COMBINATORIAL TECHNIQUES FOR PERFORMING ARITHMETIC AND LOGICAL OPERATIONS

H. Fleisher

IBM

RESEARCH CENTER

INTERNATIONAL BUSINESS MACHINES CORPORATION
YORKTOWN HEIGHTS, NEW YORK

RC-289

COMBINATORIAL TECHNIQUES FOR PERFORMING ARITHMETIC AND LOGICAL OPERATIONS

by
H. Fleisher

International Business Machines Corporation
Research Center
Yorktown Heights, New York

ABSTRACT: Counting/coding blocks are presented in this paper as a means of performing arithmetic operations such as simultaneous addition of more than two words, and multiplication while keeping circuit complexity due to the combinatorial nature of these operations within bounds.

The logical capability of these blocks is also discussed in terms of their relation to majority organs.

A cryogenic "batch adder" is shown and discussed.

RC - 289
Research Report
July 18, 1960

Combinatorial Techniques for Performing Arithmetic and Logical Operations

by H. Fleisher

1. INTRODUCTION

Batch arithmetic operations in the sense of handling things of the same kind at the same time are familiar to us since many manual calculations involve them. Their extension to machine operation is, however, not so straightforward, although a history of such attempts exists.[1] These efforts encompass strictly mechanical means, as well as electro-mechanical and electronic circuits.

Mechanical techniques suffer from the obvious defects of bulk and slow speed, while electro-mechanical and electronic methods suffer from a need for a tremendous number of relatively expensive components. This is so because of the combinatorial nature of the problem.

A method of accomplishing these batch operations and simplifying the combinatorial aspect of the problem is to use an analogue procedure involving summation of voltages or currents. For well-known reasons, these methods have serious limitations, and circuitry to batch more than a few (i.e., 6 or so) words becomes unwieldy. In a practical sense, we can argue that an analogue approach is acceptable only for batching a few words.

Clearly, another approach is needed, one that will keep the combinatorial aspect of the problem within bounds. What we mean by acceptable bounds will depend on our choice of technology as well as economic factors related to it.

The method described in this paper is that of counting the appropriate information bits and then coding that count in an ordered system based on a chosen radix. The radix we have chosen is 2, and the ordered system is standard binary code.

Figure 1 shows an example of addition of several words: there are 5 words of 6 bits each. If we examine this problem in a strict combinatorial sense, we note, on counting down a column, that we must consider all combinations of 5 things: $\binom{5}{0} = 1$, $\binom{5}{1} = 5$, $\binom{5}{2} = 10$, $\binom{5}{3} = 10$, $\binom{5}{4} = 5$, $\binom{5}{5} = 1$. Consider the case of two one-bits present in the column. There are 10 combinations of this group, and, accordingly, we would need 10 distinct circuits. Totalling, we find that we would need 32 distinct adder circuits simply to add the bits in a single column. Actually, this applies only to the lowest order column, since we must also count carries in the higher order columns. If we consider batch addition of n words on this strict combinatorial basis, we note that we would need $\sum_{i=0}^{n} \binom{n}{i} = 2^n$ distinct circuits for each column.

The counting/coding block to be discussed in this paper avoids this problem by bringing each combinatorial group together so that the count of a given group is uniquely represented. In turn, this count is coded in a manner we describe now.

2. BATCH ADDITION

Referring to Figure 1, we first discuss column "2^0", the lowest order column. A counting circuit associated with this column counts the number of 1 bits present. The associated coding circuit translates this count into binary ordered code ($2^k \ldots 2^1\ 2^0$) and the output is written into the sum and carry registers as follows: the 2^0 bit is written into the sum register for column (2^0); the 2^1 bit is written into the C_1 register but in the next higher column position (2^1); the 2^2 bit is written in the C_2 register but in the second higher column position (2^2); and so on until the coded bits have been used up. Then the operation is repeated for the next column: 2^1.

In repeating this operation for column 2^1 and all subsequent columns we note that the counting operation includes the carry bits that were inserted into the column being counted from lower order columns.

We generalize this description by this statement: if we count m one bits in a column, and state this count in binary coded form, i.e.,

$$m = b_k\ 2^k \ldots b_2\ 2^2\ b_1\ 2^1\ b_0\ 2^0$$

where each b_i is either 0 or 1, then each bit in the binary number is placed in an appropriate register. Identification of the register is obtained by these identities:

$$\left.\begin{array}{l} 2^0 = \text{SUM} \\ 2^1 = C_1 \\ 2^2 = C_2 \\ \quad \text{etc.} \end{array}\right\} \text{ for a given column.}$$

The number of sum and carry registers that is needed is related to the number of words being added: if n is the number of words, then the number of carry and sum registers is the smallest integer greater than $\log_2 n$.

3. MULTIPLICATION

Multiplication can be expressed directly as a batch addition process, provided all partial products are obtained. The partial products of binary multiplication are simply the results of column shifts to the left of the multiplicand. These partial products are under control of corresponding multiplier bits, with the highest order bit of the multiplier controlling the greatest partial product, i.e., the multiplicand which has been shifted furthest to the left.

Figure 2 illustrates the method of obtaining a product by means of a column shifted multiplicand matrix under control of a multiplier. For convenience, both multiplicand and multiplier are restricted to 6 bits: the product will have 12 bits. As shown by the crossed out row in Figure 2, a zero bit in the multiplier inhibits that row of the multiplicand matrix, and the one bits in that row are not used.

4. THE "COUNTING/CODING" BLOCK

A block diagram of the counting/coding operation is shown as a "counting/coding" block in Figure 3. The nomenclature of this block diagram is this: there are n words in the batch to be added, hence, a given column has n bits as input. The coding circuit converts the count of these bits into binary ordered code such that the highest ordered output 2^k is determined by $k \sim \log_2 n$. This is strictly true for the lowest order column of the batch: for higher order columns the input to the block is of order $n + \log_2 n$, but this does not materially affect the coding circuit.

For purposes of illustration, binary ordered coding is shown, but clearly, ordering in any radix is feasible. For example, it may be desirable under some circumstances to code directly into decimal ordered form. In this way, batch arithmetic operations in the decimal system would be performed.

5. MAJORITY ORGAN LOGIC

We next explore the relation between the "counting/coding" block and majority organ logic. We do this by means of the example, shown in Figure 4, involving 3 inputs. Figure 4a shows a three input majority organ with an accompanying table listing the output function in terms of the threshold value. In general, an output is obtained from a majority organ, if it has n inputs and a threshold of T for the condition

$$\sum_{i=1}^{n} x_i \geq T .$$

Figure 4b shows a three input counting/coding block with additional switching circuits: an AND gate, and an OR gate. The indicated outputs are shown as functions that correspond to the outputs of the majority organ for thresholds of 1, 2, and 3. A complete tabulation of functions of $u(= 2^0)$ and $v(= 2^1)$ considered as independent variables is shown in Figure 5. These functions are also shown related to the input variables x_1, x_2, and x_3. The switching circuits in Figure 4b will not generate all of these functions, and a generalized diagram is indicated in Figure 4c.

While this generalized diagram simply shows a "logic" block, we understand that the circuits in this block perform those switching operations necessary to generate all 16 functions of u, v. Since these, in turn, are functions of inputs

x_1, x_2, x_3, they constitute a special subset of the 256 Boolean functions of three variables.

Interpretation of the table shown in Figure 5 is straightforward, and we note, by inspection, that it is a way of stating a generalization of the majority decision principle.

The specific majority organ functions that are detailed in Figure 4 are these from Figure 5:

$f_{15} = 1$ = output function for T = 0

$f_0 = 0$ = output function for T = 4

$f_7 = u\bar{v} + \bar{u}v + uv = u + v = x_1 + x_2 + x_3$ = output function for T = 1

$f_3 = \bar{u}v + uv = v = x_1 x_2 + x_1 x_3 + x_2 x_3$ = output function for T = 2

$f_1 = u \cdot v = x_1 \cdot x_2 \cdot x_3$ = output function for T = 3.

Because the input variables are "counted", their identity is effectively destroyed, and only those functions of these variables can be generated that are invariant under permutations of the input variables. As an example, consider f_6 of Figure 5. In (u, v) notation, $f_6 = u\bar{v} + \bar{u}v$, which becomes $= \bar{x}_1 \bar{x}_2 x_3 + \bar{x}_1 x_2 \bar{x}_3 + x_1 \bar{x}_2 \bar{x}_3 + x_1 x_2 \bar{x}_3 + x_1 \bar{x}_2 x_3 + \bar{x}_1 x_2 x_3$ in terms of x_1, x_2, and x_3.

Expressing the latter in numeric form, we get $f_6 = \sum (001, 010, 100, 110, 101, 011)$. It is easy to see, on inspecting this form, that f_6 is invariant under permutations of the variables. On the other hand, a function such as $\bar{x}_1 \bar{x}_2 x_3 + x_1 \bar{x}_2 x_3 = \sum 001, 101$ is not invariant under permutations of the variables, and is therefore not directly realizable from the table of Figure 5.

That this invariance under permutations of the input variables must hold for the generated functions of Figure 5 is clear from the grouping of

the input variables. Each combinatorial group of variables contains all permutations pertaining to that group: the output distinguishes combinations and not permutations.

However, this is not a serious limitation since all functions of a given number of variables can be generated either by a single counting/coding/logic block or by a group containing a small number of them. We obtain this result by considering a counting/coding/logic block of a greater number of variables than needed and biasing the excess.

As an example, consider the three variable case shown in Figure 5, and assume x_1 is restricted to value 0. The table is reduced to 8 columns since the last row is inoperative.

In terms of x_2 and x_3, we obtain these functions:

$$f_{14} = f_{15} = 1$$
$$f_0 = f_1 = 0$$
$$f_2 = f_3 = x_2 \cdot x_3$$
$$f_6 = f_7 = x_2 + x_3$$
$$f_4 = f_5 = x_2 \bar{x}_3 + \bar{x}_2 x_3$$
$$f_{10} = f_{11} = x_2 \equiv x_3$$
$$f_{12} = f_{13} = \bar{x}_2 + \bar{x}_3 = \overline{x_2 \cdot x_3}$$
$$f_8 = f_9 = \bar{x}_2 \cdot \bar{x}_3 = \overline{x_2 + x_3}$$

We note that the last two functions are the well-known Sheffer and Pierce Stroke functions, or, as is currently the fashion: "NAND" and "NOR" functions.

Hence, the three input counting/coding/logic block with 0 bias on one of its inputs can generate enough functions* to realize completely all

*Either of the last two functions, being universal, is sufficient to generate all switching functions of two variables.

of the switching functions of two variables.

6. PHYSICAL REALIZATION

The realization of a counting/coding block for use in arithmetic circuits or a counting/coding/logic block for use as a controlled logical function generator is relatively straightforward but tends toward requiring a large number of components and devices. If these are the customary discrete entities such as transistors, diodes, and magnetic cores, then this number may become prohibitive.

There is, however, little emphasis on minimizing circuit elements in the nature of the counting/coding block, since its usefulness lies in its ability to reduce the combinatorial aspect of switching to more manageable proportions.

We look therefore to newer technologies for preferred circuit realizations. These should provide devices with such attributes as small size and low power consumption, as well as offer potential advantages of low cost per element and mass fabrication of entire functional units. The switching speed of these devices may be thought of as a parameter to outline a range of possible applications.

An example of such a device is the cryotron, and we show a cryotron circuit in Figure 7. This circuit produces the sum of four words, and shows the flip-flops that comprise the sum register of the two lowest orders. The counting/coding block associated with each column is here shown as a modified transfer tree driving a set of flip flops. The devices shown are crossed cryotrons. A single crossed cryotron is diagrammed in Figure 6a, and control of a loop is shown in Figure 6b.

Briefly, we explain operation of a cryotron as follows: refer to Figure 6a, and allow a sufficiently large current through the control wire; its magnetic field will force the cryotron gate to switch from superconducting to normal resistive state, and current which was in the gate line is diverted elsewhere. Figure 6b illustrates the diversion path specifically. Note that switching is confined to the loop; current entering and leaving the loop is undisturbed.

If we assume current in leg B of the loop, then application of control current to cryotron a will force it resistive, and current will be diverted to the remaining superconducting leg A. Removal of current from cryotron a will not change the path. To restore current to path B, we must apply current to the control line to force cryotron b resistive.

In similar fashion we explain operation of the circuit in Figure 7. The memory cells of the first column are labelled B_1, B_2, B_3, and B_4 and show only that part of each cell that controls the counting tree. For simplicity, these are shown as control wires K_1, K_2, K_3, and K_4, although other configurations of control of the tree directly by stored bits are possible.

The outputs of the tree are labelled O_0, O_1, O_2, O_3, O_4 to correspond to the count of 1 - bits in the column. The coding circuit is here represented by cryotron flip flops S, C_1, C_2. S is the Sum flip flop for that column; C_1 is the first carry flip flop, and is accordingly in the next higher column; and similarly, C_2, being the second order carry flip flop, is in the column one removed. Clearly, the counting trees for the higher order columns are also controlled by the carry flip-flops from the lower order columns.

As an example for the lowest order column, let $B_1 = 1$, $B_2 = 0$, $B_3 = 1$, and $B_4 = 1$. The current in memory cell B flows in the left-hand leg, in B_2 in the right-hand leg, and so on. Cryotrons a, b, c, d, are forced resistive, diverting current in the control lines: in K_1 to the lower leg, in K_2 to the upper leg, and in K_3 and K_4 to the lower legs. These currents in turn control the tree: reading from top, down: current is diverted to the left since cryotron e is resistive, thence to the right since cryotron f is resistive. On the third level of the tree, current is diverted to the left since cryotron g is resistive; on the fourth level, the current is also diverted to the left since cryotron h is resistive. The current thus emerges on output line O_3, indicating that three ones are present in the 4 bit positions. Clearly, <u>any</u> three ones in the four bit positions would result in current emerging on O_3.

Binary coding of the information on O_3 is straightforward: O_3 flips S to the 1 position, C_1 to 1, and C_2 to O.

While operating speeds of a circuit such as the batch adder shown in Figure 7 are dependent in part on specific circuit configurations, certain comparisons with more conventional circuitry can be made:

a. This circuitry, operating on two n-bit words will add in a time interval of the same order as a conventional adder.

b. A batch adder adding together as many words as there are bits per word, will obtain the sum in a time interval approximately twice that of the time interval needed for two word addition. This is clearly much faster than over and over addition needed for more conventional circuitry.

c. The batch adder, adapted to multiplication, will multiply two n-bit words, yielding a 2n-bit product in roughly the same time interval as a, above. This operation is also much faster than more conventional multiplication schemes.

7. SUMMARY

A method is described in this paper of performing arithmetic or logical operations in such a way as to simplify the combinatorial aspect of these operations. This method tends to remove specific labelling of input variables. Because of this, only certain functions that are invariant to permutations are capable of being directly generated: other functions are generated subject to constraints.

Block diagrams called "counting/coding" block and "counting/coding/logic" block are introduced as the means, symbolically, for realizing this method.

Physical realization has been achieved in the form of cryotron circuits, and an example is shown of a cryotron batch adder operating embedded in memory.

H. Fleisher
IBM Research

July 12, 1960

Revised July 18, 1960

FOOTNOTE

1. U. K. Patent 410, 129, May 9, 1934, "Improvements in or relating to Calculating and like Apparatus," by Raymond Louis Andre Valtat.

   U. S. Patent 2, 404, 250, July 16, 1946, "Computing System," by Jan A. Rajchman.

   U. K. Patent 754, 208, August 1, 1956, "Improvements relating to digital multipliers," by Rene Alphonse Eugene Higonnet and Louis Marius Moyroud.

ACKNOWLEDGEMENTS

The batch processing concepts discussed in this paper and their cryogenic circuit embodiments are results of a joint effort with R. I. Roth of the IBM Research Laboratory.

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	
		0	1	0	1	1	1	WORD REGISTERS
		0	0	0	0	0	1	
		0	1	0	0	0	1	
		0	1	1	1	1	1	
		0	0	1	1	0	0	
0	1	0	1	0	1			C_2 REGISTER
0	0	0	1	0	1	0		C_1 REGISTER
0	1	0	1	0	1	0	0	SUM REGISTER

Figure 1. Batch addition

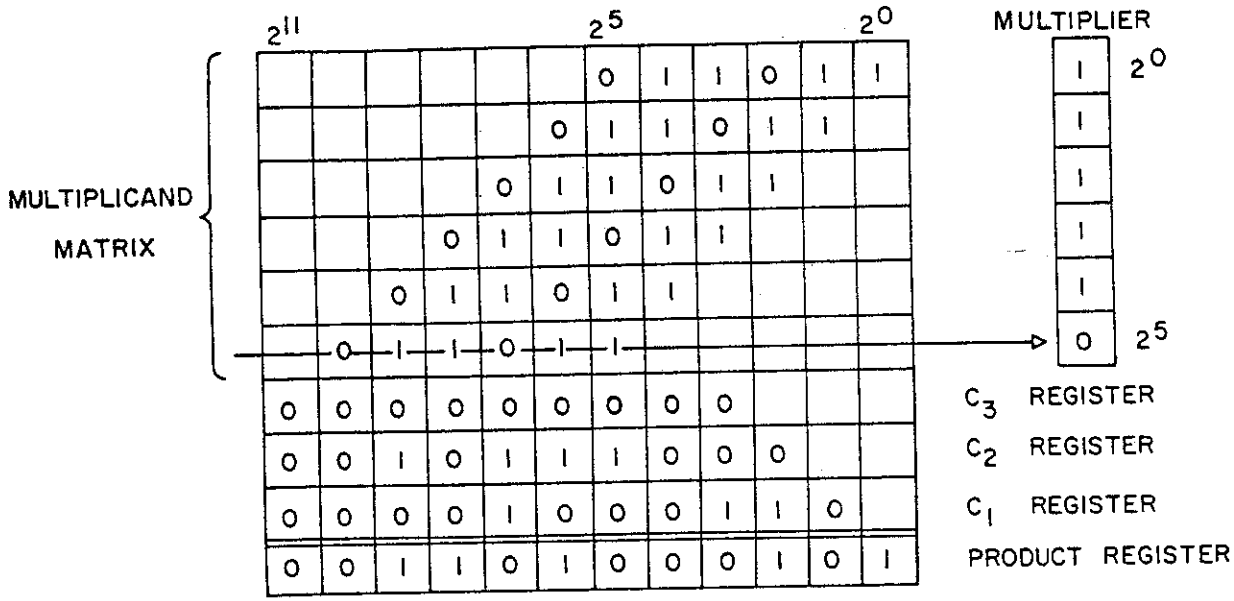


Figure 2. Multiplication

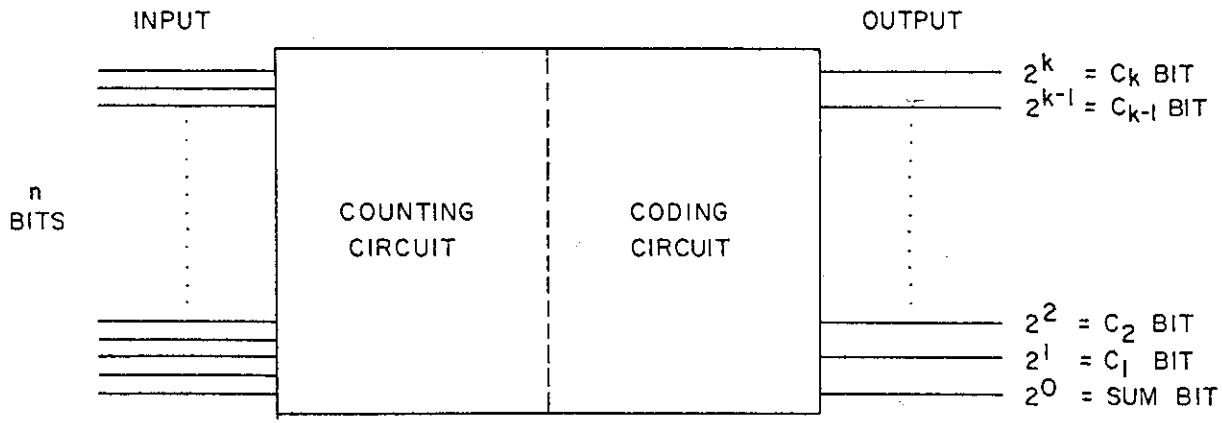
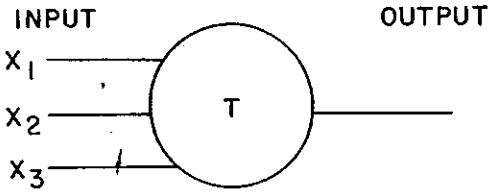


Figure 3. Counting/coding block



THRESHOLD, T	MAJORITY ORGAN OUTPUT FUNCTION
0	1
1	$X_1+X_2+X_3$
2	$X_1X_2+X_1X_3+X_2X_3$
3	$X_1\ X_2\ X_3$
4	0

Figure 4a. Majority organ and function table

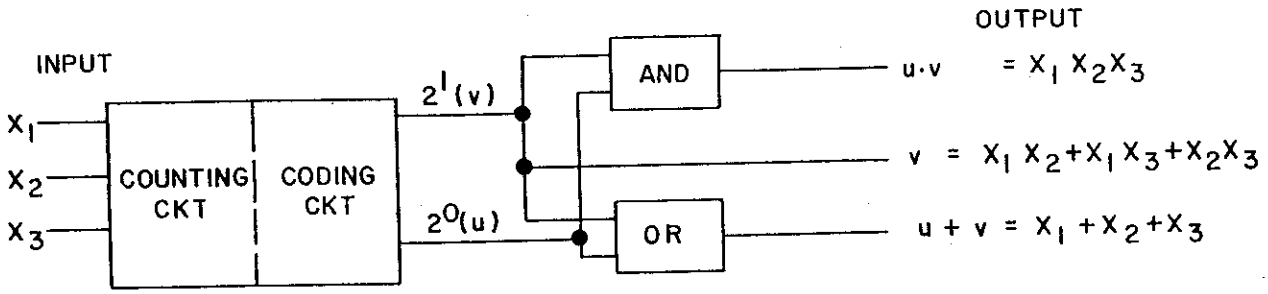


Figure 4b. Counting/coding block with additional logic circuits to realize majority organ functions

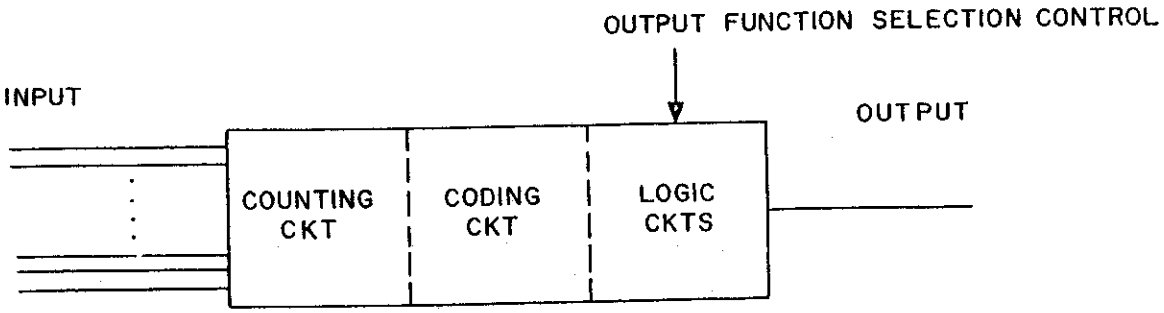


Figure 4c. Generalized counting/coding/logic block with selection of output function

x_1	x_2	x_3	$\binom{n}{j}$	u	v	f_0	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}	f_{11}	f_{12}	f_{13}	f_{14}	f_{15}
0	0	0	$\binom{3}{0}$	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	0	1	$\binom{3}{1}$	1	0	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
0	1	0																			
1	0	0																			
0	1	1	$\binom{3}{2}$	0	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	0	1																			
1	1	0																			
1	1	1	$\binom{3}{3}$	1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

Figure 5. Table of symmetric functions obtained from counting/coding/logic block

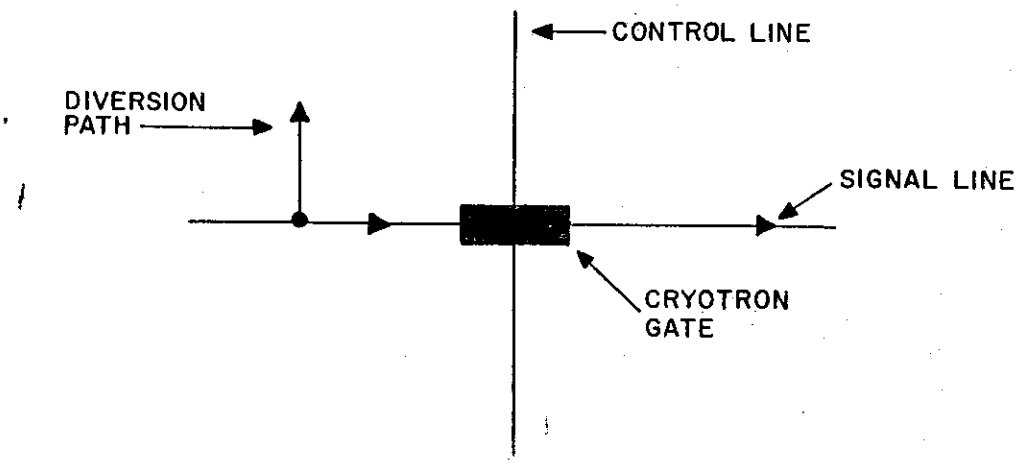


Figure 6a. Single cryotron control

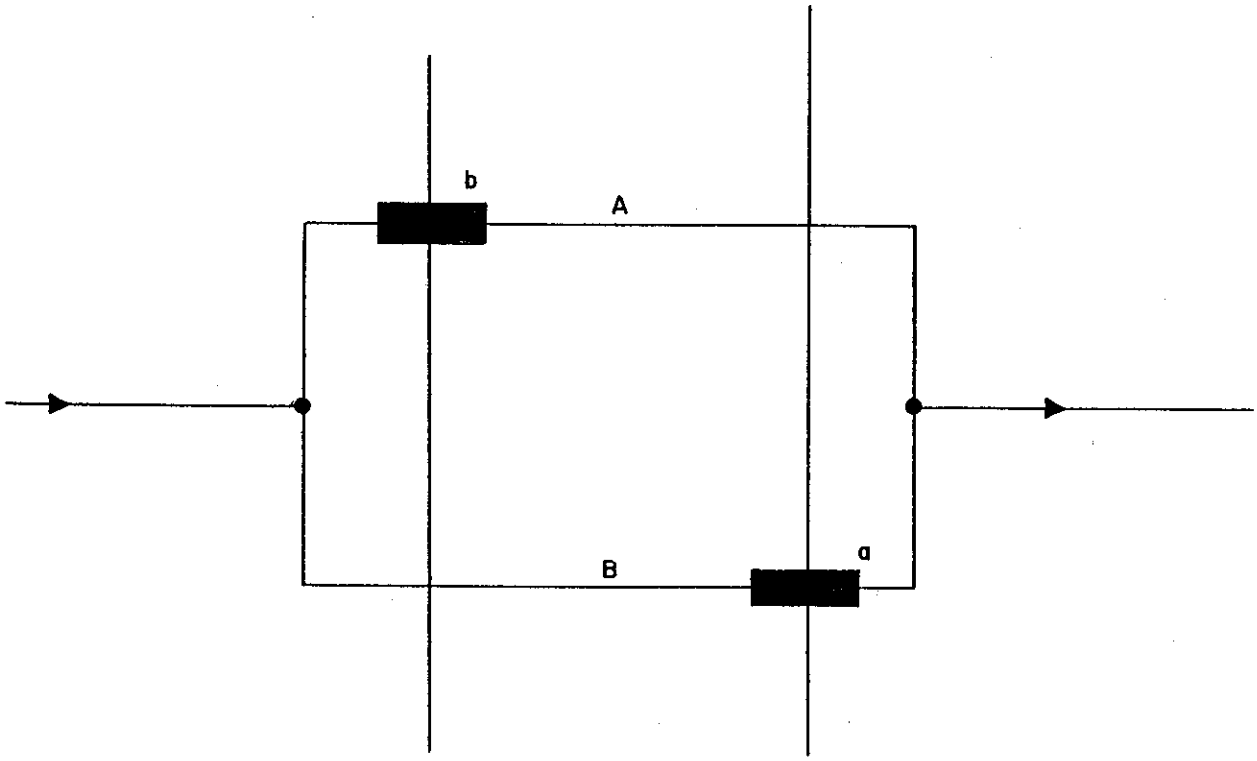


Figure 6b. Loop control

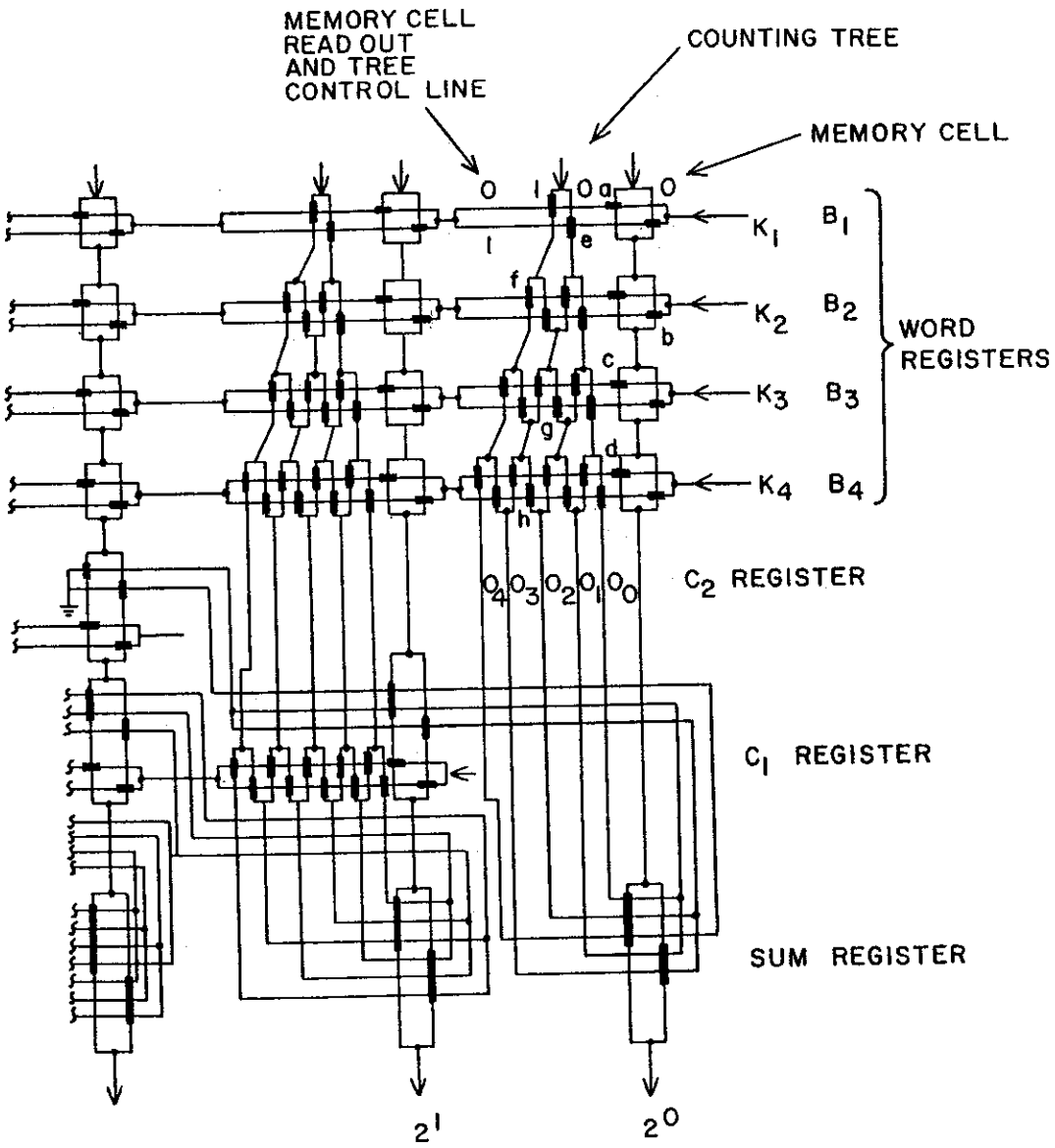


Figure 7. Cryotron batch adder