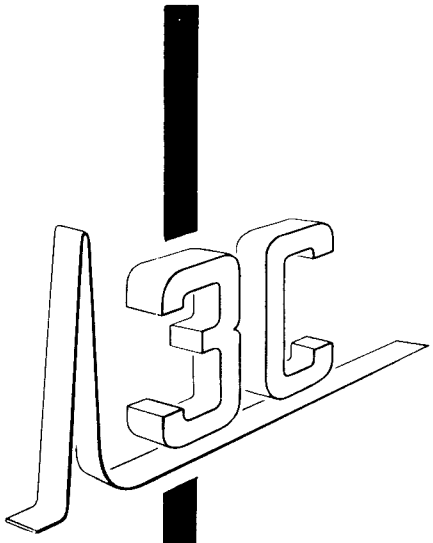


symbolic logic, binary calculation, and 3C-PACs

by ROBERT W. BROOKS

COMPUTER CONTROL COMPANY, inc.
92 broad street
wellesley 57, massachusetts

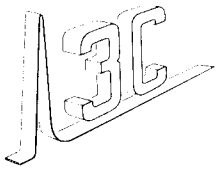
Bibliography

Brooks, R. W., "Extension of the Veitch Chart Method in Computer Design", meeting of Association for Computing Machinery, Cambridge, Massachusetts, September, 1953.

Karnaugh, M., "The Map Method for Synthesis of Combinational Logic Circuits", AIEE Summer General Meeting, Atlantic City, New Jersey, June, 1953.

Varnum, E. C., "Binary Matrix Analysis of Relay Circuits", Machine Design, 1949.

Veitch, E. W., "A Chart Method for Simplifying Truth Functions", meeting of Association for Computing Machinery, Pittsburgh, Pennsylvania, May, 1952.



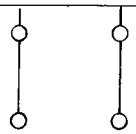
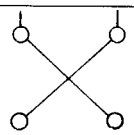
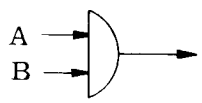
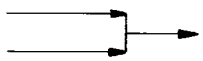
SYMBOLIC LOGIC, BINARY CALCULATION, AND 3C-PACs

Robert W. Brooks*

1. Introduction

It is a remarkable and useful fact that extremely complex logical manipulations and numerical computations can be built up from a few simple operations. The basis for such operations is a form of symbolic logic which deals with the question of whether a certain relationship is "true" or "false" under a given set of conditions. These two "true" and "false" alternatives are of great practical value since they can be represented conveniently by physical equipment; for example, by a light being on or off, a relay open or closed, a vacuum tube conducting or nonconducting, or a voltage pulse being present or absent. As is shown later, this type of instrumentation is ideal for computations with binary numbers.

Fig. 1. Representation of conditions and connectives of symbolic logic.

	Condition		Connective	
	true (assertion)	false, not (negation)	and	or
Written Symbol	A	$\bar{A}$	(A · B)	(A v B)
Schematic Representation			 gate	 buffer
Common Equipment Forms	positive d-c voltage (through feed) + pulse present	d-c voltage polarity reversal − no pulse	coincidence detector	summing circuit and limiter

Logical relationships, or statements, are built up by the use of three connectives: "and", "or", and "not", represented symbolically in Fig. 1. Not represents negation, or opposition. For example, if condition "A" is true, then condition "not A" is false. The connective and may join two conditions to indicate that "A and B" is true only when both "A" and "B" are true. The connective "and" is thus restrictive in the sense that it requires that two conditions be simultaneously true. The connective or indicates that "A or B" is true whenever either "A" or "B" (or both "A" and "B") is true. The connective or is thus less restrictive in that it can be true under three possible conditions.

---

*Vice-President, Computer Control Company, Inc.

Fig. 2. Truth table showing 4 basic conditions of 2 variables and effects of connectives not, and, or.

4 Basic Truth Conditions		not		and	or	not-and	not-or	(etc.)	
A	B	$\bar{A}$	$\bar{B}$	(A · B)	(A v B)	($\bar{A}$ · $\bar{B}$)	($\bar{A}$ v $\bar{B}$)	($\bar{A}$ · B)	(A v $\bar{B}$)
0	0	1	1	0	0	1	1	0	1
0	1	1	0	0	1	0	1	1	0
1	1	0	0	1	1	0	0	0	1
1	0	0	1	0	1	0	1	0	1

1 = true 0 = false

2. Truth Tables and Logic Charts

Several graphical methods have been devised for representing logical relationship among variables. A simple tabular method employs "1" to indicate true and "0" to indicate false. For two variables A and B there are four possibilities shown in the truth table of Fig. 2. The effect of the three connectives on the truth statements can also be seen.

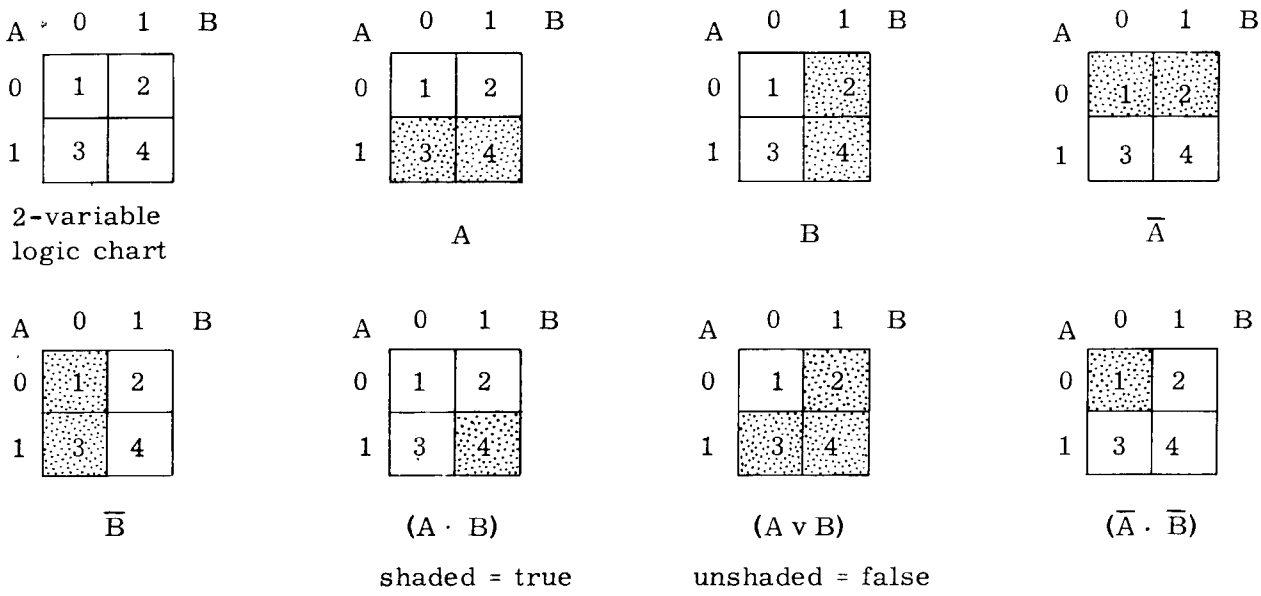


Fig. 3. Logic charts showing conditions of 2 variables.

The four combinations of two variables can also be represented conveniently on a rectangular chart having four squares, as in Fig. 3. The shaded squares indicate "true" and the unshaded squares indicate "false". The charts for two variables and the effect of the three connectives are shown in Fig. 3. There are 16 possible combinations, or 2^{2^n} with n being the number of variables.

All combinations of three variables A, B, and C can be represented on the logic chart of Fig. 4 having eight squares. The number of such possibilities is $2^{2^3} = 2^8 = 256$. The method for building up a simple logical statement in three variables such as $(\bar{A} \cdot B) \vee (C)$, is also shown in Fig. 4.

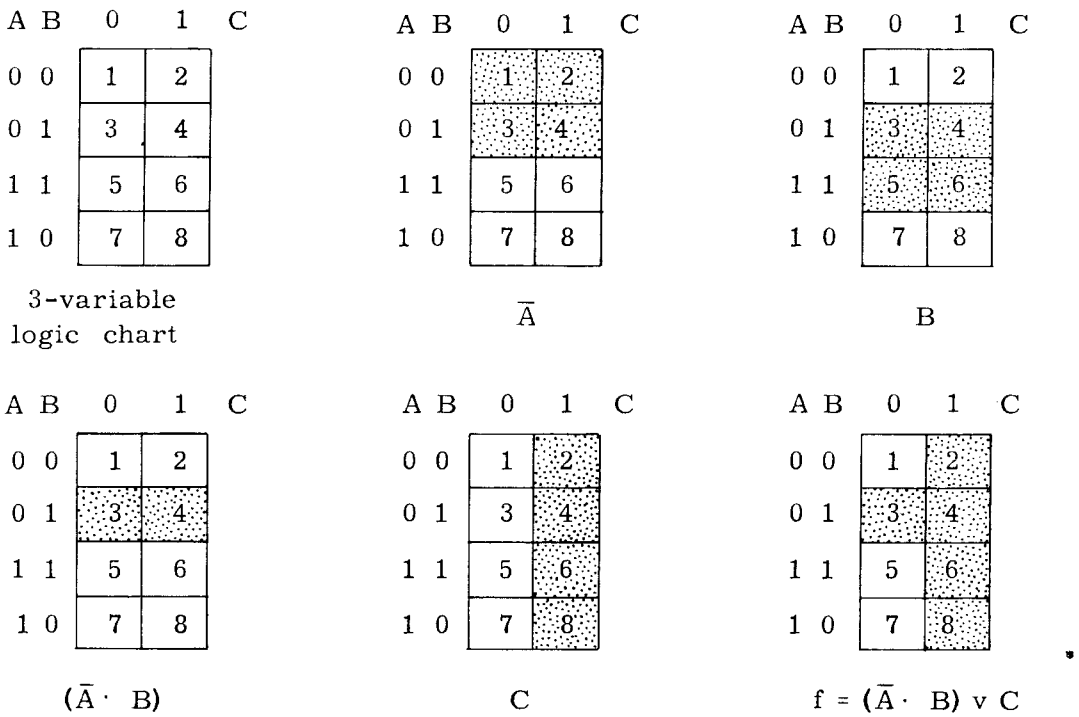


Fig. 4. Three-variable logic chart and construction of the function $(\bar{A} \cdot B) \vee C$

3. Logical Implementation

To implement logical statements with physical equipment, it is necessary merely to have signals which represent the true and false conditions (for example, positive and negative voltages), and physical components which symbolize the three connectives defined in Fig. 1. The connective not is represented by a form of "inverter" which might, for example, produce a negative signal when it receives a positive one, or produce no pulse when it receives a pulse. Such a "not" component will thus produce "not A" ($\bar{A}$) when it receives "A", or will produce "B" when it receives "not B" ($\bar{B}$).

The "and" connective is represented by a gate, shown symbolically in Fig. 1. Physically, the gate is a component which requires that all of its inputs be true in order that the output be true. If there is one false input, the output will be false.

The element which produces the "or" connective is called a buffer. In effect, the buffer is merely a direct connection between several variables so that if one or more of the variables is true, a "truth" will appear at the buffer output.

To represent any function of three variables, the combined gate and buffer instrumentation of Fig. 5 can be used. This consists of eight separate gates each having three input legs. Each of these gates corresponds to one of the eight squares in the three-variable table of Fig. 4. The truth of a statement can be determined by implementing the truth, or shaded, squares

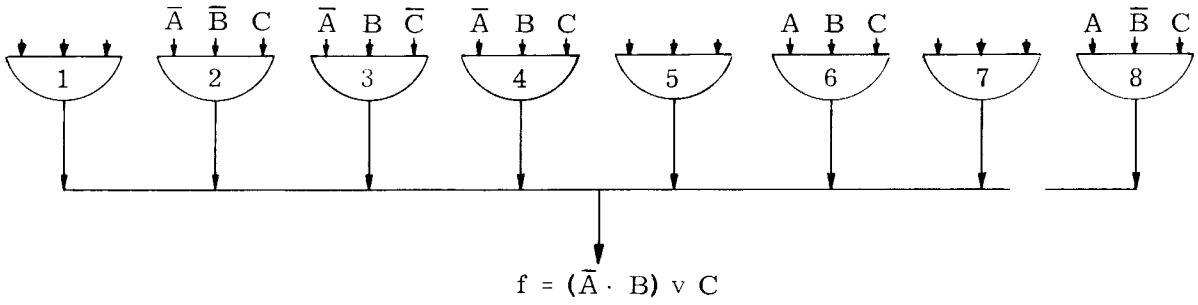


Fig. 5. Use of 8 gates and buffer for 3-variable function.

of the table. The eight-gate structure of Fig. 5 is shown set up to represent the truth of the simple statement $(\bar{A} \cdot B) \vee (C)$ of Fig. 4. The false of a statement can be represented by merely implementing the unshaded squares of the logic chart.

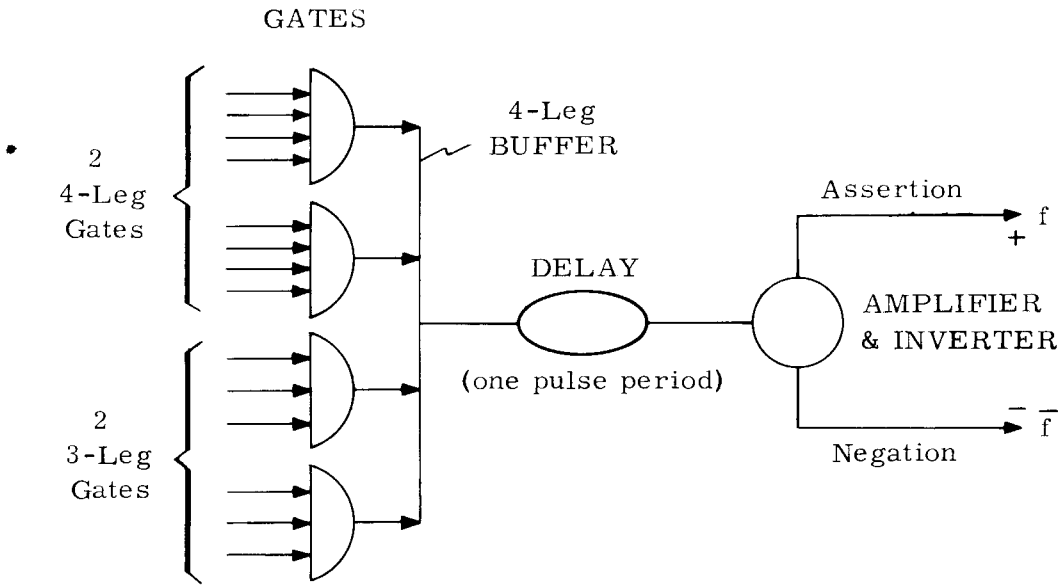


Fig. 6. 3C-PAC, a simple and versatile Gating Package.

4. The 3C-PAC, A Simple and Versatile Gating Package

In Fig. 6, the 3C-PAC Gating Package by Computer Control Company, Inc. is shown. This simple four-gate package can perform all of the functions possible with the eight-gate package of Fig. 5. In addition to being able to handle all functions of three variables, the 3C-PAC can handle 51,358 functions of four variables and many functions of five or more variables.

The four-gate 3C-PAC of Fig. 6 can replace the eight-gate package of Fig. 5 because of the use of the inverter at the output. (The purpose of the one-pulse-period delay is described later in Section 5.) The output inverter permits either the truth or the false of the output to be selected as desired. The gates are used to represent either the shaded or the unshaded squares in the chart, whichever is the lesser in number. It can be seen that four is the maximum number of gates required, and many three-variable functions can be represented by using three or fewer gates.

The general rules for implementing any function of three variables with the 3C-PAC Gating Package are as follows:

1. Shade in the squares of the three-variable chart that represent the truth of the defined function f.

2. If the number of shaded squares is four or less, implement the function f by making the connections that correspond to the shaded squares. The positive output lead indicates the truth f of the function represented; the negative output lead represents the false f.

3. If the number of shaded squares is greater than four, implement the function f by making the connections that correspond to the unshaded squares (i.e., implement the false f of the function). The negative output lead represents the truth f of the described function; the positive output lead represents the false f of the function.

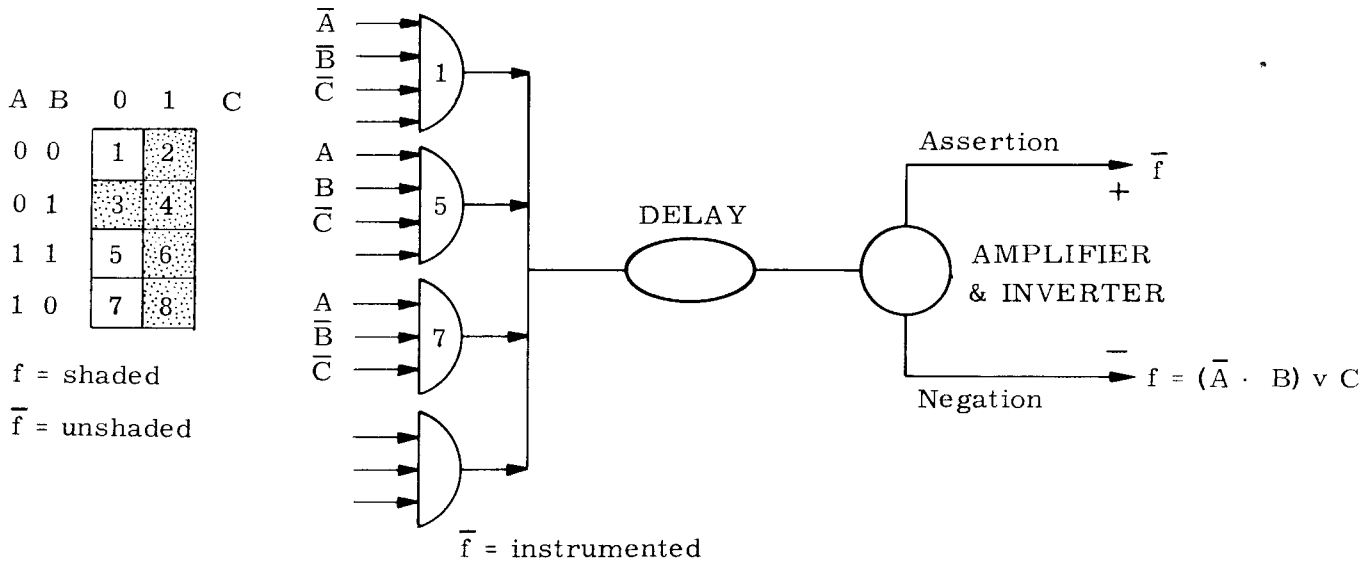


Fig. 7. Representation of 3-variable function with 3C-PAC.

In Fig. 7, the 3C-PAC is shown set up to represent the same three-variable function $(\bar{A} \cdot B) \vee C$, treated in Figs. 4 and 5. Note that the three unshaded, or false, squares are implemented, and that the true function f is taken from the negation output.

The 3C-PAC Gating Package of Fig. 6 can be seen to have two 4-leg gates. In later Section 11, it is shown how the addition of these two legs greatly extends the usefulness of the 3C-PAC when functions involving four or more variables are involved.

5. Pulse Techniques and Time Sequence

The discussion of symbolic logic in the preceding sections has not been limited to any particular type of "true" or "false" signals. They could very well be d-c voltage signals of positive or negative polarity to represent assertion or negation. Many logical manipulations and computations, however, must be performed at very high speed to be useful. The 3C-PAC Gating Package of Fig. 6 has thus been designed to operate on pulses having a one-megacycle repetition rate. Truth, or assertion, is represented by the presence of a positive pulse on the assertion lead; false, or negation, is represented by the presence of a negative pulse on the negation lead.

In operation, synchronized pulse trains enter the gate legs, and pass through the gates and buffer to provide a resultant output pulse train. One such pulse operation is performed each microsecond. The result of this operation is carried over for use during the next pulse period. The one-pulse-period delay is used to perform this carry-over function, and aids in the synchronization of pulse signals. It will be shown in later sections how the use of "serial" or "parallel" methods of instrumentation affect the time sequence of the different pulse operations. It should also be noted that accessory delay equipment is available for use with 3C-PACs when it is desired to hold an output pulse train for greater than one pulse period.

6. Binary Arithmetic

In logical analyses, the "pulse no-pulse" conditions handled by the 3C-PAC Gating Package can serve to determine the "true" or "false" of complex logical reasoning. These two physical "pulse no-pulse" conditions are also well suited to binary digital computation.

Fig. 8. Decimal and binary number systems.

	hundreds	tens	units
	10^2	10^1	10^0
	8	7	3
873 =	$8 \times 10^2 +$	7×10^1	$+ 3 \times 10^0$

Decimal

	eights	fours	twos	units
	2^3	2^2	2^1	2^0
9 =	1	0	0	1
9 =	1×2^3	$+ 0 \times 2^2$	$+ 0 \times 2^1$	$+ 1 \times 2^0$

Binary

The basis for both decimal and binary number systems is shown in Fig. 8. The decimal number system is built up from powers of 10. The binary number system, consisting of only two digits 1 and 0, is built up from powers of 2. The "double or dabble" method of evaluating binary numbers is illustrated below with the number 011100. Starting with 1 at the highest order 1, the 1 is doubled to equal 2 if the next digit is an 0. If the next digit is a 1, the 1 is dabbled; i.e., doubled plus 1 to equal 3. This process is repeated with the accumulated total as follows:

0	1	1	1	0	0	= 28
	1	(1 x 2)+1	(3 x 2)+1	(7 x 2)	(14 x 2)	
	↑	3	7	14	28	
		dabble	dabble	double	double	

Binary addition and subtraction follow the simple rules set forth in Fig. 9. Binary addition and subtraction are also illustrated in this figure. As might be expected, it is necessary to "carry" and "borrow" in binary addition and subtraction in essentially the same manner as in decimal addition and subtraction.

Fig. 9. Addition and subtraction of binary numbers.

Addition Rules:

0	0	1	1	
+0	+1	+0	+1	carry 1 to next higher column
0	1	1	10	

Sample:

0 1 1 1 = 0 + 4 + 2 + 1 = 7

0 1 0 0 = 0 + 4 + 0 + 0 = +4

1 0 1 1 = 8 + 0 + 2 + 1 = 11

Subtraction Rules:

0	1	1	0	and
-0	-0	-1	-1	borrow 1 from next higher column
0	1	0	0	

Sample:

0 1 1 1 0 1 = 0 + 16 + 8 + 4 + 0 + 1 = 29

- 0 1 0 1 0 1 = - 0 - 16 - 0 - 4 - 0 - 1 = - 21

0 0 1 0 0 0 = 0 + 0 + 8 + 0 + 0 + 0 = 8

7. Binary Addition - Serial Method

The most common method for performing binary addition is called the serial method. By this method, two binary numbers such as 29 and 21, illustrated in Fig. 10, are added by commencing with the right-hand column and progressing to the left, carrying a 1 to the next higher column when a (1 + 1) addition is indicated. The two numbers A and B can be thought of as two pulse trains entering an adding device. The 0's represent no pulse and the 1's represent a pulse. The pulse trains A and B are synchronized so that corresponding columns enter the adding network precisely at successive pulse periods. Using 3C-PACs, these two numbers "march" into the adder at the rate of one column per microsecond. As will be shown, the adding device must merely be a digital network arranged to perform the addition table defined in Fig. 9.

A "carrying" circuit supplies a third pulse input C. The "carry" signals are fed back from the output through the one-pulse-period delay. The "carry" digit is thus added to the next higher-order column, and is indicated with a "prime", as C', to show that it is generated one-pulse-period earlier than the A and B inputs. Figure 10 shows how the two digital numbers A and B are added in six pulse periods.

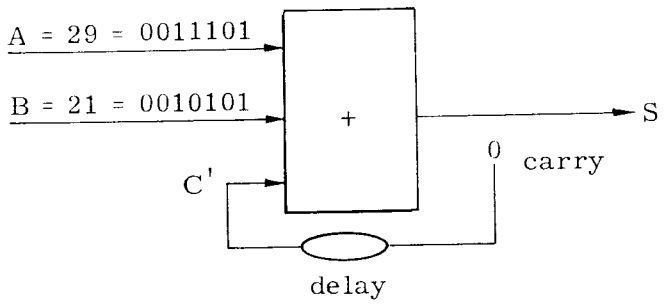
Serial Addition Implementation: Defining the following variables

A = Incoming binary digit of augend
B = Incoming binary digit of addend
C = Carry digit determined by addition performed one-pulse-period earlier
S = Sum digit of column being added

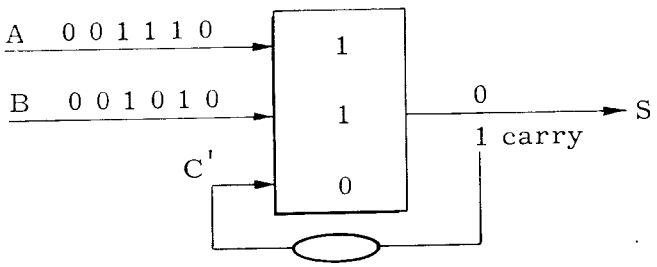
Two logical expressions can be written to represent the carry and sum functions:

$$C = (\overline{A} \cdot B \cdot C') \vee (A \cdot \overline{B} \cdot C') \vee (A \cdot B \cdot \overline{C'}) \vee (A \cdot B \cdot C')$$

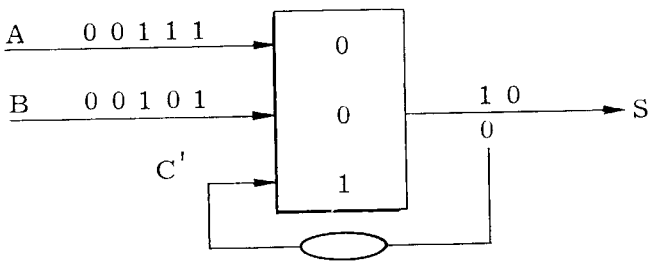
$$S = (A \cdot \overline{B} \cdot \overline{C'}) \vee (\overline{A} \cdot B \cdot \overline{C'}) \vee (\overline{A} \cdot \overline{B} \cdot C') \vee (A \cdot B \cdot C')$$



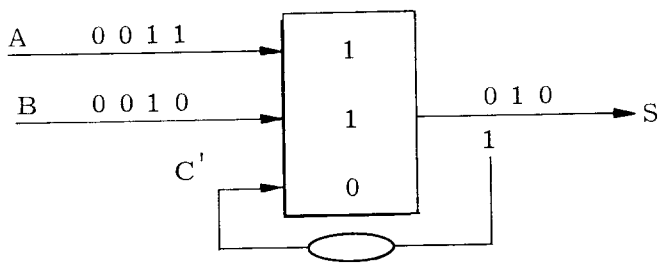
1) 1st pulse period prior to addition



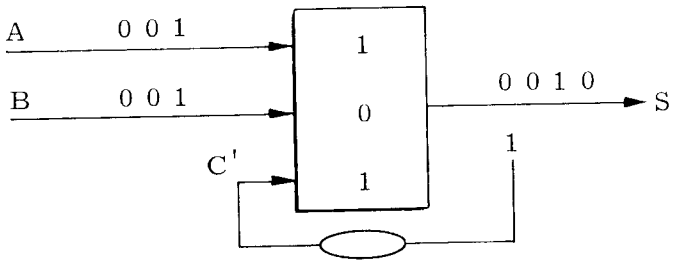
2) addition of 1st column - 1st pulse period



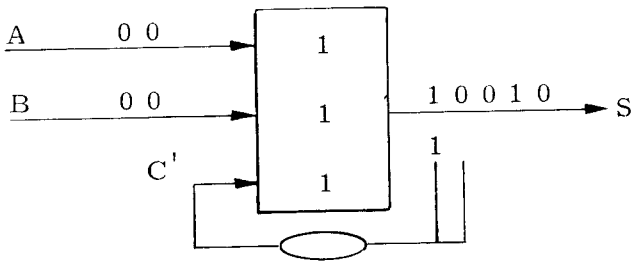
3) 2nd pulse period



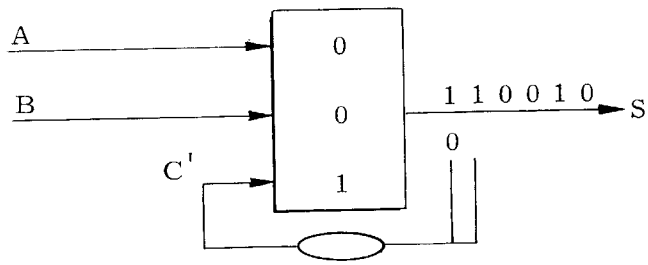
4) 3rd pulse period



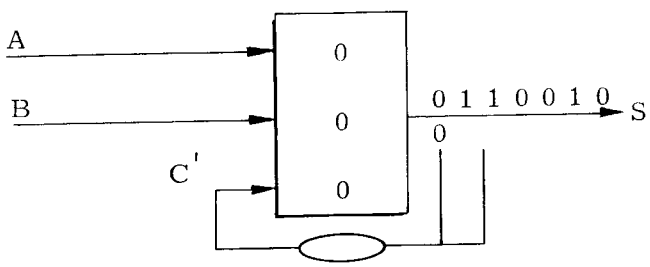
5) 4th pulse period



6) 5th pulse period



7) 6th pulse period



8) addition completed 32 + 16 + 0 + 0 + 2 + 0 = 50

Fig. 10. Steps in serial binary addition.

Set up in the form of logic charts, the Carry and Sum functions appear as shown in Fig. 11. These adding functions are easily implemented using two 3C-PAC Gating Packages, interconnected as shown in Fig. 11.

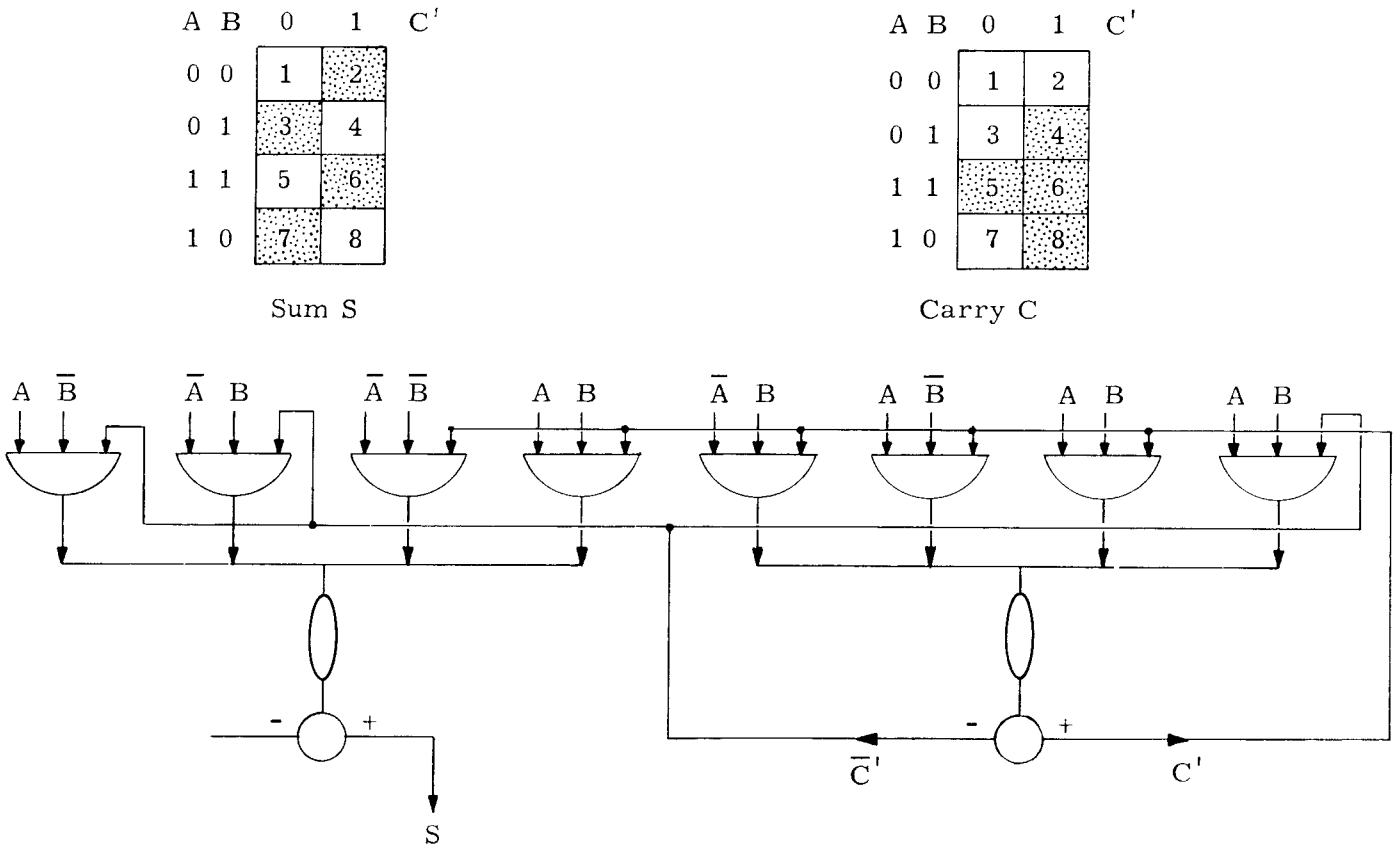


Fig. 11. Logic charts and 3C-PAC connections for serial binary addition.

8. Binary Addition - Parallel Method

The serial method of addition requires as many pulse periods of time as there are columns of digits to be added. If a faster addition is required, the parallel method may be used. This method is shown in Fig. 12. The parallel method, however, requires as many sets of Carry and Sum 3C-PACs as there are columns to be added. The two numbers to be added enter the parallel adding system of Fig. 12 repeatedly for as many pulse periods as there are carries. The correct Sum will then appear as shown. The parallel addition system of Fig. 12 can be implemented with twelve 3C-PAC Gating Packages. Signal leads of Carry and Sum packages are interconnected in the same manner shown in Fig. 11.

9. Binary Subtraction - Serial Method

Binary subtraction is performed as shown in Fig. 9 From the definitions

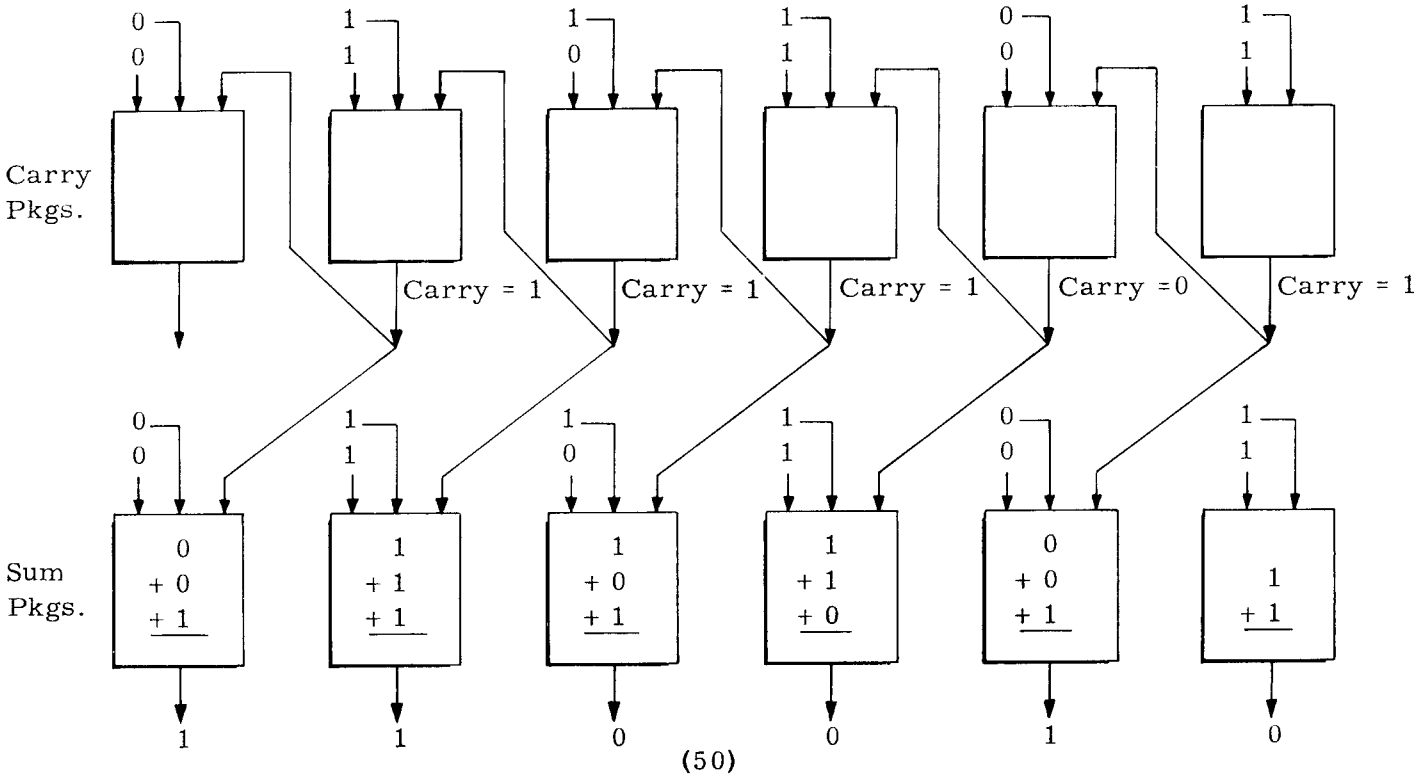
A = Incoming binary digit of minuend
B = Incoming binary digit of subtrahend
C' = Borrow digit determined by subtraction of preceding column
D = Difference digit of column being subtracted

the following logical expressions for the Borrow and Difference functions can be written

$$C = (\overline{A} \cdot \overline{B} \cdot C') \vee (\overline{A} \cdot B \cdot \overline{C}') \vee (\overline{A} \cdot B \cdot C') \vee (A \cdot B \cdot C')$$

$$D = (A \cdot \overline{B} \cdot \overline{C}') \vee (\overline{A} \cdot B \cdot \overline{C}') \vee (\overline{A} \cdot \overline{B} \cdot C') \vee (A \cdot B \cdot C')$$

The logic charts for the serial method of binary subtraction are shown in Fig. 13. These Borrow and Difference functions can be implemented with two 3C-PAC Gating Packages interconnected as shown in Fig. 13.



(50)

Fig. 12. Parallel method of binary addition.

10. Serial Adder-Subtracter with 3C-PACs

For the utmost flexibility in binary computation, both addition and subtraction can be performed with three 3C-PACs, connected as shown in Fig. 14. An additional add-subtract command function $\underline{X}$ is required. This is in the form of a pulse train which causes addition to be performed when a pulse is present, and subtraction when a pulse is absent. The variables required for the serial adder-subtracter are

A = augend or minuend
B = addend or subtrahend
C = carry or borrow (delayed from preceding pulse period)
$\underline{X}$ = add command
$\overline{\underline{X}}$ = subtract command
D = difference
S = sum
$f_1 = (X \cdot A) \vee (\overline{X} \cdot \overline{A})$ computing function

Using these variables, the serial adder-subtracter of Fig. 14 can be achieved with 3C-PACs.

A B	0	1	C′
0 0	1	2	
0 1	3	4	
1 1	5	6	
1 0	7	8	

Borrow C

A B	0	1	C′
0 0	1	2	
0 1	3	4	
1 1	5	6	
1 0	7	8	

Difference D

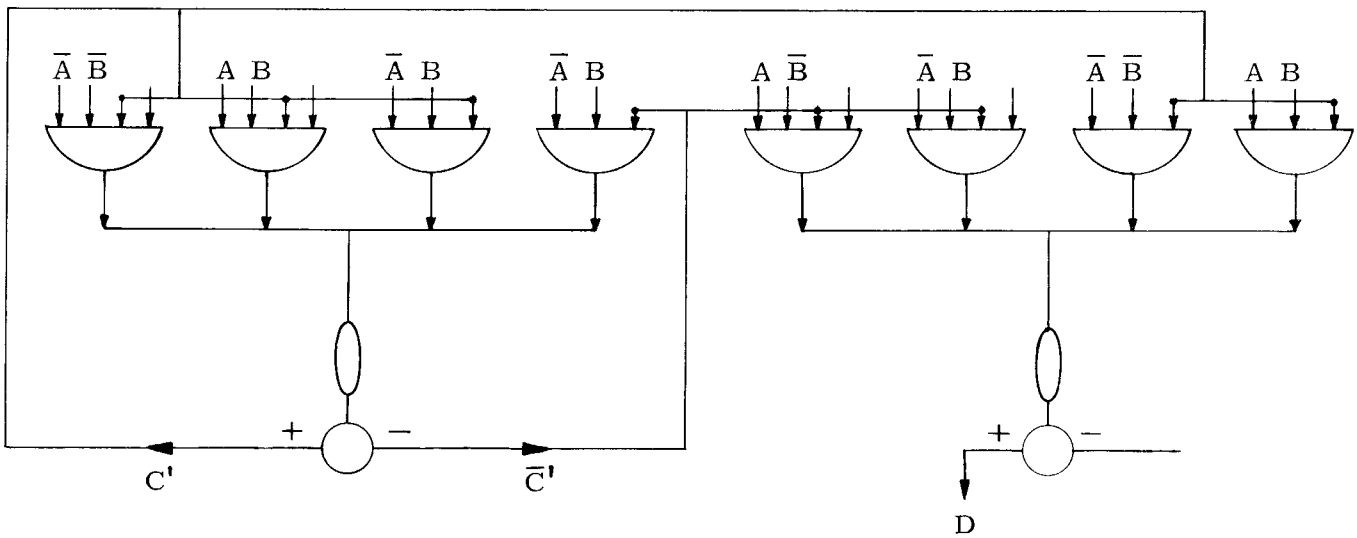


Fig. 13. Logic charts and 3C-PAC connections for serial binary subtraction.

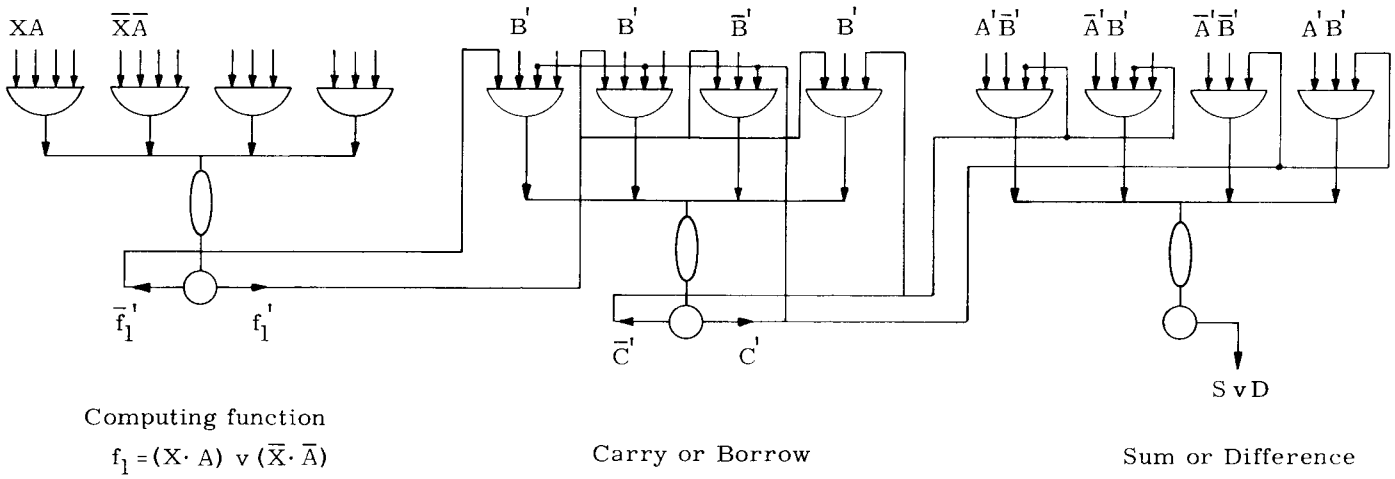


Fig. 14. Combined serial adder-subtracter with 3C-PACs.

11. Four-Variable Functions with 3C-PACs

The logic chart for a four-variable function is shown in Fig. 15. On this chart all possible functions of four variables can be defined. There are 65,536 such combinations, 51,358 of which can be implemented by a single 3C-PAC. To implement a separate square of this chart would require one 4-leg gate. Implementing all squares separately would thus require sixteen such gates. To lessen the number of gates required, however, it is possible to represent "blocks" of adjacent squares with a single gate. In defining such blocks, the first and last columns of the chart and the top and bottom rows can be considered adjacent.

Basic 4-variable logic chart

A	B	0 1	1 0	C D
0	0	1	2	3	4
0	1	5	6	7	8
1	1	9	10	11	12
1	0	13	14	15	16

Example 1: $f_1 = (A \cdot B \cdot C) \vee (A \cdot \overline{B} \cdot D) \vee (A \cdot C \cdot \overline{D}) \vee (B \cdot D)$

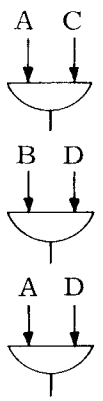
A	B	0 1	1 0	C D
0	0	1	2	3	4
0	1	5	6	7	8
1	1	9	10	11	12
1	0	13	14	15	16

f_1

Squares 10, 11, 14, 15

Squares 7, 8, 11, 12

Squares 11, 12, 15, 16



Example 2: $f_2 = A \cdot B \cdot \overline{C} \vee [\overline{A} \cdot \overline{B}] \cdot [(\overline{C} \cdot \overline{D}) \vee (C \cdot D)] \vee A \cdot \overline{B} \cdot C \cdot D$

A	B	0 1	1 0	C D
0	0	1	2	3	4
0	1	5	6	7	8
1	1	9	10	11	12
1	0	13	14	15	16

Squares 3 and 15

Squares 9 and 12

Square 1

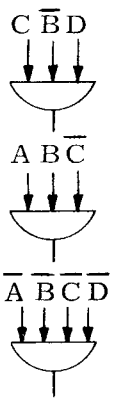


Fig. 15. Four-variable logic chart and implementation with 3C-PAC gates.

If shaded or unshaded squares are grouped together they can be implemented as follows:

(a) Eight adjacent squares forming a 2 x 4 rectangle can be represented by a 1-leg gate.

(b) Four adjacent squares forming either a 2 x 2 or a 4 x 1 rectangle can be represented by a 2-leg gate.

(c) Two adjacent squares sharing the same row or column can be represented by a 3-leg gate.

(d) Remaining single squares require a 4-leg gate for representation.

As described in previous sections, either the true (shaded) or false (unshaded) squares can be instrumented with 3C-PACs, depending upon which is the more convenient or lesser in number. If the unshaded false squares are represented, it must be remembered to reverse the output polarity of the 3C-PAC.

Examples of using the 3C-PAC for four-variable logical functions are shown in Fig. 15. Example 1 is best implemented as three sets of 2 x 2 squares. As shown, these require only three 2-leg gates. Thus the four-variable function of Example 1 can be implemented by a single 3C-PAC, using only three of the gates.

Example 2 of Fig. 15 can be implemented as two sets of two adjacent squares (2 x 1) and one single square. This complex four-variable function thus requires only three of the 3C-PAC gates, and shows how the provision of 4-leg gates increases the flexibility of the 3C-PAC.

Decimal	Binary
21	1011 = 11
x 14	x 101 = 5
21	1011
21	+ 0000
+ 21	1011
21	110111 = 55
21	
294	

Fig. 16. Decimal and binary multiplication by register shifting and addition.

```
                000110 = 6
          ------------
7 = 111 | 101010 = 42
          111
          ---
          0111
           111
           ---
           0000
           0000
           ----
```

Fig. 17. Binary division by the conventional method of trial multiplication and subtraction.

12. Multiplication, Division, Function Generation, Calculus Operations

Nearly all other mathematical operations can be reduced basically to addition and subtraction operations, and implemented with 3C-PACs as described previously. Accessory operations may be required, such as register shifting discussed in Section 13.

In Fig. 16, the multiplication of decimal and binary numbers by register shifting and addition is depicted. Binary division by the conventional method of trial multiplication (or addition) and subtraction is shown in Fig. 17. Such step-by-step computing programs are easily implemented with 3C-PACs and appropriate pulse delays.

Function generation is performed by implementing an algebraic series expansion of the desired relationship. Integration and differentiation are handled by binary numerical methods. These involve basic addition and subtraction operations, programmed by the logical interconnection of 3C-PACs.

13. Shift Registers

Registers serve as repositories for digital information. While storing such digital information, it is often desired to shift the time sequence or relative space position of the pulses. The functions of shift registers may thus include serial read-in and read-out, parallel read-in and read-out, shifting of information to right or left, etc. These operations, which can be performed with 3C-PACs, are used in parallel-to-serial and serial-to-parallel code conversion, storage of information, binary arithmetic operations, conversion of pulse information between low-speed and high-speed systems.

The variables involved in register shifting include

W = parallel read-in command
P = external information to be read-in
R_n = n^{th} bit of stored information
S_R = shift right command
S_L = shift left command

For serial read-in and read-out, the required shifting function is $(S_R \cdot P)$ for one stage, $[S_R \cdot R_{(n+1)}]$ for the n^{th} stage. For parallel read-in, the shifting function is $(W \cdot P_n)$; for parallel read-out, the output of each shifting stage is available to drive several gate inputs. For shifting information to the right, the function $[S_R \cdot R_{(n-1)}]$ is used; for shifting to the left, $[S_L \cdot R_{(n+1)}]$ is used.

The gates of a typical shift register stage are wired as shown in Fig. 18. The fourth gate $(R_n \cdot \overline{W} \cdot \overline{S}_L \cdot \overline{S}_R)$ causes the information to circulate and be stored, or "memorized", until it is desired to replace it with new information.

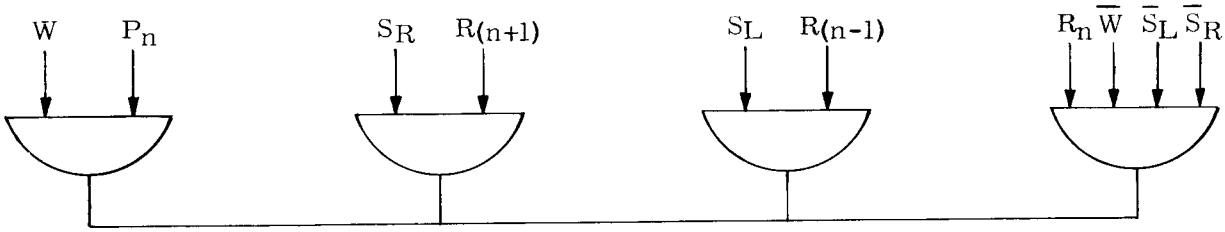


Fig. 18. Typical stage of a binary digital shift register.

Other uses for 3C-PACs include the translation of codes, such as the conversions of teletype code to IBM code, Gray code to binary code, decimal code to binary code, etc. Further reports by Computer Control engineers will discuss such utilization of these versatile 3C-PAC logical packages.