R. R. Seeber
A. B. Lindquist

Associative Memory with Ordered Retrieval

Abstract: A basic associative memory utilizing cryogenic circuitry is described and its functions are compared with those of previously published associative memory descriptions. The ordered-retrieval sorting algorithm is described, along with its implementation by means of a ternary interrogating counter. A sorting example is given. The sorting efficiency is discussed and an efficiency formula is given. The required additions to the basic memory are outlined. Finally, some of the basic cryotron circuits are illustrated and their operation described.

Introduction

The sorting of data for processing is a recurring problem of major proportions, particularly in business data processing. The concept of an associative memory somewhat reduces the need for sorting. For example, when a binary search is employed in a table look-up operation, the data must be in sorted order so that the search may be made; with an associative memory we may have direct access to items in a table by association without regard to the order in which they are stored, thus eliminating the need for sorting. But there are still many places where sorting is necessary, particularly in preparing data for output. The associative memory system proposed here applies to any sorting operation that involves a high-speed internal sort. An implementation is shown using a cryogenic memory, but the principles can be applied also to other technologies.

Earlier papers[1-4] on associative memory have dealt with storage and retrieval of information and, with one exception, have not dealt with the sorting problem. The "Associative Self-Sorting Memory"[5] permitted data entering memory to be sorted in order during the entering operation. However, this operation required substantial additional hardware beyond that required for a conventional associative memory in that double registers were provided for buffering of data and transfer circuits were provided between each pair of registers. In the present proposal, Associative Memory with Ordered Retrieval, in addition to the circuits of a conventional associative memory there is required only one relatively simple circuit extending throughout memory plus additional logic in the control circuits.

An efficient algorithm enables the locating of the lowest-ranking word in memory, where the words may be in any random order and the sorting field may be selected at the time of sorting.

With an associative memory, one may retrieve randomly stored words in order by using a counter to establish the interrogating tag. For example, if 90 employees have employee numbers consisting of two decimal digits and we wish to retrieve their records in employee-number order, we can start by retrieving employee number 00, then 01, 02, . . . , 99. On ten of these retrieval tries, no retrieval will be made since only 90 of the codes have been used. But by advancing a counter a unit at a time, we have, in this case, a highly efficient ordered retrieval, since for 100 retrieval tries, we have succeeded in 90 cases. But if our sorting key was such that we had a non-dense set to be sorted, our efficiency would be quite different. For example, a ten-digit part number might be used where there were only 1000 part numbers. Employing the above method would require 10^{10} retrieval tries for 10^3 successful retrievals.

For non-dense keys we need a more efficient algorithm which will permit us to find the lowest-ranking key without examining all possible combinations. That is what the algorithm provides. Consider how the human brain approaches the problem of sorting the following four keys:

2 6 3 4 5
4 6 9 2 1
8 6 4 7 1
4 7 5 2 7

By looking only at the first digit we could complete our sort except for the second and fourth keys, which both begin with 4. In these two cases we need look at only one more digit to determine their correct order. Thus the sort can be completed without any reference to the right-hand digits. The following shows this separation into the digits used for the sort, to the left of the line, and those not used.

$$\begin{array}{l} \overline{2}\,|\,6\ 3\ 4\ 5 \\ 4\ \boxed{6}\,9\ 2\ 1 \\ 8\,\boxed{6}\,4\ 7\ 1 \\ \underline{4\ 7}\,|\,5\ 2\ 7 \end{array}$$

The ordered-retrieval algorithm operates on a similar basis, working on the binary bits of the key. An interrogation starts with the leftmost bit, associatively separating the words in memory into those that match and those that do not match an interrogating zero in that single bit position. If there is more than one match, we must analyze the matching words further to the right to the point where there is usually but a single match. Having retrieved the lowest-ranking word, we can then proceed to the next higher word in much the same fashion. This will be more fully explained in the section on automatic ordered retrieval after a review of cryotron associative memories.

This paper, then, gives an algorithm for use with an associative memory. With this algorithm a set of randomly ordered words is retrieved in order by some non-preassigned key. This scheme is in contrast to the operation of table look-up, where usually the words are in order and we desire to retrieve only one of them.

Cryotron associative memories

Before we consider the automatic ordered retrieval properties, it is desirable to discuss the salient features of the associative portion and to indicate how they differ from other such memories. Figure 1 shows a simplified block diagram of a cryogenic implementation of the associative portion of the memory with ordered retrieval.

The memory cycle consists of two phases established by the memory controls. During the first phase the contents of the unmasked positions of the interrogation register condition the comparing circuits to their corresponding positions in the memory. For the positions that were masked, a mask signal is sent through the memory. A mask signal forces that position in memory to indicate an equal condition.

In the memory, a comparing current is established on an equal line going into the high-order position of each word. As long as the memory positions are equal to their corresponding positions in the interrogation register or are masked, the comparing current will remain on the equal line. If at any position there is an unequal condition, the comparing current will be diverted to the unequal line. At the end of each word register in memory, an indicator bit is set to "1" by current on the equal line, indicating an equal word register, or to a "0" by current on the unequal line. The indicator bit is part of the register select controls and is referred to as the *equal bit*. An *equal word* is defined as having its tag equal to the interrogating tag.

During the second phase of the memory cycle, the equal-word register is read out of or written into, depending upon the operation desired. These operations are performed by the memory control, register select control, and the READ-WRITE circuits. Details of some of these circuits will be discussed in the section on circuitry.

In this memory system provision is made for storage of data into the first vacant word register. A vacancy bit in each register is used to indicate whether that register is occupied or vacant. A "0" or "1" in the vacancy bit may respectively represent a vacant or occupied word register. The vacancy bit is designed so that it is possible to write into it during a READ operation as well as during a WRITE operation. Thus writing a "0" into the vacancy bit of a word register while reading that register results in a destructive READ operation. Notice in Fig. 1 that in order to accomplish the WRITE WHILE READ operation, the READ line connects to the WRITE line before going into the vacancy bit.

Provision is also made in this memory for handling multiple-matched words. When a plurality of matched words results, the first matched word register is read and marked. By definition a matched word is an unmarked equal word. The matched and marked word registers are identified by match and mark bits respectively. These bits, like the equal bit, are part of the register select controls. Therefore, during the next memory cycle, using the same tag, the next matched word is read and marked. This time the matched word is the second equal word. After all matched words have been read and marked, the mark bits are reset for all equal words.

In summary we would like to point out some of the systems differences between this associative memory and the others discussed in the literature.[1-3] First of all, this associative memory uses a parallel-by-bit, parallel-by-word approach which differs from the serial-by-bit, parallel-by-word method of Petersen[3]; the parallel-by-bit search of the present system results in a more efficient sort than the serial-by-bit approach. Second, with regard to sensing, indicator bits on each word indicate as to whether that word was a match or no-match during an interrogation; these indicator bits differ from the YES, NO answer of Slade's catalog memory.[1,2] Finally, the readout of this associative memory differs from both those of Petersen and Slade. In our ordered-retrieval system the direct operation of the output circuits results in a parallel readout. Petersen accomplishes the readout through an encoding from the detector plane which, by decoding, activates the X and Y READ drivers resulting in a parallel destructive or nondestructive READ. Slade's readout is a serial-by-bit system.

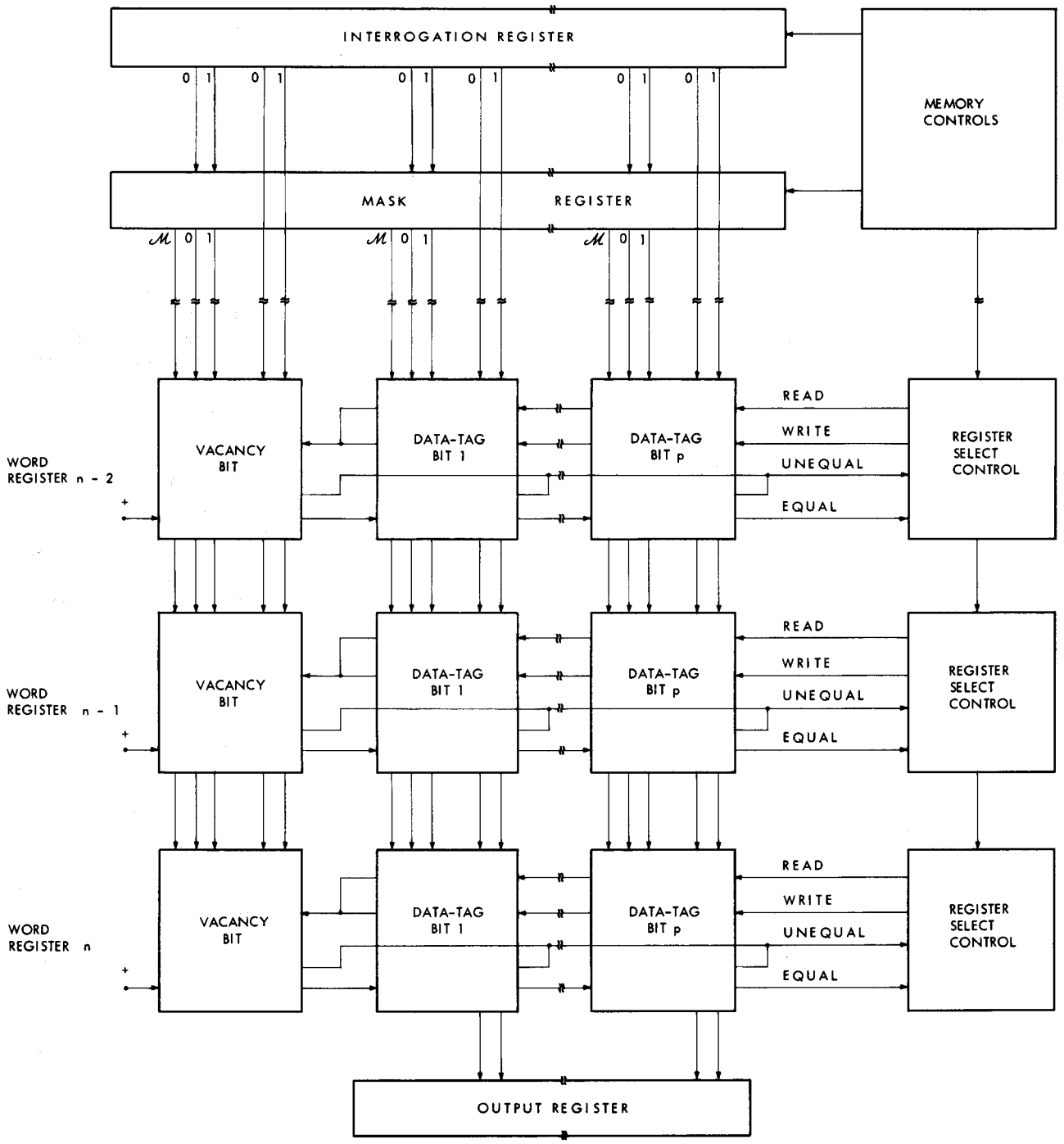
Automatic ordered retrieval

A. *Basic algorithm*

The object of the sorting mechanism is to generate an interrogating tag which will result in an ordered retrieval of the desired data. This mechanism should accomplish this object in a minimum number of memory interrogations. The mechanism capable of doing this is similar to a ternary counter, although logic operating on a mask register (see References 3 and 5) would also suffice.

The three states of each ternary counter position are designated as $\mathscr{M}$, "0", and "1" representing mask, binary "0" and binary "1". When a counter position is in the $\mathscr{M}$ state, matching of the word is not required in that bit position. When a counter position is in the

Figure 1 **Block diagram of an associative memory.**



"0" state, a binary "0" is searched for in that position and similarly for a binary "1".

Except at the start of the sort operation, the rules for the operation of the ternary counter are a function of the match indicator. This indicator tells how many registers in memory match the setting of the ternary counter, but only in abbreviated form as indicated below.

Match Indicator: 0 = no matches
1 = one match
P = more than one match

At the start of the sort operation, all the positions of the ternary counter are reset to the $\mathcal{M}$ state and a fixed tag identifying the data to be sorted is placed in the interrogation register. The first interrogation of the memory, using the all-$\mathcal{M}$ state of the ternary counter and the fixed tag in the interrogation register, determines whether there are any words to be sorted; then the highest ordered position of the sorting field in the ternary counter is located and advanced to the next state. The cycle of a ternary counter position is from $\mathcal{M}$, to "0", to "1" and then back to the $\mathcal{M}$ state. The rightmost position of the ternary counter is defined as the lowest-ordered position of the sorting field, and thereafter, each position to the left is considered the next higher ordered position. Thus, in the memory, if a field to be sorted consists of ten positions, the rightmost ten positions of the ternary counter will be used as the sort-interrogating tag.

After each of the remaining memory interrogations, the match indicator (MI) is consulted to determine the next state of the ternary counter. Before stating the rules for establishing the next state of the ternary counter, it is desirable to mention two definitions. First, the lowest-ordered position of the ternary counter whose state is either a "0", or a "1" will be defined as the operating position (OP). Second, the position immediately to the right (that is, the next lowest order) of the OP will be defined as the next position (NP). The rules that are now stated tell not only how to advance the ternary counter, but they also tell when to select a matched word from memory so that it will be retrieved in the desired order.

Rule 1 When the $MI = P$, if the OP is the lowest-order position of the ternary counter, indicating multiple like sorting keys, select the first matched register and mark the selected register; otherwise do not select. Also when the $MI = P$, the NP is to be advanced in preparation for the succeeding interrogation. It should be observed that when the OP is the lowest-order position of the ternary counter, the state of the ternary counter does not change until all multiple like sorting keys have been selected, at which time the mark bits are reset.

Rule 2 When the $MI = 1$, select the matched word and advance the OP.

Rule 3 When the $MI = 0$, do not select, but advance the OP.

In addition to the above ordered-retrieval rules, when a ternary counter position is advanced from the "1" state to the $\mathcal{M}$ state, a carry is produced and directed into the next higher ordered position. A carry generated in the highest-order position of the sorting field indicates the end of the sort operation.

Table 1 shows the detailed steps involved in sorting a six-bit key. Only six bits of the ternary counter and the sorting key of the memory register data words are shown. Since all data words are to be sorted, no fixed tag is necessary.

The interrogations are numbered 1 through 15, and the states of the ternary counter and match indicator are shown for each interrogation. In addition the rule is shown that is applied to obtain the state of the ternary counter for the next interrogation. There is also shown, to the right of each sorting key, a number which indicates the interrogation when this key, along with its data, was retrieved.

Note that at the start of the sort operation, the ternary counter is set to the all-$\mathcal{M}$ state. Then, after the first interrogation, the high-order bit position of the sorting field in the ternary counter, bit position 1 in this example, is advanced to a "0". Thereafter the match indicator dictates how the ternary counter is to be advanced. For example, after the seventh interrogation using a tag of $001\mathcal{M}\mathcal{M}\mathcal{M}$, the $MI = 0$, which according to Rule 3 tells us to advance the OP. The OP, as defined for interrogation 7, is bit position 3. Since bit position 3 is in the "1" state, it is advanced to the $\mathcal{M}$ state which results in a carry into bit position 2. Bit position 2 is thus advanced from the "0" state to the "1" state. As a result, the new tag for the next interrogation, interrogation 8, is $01\mathcal{M}\mathcal{M}\mathcal{M}\mathcal{M}$.

In summary, we first see if there are any words to be sorted. This is the interrogation with the ternary counter set to the all-$\mathcal{M}$ state; if MI is "0", no sort need be done; if MI is "1", that one word is the only word to be "sorted" and it is immediately retrieved; but if MI is a plurality, P, we must proceed to a binary subdivision in our interrogation by interrogating on "0" in the highest-order bit in seeking for the lowest-ranking word. Each setting of the ternary counter (except the final all-$\mathcal{M}$ setting) constitutes at least one interrogation, that is, retrieval try; if there are n multiple like tags, n interrogations or tries (all successful) would be made with the OP in the lowest-order counter position without changing the setting of the ternary counter. When the last word has been found, the algorithm usually requires additional retrieval tries to prove that there are no more words in the sort. Some of these steps could be avoided if we knew in advance how many words were in our sort, but we consider the more general case where the size of the group is unknown, counting these retrieval tries as part of the cost of the sort.

Table 1 **Example of an ordered-retrieval sort.**

Interrogation	State of the Ternary Counter Bit Positions 1	2	3	4	5	6	State of MI	Rule Applied
1	$\mathcal{M}$	$\mathcal{M}$	$\mathcal{M}$	$\mathcal{M}$	$\mathcal{M}$	$\mathcal{M}$	*P*	*Start of Sort*
2	0	$\mathcal{M}$	$\mathcal{M}$	$\mathcal{M}$	$\mathcal{M}$	$\mathcal{M}$	*P*	1
3	0	0	$\mathcal{M}$	$\mathcal{M}$	$\mathcal{M}$	$\mathcal{M}$	*P*	1
4	0	0	0	$\mathcal{M}$	$\mathcal{M}$	$\mathcal{M}$	*P*	1
5	0	0	0	0	$\mathcal{M}$	$\mathcal{M}$	1	2
6	0	0	0	1	$\mathcal{M}$	$\mathcal{M}$	1	2
7	0	0	1	$\mathcal{M}$	$\mathcal{M}$	$\mathcal{M}$	0	3
8	0	1	$\mathcal{M}$	$\mathcal{M}$	$\mathcal{M}$	$\mathcal{M}$	1	2
9	1	$\mathcal{M}$	$\mathcal{M}$	$\mathcal{M}$	$\mathcal{M}$	$\mathcal{M}$	*P*	1
10	1	0	$\mathcal{M}$	$\mathcal{M}$	$\mathcal{M}$	$\mathcal{M}$	*P*	1
11	1	0	0	$\mathcal{M}$	$\mathcal{M}$	$\mathcal{M}$	0	3
12	1	0	1	$\mathcal{M}$	$\mathcal{M}$	$\mathcal{M}$	*P*	1
13	1	0	1	0	$\mathcal{M}$	$\mathcal{M}$	1	2
14	1	0	1	1	$\mathcal{M}$	$\mathcal{M}$	1	2
15	1	1	$\mathcal{M}$	$\mathcal{M}$	$\mathcal{M}$	$\mathcal{M}$	1	2
	← $\mathcal{M}$	$\mathcal{M}$	$\mathcal{M}$	$\mathcal{M}$	$\mathcal{M}$	$\mathcal{M}$		*End of Sort*

Register No.	Register Bit Positions 1	2	3	4	5	6	Interrogation When Retrieved
1	1	0	1	1	0	1	14
2	1	1	0	1	0	1	15
3	0	0	0	1	0	1	6
4	0	1	1	0	0	1	8
5	1	0	1	0	1	1	13
6	0	0	0	0	1	0	5

In the example, the two lowest-order positions of the sorting field did not enter into the sort operation; it is the generalization of this aspect that gives the algorithm its efficiency. Fifteen retrieval tries were required to retrieve the six words, an average of two-and-one-half retrieval tries per successful retrieval. This ratio of two-and-one-half falls within the range of two to three found by experiment on small random-sample sorts that have been tried. However, "worst case" situations can be constructed in which this ratio of retrieval tries to successful retrievals is somewhat greater than the number of bits in the sorting key.

By counting all the retrieval tries required for retrieving all possible sets of sorting keys, a mathematical formula* has been derived for this ratio, $R(b, w)$, of the number of retrieval tries to the number of successful retrievals as a function of the number of bits, b, in the sorting key and the number of words, w, to be retrieved.

We further distinguish two cases:

Case I Equally weighted: where each set of sorting keys is counted only once.

Case II Randomly weighted: where each set of sorting keys is counted with the appropriate frequency on the assumption that they occur as random combinations with the "0" and "1" equally likely in any given position of the key.

Under these assumptions we have:

$$R(b, w) = \frac{1}{w}\left\{(t + w - 1) + \frac{t(t-1)^{[w]}}{t^{[w]}} - \frac{2t}{t^{[w]}}\sum_{i=1}^{b}\left(w(t - 2^i)^{[w-1]} + 2^{-i}(t - 2^i)^{[w]}\right)\right\},$$

where $t = 2^b$ and the square brackets for exponents have different meanings for the two cases, viz.:

Case I Equally weighted: ascending factorial; i.e., $x^{[w]} = x(x + 1)(x + 2) \ldots (x + w - 1)$.

Case II Randomly weighted: simple power; i.e., $x^{[w]} = x^w$.

Table 2 and Table 3 show the evaluation of this

* The derivation of this formula is available from the authors on request.

formula for a selection of values of b and w for Case *I* and Case *II*, respectively. Note that the limiting values are the same for the two cases.

B. *Modifications to the algorithm*

The circuit descriptions will be given for the ordered-retrieval algorithm as described above. However, there exist modifications of that basic algorithm which are applicable to different hardware configurations. We now briefly describe some of these modifications.

By adding some logic to the ternary counter, we may save an occasional interrogation by noting that if the match indicator gives a "0" indication on the interrogation following a *P* indication, we may simultaneously advance both the *OP* and the *NP*, thus saving one interrogation. Also, if we have a more elaborate match indicator circuit which gives the indications, "0", "1", "2", *P* where *P* is now greater than two, then we can save an occasional interrogation by designing the ternary counter to "remember" the setting when the first of a series of *MI*'s equal to "2" was indicated. Then when a retrieval is made, the second of the two matches can be retrieved from that remembered setting, thus eliminating going through some of the intermediate steps of the normal sequence of the algorithm when retrieving the next following, i.e., third, word. Incorporating these two modifications into our system would yield the improved ratio $R^*(b, w)$, as indicated in Table 4 for the equally weighted case only.

Where the serial-by-bit interrogation is available, as in Petersen,[3] the modification becomes much less efficient, since after each successful retrieval or indication of a "0" by the *MI*, the entire serial interrogation must be started over from the highest-order bit position, because the mismatch indicator bits must be reset. However, the algorithm does assure that the interrogation proceed to the right only so far as is necessary for each interrogation sequence and that all words be retrieved.

In the case of the parallel interrogation with the YES, NO response as in Slade,[1,2] the algorithm modification provides a means of reading out all words (in order) including the case of multiple like tags. Here the sort must proceed to the lowest-order bit in each case but need not return to the highest-order bit after a successful retrieval.

Although the parallel-by-bit, parallel-by-word approach of our proposed system leads to a more efficient use of the ordered-retrieval algorithm, it

Table 2 **R(b, w) equally weighted.**

w \ b	1	2	3	4	5	6	7 ...	∞
1	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
2	1.8333	2.3000	2.5000	2.5588	2.5606	2.5462	2.5310	2.5000
3	1.5000	2.0000	2.3556	2.5359	2.5989	2.6070	2.5968	2.5556
4	1.3500	1.8071						2.6310
5	1.2667	1.6714						2.6838
6	1.2143	1.5714						2.7188
7	1.1786	1.4952						2.7430
8	1.1528	1.4356						2.7607
⋮								
∞	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	

Table 3 **R(b, w) randomly weighted.**

w \ b	1	2	3	4	5	6	7 ...	∞
1	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
2	1.7500	2.1250	2.3125	2.4063	2.4531	2.4766	2.4883	2.5000
3	1.4167	1.8958	2.2031	2.3737	2.4632	2.5090	2.5322	2.5556
4	1.2813	1.7539						2.6310
5	1.2125	1.6398						2.6838
6	1.1719	1.5457						2.7188
7	1.1451	1.4691						2.7430
8	1.1260	1.4075						2.7607
⋮								
∞	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	

Table 4 **R*(b, w) equally weighted.**

w \ b	1	2	3	4	5	6	7	...	∞
1	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000		1.0000
2	1.8333	2.2000	2.2679	2.2942	2.2878	2.2769	2.2675		2.2500
3	1.5000	1.9333	2.1333	2.2190	2.2425	2.2414	2.2323		2.2083
4	1.3500	1.7642							2.2478
5	1.2667	1.6428							2.2881
6	1.2143	1.5517							2.3192
7	1.1786	1.4810							2.3421
8	1.1528	1.4250							2.3596
⋮									
∞	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000		

should be noted that the approaches of Petersen and of Slade require a technology much more nearly available than the technology which we require.

C. *Additional functional features*

In data processing type problems, the data record is usually defined by more than one field. For example, a payroll record might be defined by specifying an employee number, name, age, department, rate of pay, et cetera. In certain applications it might be desirable to retrieve these payroll records in ascending order by employee number. In another application the order of retrieval might be by department and by man number in ascending order. Still another job might call for the records to be retrieved in descending order by age and in ascending order by department and man number.

To provide this flexibility, a character-select matrix has been designed. This matrix directs, by character, the outputs from the ternary counter into the desired character positions in the memory. This feature thus allows, in memory, minor sorting keys to be placed physically to the left of major keys and allows any key to appear in nonconsecutive character positions. In addition, the matrix will furnish the complement output of any character position in the ternary counter that is desired. The complement of $\mathscr{M}$, "0" and "1" is defined to be $\mathscr{M}$, "1" and "0" respectively. The only change that is required in order to perform a descending sort on a desired field is to complement the output of that field as it comes from the ternary counter. Thus this feature provides for ascending and descending ordering or any combination thereof of the desired sorting fields.

Another feature which is very powerful and yet very simple to achieve in the AMOR system is the next higher or next lower value retrieval feature. By providing circuits for storing a number into the ternary counter, the AMOR system can start sorting from this number in either an ascending or descending order. It should be noted here that in order to perform a descending sort, starting from a given number, that number must be stored in the ternary counter in its complement form, i.e., $\mathscr{M} \rightarrow \mathscr{M}$, "0"→"1", and "1"→"0".

D. *Memory organization*

The shaded blocks in Fig. 2 represent the hardware units that have been added to the associative memory previously shown in Fig. 1. Although most of these hardware units are not necessary to implement the ordered-retrieval algorithm, they do provide some desirable features. For example, the ternary counter and the match indicator provide for automatic ordered retrieval—automatic in the sense that the data is retrieved and the sort-interrogate tag is generated without stored-program intervention. The character-select matrix allows the output of each character position of the ternary counter to be connected to any character position in the memory. The matrix also provides for the complement output of each character position. The sort-select register determines whether the interrogating tag is to come from the interrogation register or the ternary counter.

Circuits

A. *Basic devices*

The crossed thin-film cryotron[6,7] is used in the design of this memory system. A symbol for the cryotron is shown in Fig. 3a. The cryotrons are designed so that a full current flowing through the control path will make the gate path resistive. For example, refer to Fig. 3b, which shows a two-state storage device and its associated input and output circuits. The storage device consists of a current source *1*, and a current sink *2*, and two paths between them. Current flowing through the right path represents a "0", whereas current flowing through the left path represents a "1" in storage.

To store a "0", current is put on the S_0 line and no current on the S_1 line. This makes cryotron *5* resistive and leaves cryotron *6* superconductive. Thus the current from source *1* is directed through the right path to current sink *2*. Once the current is established in the

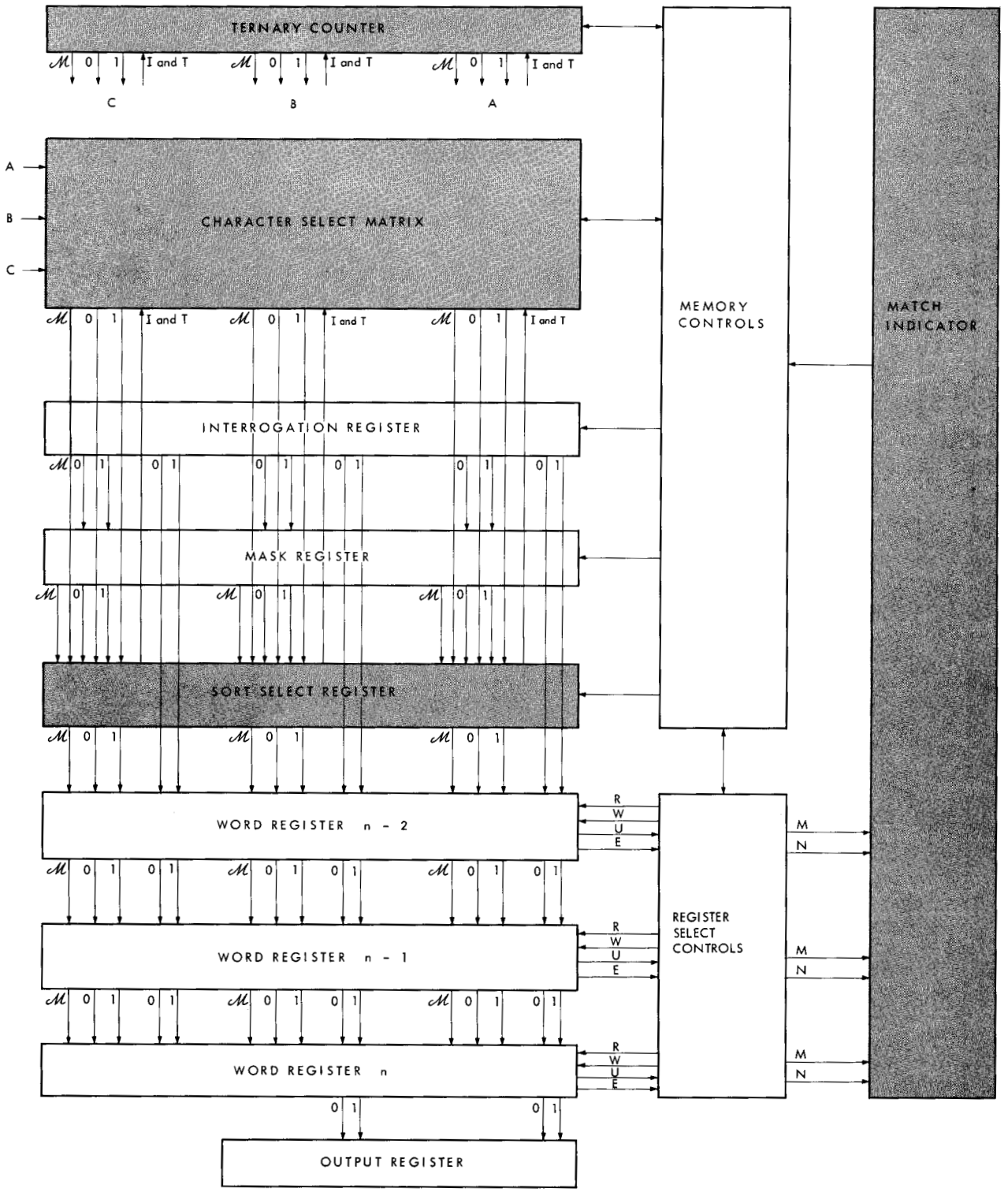


Figure 2 **Block diagram of associative memory with ordered retrieval.**

right path, the current on line S_0 may be released. However, the current on the right path will continue to flow, and no current will flow on the left path even though it is superconductive.

To sense the state of the storage device, current is applied to node *4*. With current on the right path of the storage device, cryotron *8* would be resistive and cryotron *7* superconductive. Thus the current from node *4* will be directed through path W_0, indicating a "0" stored.

Adding another path to the storage device makes it a three-state storage device. The input and output circuits would use the same principles as the two-state device.

B. *Associative memory bit*

Shown in Fig. 4 is the associative memory bit used in the present system. Notice that the bit is a two-state storage device with READ, WRITE and COMPARE circuits. The vertical interrogate lines labeled $\mathcal{M}$, 0, 1 carry the interrogating currents from the interrogation and mask registers. The currents on the compare lines are controlled by the currents on both the vertical interrogate lines and the lines of the storage devices. For example, assume there is current flowing down the "0" interrogate line and there is current on the "0" line of the storage bit. Thus with current coming in on the equal compare line from the next higher-ordered bit, the current would be directed through the lower path of the compare paths to the equal line going into the next lower-ordered bit position. Had the storage bit been set to a "1", the current on the equal line would have been directed through the upper path to the unequal line. Once the compare current flows in the unequal line it is directed into the register select control.

The other two vertical lines are the input, output lines. For example, to store a "1", current is put on the "1" input line and on the WRITE line. The current on the WRITE line is then directed through the upper path of the WRITE circuit which sets the storage bit to a "1". To read the contents of the storage bit, current is put on the READ line. With a "1" stored in the storage bit, the current on the READ line will be directed through the upper path of the READ circuit. The current on the upper path of the READ circuit makes the "0" output line resistive and leaves the "1" output line superconductive. Thus when a current source is connected to the output lines, the current will be directed down the "1" output line indicating a "1" stored in that bit position of the selected word register.

C. *Ternary counter*

Figure 5 shows a position of the ternary counter. Note that it contains two three-state storage circuits. During the first phase of the memory cycle the interrogate and transfer circuit sets the upper storage circuit equal to the lower storage circuit. In addition, at this time, the output of the lower storage circuit is sent through the character-select matrix to the desired character position in the memory.

During the second phase, the state of the upper storage device is tested by current on either the advance *NP* or advance *OP* line to determine if it is the *OP* position. Recall that the operating position (*OP*) is defined as the lowest-ordered position whose state is either a "0" or a "1" and that the next position (*NP*) is defined as the position immediately to the right of the *OP*. Notice that if the upper storage device is not a "0" or "1", then the current on either the advance *NP* or advance *OP* line will be directed to the next higher ordered position. If this storage device is in

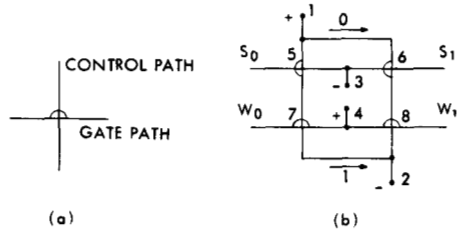


Figure 3 **Two-state storage device.**

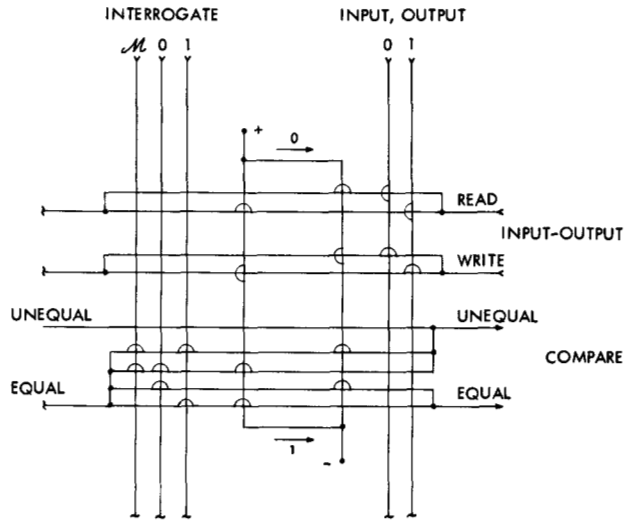


Figure 4 **Associative memory bit.**

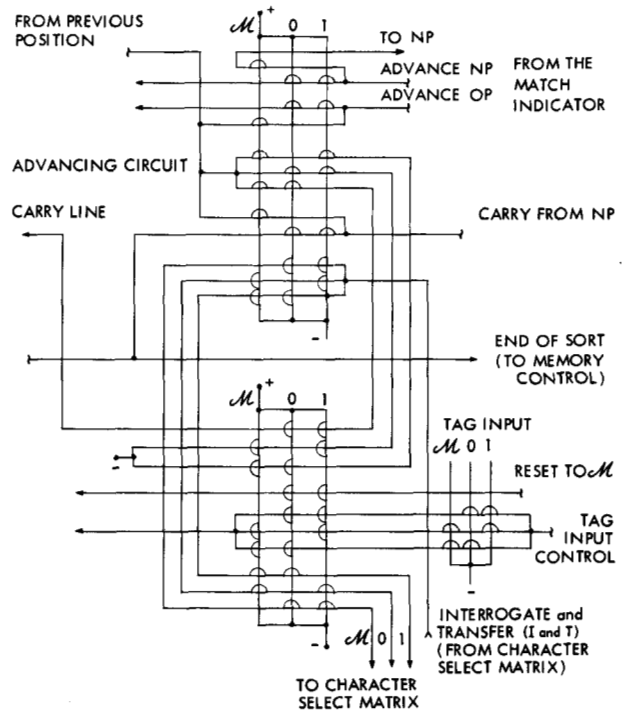


Figure 5 **A ternary counter position.**

the "0" or "1" state, then the current on either the advance *NP* or advance *OP* line will be directed to the advancing circuit of either the *NP* or the *OP*, respectively.

The advancing circuit advances the state of the lower storage device one unit, i.e., from $\mathscr{M}$ to "0", "0" to "1" or "1" to $\mathscr{M}$. In addition, if the lower storage device is being advanced from the "1" to the $\mathscr{M}$ state, current will appear on the carry line going into the next higher ordered position. As can be seen from Fig. 5, any current on the carry line will be directed either to the advancing circuit or to the end of sort line, depending upon the state of the upper storage device.

The reset to $\mathscr{M}$ and tag input circuits are controlled by the memory controls. These circuits are used to establish the state of the ternary counter at the start of a sort operation.

D. *Match indicator*

The match indicator which determines the number of matches during an interrogate phase is shown in Fig. 6. The match indicator also establishes the controls for advancing the ternary counter and for selecting the matched words. During the first phase of the memory cycle, the interrogate phase, current will be directed onto either the "0", "1" or *P* count lines and set the match indicator storage circuit. The circuits for accomplishing this are shown in the lower portion of Fig. 6. As can be seen in the Figure, the count line that the current appears on will be determined by the number of matched words in the memory. If there is more than one matched word, the current will be directed to the plurality (*P*) line.

During the second phase of the memory cycle, the advance control and the select control circuits are activated. It should be noted that the advance control circuits obey the sorting rules established earlier. Also note that when the match indicator storage circuit is set to *P*, the select control circuits direct current into the lowest-ordered position of the ternary counter. If this position is set to either a "0" or a "1", and the match indicator is set to *P*, this indicates multiple-matched sorting keys. When this happens, the first matched word register is selected. The select and no-select lines are shown going into the memory register select controls.

The details of the memory register select controls are not shown. However, three two-state storage circuits are actually needed for each word register, indicating the equal, marked and matched conditions. In addition controls are necessary for establishing current on either the READ or WRITE line of the first matched word register, for marking of the first matched word register, and for unmarking all equal word registers when desired.

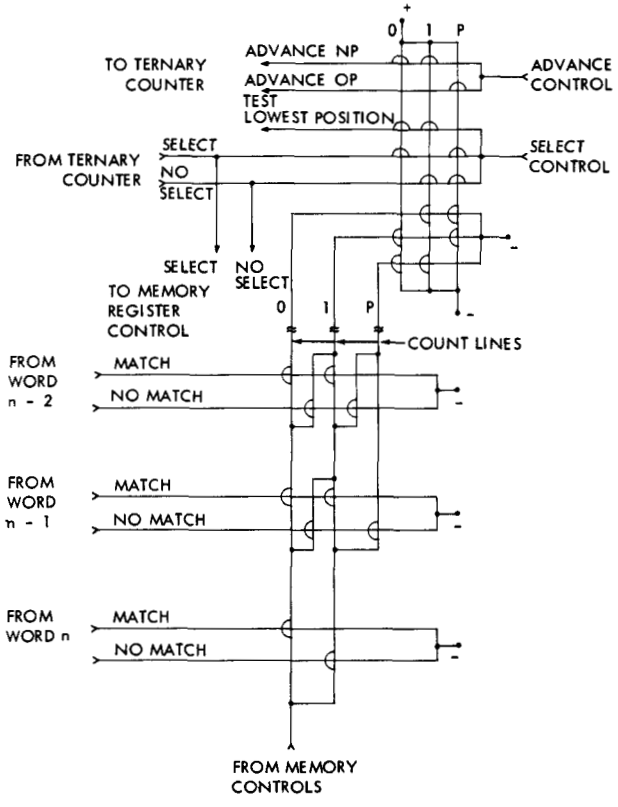


Figure 6 **Match indicator.**

Conclusions

We have constructed an algorithm to be used with an associative memory to retrieve randomly stored data in an ordered sequence. Modifications of this sorting algorithm using the different methods of interrogation and the different types of match indications were presented. We have also shown a cryogenic implementation of an associative memory using one form of the ordered-retrieval algorithm.

Some of the main features of our ordered-retrieval system are as follows. The sorting key is determined at the time of the sort by a stored program. The key may be made up of subkeys whose order within the word is irrelevant. The sort may be in ascending or descending order or a combination. A next higher or next lower value retrieval may be made. An ordered retrieval is performed without disturbing the original ordering of the stored data.

Most of the possible applications of these features are readily apparent. For example, the next value retrieval feature would be an aid to an interpolation routine operating on randomly stored data. These features are, of course, limited to data that may be stored within an associative memory system. Thus we have a high-speed internal sort which, of itself, does not solve all sorting problems. However, an internal sort is an integral part of file sorting. As technology increases the input-output data rate, there will be an increasing need for a fast internal sort.

References

1. A. E. Slade and H. O. McMahon, "A Cryotron Catalog Memory System", *Proceedings of the Eastern Joint Computer Conference*, December 1956.
2. A. E. Slade and C. R. Smallman, "Thin Film Cryotron Catalog Memory", *Symposium on Superconductive Techniques for Computing Systems*, May 1960.
3. J. R. Kiseda, H. E. Petersen, W. C. Seelbach, and M. Teig, "A Magnetic Associative Memory", *IBM Journal*, **5**, No. 2, 106 (1961).
4. R. R. Seeber, "Cryogenic Associative Memory", presented at National Conference of the Association for Computing Machinery, Milwaukee, August 1960.
5. R. R. Seeber, "Associative Self-Sorting Memory", presented at Eastern Joint Computer Conference, December 1960.
6. D. A. Buck, "The Cryotron—A Superconductive Computer Component", *Proceedings of the IRE*, **44**, No. 4, 482 (1956).
7. W. B. Ittner III and C. J. Kraus, "Superconducting Computers", *Scientific American*, **205**, 1, 124 (1961).

Received July 10, 1961