5$
Double-limit Search		
Between-limit Search, $X > Y$		
$<X \,\&\, >Y$	$n+12$	$2n+10$
$<X \,\&\, \geq Y$	$n+12$	$2n+10$
$\leq X \,\&\, >Y$	$2n+10$	$2n+10$
$\leq X \,\&\, \geq Y$	$2n+10$	$2n+10$
Outside-limit Search, $X < Y$		
$<X \,\&\, >Y$	7	$2n+10$
$<X \,\&\, \geq Y$	7	$2n+10$
$\leq X \,\&\, >Y$	$n+5$	$2n+10$
$\leq X \,\&\, \geq Y$	$n+5$	$2n+10$
Extremum Search		
1) Least-value Search	n	$7n$
2) Greatest-value Search	n	$4n$
Adjacency Search		
1) Nearest-above Search	$2n+5$	$8n+5$
2) Nearest-below Search	$n+7$	$5n+5$

We see that the delay time in all these searches is proportional to n, the number of bits in a word and is independent of the number of words in the memory.

Several methods of ordered retrieval and multiple response resolution have been proposed in the past. It would be of great value to evaluate the method of ordered retrieval presented in this paper in terms of these other schemes. In particular, we compare this new algorithm with those of Frei and Goldberg [4], Seeber and Lindquist [13], Lewin [10], Miiller [11], and Foster [2]. In order to evaluate these various schemes, it is necessary to determine the significant characteristics that we wish to examine and to determine the comparable features of these diverse methods.

In order to facilitate these comparisons, the methods mentioned will be classified into two types, those with an algorithm to order the retrieval according to the contents of the stored words and those which use an external priority scheme, usually some form of priority tree, to order the retrieval according to the physical location in memory. Among schemes of the first type are those of Frei and Goldberg, Seeber and Lindquist, Miiller and Lewin. Miiller's scheme uses the contents of the responding words to resolve multiple response conflicts but it does not necessarily order the selections in ascending or descending order.

Among those schemes that use an external priority circuit to resolve conflicts are those of Weinstein [16] and Foster. These schemes are not strictly comparable to the proposed algorithm since they cannot be used for sorting. Likewise the Miiller scheme is not absolutely comparable to our proposed scheme but is similar enough that we will include it in the comparison.

The two main considerations for comparison are obviously the speed with which a method retrieves stored data and the cost in terms of amount of logic required. Rather than attempting an exhaustive analysis of the implementation cost for each of the various schemes, we shall look at the more readily available information as to the rate of cost increase for increasing memory size. In particular we are interested in the memory cost as a function of memory size.

We shall limit our discussion of speed comparisons to the number of search cycles required to retrieve each stored word. For several of the schemes under consideration, a significant parameter is the density of flagged words, that is, the ratio of the number of words to be retrieved to the number of words addressable with the given tag field size. We will assume that the number of words addressable by the tag field is the same as the length of the memory.

The chart of Table I shows as direct a comparison as possible between the aforementioned searches and the search scheme proposed. The headings include relative speed (in terms of number of cycles needed to retrieve each flagged word), comments upon dependencies of logic complexities to memory size, relative complexities of the hardware needed for implementation, and comments upon class of problems handled. Fig. 11 shows a plot of words to be retrieved for a memory with a five bit tag field in each word, corresponding to a memory size of 32 words.

It can be seen that our proposed scheme is equal to or better than all of the presented schemes in terms of speed, and in terms of the number of cycles needed to retrieve a word from memory. In terms of the absolute speed, the Foster method is somewhat faster in terms of gate delays per retrieval since it used an external priority logic tree. The Foster scheme, however, is not useful as a tool for ordered retrieval, but only for multiple response resolution. At two retrievals per memory cycle, our proposed scheme is by far the fastest ordered retrieval scheme, even faster than the Miiller scheme which does not even produce ordering, only resolution. As far as the complexity of the hardware goes, our scheme is well within the realizable realm of LSI technology and in fact is no more complex than that used by Seeber and Lindquist or than that used by Yang and Yau [17] in their implementation of Lewin's algorithm.

X. Conclusion

The concept and the design of this scheme was investigated by the authors several years ago [15]. This paper contains the basic principles and the subsequent extensions and applications of this scheme. Among the different search operations we have presented, which include the equality–inequality searches, the threshold searches and the extremum searches, the extremum searches have the greatest improvement over the previous schemes in the literature. They are useful in many applications, particularly in the resolution of multiple responses. We have also indicated many issues in the design and implementation of this memory and some techniques to speed up the processing.

We conclude that such a scheme as we have presented may be a useful and realizable tool for associative processing in any application where ordered retrieval is important. One of the applications is to use it as a multiple match resolver as what we have described in Section VIII. There are many other applications which vary from priority interrupt processing to standard file processing and are too numerous to be mentioned here.

References

[1] T. Feng, "Data manipulating functions in parallel processor and their implementations," *IEEE Trans. Comput.*, vol. C-23, pp. 309–318, Mar. 1974.

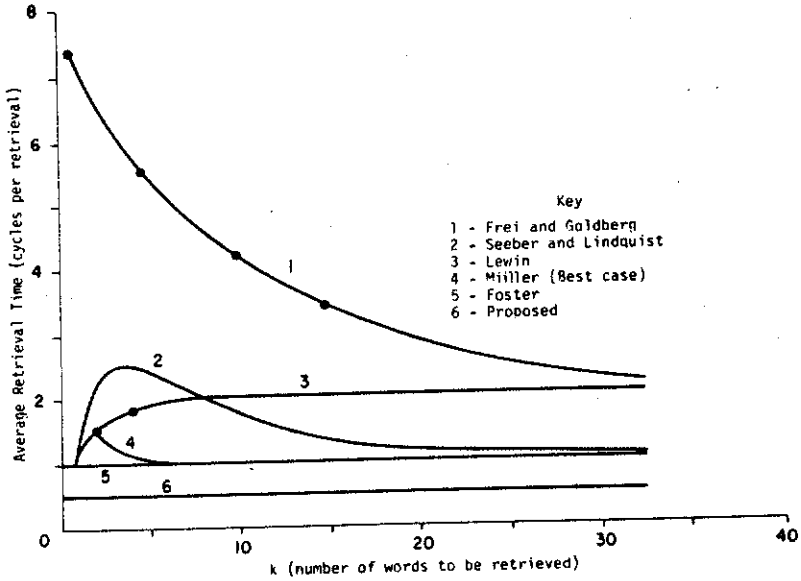


Fig. 11. Comparison of retrieval speeds for a 5 bit tag field with k words flagged.

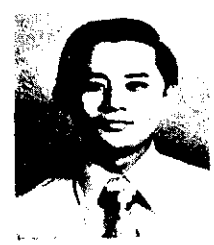
[2] C. C. Foster, "Determination of priority in associative memories," *IEEE Trans. Electron. Comput.*, vol. EC-17, pp. 788–789, Aug. 1968.
[3] ——, *Content Addressable Parallel Processors.* New York: Van Nostrand Reinhold, 1976.
[4] E. H. Frei and J. Goldberg, "A method for resolving multiple responses in a parallel search file," *IRE Trans. Electron. Comput.*, vol. EC-10, p. 718, Dec. 1961.
[5] A. G. Hanlon, "Content-addressable and associative memory systems: A survey," *IEEE Trans. Electron. Comput.*, vol. EC-15, pp. 509–521, Aug. 1966.
[6] W. Hilberg, "Simultaneous multiple response in associative memories and readout of the detector matrix," *IEEE Trans. Electron. Comput.*, vol. EC-15, pp. 117–118, Feb. 1966.
[7] W. H. Kautz, "Cellular logic-in memory arrays," *IEEE Trans. Electron. Comput.*, vol. EC-18, pp. 719–727, Aug. 1969.
[8] ——, "An augmented context-addressed memory array for implementation with large-scale integration," *J. Assoc. Comput. Mach.*, vol. 18, pp. 19–33, Jan. 1971.
[9] D. Landis, "Multiple-response resolution in associative systems," *IEEE Trans. Comput.*, vol. C-26, pp. 230–235, Mar. 1977.
[10] M. H. Lewin, "Retrieval of ordered lists from a content-addressed memory," *RCA Rev.*, vol. 23, pp. 215–229, June 1962.
[11] H. S. Miller, "Resolving multiple responses in an associative memory," *IEEE Trans. Electron. Comput.* (Short Notes), vol. EC-13, pp. 614–616, Oct. 1964.
[12] B. Parhami, "Associative memories and processors: An overview and selected bibliography," *Proc. IEEE*, vol. 61, pp. 722–730, June 1973.
[13] R. R. Seeber and A. B. Lindquist, "Associative memory with ordered retrieval," *IBM J. Res. Develop.*, vol. 6, p. 126, Jan. 1962.
[14] A. E. Slade and H. O. McMahon, "A cryotron catalog memory system," *Proc. Eastern Joint Comput. Conf.*, Dec. 1956, pp. 115–119.
[15] J. L. Turner, "A design for a fast sorting associative memory," Master of Science Thesis, University of Texas at Austin, Aug. 1972.
[16] H. Weinstein, "Proposals for ordered sequential detection of simultaneous multiple responses," *IEEE Trans. Electron. Comput.* (Corresp.), vol. EC-12, pp. 564–567, Oct. 1963.
[17] C. C. Yang and S. S. Yau, "Cutpoint cellular associative memory," *IEEE Trans. Electron. Comput.*, vol. EC-15, pp. 522–528, Aug. 1966.

C. V. Ramamoorthy (M'57–F'78) received the undergraduate degrees in physics and technology from the University of Madras, India, the M.S. degree and the professional degree of Mechanical Engineer, both from the University of California, Berkeley, and the M.A. and Ph.D. degrees in applied mathematics and computer theory from Harvard University, Cambridge, MA.

He was associated with Honeywell's Electronic Data Processing Division from 1956 to 1971, last as Senior Staff Scientist. He was a Professor in the Department of Electrical Engineering and Computer Sciences at the University of Texas, Austin. He is currently a Professor in the Department of Electrical Engineering and Computer Sciences, University of California, Berkeley. He was Chairman of the Education Committee of the IEEE Computer Society and Chairman of the Committee to develop E.C.P.D. Accreditation for Computer Science and Engineering Degree Programs. He is currently the Chairman of the AFIPS Education Committee and a member of the Science and Technology Advisory Group of the U.S. Air Force.

James L. Turner was born in Fort Worth, TX on February 10, 1947. He received the B.S. degree in engineering physics and the B.S. degree in electrical engineering from the University of Oklahoma, Norman in 1970 and the M.S. degree in electrical engineering from the University of Texas, Austin in 1972.

During his graduate study at Austin he was a Research Associate with the University of Texas Electronic Research Center in the area of Associative Processing. He has since been involved in work on software reliability and is currently Manager of Software Applications for the Systems Division of National Semiconductor, Santa Clara, CA.



Benjamin W. Wah (S'73) was born in Hong Kong on September 7, 1952. He received the B.S. and M.S. degrees in electrical engineering and computer science from the Columbia University, New York, NY, in 1974 and 1975, respectively, and the M.S. degree in computer science from the University of California, Berkeley, in 1976.

He is currently a Research Assistant in the Electronics Research Laboratory, University of California, Berkeley, where he is working toward the Ph.D. degree in computer science. His current research interests include memory architecture, distributed systems and computer system performance evaluation.