

# A NEW APPROACH TO DIGITAL COMPUTER LOGIC

FANTASTIC DO-IT-YOURSELF COMPUTER PROJECTS  
USING ONLY 4 SIMPLE ELECTRONIC CIRCUITS !!!!!

● BINARY COUNTERS

● ELECTRONIC DICE

● SHIFT REGISTERS

● COMPARATORS

● BCD COUNTERS

● BCD DECODER

● ADDERS & SUBTRACTORS

● SAMPLE & HOLD LOGIC

● MULTIPLIER

● DIVIDER

● SQUARE ROOTER

● TIME MACHINE

# LIBE



A NEW APPROACH TO

DIGITAL COMPUTER LOGIC

**IB-2**

by

Burt Libe

Copyright © 1968, 1970

by

Libe Company and Burt Libe

All rights reserved under International and Pan-American Copyright Conventions. No reproduction of this book in whole or in part may be made without written permission of the publisher. Quotations of ten lines or less are excepted, provided acknowledgement is given.

Printed in the United States of America

Library of Congress Catalog Card No.

Published by:



LIBE COMPANY  
P.O. BOX 1196  
LOS ALTOS, CALIFORNIA 94022

This book is for those who need a thorough basic discussion of digital computers and the binary number system. It is intended as a key to unlock the door to a fascinating world of the electronic computer circuits!!

## TABLE OF CONTENTS

1	INTRODUCTION
2	LOGIC OPERATIONS, NOTATION, AND SYMBOLISM
2.1	"AND" LOGIC OPERATION
2.2	"OR" LOGIC OPERATION
2.3	"SUM" LOGIC OPERATION
2.4	"EOR" (EXCLUSIVE "OR") LOGIC OPERATION
2.5	"NAND" LOGIC OPERATION
2.6	"NOR" LOGIC OPERATION
2.7	FACTUAL EXAMPLES - THE BOOLEAN APPROACH
2.8	BINARY NUMBER "FACT" REPRESENTATION
2.9	LOGIC PARENTHESES AND BRACKETS
3	FUNDAMENTAL LOGIC THEOREMS, LAWS, AND IDENTITIES
3.1	DEMORGAN'S THEOREMS
3.2	COMMUTATIVE LAWS
3.3	ASSOCIATIVE LAWS
3.4	DISTRIBUTIVE LAWS
3.5	FUNDAMENTAL IDENTITIES
3.6	NEGATION AND DOUBLE NEGATION
3.7	SPECIAL IDENTITIES
3.8	ADDITION IDENTITIES
4	TRUTH TABLES
4.1	LOGIC OPERATOR TRUTH TABLE REPRESENTATIONS
4.2	PROOF OF DEMORGAN'S THEOREMS BY TRUTH TABLES
4.3	PROOF OF DISTRIBUTIVE LAWS BY TRUTH TABLES
4.4	PROOF OF FUNDAMENTAL IDENTITIES BY TRUTH TABLES
4.5	PROOF OF ADDITION IDENTITIES BY TRUTH TABLES
5	ELECTRONIC COMPUTER LOGIC NOTATION
5.1	EXAMPLES OF ELECTRONIC COMPUTER LOGIC EXPRESSIONS
6	THE BINARY NUMBER SYSTEM
6.1	TRUTH TABLES AND THE BINARY COUNT
6.2	BINARY ADDITION
6.3	BINARY SUBTRACTION
6.4	BINARY MULTIPLICATION
6.5	BINARY DIVISION
6.6	BINARY FRACTIONALS
6.6.1	CONVERSION FROM DECIMAL FRACTIONALS TO BINARY FRACTIONALS
6.6.2	CONVERSION FROM BINARY FRACTIONALS TO DECIMAL FRACTIONALS
6.7	BINARY FRACTIONS
6.8	SQUARE ROOTS
6.8.1	EXTRACTING THE SQUARE ROOT
6.8.2	THE BINARY SQUARE ROOT
6.9	ROUNDING "UP" AND ROUNDING "DOWN"
6.9.1	BINARY "ROUNDING OFF"
6.10	THE BINARY COMPLEMENT
7	BINARY ELECTRONIC COMPUTER CIRCUITS
7.1	THE FLIP-FLOP
7.2	THE PULSE GENERATOR
7.3	LOGIC GATES
7.3.1	THE "AND" GATE
7.3.2	THE "OR" GATE

## TABLE OF CONTENTS (Continued)

7.3.3	THE "AND" AND "OR" GATE
7.3.4	NEGATIVE "AND" AND "OR" GATES
7.3.5	THE "EXCLUSIVE OR" GATE ("EOR" GATE)
7.3.6	THE "SUMMATION" GATE ("SUM" GATE)
7.3.7	THE "INVERTER" GATE ("NOT" GATE)
7.4	GATE OUTPUT "TRIGGERING" AND DIRECT DISPLAY
8	ELECTRONIC COMPUTER CIRCUIT OPERATION
8.1	CAUTIONS!!!!
8.2	PHYSICAL LAYOUT, MOUNTING, AND CONNECTIONS
8.2.1	MOUNTING BOARDS
8.2.2	WIRES
8.2.3	LABELS
8.3	LOGIC DIAGRAMS AND WIRING
8.3.1	POWER PIN CONNECTIONS
8.4	POWER SOURCES
8.5	SENSES AND COMMANDS
8.6	SET AND RESET
8.7	INTRINSIC PROBLEMS AND DEBUGGING
8.8	CARE AND REPAIR OF UNITS
9	BASIC NON-GATED COMPUTER PROJECTS
9.1	THE BINARY "UP" COUNTER
9.2	THE BINARY "DOWN" COUNTER
9.3	THE BINARY SHIFT REGISTER (LEFT)
9.4	THE BINARY SHIFT REGISTER (RIGHT)
9.5	THE COMPLEMENTARY TRANSFORMATION REGISTER
9.6	THE NON-GATED BINARY ADDER
9.7	THE NON-GATED BINARY SUBTRACTER
10	ADVANCED COMPUTER PROJECTS
10.1	THE BINARY CODED DECIMAL (BCD) COUNTER
10.2	THE LOGIC ADDER
10.3	THE LOGIC SUBTRACTER
10.4	THE SHIFT ADDER
10.5	"SAMPLE-AND-HOLD" LOGIC
10.6	GATED "UP-DOWN" COUNTER
10.7	THE FULL-LOGIC BINARY MULTIPLIER
10.8	THE CUMULATIVE-ADDITION MULTIPLIER
10.9	THE DIVIDER
10.10	THE SQUARE ROOTER
10.11	COMPARATORS
10.11.1	THE "GREATER THAN" COMPARATOR
10.11.2	THE "LESS THAN" COMPARATOR
10.11.3	THE "EQUAL TO" COMPARATOR
10.11.4	THE "UNEQUAL TO" COMPARATOR
10.11.5	THE "GREATER THAN OR EQUAL TO" COMPARATOR
10.11.6	THE "LESS THAN OR EQUAL TO" COMPARATOR
10.11.7	GENERAL COMPARISON
10.12	THE TIME MACHINE
10.12.1	THE TIME MACHINE CLOCK
10.13	ELECTRONIC DICE
10.14	THE BINARY-TO-DECIMAL DECODER

## TABLE OF CONTENTS (Continued)

11	UNLIMITED HORIZONS
12	GLOSSARY

TABLE OF CONTENTS

1	THE PAPER AIRCRAFT	10
2	THE PAPER AIRCRAFT	11
3	THE PAPER AIRCRAFT	12
4	THE PAPER AIRCRAFT	13
5	THE PAPER AIRCRAFT	14
6	THE PAPER AIRCRAFT	15
7	THE PAPER AIRCRAFT	16
8	THE PAPER AIRCRAFT	17
9	THE PAPER AIRCRAFT	18
10	THE PAPER AIRCRAFT	19
11	THE PAPER AIRCRAFT	20
12	THE PAPER AIRCRAFT	21
13	THE PAPER AIRCRAFT	22
14	THE PAPER AIRCRAFT	23
15	THE PAPER AIRCRAFT	24
16	THE PAPER AIRCRAFT	25
17	THE PAPER AIRCRAFT	26
18	THE PAPER AIRCRAFT	27
19	THE PAPER AIRCRAFT	28
20	THE PAPER AIRCRAFT	29
21	THE PAPER AIRCRAFT	30
22	THE PAPER AIRCRAFT	31
23	THE PAPER AIRCRAFT	32
24	THE PAPER AIRCRAFT	33
25	THE PAPER AIRCRAFT	34
26	THE PAPER AIRCRAFT	35
27	THE PAPER AIRCRAFT	36
28	THE PAPER AIRCRAFT	37
29	THE PAPER AIRCRAFT	38
30	THE PAPER AIRCRAFT	39
31	THE PAPER AIRCRAFT	40
32	THE PAPER AIRCRAFT	41
33	THE PAPER AIRCRAFT	42
34	THE PAPER AIRCRAFT	43
35	THE PAPER AIRCRAFT	44
36	THE PAPER AIRCRAFT	45
37	THE PAPER AIRCRAFT	46
38	THE PAPER AIRCRAFT	47
39	THE PAPER AIRCRAFT	48
40	THE PAPER AIRCRAFT	49
41	THE PAPER AIRCRAFT	50
42	THE PAPER AIRCRAFT	51
43	THE PAPER AIRCRAFT	52
44	THE PAPER AIRCRAFT	53
45	THE PAPER AIRCRAFT	54
46	THE PAPER AIRCRAFT	55
47	THE PAPER AIRCRAFT	56
48	THE PAPER AIRCRAFT	57
49	THE PAPER AIRCRAFT	58
50	THE PAPER AIRCRAFT	59
51	THE PAPER AIRCRAFT	60
52	THE PAPER AIRCRAFT	61
53	THE PAPER AIRCRAFT	62
54	THE PAPER AIRCRAFT	63
55	THE PAPER AIRCRAFT	64
56	THE PAPER AIRCRAFT	65
57	THE PAPER AIRCRAFT	66
58	THE PAPER AIRCRAFT	67
59	THE PAPER AIRCRAFT	68
60	THE PAPER AIRCRAFT	69
61	THE PAPER AIRCRAFT	70
62	THE PAPER AIRCRAFT	71
63	THE PAPER AIRCRAFT	72
64	THE PAPER AIRCRAFT	73
65	THE PAPER AIRCRAFT	74
66	THE PAPER AIRCRAFT	75
67	THE PAPER AIRCRAFT	76
68	THE PAPER AIRCRAFT	77
69	THE PAPER AIRCRAFT	78
70	THE PAPER AIRCRAFT	79
71	THE PAPER AIRCRAFT	80
72	THE PAPER AIRCRAFT	81
73	THE PAPER AIRCRAFT	82
74	THE PAPER AIRCRAFT	83
75	THE PAPER AIRCRAFT	84
76	THE PAPER AIRCRAFT	85
77	THE PAPER AIRCRAFT	86
78	THE PAPER AIRCRAFT	87
79	THE PAPER AIRCRAFT	88
80	THE PAPER AIRCRAFT	89
81	THE PAPER AIRCRAFT	90
82	THE PAPER AIRCRAFT	91
83	THE PAPER AIRCRAFT	92
84	THE PAPER AIRCRAFT	93
85	THE PAPER AIRCRAFT	94
86	THE PAPER AIRCRAFT	95
87	THE PAPER AIRCRAFT	96
88	THE PAPER AIRCRAFT	97
89	THE PAPER AIRCRAFT	98
90	THE PAPER AIRCRAFT	99
91	THE PAPER AIRCRAFT	100



## 1. INTRODUCTION

This book represents an original new basic approach to computer logic to communicate the enjoyment of this vast field to the beginning experimenter. By making repeated use of only four simple LIBE\* electronic circuits, the computer hobbyist can derive great satisfaction from building up the many projects described in this text. These four circuits are the basic keys to the world of the computer!

The logic symbolism used in the text was not chosen to totally agree with that used in other publications, but to communicate basic concepts to the reader. The notation and symbolism contained herein will be used for all future LIBE literature, papers, and discussions on computer logic and circuits.

All information is presented in a direct manner for use as a textbook or handbook reference. The reader is further encouraged to seek out other books after becoming familiar with the contents of this publication. Readers partially familiar with some of the material in this book may skip over to the sections which are of interest.

\* T.M.

## 2. LOGIC OPERATIONS, NOTATION AND SYMBOLISM

Logic is the SCIENCE OF REASONING. In other words, a set of FACTS are used to reason out a CONCLUSION. The computer logic GATE uses these facts (INPUTS) and determines a conclusion (OUTPUT).

The logic functions discussed here are: "AND", "OR", "SUM", "EOR", "NAND", and "NOR". Discussion of this section is limited to logic functions only. Section 7 contains an intensive discussion on gates.

LOGIC NOTATION consists mainly of the "dot" ( $\cdot$ ), the "wedge" ( $\wedge$ ), the "plus" ( $+$ ), the "triangle" ( $\nabla$ ), the "equal" ( $=$ ), and the overhead "bar" ( $\overline{\quad}$ ). The first five symbols will be referred to, throughout the rest of this book, as LOGIC OPERATORS. Parentheses, brackets, braces, and overhead bars are used to group together specific parts of logic expressions. The "equal" sign denotes logical equality. The overhead "bar" is used to denote a "FALSE" logic FACT which is opposite to the corresponding "TRUE" fact. The entire expression contained under an overhead "bar" is "FALSE". Examples are:

$$\begin{aligned}\overline{A} &= \text{"not" } A \\ \overline{B} &= \text{"not" } B \\ \overline{A \cdot B \cdot C} &= \text{"not" } (A \text{ and } B \text{ and } C)\end{aligned}$$

### 2.1 "AND" LOGIC OPERATION

"AND" logic operation is denoted by the "dot" ( $\cdot$ ). We denote the "AND" operation on four facts (A, B, C, D)\* as follows:

$$A \cdot B \cdot C \cdot D$$

If we consider the above expression "A and B and C and D", we draw the following conclusion: If and only if all facts are "TRUE", then the conclusion is "TRUE". If one or more facts are "FALSE", then the conclusion is "FALSE". Examples are:

$$\begin{aligned}A \cdot B \cdot C \cdot D &= \text{conclusion (TRUE)} \\ A \cdot \overline{B} \cdot C \cdot D &= \text{conclusion (FALSE)} \\ \overline{A} \cdot \overline{B} \cdot C \cdot D &= \text{conclusion (FALSE)} \\ \overline{A} \cdot \overline{B} \cdot \overline{C} \cdot \overline{D} &= \text{conclusion (FALSE)}\end{aligned}$$

The "AND" operation, in effect, answers the question: "Are all the facts 'TRUE'?"

### 2.2 "OR" LOGIC OPERATION

"OR" logic operation is denoted by the "wedge" ( $\vee$ ). We denote the "or" operation on four facts (A, B, C, D) as follows:

$$A \vee B \vee C \vee D$$

If we consider the expression "A or B or C or D" above, we draw the following conclusion: If and only if all facts are "FALSE", then the conclusion is "FALSE". If one or more facts are "TRUE", then the conclusion is "TRUE". Examples are:

\*Each letter A, B, C, D stands for a specific fact.

## 2.2 "OR" LOGIC OPERATION (Continued)

$$\overline{A}\overline{B}\overline{C}\overline{D} = \text{conclusion (FALSE)}$$

$$A\overline{B}\overline{C}\overline{D} = \text{conclusion (TRUE)}$$

$$A\overline{B}\overline{C}D = \text{conclusion (TRUE)}$$

$$\overline{A}\overline{B}\overline{C}D = \text{conclusion (TRUE)}$$

The "OR" operation, in effect, answers the question: "Is at least one fact 'TRUE'?"

## 2.3 "SUM" LOGIC OPERATION

The "SUM" logic operation is denoted by the "plus" (+). We denote the "sum" operation on four facts (A, B, C, D) as follows:

$$A+B+C+D$$

If we consider the expression "A plus B plus C plus D" above, we draw the following conclusion: If and only if an odd number of facts are "TRUE", then the conclusion is "TRUE". If an even number of facts are "TRUE", then the conclusion is "FALSE". The conclusion, in this case, may be considered the PARITY (the condition of being even or odd) of "TRUE" facts. The conclusion depends only on an even or odd number of "TRUE" facts. Examples are:

$$A+B+C+D = \text{conclusion (FALSE)}$$

$$A+\overline{B}+C+D = \text{conclusion (TRUE)}$$

$$A+\overline{B}+\overline{C}+D = \text{conclusion (FALSE)}$$

$$\overline{A}+\overline{B}+\overline{C}+D = \text{conclusion (TRUE)}$$

$$\overline{A}+\overline{B}+C+D = \text{conclusion (FALSE)}$$

$$\overline{A}+\overline{B}+\overline{C}+\overline{D} = \text{conclusion (FALSE)}$$

The "SUM" operation, in effect, answers the question: "Is the number of 'TRUE' facts odd?"

## 2.4 "EOR" (EXCLUSIVE "OR") LOGIC OPERATION

The "EOR" logic operation is denoted by the "triangle" ( $\nabla$ ). We denote the "eor" operation on four facts (A, B, C, D) as follows:

$$A\nabla B\nabla C\nabla D$$

If we consider the expression "A eor B eor C eor D" above, we draw the following conclusion: If and only if just one fact is "TRUE", then the conclusion is "TRUE". However, if all facts are "FALSE", or more than one fact is "TRUE", then the conclusion is "FALSE". Examples are:

$$A\nabla B\nabla C\nabla D = \text{conclusion (FALSE)}$$

$$\overline{A}\nabla\overline{B}\nabla\overline{C}\nabla\overline{D} = \text{conclusion (TRUE)}$$

$$\overline{A}\nabla\overline{B}\nabla C\nabla\overline{D} = \text{conclusion (TRUE)}$$

$$\overline{A}\nabla\overline{B}\nabla\overline{C}\nabla\overline{D} = \text{conclusion (FALSE)}$$

$$A\nabla B\nabla C\nabla\overline{D} = \text{conclusion (FALSE)}$$

#### 2.4 "EOR" (EXCLUSIVE "OR") LOGIC OPERATION (Continued)

The "EOR" operation, in effect, answers the question: "Is there only one 'TRUE' fact?"

#### 2.5 "NAND" LOGIC OPERATION

"NAND" logic is simply the negative of "AND" logic operation. "NAND" logic operation is denoted by a "dot" and an overhead "bar" across the entire expression. The "NAND" operation on four facts is denoted as follows:

$$\overline{A \cdot B \cdot C \cdot D}$$

If we consider the above expression "not (A and B and C and D)", we draw the following conclusion: If and only if all facts are "TRUE", then the conclusion is "FALSE". If one or more facts are "FALSE", then the conclusion is "TRUE". Examples are:

$$\overline{A \cdot B \cdot C \cdot D} = \text{conclusion (FALSE)}$$

$$\overline{A \cdot \overline{B} \cdot C \cdot D} = \text{conclusion (TRUE)}$$

$$\overline{A \cdot \overline{B} \cdot C \cdot \overline{D}} = \text{conclusion (TRUE)}$$

$$\overline{A \cdot \overline{B} \cdot \overline{C} \cdot \overline{D}} = \text{conclusion (TRUE)}$$

The "NAND" operation, in effect, answers the question: "Is at least one fact 'FALSE'?"

"NAND" is a combination of the two words "NOT AND". Further discussion of this "negative" logic will be held to a minimum because the positive "AND" logic is easier to explain. "NAND" logic permits the use of simpler electronic gate circuitry.

#### 2.6 "NOR" LOGIC OPERATION

"NOR" logic is simply the negative of "OR" logic operation. "NOR" logic operation is denoted by a "wedge" and an overhead "bar" across the entire expression. The "NOR" operation on four facts is denoted as follows:

$$\overline{A \vee B \vee C \vee D}$$

If we consider the above expression "not (A or B or C or D)", we draw the following conclusion: If and only if all facts are "FALSE", then the conclusion is "TRUE". If one or more facts are "TRUE", then the conclusion is "FALSE". Examples are:

$$\overline{\overline{A} \vee \overline{B} \vee \overline{C} \vee \overline{D}} = \text{conclusion (TRUE)}$$

$$\overline{A \vee \overline{B} \vee C \vee \overline{D}} = \text{conclusion (FALSE)}$$

$$\overline{A \vee \overline{B} \vee \overline{C} \vee D} = \text{conclusion (FALSE)}$$

$$\overline{A \vee \overline{B} \vee C \vee D} = \text{conclusion (FALSE)}$$

The "NOR" operation, in effect, answers the question: "Are all facts 'FALSE'?"

"NOR" is a combination of the two words "NOT OR". Further discussion of this "negative" logic will be held to a minimum because the positive "OR" logic is easier to explain. "NOR" logic permits the use of simpler electronic gate circuitry.

## 2.7 FACTUAL EXAMPLES-THE BOOLEAN APPROACH

The BOOLEAN APPROACH to logic is to impose the condition that all logic statements, reasons, conclusions, facts, etc., are either "TRUE" or "FALSE". There are no conditional steps between "TRUE" and "FALSE".

Now, instead of using just letters, let us assign some statements of fact to the letters A, B, C, and D.

- A: Arizona is in the United States.
- B: Boston is in New York.
- C: California is on the West Coast.
- D: Denver is in Colorado.

Now, through research, suppose we determine that A is "TRUE", B is "FALSE", C is "TRUE", and D is "TRUE". We would have to replace " $\bar{B}$ " for statement (fact) "B" to denote that it is "FALSE".

Consider:  $A \cdot \bar{B} \cdot C \cdot D =$  conclusion (FALSE). The conclusion represents a decision on the collection of information about the United States. A "FALSE" conclusion for the "AND" function means that not all the facts are "TRUE".

Now consider:  $A \vee \bar{B} \vee C \vee D =$  conclusion (TRUE). A "TRUE" conclusion for the "OR" logic function means that one or more facts are "TRUE".

Consider:  $A + \bar{B} + C + D =$  conclusion (TRUE). A "TRUE" conclusion for the "SUM" logic function means that an odd number (either 1 or 3 in this case) of facts are "TRUE".

Consider:  $A \vee \bar{B} \vee C \vee D =$  conclusion (FALSE). A "FALSE" conclusion for the "EOR" logic function means that more than one fact or none of the facts are "TRUE".

It can be seen from the above examples that logic operations are basic tools for making decisions and evaluations about a group of facts without really knowing the details and/or validity of each fact. The CONCLUSION represents a specific decision or evaluation. In the examples above, the statement and validity of each fact was noted for reference only. The CONCLUSION is also of a more general nature and it is not usually possible to use CONCLUSION information to derive information about a specific fact. Note the "FALSE" conclusion for the "AND" operation on the four facts. Noting the conclusion, we know that not all facts are "TRUE". However, it is not possible to tell from this conclusion exactly which facts are "TRUE" and which are "FALSE". The same holds true for all the other logic operations.

## 2.8 BINARY NUMBER "FACT" REPRESENTATION

The BINARY number system is a number system with 2 as the number base. A BINARY NUMBER can be only a "1" or a "0". Since a computer only "sees" a "1" or a "0", we will employ these two numbers to represent the validity of facts as follows:

"1" = "TRUE"

"0" = "FALSE"

The above numerical representations will be used wherever possible.

## 2.9 LOGIC PARENTHESES AND BRACKETS

Logic PARENTHESES and BRACKETS are very important because they keep the logic statement clear when an expression can be interpreted more than one way. Consider the following example:

$$A \vee B \cdot C$$

What does the above expression mean? Is it the quantity A "OR" B, "AND" C? Or is it A "OR" the quantity B "AND" C? These two expressions are not logically equivalent, as can be seen from the "truth table" method of proof in chapter 4. (The reader can verify this.) In order to clear up this confusion, we must use PARENTHESES to represent the two different expressions respectively:

$$(1) \quad (A \vee B) \cdot C$$

$$(2) \quad A \vee (B \cdot C)$$

It is very important that parentheses be used when confusion can occur. In some cases, where the Associative Law (see chapter 3) holds for certain expressions, the parentheses may be left out. Example:  $(A \vee B) \vee [(C \vee D) \vee E] = A \vee B \vee C \vee D \vee E$ . If more than one set of parentheses is needed in the same expression, then various forms of different brackets may be used such as: "[ ]" or "{ }" or "( )". Consider the following example:

$$(A+B)+C = \left\{ \left[ (A \cdot \bar{B}) \vee (\bar{A} \cdot B) \right] \cdot \bar{C} \right\} \vee \left\{ (A \cdot \bar{B}) \vee (\bar{A} \cdot B) \cdot C \right\}$$

When simplifying expressions such as above, the ASSOCIATIVE LAWS will be used (chapter 3) to clear the parentheses.

### 3. FUNDAMENTAL LOGIC THEOREMS, LAWS, AND IDENTITIES

The fundamental theorems, laws, and identities presented here are the necessary "tools" for all logical manipulations and problem work. Each group is classified together with a brief explanation followed by a tabulated presentation. These logical equalities should be referred to when necessary and will be referred to in this text by their corresponding identification numbers on the right.

#### 3.1 DEMORGAN'S THEOREMS

The two expressions called "DeMorgan's Theorems" are the most important laws for computer logic. They define the relationship between the "AND" and "OR" logic operations.

The first theorem defines the negative of a group of "AND" logic facts to be equal to the corresponding group of "OR" of the negatives of these respective logic facts.

The second theorem defines the negative of a group of "OR" logic facts to be equal to the corresponding group of "AND" of the negatives of these respective logic facts.

Specific examples for two facts A, B are:

$$\overline{A \cdot B} = \overline{A} \vee \overline{B} \quad (\text{DM \#1})$$

$$\overline{A \vee B} = \overline{A} \cdot \overline{B} \quad (\text{DM \#2})$$

For 3 facts A, B, C we have:

$$\overline{A \cdot B \cdot C} = \overline{A} \vee \overline{B} \vee \overline{C} \quad (\text{DM \#3})$$

$$\overline{A \vee B \vee C} = \overline{A} \cdot \overline{B} \cdot \overline{C} \quad (\text{DM \#4})$$

For 4 facts A, B, C, D we have:

$$\overline{A \cdot B \cdot C \cdot D} = \overline{A} \vee \overline{B} \vee \overline{C} \vee \overline{D} \quad (\text{DM \#5})$$

$$\overline{A \vee B \vee C \vee D} = \overline{A} \cdot \overline{B} \cdot \overline{C} \cdot \overline{D} \quad (\text{DM \#6})$$

#### 3.2 COMMUTATIVE LAWS

The commutative laws refer to the order of the logic facts. To commute logic facts means to change the order of either two or more facts. Note that equality holds for the four logic operators discussed in this book.

Specific examples for 2 facts A, B are:

$$A \cdot B = B \cdot A \quad (\text{CL \#1})$$

$$A \vee B = B \vee A \quad (\text{CL \#2})$$

$$A + B = B + A \quad (\text{CL \#3})$$

$$A \nabla B = B \nabla A \quad (\text{CL \#4})$$

For 3 facts A, B, C we have:

$$A \cdot B \cdot C = A \cdot C \cdot B = B \cdot A \cdot C = B \cdot C \cdot A = C \cdot A \cdot B = C \cdot B \cdot A \quad (\text{CL \#5})$$

$$A \vee B \vee C = A \vee C \vee B = B \vee A \vee C = B \vee C \vee A = C \vee A \vee B = C \vee B \vee A \quad (\text{CL \#6})$$

$$A + B + C = A + C + B = B + A + C = B + C + A = C + A + B = C + B + A \quad (\text{CL \#7})$$

$$A \nabla B \nabla C = A \nabla C \nabla B = B \nabla A \nabla C = B \nabla C \nabla A = C \nabla A \nabla B = C \nabla B \nabla A \quad (\text{CL \#8})$$

### 3.3 ASSOCIATIVE LAWS

The associative laws refer to the grouping of logic facts. The use of logical parentheses, brackets, and the overhead bar are very important in order to show the grouping of logic facts. As there is only one way to group two facts, we must start with three. Note that three facts can be grouped by two's, or as a single group of three. Note the inequality ( $\neq$ ) for the "EOR" operator ( $\nabla$ ).

Specific examples for 3 facts A, B, C are:

$$A \cdot (B \cdot C) = (A \cdot B) \cdot C = A \cdot B \cdot C \quad (\text{AL \#1})$$

$$A \vee (B \vee C) = (A \vee B) \vee C = A \vee B \vee C \quad (\text{AL \#2})$$

$$A + (B + C) = (A + B) + C = A + B + C \quad (\text{AL \#3})$$

$$A \nabla (B \nabla C) = (A \nabla B) \nabla C \neq A \nabla B \nabla C \quad (\text{AL \#4})$$

### 3.4 DISTRIBUTIVE LAWS

The distributive laws define "logical multiplication." This term is used because of the analogy to numerical multiplication. Consider the following numerical example:

$$2x(3+4) = (2x3)+(2x4)$$

Note the position of the "times" sign (x) and the "plus" sign on both sides of the equality. In "logical multiplication" we replace the "times" sign and "plus" sign with a logic operator. However, the respective positions of the logic operators must be the same as the corresponding "times" and "plus" signs. (Note: numerical "plus" and logic "plus" signs should not be confused.) The quantity to the right of the "equal" sign is the distributed expression, while the quantity to the left is the factored expression. We need at least three logic facts in order to form a "distributed" expression. The twelve examples shown for the three logic facts A, B, C show all the possibilities with the four logic operators used in this book. Note that only the first four examples are an equality. The distributive law does not hold for the other eight expressions, hence the inequalities ( $\neq$ ).

Specific examples for 3 facts A, B, C are:

$$A \cdot (B \vee C) = (A \cdot B) \vee (A \cdot C) \quad (\text{DL \#1})$$

$$A \cdot (B + C) = (A \cdot B) + (A \cdot C) \quad (\text{DL \#2})$$

$$A \cdot (B \nabla C) = (A \cdot B) \nabla (A \cdot C) \quad (\text{DL \#3})$$

$$A \vee (B \cdot C) = (A \vee B) \cdot (A \vee C) \quad (\text{DL \#4})$$

$$A \vee (B + C) \neq (A \vee B) + (A \vee C) \quad (\text{DL \#5})$$

$$A \vee (B \nabla C) \neq (A \vee B) \nabla (A \vee C) \quad (\text{DL \#6})$$

$$A + (B \cdot C) \neq (A + B) \cdot (A + C) \quad (\text{DL \#7})$$

$$A + (B \vee C) \neq (A + B) \vee (A + C) \quad (\text{DL \#8})$$

$$A + (B \nabla C) \neq (A + B) \nabla (A + C) \quad (\text{DL \#9})$$

$$A \nabla (B \cdot C) \neq (A \nabla B) \cdot (A \nabla C) \quad (\text{DL \#10})$$

$$A \nabla (B \vee C) \neq (A \nabla B) \vee (A \nabla C) \quad (\text{DL \#11})$$

$$A \nabla (B + C) \neq (A \nabla B) + (A \nabla C) \quad (\text{DL \#12})$$



### 3.5 FUNDAMENTAL IDENTITIES

The fundamental identities define the relationship between two logic quantities. Either or both of the quantities may be a logic fact (indicated by the letter A), its converse ("not A", or  $\bar{A}$ ), the number "1", or the number "0". Note that the "plus" (+) and "triangle" ( $\nabla$ ) operators are the same for two logic quantities. The interrelationship between logic facts and binary numbers is shown here.

$A \cdot A = A$	(FI #1)	$A + A = 0$	(FI #33)
$A \cdot \bar{A} = 0$	(FI #2)	$A + \bar{A} = 1$	(FI #34)
$A \cdot 1 = A$	(FI #3)	$A + 1 = \bar{A}$	(FI #35)
$A \cdot 0 = 0$	(FI #4)	$\bar{A} + 0 = A$	(FI #36)
$\bar{A} \cdot A = 0$	(FI #5)	$\bar{A} + A = 1$	(FI #37)
$\bar{A} \cdot \bar{A} = \bar{A}$	(FI #6)	$\bar{A} + \bar{A} = 0$	(FI #38)
$\bar{A} \cdot 1 = \bar{A}$	(FI #7)	$\bar{A} + 1 = A$	(FI #39)
$\bar{A} \cdot 0 = 0$	(FI #8)	$\bar{A} + 0 = \bar{A}$	(FI #40)
$1 \cdot A = A$	(FI #9)	$1 + A = \bar{A}$	(FI #41)
$1 \cdot \bar{A} = \bar{A}$	(FI #10)	$1 + \bar{A} = A$	(FI #42)
$1 \cdot 1 = 1$	(FI #11)	$1 + 1 = 0$	(FI #43)
$1 \cdot 0 = 0$	(FI #12)	$1 + 0 = 1$	(FI #44)
$0 \cdot \bar{A} = 0$	(FI #13)	$0 + A = A$	(FI #45)
$0 \cdot A = 0$	(FI #14)	$0 + \bar{A} = \bar{A}$	(FI #46)
$0 \cdot 1 = 0$	(FI #15)	$0 + 1 = 1$	(FI #47)
$0 \cdot 0 = 0$	(FI #16)	$0 + 0 = 0$	(FI #48)
$A \nabla A = A$	(FI #17)	$A \nabla A = 0$	(FI #49)
$A \nabla \bar{A} = 1$	(FI #18)	$A \nabla \bar{A} = 1$	(FI #50)
$A \nabla 1 = 1$	(FI #19)	$A \nabla 1 = \bar{A}$	(FI #51)
$A \nabla 0 = A$	(FI #20)	$A \nabla 0 = A$	(FI #52)
$\bar{A} \nabla A = 1$	(FI #21)	$\bar{A} \nabla A = 1$	(FI #53)
$\bar{A} \nabla \bar{A} = \bar{A}$	(FI #22)	$\bar{A} \nabla \bar{A} = 0$	(FI #54)
$\bar{A} \nabla 1 = 1$	(FI #23)	$\bar{A} \nabla 1 = A$	(FI #55)
$\bar{A} \nabla 0 = \bar{A}$	(FI #24)	$\bar{A} \nabla 0 = \bar{A}$	(FI #56)
$1 \nabla A = 1$	(FI #25)	$1 \nabla A = \bar{A}$	(FI #57)
$1 \nabla \bar{A} = 1$	(FI #26)	$1 \nabla \bar{A} = A$	(FI #58)
$1 \nabla 1 = 1$	(FI #27)	$1 \nabla 1 = 0$	(FI #59)
$1 \nabla 0 = 1$	(FI #28)	$1 \nabla 0 = 1$	(FI #60)
$0 \nabla A = A$	(FI #29)	$0 \nabla A = A$	(FI #61)
$0 \nabla \bar{A} = \bar{A}$	(FI #30)	$0 \nabla \bar{A} = \bar{A}$	(FI #62)
$0 \nabla 1 = 1$	(FI #31)	$0 \nabla 1 = 1$	(FI #63)
$0 \nabla 0 = 0$	(FI #32)	$0 \nabla 0 = 0$	(FI #64)

### 3.6 NEGATION AND DOUBLE NEGATION

Simple negation employs the use of the overhead "bar" to denote "not A" when placed over logic fact A. This quantity ( $\bar{A}$ ) is called the "negative of A", the "converse of A", or the "invert of A".

Double negation means to invert A twice, or to invert the negative of A ( $\bar{\bar{A}}$ ). However, if we invert a "negative", it returns to a "positive". Thus, the law of double negation is written as follows:

$$\bar{\bar{A}} = A \quad (\text{DN \#1})$$

### 3.7 SPECIAL IDENTITIES

Logic identities other than those in the preceding categories are listed here.

The first is the equality of the "eor" and "sum" for two quantities. This is expressed as follows:

$$A+B = A \vee B \quad (\text{SI \#1})$$

For 3 quantities A, B, C we must use parentheses as follows:

$$A+B+C = A \vee (B \vee C) = (A \vee B) \vee C \quad (\text{SI \#2})$$

For 4 quantities A, B, C, D we must use parentheses as follows:

$$A+B+C+D = (A \vee B) \vee (C \vee D) \quad (\text{SI \#3})$$

For 5 quantities A, B, C, D, E we must use parentheses as follows:

$$A+B+C+D+E = [(A \vee B) \vee (C \vee D)] \vee E = (A \vee B) \vee [(C \vee D) \vee E] = (C \vee D) \vee [(A \vee B) \vee E] \quad (\text{SI \#4})$$

### 3.8 ADDITION IDENTITIES

The most important addition identities for 2 quantities (A, B) are:

$$S = A+B = (A \cdot \bar{B}) \vee (\bar{A} \cdot B) \quad (\text{AI \#1})$$

$$S = A+B = (A \vee B) \cdot (\bar{A} \vee \bar{B}) \quad (\text{AI \#2})$$

These expressions will be used throughout the book to expand and simplify logic expressions.

The following addition identities are entered in this section for reference only. They will be further discussed in chapter 10. Note how complicated the expressions become as the number of logic quantities increases.

"S" means "SUM"

"C<sub>1</sub>" means "CARRY-ONE"

"C<sub>2</sub>" means "CARRY-TWO"

The above notation will be explained later.

## 3.8 ADDITION IDENTITIES (Continued)

For 2 quantities A, B:

$$S = A+B = A \vee B \quad (\text{AI \#3})$$

$$S = (A \cdot \bar{B}) \vee (\bar{A} \cdot B) \quad (\text{AI \#4})$$

$$S = (A \vee B) \cdot (\bar{A} \vee \bar{B}) \quad (\text{AI \#5})$$

$$C_1 = A \cdot B \quad (\text{AI \#6})$$

For 3 quantities A, B, C:

$$S = A+B+C \quad (\text{AI \#7})$$

$$S = (A \cdot B \cdot C) \vee (A \cdot \bar{B} \cdot \bar{C}) \vee (\bar{A} \cdot B \cdot \bar{C}) \vee (\bar{A} \cdot \bar{B} \cdot C) \quad (\text{AI \#8})$$

$$C_1 = (A \cdot B) \vee (A \cdot C) \vee (B \cdot C) \quad (\text{AI \#9})$$

For 4 quantities A, B, C, D:

$$S = A+B+C+D \quad (\text{AI \#10})$$

$$S = (A \cdot B \cdot C \cdot \bar{D}) \vee (A \cdot B \cdot \bar{C} \cdot D) \vee (A \cdot \bar{B} \cdot C \cdot D) \vee (\bar{A} \cdot B \cdot C \cdot D) \vee$$

$$(A \cdot \bar{B} \cdot \bar{C} \cdot \bar{D}) \vee (\bar{A} \cdot B \cdot \bar{C} \cdot \bar{D}) \vee (\bar{A} \cdot \bar{B} \cdot C \cdot \bar{D}) \vee (\bar{A} \cdot \bar{B} \cdot \bar{C} \cdot D) \quad (\text{AI \#11})$$

$$C_1 = (A \cdot B \cdot \bar{D}) \vee (A \cdot C \cdot \bar{D}) \vee (A \cdot \bar{C} \cdot D) \vee (A \cdot \bar{B} \cdot D) \vee (\bar{A} \cdot B \cdot C) \vee$$

$$(\bar{A} \cdot B \cdot D) \vee (\bar{A} \cdot C \cdot D) \quad (\text{AI \#12})$$

$$C_2 = A \cdot B \cdot C \cdot D \quad (\text{AI \#13})$$

For 5 quantities A, B, C, D, E:

$$S = A+B+C+D+E \quad (\text{AI \#14})$$

$$S = (A \cdot B \cdot C \cdot D \cdot E) \vee (A \cdot B \cdot C \cdot \bar{D} \cdot \bar{E}) \vee (A \cdot B \cdot \bar{C} \cdot \bar{D} \cdot E) \vee$$

$$(A \cdot \bar{B} \cdot \bar{C} \cdot D \cdot E) \vee (\bar{A} \cdot \bar{B} \cdot C \cdot D \cdot E) \vee (A \cdot B \cdot \bar{C} \cdot D \cdot \bar{E}) \vee$$

$$(A \cdot \bar{B} \cdot C \cdot \bar{D} \cdot E) \vee (\bar{A} \cdot B \cdot \bar{C} \cdot D \cdot E) \vee (A \cdot \bar{B} \cdot C \cdot D \cdot \bar{E}) \vee$$

$$(\bar{A} \cdot B \cdot C \cdot \bar{D} \cdot E) \vee (\bar{A} \cdot B \cdot C \cdot D \cdot \bar{E}) \vee (A \cdot \bar{B} \cdot \bar{C} \cdot \bar{D} \cdot \bar{E}) \vee$$

$$(\bar{A} \cdot B \cdot \bar{C} \cdot \bar{D} \cdot \bar{E}) \vee (\bar{A} \cdot \bar{B} \cdot C \cdot \bar{D} \cdot \bar{E}) \vee (\bar{A} \cdot \bar{B} \cdot \bar{C} \cdot D \cdot \bar{E}) \vee$$

$$(\bar{A} \cdot \bar{B} \cdot \bar{C} \cdot \bar{D} \cdot E) \quad (\text{AI \#15})$$

$$C_1 = (A \cdot B \cdot \bar{C} \cdot \bar{D}) \vee (A \cdot C \cdot \bar{D} \cdot \bar{E}) \vee (A \cdot \bar{C} \cdot D \cdot \bar{E}) \vee (A \cdot \bar{B} \cdot \bar{C} \cdot E) \vee$$

$$(\bar{A} \cdot B \cdot C \cdot \bar{D}) \vee (\bar{A} \cdot B \cdot \bar{C} \cdot D) \vee (\bar{A} \cdot B \cdot \bar{D} \cdot E) \vee (\bar{A} \cdot \bar{B} \cdot C \cdot D) \vee$$

$$(\bar{A} \cdot \bar{B} \cdot C \cdot E) \vee (\bar{A} \cdot \bar{C} \cdot D \cdot E) \vee (\bar{A} \cdot C \cdot D \cdot \bar{E}) \vee (A \cdot \bar{B} \cdot C \cdot \bar{D}) \vee$$

$$(A \cdot \bar{B} \cdot C \cdot \bar{E}) \quad (\text{AI \#16})$$

$$C_2 = (A \cdot B \cdot C \cdot D) \vee (A \cdot B \cdot C \cdot E) \vee (A \cdot B \cdot D \cdot E) \vee (A \cdot C \cdot D \cdot E) \vee$$

$$(B \cdot C \cdot D \cdot E) \quad (\text{AI \#17})$$

#### 4. TRUTH TABLES

The truth table is a tabular representation of a logic expression and facts which indicates all possibilities of "TRUE" and "FALSE". The "1" will be used to denote the "TRUE" possibility and the "0" will denote the "FALSE" possibility. The relationship between "TRUE" and "FALSE" logical facts has now become more complex. Now the fact A is undefined as to whether it is "TRUE" or "FALSE" and may be either.  $\bar{A}$  can also be "TRUE" if "A" is "FALSE". The relationship that does hold is that no matter whether A is assumed to be "TRUE" or "FALSE",  $\bar{A}$  must always be assumed the opposite.

To set up a table, we must count the number of logic FACTS present (that is, A, B, C, D, E, etc.). If there are 3 facts present, then we must calculate  $2^3$ . Since  $2^3 = 8$ , we will have 8 possibilities, half of which are "1" and half of which are "0". For the first fact, the possibilities are divided up: 4 "0's" and 4 "1's". For the second fact, the possibilities are alternately divided up: 2-2-2-2. The third fact has alternate "1's" and "0's".

As examples, truth tables will be used to prove some of the identities in chapter 3.

##### 4.1 LOGIC OPERATOR TRUTH TABLE REPRESENTATIONS

Truth table representations are shown for two facts A, B and three facts A, B, C for the four logic operators ( $\cdot$ ,  $\vee$ ,  $+$ ,  $\nabla$ ). In addition, the "negation" truth tables are shown for one fact A, two facts A, B, and three facts A, B, C.

The representations for 2 quantities are as follows:

A	B	$A \cdot B$	$A \vee B$	$A + B$	$A \nabla B$	$\bar{A}$	$\bar{B}$
0	0	0	0	0	0	1	1
0	1	0	1	1	1	1	0
1	0	0	1	1	1	0	1
1	1	1	1	0	0	0	0

#### 4.1 LOGIC OPERATOR TRUTH TABLE REPRESENTATIONS (Continued)

For 3 quantities we have:

A	B	C	$A \cdot B \cdot C$	$A \vee B \vee C$	$A+B+C$	$(A \vee B) \vee C$	$A \vee (B \vee C)$	$A \vee B \vee C$	$\bar{A}$	$\bar{B}$	$\bar{C}$
0	0	0	0	0	0	0	0	0	1	1	1
0	0	1	0	1	1	1	1	1	1	1	0
0	1	0	0	1	1	1	1	1	1	0	1
0	1	1	0	1	0	0	0	0	1	0	0
1	0	0	0	1	1	1	1	1	0	1	1
1	0	1	0	1	0	0	0	0	0	1	0
1	1	0	0	1	0	0	0	0	0	0	1
1	1	1	1	1	1	1	1	0	0	0	0

Truth tables for a greater number of facts (A, B, C, D, E.....) can be made up from the definitions in chapter 2.

#### 4.2 PROOF OF DEMORGAN'S THEOREMS BY TRUTH TABLES

The proofs of DM #1, DM #2, DM #3, and DM #4 are illustrated by use of the following truth tables. Note that the "1" and "0" sequence for the "truth" columns which represent the quantity left of the "equal" sign match exactly the "truth" column to the right of the "equal" sign. Two or more quantities are logically equal if their "truth" columns match exactly.

Consider the following:

$$\overline{A \cdot B} = \bar{A} \vee \bar{B} \quad (\text{DM \#1})$$

A	B	$\bar{A}$	$\bar{B}$	$A \cdot B$	$\overline{A \cdot B}$	$\bar{A} \vee \bar{B}$
0	0	1	1	0	1	1
0	1	1	0	0	1	1
1	0	0	1	0	1	1
1	1	0	0	1	0	0

## 4.2 PROOF OF DEMORGAN'S THEOREMS BY TRUTH TABLES (Continued)

$$\overline{A \vee B} = \bar{A} \cdot \bar{B}$$

(DM #2)

A	B	$\bar{A}$	$\bar{B}$	$A \vee B$	$\overline{A \vee B}$	$\bar{A} \cdot \bar{B}$
0	0	1	1	0	1	1
0	1	1	0	1	0	0
1	0	0	1	1	0	0
1	1	0	0	1	0	0

$$\overline{A \cdot B \cdot C} = \bar{A} \vee \bar{B} \vee \bar{C}$$

(DM #3)

A	B	C	$\bar{A}$	$\bar{B}$	$\bar{C}$	$A \cdot B \cdot C$	$\overline{A \cdot B \cdot C}$	$\bar{A} \vee \bar{B} \vee \bar{C}$
0	0	0	1	1	1	0	1	1
0	0	1	1	1	0	0	1	1
0	1	0	1	0	1	0	1	1
0	1	1	1	0	0	0	1	1
1	0	0	0	1	1	0	1	1
1	0	1	0	1	0	0	1	1
1	1	0	0	0	1	0	1	1
1	1	1	0	0	0	1	0	0

$$\overline{A \vee B \vee C} = \bar{A} \cdot \bar{B} \cdot \bar{C}$$

(DM #4)

A	B	C	$\bar{A}$	$\bar{B}$	$\bar{C}$	$A \vee B \vee C$	$\overline{A \vee B \vee C}$	$\bar{A} \cdot \bar{B} \cdot \bar{C}$
0	0	0	1	1	1	0	1	1
0	0	1	1	1	0	1	0	0
0	1	0	1	0	1	1	0	0
0	1	1	1	0	0	1	0	0
1	0	0	0	1	1	1	0	0
1	0	1	0	1	0	1	0	0
1	1	0	0	0	1	1	0	0
1	1	1	0	0	0	1	0	0

### 4.3 PROOF OF DISTRIBUTIVE LAWS BY TRUTH TABLES

As examples, only DL #1 and DL #7 will be represented by truth tables. Note that DL #7 is an inequality and that the "truth" columns are different.

$$A \cdot (B \vee C) = (A \cdot B) \vee (A \cdot C) \quad (\text{DL \#1})$$

A	B	C	(B∨C)	(A·B)	(A·C)	A·(B∨C)	(A·B)∨(A·C)
0	0	0	0	0	0	0	0
0	0	1	1	0	0	0	0
0	1	0	1	0	0	0	0
0	1	1	1	0	0	0	0
1	0	0	0	0	0	0	0
1	0	1	1	0	1	1	1
1	1	0	1	1	0	1	1
1	1	1	1	1	1	1	1

Since the last two "truth" columns match, the above distribution expression is a proven equality.

Now consider the following inequality:

$$A + (B \cdot C) \neq (A + B) \cdot (A + C) \quad (\text{DL \#7})$$

A	B	C	(B·C)	(A+B)	(A+C)	A+(B·C)	(A+B)·(A+C)
0	0	0	0	0	0	0	0
0	0	1	0	0	1	0	0
0	1	0	0	1	0	0	0
0	1	1	1	1	1	1	1
1	0	0	0	1	1	1	1
1	0	1	0	1	0	1	0
1	1	0	0	0	1	1	0
1	1	1	1	0	0	0	0

Since the last two "truth" columns do not match, the expressions are not equal and the distributive law for this expression does not hold. Note that only the first four distributive expressions (DL #1, DL #2, DL #3, and DL #4) are equalities.

#### 4.4 PROOF OF FUNDAMENTAL IDENTITIES BY TRUTH TABLES

The truth table proofs of fundamental identities are very simple. A few examples are shown in order to illustrate this fact.

$$A \cdot A = A \text{ (FI #1)}$$

A	A	A · A
0	0	0
1	1	1

$$A \cdot \bar{A} = 0 \text{ (FI #2)}$$

A	A	A · $\bar{A}$
0	1	0
1	0	0

$$A \cdot 1 = A \text{ (FI #3)}$$

A	1	A · 1
0	1	0
1	1	1

$$A \cdot 0 = 0 \text{ (FI #4)}$$

A	0	A · 0
0	0	0
1	0	0

$$A \vee A = A \text{ (FI #17)}$$

A	A	A ∨ A
0	0	0
1	1	1

$$A \vee \bar{A} = 1 \text{ (FI #18)}$$

A	$\bar{A}$	A ∨ $\bar{A}$
0	1	1
1	0	1

$$A \vee 1 = 1 \text{ (FI #19)}$$

A	1	A ∨ 1
0	1	1
1	1	1

$$A \vee 0 = A \text{ (FI #20)}$$

A	0	A ∨ 0
0	0	0
1	0	1

$$A + A = 0 \text{ (FI #33)}$$

A	A	A + A
0	0	0
1	1	0

$$A + \bar{A} = 1 \text{ (FI #34)}$$

A	$\bar{A}$	A + $\bar{A}$
0	1	1
1	0	1



4.4 PROOF OF FUNDAMENTAL IDENTITIES BY TRUTH TABLES (Continued)

$$A+1 = \bar{A} \text{ (FI \#35)}$$

A	$\bar{A}$	1	A+1
0	1	1	1
1	0	1	0

$$A+0 = A \text{ (FI \#36)}$$

A	0	A+0
0	0	0
1	0	1

$$A \nabla A = 0 \text{ (FI \#49)}$$

A	A	$A \nabla A$
0	0	0
1	1	0

$$A \nabla 1 = \bar{A} \text{ (FI \#51)}$$

A	$\bar{A}$	1	$A \nabla 1$
0	1	1	1
1	0	1	0

4.5 PROOF OF ADDITION IDENTITIES BY TRUTH TABLES

The proofs of the two important addition identities AI #1 and AI #2 are shown here.

$$A+B = (A \cdot \bar{B}) \vee (\bar{A} \cdot B) \text{ (AI \#1)}$$

A	B	$\bar{A}$	$\bar{B}$	$(A \cdot \bar{B})$	$(\bar{A} \cdot B)$	A+B	$(A \cdot \bar{B}) \vee (\bar{A} \cdot B)$
0	0	1	1	0	0	0	0
0	1	1	0	0	1	1	1
1	0	0	1	1	0	1	1
1	1	0	0	0	0	0	0










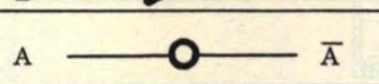
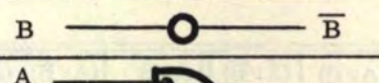
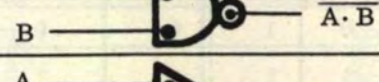
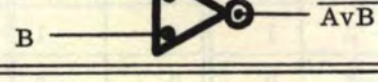
$$A+B = (A \vee B) \cdot (\bar{A} \vee \bar{B}) \text{ (AI \#2)}$$

A	B	$\bar{A}$	$\bar{B}$	$(A \vee B)$	$(\bar{A} \vee \bar{B})$	A+B	$(A \vee B) \cdot (\bar{A} \vee \bar{B})$
0	0	1	1	0	1	0	0
0	1	1	0	1	1	1	1
1	0	0	1	1	1	1	1
1	1	0	0	1	0	0	0

## 5. ELECTRONIC COMPUTER LOGIC NOTATION

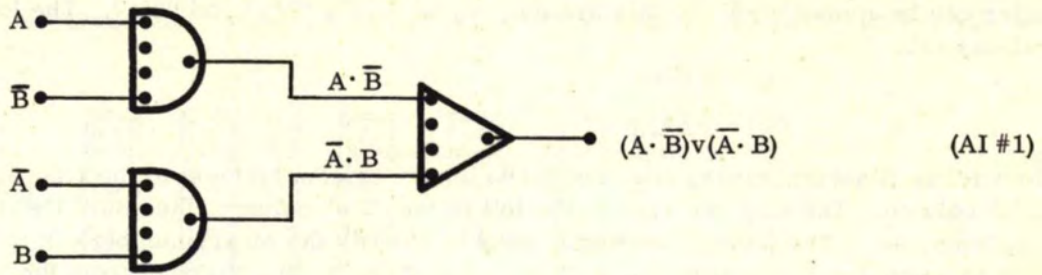
Up to this point, we have been discussing only pure logic operation on facts and conclusions. We will now relate this logic to electronic gate circuits. The 5 types of gates discussed here are: "AND", "OR", "SUM", "EOR", and "INVERTER".

Inputs to these gates are supplied by electronic FLIP-FLOP circuits (see chapter 6). The following table relates the written logic symbol with the electronic gate symbols. This table should be referred to when necessary.

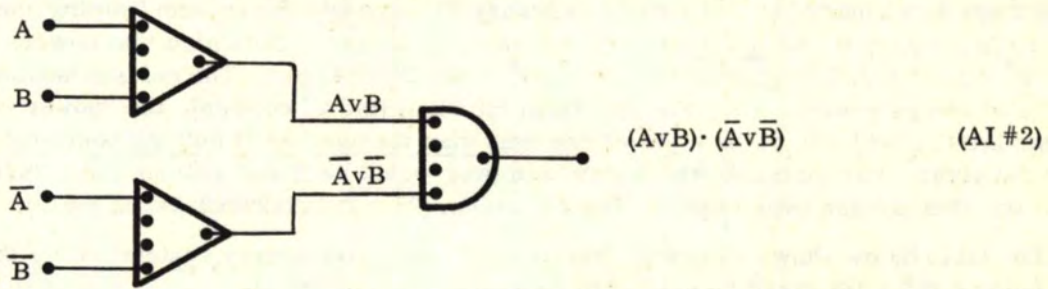
WRITTEN LOGIC SYMBOL	ELECTRONIC GATE SYMBOL	LOGIC OPERATION DEFINITION
· "DOT"		"AND"
∨ "WEDGE"		"OR"
+ "PLUS"		"SUM"
∇ "TRIANGLE"		"EXCLUSIVE OR"
¯ "BAR"		"INVERT" ("NOT")
$A \cdot B$		A "AND" B
$A \vee B$		A "OR" B
$A + B$		A "SUM" B
$A \nabla B$		A "EOR" B
$\bar{A}$		"NOT" A
$\bar{B}$		"NOT" B
$\overline{A \cdot B}$		"NOT" ( A "AND" B )
$\overline{A \vee B}$		"NOT" ( A "OR" B )

5.1 EXAMPLES OF ELECTRONIC COMPUTER LOGIC EXPRESSIONS

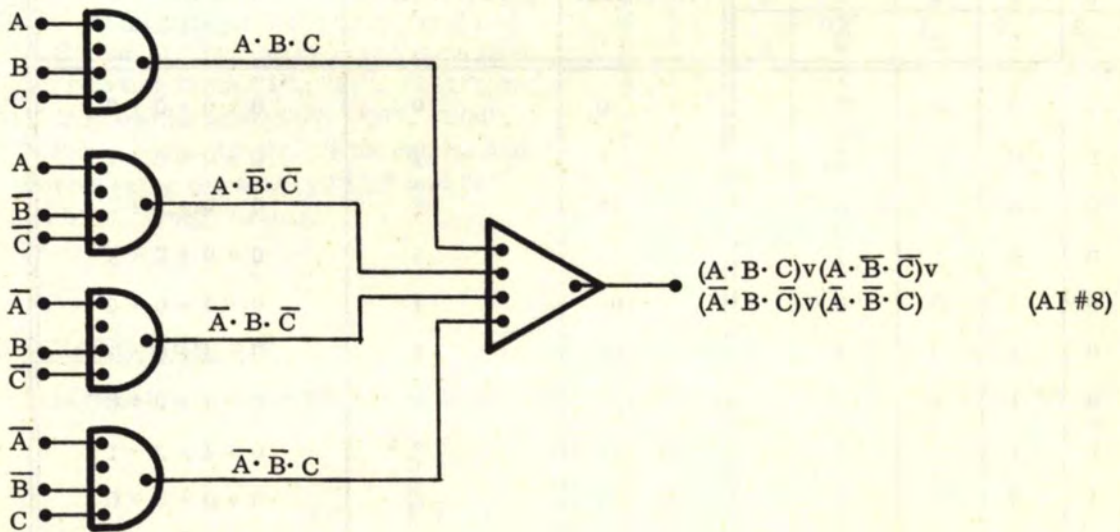
Consider the expression  $(A \cdot \bar{B}) \vee (\bar{A} \cdot B)$ . The "gate" representation is as follows:



Now consider the expression  $(A \vee B) \cdot (\bar{A} \vee \bar{B})$ . The "gate" representation is as follows:



The complicated expression AI #8 is represented as follows:



## 6.0 THE BINARY NUMBER SYSTEM

This simple number system must be understood before the value of the binary digital computer can be appreciated. In this system, we have only "1's" and "0's". The logical equivalents are:

"1" = TRUE

"0" = FALSE

Now let us illustrate binary counting. The binary column farthest to the right is called the "1's" column. The next one over to the left is the "2's" column, then the "4's" column, "8's" column, etc. The column heading is used to identify the binary numbers in terms of our own common decimal system (1, 2, 3, 4, 5, 6, 7, 8, 9, 0). Starting from the "1's" column, the column heading numbers are doubled as we go to the adjoining left column. In converting a binary number back to our own system, we note the column headings under which there is a binary "1". If there is a binary "1", we add the column heading number; if there is a binary 0, we ignore the column heading number. Note also that powers of 2 are:  $2^0 = 1$ ;  $2^1 = 2$ ;  $2^2 = 4$ ;  $2^3 = 8$ ;  $2^4 = 16$ ;  $2^5 = 32$ ;  $2^6 = 64$ ; etc. The column headings may also be shown as powers of 2. Starting from the right (the  $2^0$  column), the "power of 2" column heading will always be a power one less than the number of column positions over from the right. For instance, the 6th column over would be the  $2^5$  column ( $2^5 = 2 \times 2 \times 2 \times 2 \times 2 = 32$ ); the 10th column over would be the  $2^9$  column ( $2^9 = 2 \times 2 \times 2 \times 2 \times 2 \times 2 \times 2 \times 2 \times 2 = 512$ ).

The table below shows a "count" from 1 to 15, using the binary system. Note that a fifth column ( $2^4 = 16$ ) would be needed to represent the number 16.

BINARY COLUMN HEADING				BINARY NUMBER	DECIMAL EQUIVALENT	CONVERSION FROM BINARY TO DECIMAL
"8"	"4"	"2"	"1"			
$2^3$	$2^2$	$2^1$	$2^0$			
0	0	0	0	0	0	$0 + 0 + 0 + 0$
0	0	0	1	1	1	$0 + 0 + 0 + 1$
0	0	1	0	10	2	$0 + 0 + 2 + 0$
0	0	1	1	11	3	$0 + 0 + 2 + 1$
0	1	0	0	100	4	$0 + 4 + 0 + 0$
0	1	0	1	101	5	$0 + 4 + 0 + 1$
0	1	1	0	110	6	$0 + 4 + 2 + 0$
0	1	1	1	111	7	$0 + 4 + 2 + 1$
1	0	0	0	1000	8	$8 + 0 + 0 + 0$
1	0	0	1	1001	9	$8 + 0 + 0 + 1$
1	0	1	0	1010	10	$8 + 0 + 2 + 0$
1	0	1	1	1011	11	$8 + 0 + 2 + 1$
1	1	0	0	1100	12	$8 + 4 + 0 + 0$
1	1	0	1	1101	13	$8 + 4 + 0 + 1$
1	1	1	0	1110	14	$8 + 4 + 2 + 0$
1	1	1	1	1111	15	$8 + 4 + 2 + 1$

## 6.1 TRUTH TABLES AND THE BINARY COUNT

Note that for one fact A, two facts (A, B), three facts (A, B, C), and 4 facts (A, B, C, D), that the "truth table" representations represent nothing more than a binary count as shown below:

A
0
1

A	B
0	0
0	1
1	0
1	1

A	B	C
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

A	B	C	D
0	0	0	0
0	0	0	1
0	0	1	0
0	0	1	1
0	1	0	0
0	1	0	1
0	1	1	0
0	1	1	1
1	0	0	0
1	0	0	1
1	0	1	0
1	0	1	1
1	1	0	0
1	1	0	1
1	1	1	0
1	1	1	1

For one fact A, we count up from "0" to "1". For 2 facts (A, B), we count up from "00" to "11". For 3 facts (A, B, C) we count up from "000" to "111". For 4 facts (A, B, C, D) we count up from "0000" to "1111". Note that for the invert ("not") of A, B, C, and D, the binary count goes backwards, starting from "1", "11", "111", and "1111" and ending up at "0", "00", "000", and "0000", respectively. This can be seen by interchanging the binary "1's" and "0's" in the above "truth" tables.

## 6.2 BINARY ADDITION

It is possible to perform addition with binary numbers. The rules are as follows:

$0 + 0 = 0$
$0 + 1 = 1$
$1 + 0 = 1$
$1 + 1 = 0, \text{ carry } 1$

Examples are:

	← "carries" →		← "carries" →	
a. $\begin{array}{r} 11 \\ 1001 \\ +1101 \\ \hline 10110 \end{array}$	b. $\begin{array}{r} 11 \\ 110 \\ +111 \\ \hline 1101 \end{array}$	c. $\begin{array}{r} 1111 \\ 11101 \\ + 1011 \\ \hline 101000 \end{array}$		

6.2 BINARY ADDITION (Continued)

Decimal system equivalents:

a.     9 +13 ---- 22	b.     6 +7 ---- 13	c.     29 +11 ---- 40
-------------------------------	------------------------------	--------------------------------

6.3 BINARY SUBTRACTION

We can perform binary subtraction by the following rules:

$$0 - 0 = 0$$

$$0 - 1 = 1, \text{ borrow } 1$$

$$1 - 0 = 1$$

$$1 - 1 = 0$$

Examples are:

a.     0 0 ← "borrows" <del>1</del> 01 <del>1</del> 0 -1101 ---- 1001	b.     00 ← "borrows" <del>1</del> 101 -111 ---- 110	c.     01011 <del>1</del> 0 <del>1</del> 0 <del>0</del> 0 - 1011 ---- 11101
---	--	---

Decimal system equivalents:

a.     22 -13 ---- 9	b.     13 - 7 ---- 6	c.     40 -11 ---- 29
-------------------------------	-------------------------------	--------------------------------

6.4 BINARY MULTIPLICATION

Binary multiplication is performed like regular decimal number multiplication, except that it simplifies down greatly. In the examples below, the top number (above the line) is the MULTIPLICAND and the bottom number is the MULTIPLIER. Starting with the number farthest right in the multiplier, we note whether it is a "0" or "1". If it is "0", we write nothing (or "0") and proceed to the next adjacent MULTIPLIER number to the left. If it is a "1", then we copy down the MULTIPLICAND exactly as is and be sure that the last MULTIPLICAND digit to the right and the MULTIPLIER digit being used line up in the same column. Repeat for each MULTIPLIER digit.

Consider the following examples:

a.     10110 x 1101 ---- 10110 10110 10110 ---- 10110 ---- 100011110	b.     1101 x 111 ---- 1101 1101 1101 ---- 1101 ---- 1011011	c.     101000 x 1011 ---- 101000 101000 101000 ---- 101000 ---- 110111000
---	---	--

#### 6.4 BINARY MULTIPLICATION (Continued)

Decimal system equivalents are:

a.        22 <u>x13</u> 66 <u>22</u> 286	b.        13 <u>x 7</u> 91	c.        40 <u>x11</u> 40 <u>40</u> 440
--	----------------------------------	--

As can be seen from the above examples, once all the binary numbers are properly lined up, the columns need only be added up to give the answer.

The general rules of binary multiplication are:

0 x 0 = 0
0 x 1 = 0
1 x 0 = 0
1 x 1 = 1

MULTIPLICAND

(MULTIPLICAND x MULTIPLIER = PRODUCT)

x MULTIPLIER

PRODUCT

#### 6.5 BINARY DIVISION

Binary division is very similar to ordinary decimal division, except that it is much simpler. The REMAINDER must be closely observed. Starting with the left binary DIVIDEND digit, we continue over to the right until we have enough binary numbers (reach enough "places") to start the division process to obtain a QUOTIENT (or answer). If the division can continue into a fractional answer, we use a binary fractional point, or BINARY POINT (.), which is the same as the decimal fractional point (DECIMAL POINT) to denote the binary fractional.

When we start the division process, we place a "1" above the DIVIDEND binary digit farthest to the right which yields a number large enough to start the division process. Then we subtract the DIVISOR, being sure that the last binary DIVISOR digit is in the same column as the "1" entered in the QUOTIENT (see examples). Then we "bring down" and "tack on" to the REMAINDER the next binary digit from the DIVIDEND and see if we can subtract the DIVISOR from the new REMAINDER. If we can not, then we write a "0" in the next QUOTIENT digit position, "bring down" and "tack on" to the REMAINDER the next binary DIVIDEND digit. Repeat this process until subtraction of DIVISOR from REMAINDER can be performed. Then write a "1" in the next QUOTIENT binary digit position, subtract, and "bring down" and "tack on" the next DIVIDEND binary digit. Continue this process until division is complete.

Consider the following examples:

6.5 BINARY DIVISION (Continued)

$$\begin{array}{r} \text{QUOTIENT} \\ \text{DIVISOR } \overline{) \text{ DIVIDEND}} \end{array} \quad (\text{DIVIDEND } \div \text{ DIVISOR} = \text{QUOTIENT})$$

a. 
$$\begin{array}{r} 11 \\ 1101 \overline{) 100111} \\ \underline{- 1101} \\ 1101 \\ \underline{- 1101} \\ 0 \end{array}$$

"BRING DOWN" and "REMAINDERS" are indicated with arrows pointing to the digits being brought down and the remainders at each step.

b. 
$$\begin{array}{r} 11.001001\dots\dots \\ 111 \overline{) 10110.} \\ \underline{- 111} \\ 1000 \\ \underline{- 111} \\ 111 \\ \underline{- 111} \\ 1000 \\ \underline{- 111} \\ 1000 \\ \underline{- 111} \\ 1 \end{array}$$

"BRING DOWNS" are indicated with arrows pointing to the digits being brought down.

c. 
$$\begin{array}{r} 111 \\ 10001 \overline{) 1110111} \\ \underline{- 10001} \\ 11001 \\ \underline{- 10001} \\ 10001 \\ \underline{- 10001} \\ 10001 \\ \underline{- 10001} \\ 0 \end{array}$$

RE - MAINDERS and "BRING DOWNS" are indicated with arrows.

Decimal system equivalents are:

a. 
$$\begin{array}{r} 3 \\ 13 \overline{) 39} \\ \underline{39} \\ 0 \end{array}$$

b. 
$$\begin{array}{r} 3.14\dots\dots \\ 7 \overline{) 22} \\ \underline{21} \\ 10 \\ \underline{7} \\ 30 \\ \underline{28} \\ 2 \end{array}$$

c. 
$$\begin{array}{r} 7 \\ 17 \overline{) 119} \\ \underline{119} \\ 0 \end{array}$$

6.6 BINARY FRACTIONALS

The binary "FRACTIONAL" is similar to the "DECIMAL" fractional representation of our system. Consider the common  $\pi = 3.1416$ . The first "1" represents  $1/10$  or  $10^{-1}$ . The "4" represents  $4/100$  or  $4 \times 10^{-2}$ . The second "1" represents  $1/1000$  or  $1 \times 10^{-3}$ , and the "6" represents  $6/10,000$  or  $6 \times 10^{-4}$ . As we can see, we have the "tenths", "hundredths", "thousandths", and "ten-thousandths" positions to the right of the decimal point. Note that the first position to the right of the decimal represents  $10^{-1}$  ( $1/10^1$ , or  $1/10$ ), the second position represents  $10^{-2}$  ( $1/10^2$  or  $1/100$ ), the third position represents  $10^{-3}$  ( $1/10^3$  or  $1/1000$ ), the fourth position represents  $10^{-4}$  ( $1/10^4$  or  $1/10,000$ ), and so on. Generally, for "X" positions to the right of the decimal point, we have the  $10^{-X}$  position ( $1/10^X$  or  $1/100\dots\dots 00$  with "X" number of 0's in the denominator).

The BINARY POINT denotes a BINARY FRACTIONAL and the only difference is that for "X" positions to the right of the BINARY POINT, we have  $2^{-X}$  (instead of  $10^{-X}$ ). Then in terms of our decimal system, the first position to the right of the binary point represents  $2^{-1}$  ( $1/2^1$  or  $1/2$ ), the second position represents  $2^{-2}$  ( $1/2^2$  or  $1/4$ ), the third position represents  $2^{-3}$  ( $1/2^3$  or  $1/8$ ), the fourth position represents  $2^{-4}$  ( $1/2^4$  or  $1/16$ ), and so on.

However, since "2" is represented as "10" in the binary system, the first position to the right of the BINARY POINT becomes  $10^{-1}$  ( $1/10^1$  or  $1/10$ ), the second position becomes



## 6.6 BINARY FRACTIONALS (Continued)

$10^{-2}$  ( $1/10^2$  or  $1/100$ ), the third position becomes  $10^{-3}$  ( $1/10^3$  or  $1/1000$ ), the fourth position becomes  $10^{-4}$  ( $1/10^4$  or  $1/10000$ ), and so on. Note that in binary, "10" = 2, "100" = 4, "1000" = 8, "10000" = 16, and so on. The simplicity about a BINARY FRACTIONAL position is that it is either a "1" or a "0" and nothing else. In our own decimal system, we may have any one of the numbers 1, 2, 3, 4, 5, 6, 7, 8, or 9 in a given decimal fractional position.

Consider the following decimal and binary fractional equivalents:

	DECIMAL	BINARY
a.	$\pi = 3.14159\dots$	$= 11.0010010001\dots$
b.	$1/2 = .5$	$= .1$
c.	$1/3 = .33\overline{3}\dots$	$= .01010101\overline{01}\dots$
d.	$1/10 = .1$	$= .000110011\overline{0011}\dots$
e.	$e = 2.71828\dots$	$= 10.101101111110\dots$
f.	$\sqrt{2} = 1.414\dots$	$= 1.0110101\dots$

\*The overhead bracket ( $\overline{\quad}$ ) means that the portion within it is repeated over and over indefinitely.

Note that some fractions are finite decimal fractionals, but infinite binary fractionals as in example (d). An INFINITE fractional is one where the division of two finite numbers goes on indefinitely and the numbers repeat in a given sequence. The overhead bracket ( $\overline{\quad}$ ) shows the repeating sequence. An IRRATIONAL fractional is a fractional which may be carried out indefinitely, but has no repeating sequence. The fractionals for  $1/3$  (in the binary and decimal systems) and  $1/10$  (in the binary system) are INFINITE, while the fractionals for " $\pi$ ", "e" and " $\sqrt{2}$ " are IRRATIONAL (both " $\pi$ " and "e" are widely used mathematical constants either or both of which the reader may be familiar).

### 6.6.1 CONVERSION FROM DECIMAL FRACTIONALS TO BINARY FRACTIONALS

Consider a fractional already in the decimal form. To convert to the equivalent binary fractional, we proceed as follows: The whole number to the left of the decimal fractional point is converted into binary by the method described in 6.0. We treat the portion to the right of the decimal fractional point separately and multiply that number by 2. If the answer is less than 1, we write a "0" in a "reference" column to the left. If the number is greater than 1, we remove the "1" and bring it over to the "reference" column and again multiply the rest of the fraction by 2 and repeat the process above. The multiplication by 2 can be carried out as far as one wishes. The "reference" column will then contain the BINARY FRACTIONAL to the right of the BINARY POINT.

Consider again  $3.14159\dots$ . The "3" converts to binary "11". Now work with ".14159" as follows:

6. 6. 1 CONVERSION FROM DECIMAL FRACTIONALS TO BINARY FRACTIONALS(Continued)

REFERENCE COLUMN	"FRACTIONAL"
	.14159
	<u>x 2</u>
0	.28318
	<u>x 2</u>
0	.56636
	<u>x 2</u>
1	.13272
	<u>x 2</u>
0	.26544
	<u>x 2</u>
0	.53088
	<u>x 2</u>
1	.06176
	<u>x 2</u>
0	.12352
	<u>x 2</u>
0	.24704
	<u>x 2</u>
0	.49408
	<u>x 2</u>
0	.98816
	<u>x 2</u>
1	.97632
	<u>x 2</u>
1	.95264
	<u>x 2</u>
1	.90528
	<u>x 2</u>
1	.81056
	<u>x 2</u>
1	.62112
	<u>x 2</u>
1	.24224
	<u>x 2</u>
0	.48448
	<u>x 2</u>
0	.96896
	<u>x 2</u>
1	.93792

Now, using the "11" for the whole number part, and the "reference" column for the fractional part, the decimal fractional 3.14159.... converts to the binary fractional 11.0010010000111111001.....

The decimal fractional 3.14159.... has only 5 decimal fractional places. In order to maintain this accuracy in a binary fractional, we must have roughly 3.2 times as many fractional places. Therefore, the binary equivalent for 3.14159.... must have  $5 \times 3.2$ , or 16 binary fractional digits computed to maintain the accuracy of 5 decimal fractional digits (since 19 were computed, the accuracy is at least good as 5 decimal fractional digits).

A second method of conversion is merely to subtract  $1/2$ ,  $1/4$ ,  $1/8$ ,  $1/16$  ...., etc., in decimal fractional form (.5, .25, .125, .0625 ....). Starting with .5, we try to

### 6.6.1 CONVERSION FROM DECIMAL FRACTIONALS TO BINARY FRACTIONALS (Continued)

subtract from the decimal fractional. If we can subtract, a "1" is entered for the first binary fractional digit. If we can not subtract, then a binary "0" is entered. Subtract and repeat again for .25, and so on. This method is very tiresome because the decimal fractionals get very complicated as we calculate more and more binary fractionals. Also, we need a table of decimal fractionals for  $1/2$ ,  $1/4$ ,  $1/8$ ,  $1/16$ , ... etc. Now let us convert 3.14159..... to a binary fractional by this method. Again, we convert the whole number "3" to binary "11" by the method in section 6.0. Then we proceed to work on the decimal fractional as follows:

.14159	
	0, cannot subtract .5
	0, cannot subtract .25
- .125	1, subtract .125
<u>.01659</u>	
	0, cannot subtract .0625
	0, cannot subtract .03125
- .015625	1, subtract .015625
<u>.000965</u>	
	0, cannot subtract .0078125
	0, cannot subtract .00390625
	0, cannot subtract .001953125
	0, cannot subtract .0009765625
- .000489	1, subtract .00048828125

TABLE OF DECIMAL FRACTIONALS FOR  $2^{-X}$

X	$2^{-X}$
1	.5
2	.25
3	.125
4	.0625
5	.03125
6	.015625
7	.0078125
8	.00390625
9	.001953125
10	.0009765625
11	.00048828125
12	.000244140625
13	.0001220703125
14	.00006103515625
15	.000030517578125
16	.0000152587890625
17	.00000762939453125
18	.000003814697265625
19	.0000019073486328125
20	.00000095367431640625

We have calculated eleven places above and 3.14159..... converts to 11.00100100001..... The reader can now see the difficulty in calculating out more places.

6.6.2 CONVERSION FROM BINARY FRACTIONALS TO DECIMAL FRACTIONALS

To convert back from BINARY FRACTIONALS to decimal fractionals, we may refer to the  $2^{-X}$  table in 6.6.1 and add up all the decimal fractional equivalents for the columns which have a binary fractional digit of "1". For instance, to convert .11, we add up .5 + .25 = .75, which is relatively simple. Another example, .1101 = .5 + .25 + .0625 = .8125. However, the longer binary fractionals are more complicated to handle in this manner.

The second method is that of multiplying the binary fractional by "1010" in binary and moving over into a "reference" column all binary digits left of the binary point of the resulting answer, leaving the remaining binary fractional for further multiplication by binary "1010". We repeat this process until the desired number of decimal fractional digits are obtained. Note that each "decimal fractional digit" will appear in binary form in the reference column. A simple way to multiply by "1010" is to move the BINARY POINT over one place, then add to this the same number with the BINARY POINT moved over 3 places.

Consider our calculation of 11.0010010000111111001. We treat this as follows:

"REFERENCE"  
COLUMN

	.0 0 1 0 0 1 0 0 0 0 1 1 1 1 1 1 0 0 1
	+ .0 0 1 .0 0 1 0 0 0 0 1 1 1 1 1 1 0 0 1
	1 .0 1 1 0 1 0 1 0 0 1 1 1 0 1 1 1 0 1
1	+ .0 1 1 .0 1 0 1 0 0 1 1 1 0 1 1 1 0 1
	1 0 0 .0 0 1 0 1 0 0 0 1 0 1 0 1 0 0 0 1
100	+ .0 0 1 .0 1 0 0 0 1 0 1 0 1 0 0 0 1
	1 .0 0 1 0 1 1 0 1 0 0 1 0 1 0 1
1	+ .1 0 0 .1 0 1 1 0 1 0 0 1 0 1 0 1
	1 0 1 .1 1 1 0 0 0 0 1 1 1 0 1 0 0 1
101	+ .1 1 1 .0 0 0 0 1 1 1 0 1 0 0 1
	1 0 0 0 .1 1 0 1 0 0 1 0 0 0 1 1 0 1
1000	+ .1 1 0 .1 0 0 1 0 0 0 1 1 0 1
	1 0 0 0 .0 0 1 1 0 1 1 0 0 0 0 1
1000	

Now observe that the "reference column" indicates 6 positions of the decimal fractional number and that these position numbers are in binary (1, 100, 1, 101, 1000, 1000). Now, we must convert these binary numbers to decimal numbers by the method in section 6.0 to obtain 1, 4, 1, 5, 8, 8. The whole number to the left of the binary fractional point (11) converts to 3. Therefore, the converted binary fractional is 3.141588 or, rounding up the last 8, we have 3.14159.

While the above method is not simple, the first method is much more difficult for long binary fractionals.

## 6.7 BINARY FRACTIONS

The binary fraction number, like our own decimal fraction number, has both a **NUMERATOR** and **DENOMINATOR** as shown below:

$$\text{FRACTION} = \frac{\text{NUMERATOR}}{\text{DENOMINATOR}}$$

However, both the **NUMERATOR** and **DENOMINATOR** are expressed in BINARY instead of the decimal system. Consider the following examples:

$$\text{a. } \frac{11}{100} \quad \text{b. } \frac{1}{11} \quad \text{c. } \frac{11}{1000} \quad \text{d. } \frac{1001}{10001} \quad \text{e. } \frac{1}{10}$$

The decimal fraction number equivalents of the above are:

$$\text{a. } 3/4 \quad \text{b. } 1/3 \quad \text{c. } 3/8 \quad \text{d. } 9/17 \quad \text{e. } 1/2$$

The use of binary fraction numbers is not as common as the binary FRACTIONAL representation. In order to convert a binary **FRACTION** to a binary FRACTIONAL, simply divide out the **NUMERATOR** by the **DENOMINATOR**.

## 6.8 SQUARE ROOTS

The square root of a number is a second number which, when multiplied by itself, yields back the first number. For example, the square root of 4 is 2 because  $2 \times 2 = 4$ . The square root of 225 is 15 because  $15 \times 15 = 225$ . The above examples are relatively easy because the numbers are exact. However, except for numbers which are perfect "squares" (such as 1, 4, 9, 16, 25, 225, 1.69, 5625, etc.) all other square roots are IRRATIONAL fractionals in both decimal and binary systems. The "RADICAL" ( $\sqrt{\quad}$ ) is used to indicate that a square root must be extracted from a number. " $\sqrt{2}$ " is read: "the square root of 2."

### 6.8.1 EXTRACTING THE SQUARE ROOT

If the number is a whole number, we must divide all the digits into groups of "two" by use of the "spacer" mark (^) starting with the number at the extreme right. Any odd digit left alone by the grouping process must occur at the extreme left. A number consisting of one or two digits need not be grouped.

A fractional must be grouped in "twos" starting at the decimal point in which case an odd digit would be left at the extreme right. Zeros may be added on to carry out the operation further. A zero may be added on a lone digit at the extreme right in order to complete a grouping of "two". Zeros may then be added on, two at a time, indefinitely.

A whole number may be made into a fractional by placing a decimal point at the extreme right and by adding zeros in groups of "two" ( $2 = 2.00\wedge00\wedge00$ ). A fractional which contains a whole number must be grouped in "twos" starting at the decimal point and proceeding to the right for the part to the right of the decimal point, and proceeding to the left for the part to the left of the decimal point ( $3\wedge97\wedge86.48\wedge73\wedge51$ ). The same grouping rules apply to binary numbers.

Consider the square root of 1521. We group this number in "twos" as follows and insert a "radical" over it:  $\sqrt{15\wedge21}$ . To start the operation, we observe the group farthest to the left, which is 15. We then determine which number is the largest square less than 15. Considering the possibilities from 1 to 9, we have:  $1 \times 1 = \underline{1}$ ;  $2 \times 2 = \underline{4}$ ;

6.8.1 EXTRACTING THE SQUARE ROOT (Continued)

$3 \times 3 = 9$ ;  $4 \times 4 = 16$ ;  $5 \times 5 = 25$ ;  $6 \times 6 = 36$ ;  $7 \times 7 = 49$ ;  $8 \times 8 = 64$ ;  $9 \times 9 = 81$ . The preceding information indicates that 9 is the number we want (16 is too large). We subtract 9 from 15 and write  $\sqrt{9}$ , or "3", above the grouped "15" pair. The REMAINDER is 6 and we "bring down" the next pair of numbers (21).

$$\begin{array}{r}
 3 \quad \leftarrow \text{SQUARE ROOT} \\
 \sqrt{15 \wedge 21} \\
 \underline{9} \\
 6 \quad 21 \\
 \leftarrow \text{REMAINDER}
 \end{array}$$

"L" LINE  $\longrightarrow$  6

To the left of the REMAINDER, we draw an "L" shaped line ("L" LINE) with a long bottom line as shown above. Then DOUBLE the number in the "SQUARE ROOT" (on top of the radical) and enter it above the bottom of the "L" line and leave room to write in another digit after it. The next digit on the "L" line can be anything from 0 through 9. That means the "L" line number can be any number from 60 to 69. Now observe again that the REMAINDER is 621. Also, the second number on the "L" line will be the same as the next digit in the "SQUARE ROOT" which appears above the next pair of numbers to the right (21). Now we have to determine the correct number between 60 and 69, and the digit between 0 and 9 whose product is equal to or less than the REMAINDER 621. In order to estimate the number, we will use 60. Now  $60 \times ? = 621$ . The number 9 looks possible because  $60 \times 9 = 540$ , which is less than 621. For the final test, we must place this "9" after the "6" and multiply again to be sure that the product is equal to or less than 621. Testing, we have  $69 \times 9 = 621$ , which is exact. This shows that 1521 is a perfect square and that 39 is its square root. Observe below:

$$\begin{array}{r}
 3 \quad 9 \\
 \sqrt{15 \wedge 21} \\
 \underline{9} \\
 6 \quad 21 \\
 \underline{6 \quad 21} \\
 0
 \end{array}$$

"L" LINE  $\longrightarrow$  69

Now, consider a more difficult example--a number whose square root is IRRATIONAL. The number "2" is a good example. In this example, we will calculate the  $\sqrt{2}$  to 5 places. First of all, since 2 is a whole number, we will add a decimal point to the right and add 5 pairs of zeros and a radical. The calculation is done as follows:

$$\begin{array}{r}
 1.41421 \\
 \sqrt{2.00 \wedge 00 \wedge 00 \wedge 00 \wedge 00} \\
 \underline{1} \\
 1 \quad 00 \\
 \underline{96} \\
 4 \quad 00 \\
 \underline{2 \quad 81} \\
 1 \quad 19 \quad 00 \\
 \underline{1 \quad 12 \quad 96} \\
 6 \quad 04 \quad 00 \\
 \underline{5 \quad 65 \quad 64} \\
 38 \quad 36 \quad 00 \\
 \underline{28 \quad 28 \quad 41} \\
 10 \quad 07 \quad 59
 \end{array}$$

"L" LINE  $\longrightarrow$  24

"L" LINE  $\longrightarrow$  281

"L" LINE  $\longrightarrow$  2824

"L" LINE  $\longrightarrow$  28282

"L" LINE  $\longrightarrow$  282841

REMAINDERS

6.8.1 EXTRACTING THE SQUARE ROOT (Continued)

Note that with the exception of the first "SQUARE ROOT" digit to the left, the last digits of the "L" line numbers match the "SQUARE ROOT" digits exactly in the corresponding positions. The number "1" ( $1 \times 1 = 1$ ) proved to be the largest square less than 2 and was subtracted from it, leaving a REMAINDER of 1. The square root of "1" (also 1) was entered above the "2". Note that since 2 is the only number left of the decimal point, it is grouped alone. The decimal point is placed in its respective position above the radical above the first decimal point. We then "bring down" two 0's to complete the first REMAINDER of 100. We then double 1 to form the first digit of the first "L" line number. Now we need a 20-29 and 1-9 number whose product is equal to or less than 100. In testing, we find that  $20 \times 0 = 0$ ;  $21 \times 1 = 21$ ;  $22 \times 2 = 44$ ;  $23 \times 3 = 69$ ;  $24 \times 4 = 96$ ;  $25 \times 5 = 125$ . Since 96 is the largest number less than 100, we then subtract it from 100, forming part of the second REMAINDER. We complete the first "L" line number (24) and enter in the other "4" as the SQUARE ROOT digit above the first pair of zeros (the square root now reads "1.4"). We now double the "14", ignoring the decimal, to get "28". This "28" is the first part of the second "L" line number (an easier way is just to double the last digit in the preceding "L" line number, making sure to carry the extra "1" when doubling, if the last preceding "L" line number is greater than 4).

Now we bring down the second pair of zeros to complete the second REMAINDER of 400. Now we need a 280-289 and 1-9 number whose product is equal to or less than 400. In testing, we have:  $280 \times 0 = 0$ ;  $281 \times 1 = 281$ ;  $282 \times 2 = 564$ . Since 281 is the largest number less than 400, we then subtract it from 400, forming part of the third REMAINDER (119). We complete the second "L" line number (281) and enter the "1" as the next SQUARE ROOT digit above the second pair of zeros (the SQUARE ROOT now reads "1.41"). We now double the "141", ignoring the decimal, to get "282". This "282" is the first part of the third "L" line number.

Now bring down the third pair of zeros to complete the third REMAINDER (11900). Repeat the checking process for numbers between 2820-2829 and 0-9 to obtain  $2824 \times 4 = 11296$ , the largest product less than 11900. The process is continued for as many places as necessary.

The following are additional examples for the reader to observe the square root process:

a.

$$\begin{array}{r} 1.73205 \\ \sqrt{3.00\wedge00\wedge00\wedge00\wedge00} \\ \underline{1} \\ 27 \overline{)200} \\ \underline{189} \\ 343 \overline{)1100} \\ \underline{1029} \\ 3462 \overline{)7100} \\ \underline{6924} \\ 34640 \overline{)17600} \\ \underline{0} \\ 346405 \overline{)1760000} \\ \underline{1732025} \\ 27975 \end{array}$$

b.

$$\begin{array}{r} 40.96 \\ \sqrt{16\wedge77.72\wedge16} \\ \underline{16} \\ 80 \overline{)077} \\ \underline{0} \\ 809 \overline{)7772} \\ \underline{7281} \\ 8186 \overline{)49116} \\ \underline{49116} \\ 0 \end{array}$$

6.8.1 EXTRACTING THE SQUARE ROOT (Continued)

c.

$$\begin{array}{r} \sqrt{1\ 1.046} \\ \sqrt{1\ 22.01\ 50\ 00} \\ \underline{21} \phantom{0} 22 \\ \phantom{21} \underline{21} \\ \phantom{220} 1\ 01 \\ \phantom{220} \phantom{1} \underline{0} \\ \phantom{2204} 1\ 01\ 50 \\ \phantom{2204} \phantom{1} \phantom{01} \underline{88} 16 \\ \phantom{22086} 13\ 34\ 00 \\ \phantom{22086} \phantom{13} \underline{13} 25\ 16 \\ \phantom{22086} \phantom{13} \phantom{25} \phantom{16} \underline{9} 84 \end{array}$$

Note in example (a) that when we use the number "5" to complete an "L" line number (as in the 5th "L" line), the multiplied product of the "L" line and "5" has exactly the same digits of the SQUARE ROOT which have been calculated up to that point, but the last "5" is replaced by "25". Note that  $346405 \times 5 = 1732025$ , which illustrates this point in example (a).

6.8.2 THE BINARY SQUARE ROOT

The general rules are the same except that all digits, numbers, and operations are performed and written in the binary system. The process simplifies greatly because doubling a binary number means only adding an extra "0" onto the number. For example, doubling 1011 gives 10110 ( $1011 \times 10 = 10110$ ). Multiplication simplifies down because we either write down the "L" line number and subtract, or we have "0". We do not have to test several possibilities of "L" line digits because we can have only "1" or "0". Now let us consider  $\sqrt{2}$  in binary. This is written as " $\sqrt{10}$ ". We will determine  $\sqrt{10}$  to 8 places. Observe the steps in the example below.

$$\begin{array}{r} \sqrt{1.01101010} \leftarrow \text{BINARY SQUARE ROOT} \\ \sqrt{10.00\ 00\ 00\ 00\ 00\ 00\ 00} \\ \underline{1} \\ 100 \phantom{00} \phantom{00} \phantom{00} \phantom{00} \phantom{00} \phantom{00} \phantom{00} \\ \phantom{100} \underline{0} \\ \phantom{1001} 1\ 00\ 00 \leftarrow \text{BINARY REMAINDERS} \\ \phantom{1001} \phantom{1} \underline{10} 01 \\ \phantom{10101} 1\ 11\ 00 \\ \phantom{10101} \phantom{1} \underline{1} 01\ 01 \\ \phantom{101100} 1\ 11\ 00 \\ \phantom{101100} \phantom{1} \underline{0} \\ \phantom{1011001} 1\ 11\ 00\ 00 \leftarrow \text{BINARY REMAINDERS} \\ \phantom{1011001} \phantom{1} \underline{1} 01\ 10\ 01 \\ \phantom{10110100} 1\ 01\ 11\ 00 \\ \phantom{10110100} \phantom{1} \underline{0} \\ \phantom{101101001} 1\ 01\ 11\ 00\ 00 \leftarrow \text{BINARY REMAINDERS} \\ \phantom{101101001} \phantom{1} \underline{1} 01\ 10\ 10\ 01 \\ \phantom{1011010100} 1\ 11\ 00 \\ \phantom{1011010100} \phantom{1} \underline{0} \\ \phantom{1011010100} 1\ 11\ 00 \end{array}$$

"L" LINE NUMBERS



6.8.2 THE BINARY SQUARE ROOT (Continued)

Note again either the "L" line numbers, or "0" were the only possibilities for the subtractions. Also, only the "L" line numbers ending in "1" were subtracted. If the "L" line numbers ended in "0", then "0" was subtracted. The number "1" ( $1 \times 1 = 1$ ) is the largest square less than 10 and was subtracted from it, leaving a REMAINDER of "1". "1" was entered above the first pair of numbers "10" and the second BINARY POINT was entered in above the radical in the proper position. The "1" was doubled to yield "10" which was entered as part of the first "L" line number. The first pair of zeros was brought down to complete the first REMAINDER. The final digit of the first "L" line number could only be "0" or "1" leaving the two possibilities of 100 and 101. Since 101 is larger than the REMAINDER, the only other possibility left is 100. We completed the first "L" line number by adding on a "0" and entered another "0" above the first pair of zeros to the right of the binary point. Subtracting "0" and "bringing down" the second pair of zeros (right of the binary point) completed the second REMAINDER (10000). Doubling the "10" in the "SQUARE ROOT" digits, ignoring the BINARY POINT, gave "100" which was entered as part of the second "L" line number. The two possibilities for completing the second "L" line number were 1001 and 1000. Since the larger 1001 was still smaller than the remainder 10000, the second "L" line number became 1001. The "1" was entered above the second pair of zeros and the "L" line number was subtracted from the second REMAINDER, leaving 111. "Bringing down" the third pair of zeros completed the third REMAINDER (11100) and doubling the "101" in the SQUARE ROOT yielded the first 4 digits of the third "L" line number. In testing, the "L" line number 10101 could be subtracted from REMAINDER 11100 and the resulting "1" was entered above the third pair of zeros. The process was carried out to 8 binary places.

The following additional examples will let the reader better observe the process of extracting a binary square root.

a. 
$$\sqrt{11.00\wedge00\wedge00\wedge00\wedge00\wedge00\wedge00}$$

$$\begin{array}{r} \phantom{11.00} 1 \\ \hline 101 \phantom{00} \phantom{00} \phantom{00} \phantom{00} \phantom{00} \phantom{00} \phantom{00} \phantom{00} \\ \hline 1100 \phantom{00} \phantom{00} \phantom{00} \phantom{00} \phantom{00} \phantom{00} \phantom{00} \phantom{00} \\ \hline \phantom{1100} 0 \\ \hline 11001 \phantom{00} \phantom{00} \phantom{00} \phantom{00} \phantom{00} \phantom{00} \phantom{00} \phantom{00} \\ \hline \phantom{11001} 11001 \\ \hline 110101 \phantom{00} \phantom{00} \phantom{00} \phantom{00} \phantom{00} \phantom{00} \phantom{00} \phantom{00} \\ \hline \phantom{110101} 110101 \\ \hline 1101101 \phantom{00} \phantom{00} \phantom{00} \phantom{00} \phantom{00} \phantom{00} \phantom{00} \phantom{00} \\ \hline \phantom{1101101} 1101101 \\ \hline 11011100 \phantom{00} \phantom{00} \phantom{00} \phantom{00} \phantom{00} \phantom{00} \phantom{00} \phantom{00} \\ \hline \phantom{11011100} 0 \\ \hline 110111001 \phantom{00} \phantom{00} \phantom{00} \phantom{00} \phantom{00} \phantom{00} \phantom{00} \phantom{00} \\ \hline \phantom{110111001} 11011001 \\ \hline 1101110101 \phantom{00} \phantom{00} \phantom{00} \phantom{00} \phantom{00} \phantom{00} \phantom{00} \phantom{00} \\ \hline \phantom{1101110101} 110110101 \\ \hline \phantom{1101110101} 110110101 \\ \hline \phantom{1101110101} 10110011 \end{array}$$

b. 
$$\sqrt{10.01}$$

$$\begin{array}{r} \phantom{10.01} 1 \\ \hline 100 \phantom{01} \\ \hline \phantom{100} 0 \\ \hline 1001 \phantom{01} \\ \hline \phantom{1001} 1001 \\ \hline \phantom{1001} 0 \end{array}$$

## 6.9 ROUNDING "UP" AND ROUNDING "DOWN"

Rounding "off" is the process of shortening the length of a fractional by dropping one or more digits at the extreme right.

In the decimal fractional .3333333333, we may "round off" to two places as .33. In this case we rounded "DOWN", meaning we just dropped the remaining digits. Now consider the fractional .66666666666666. In "rounding off" to two places, we have .67. In this case, we rounded "UP" (meaning that all digits were dropped after the second digit, but the value of the second digit was increased by "1"). "Rounding off" to "N" digits means that we retain "N" number of digits in the fractional and drop all the rest of the fractional digits.

The "round off" rule for the decimal system is to observe the first digit to be dropped. If that digit is a 5, 6, 7, 8, or 9, then we round "UP". However, if that digit is a 0, 1, 2, 3, or 4, then we round "DOWN" and simply drop the digits. Examples are 1.41421, 3.14159, 2.71828, and 1.73205. If we "round off" the preceding to two-place fractionals, we have: 1.41, 3.14, 2.72, and 1.73, respectively. Note that "2.71" became 2.72 before dropping the "828". Note the first digit to be dropped is an 8 and the rule says we must add "1" to the 2.71 before dropping the "828". In "rounding off" to three-place fractionals, we have 1.414, 3.142, 2.718, and 1.732, respectively. Note that "3.141" became 3.142 before dropping the "59" because of the "5". In "rounding off" to four-place fractionals, we have 1.4142, 3.1416, 2.7183, and 1.7321, respectively. Note that "3.1415" became 3.1416 before dropping the "9"; "2.7182" became 2.7183 before dropping the "8"; and "1.7320" became 1.7321 before dropping the "5". A further example is: 1.0995. In "rounding off" to three fractional places, we have 1.100 because of the "carries" generated by the 9's. Also, the fractional 1.99995 "rounded off" to four places would be 2.0000, again because of the "carries". This concept can be extended to numbers which are not fractionals. Consider 346. In "rounding off" to two significant digits, we start from the extreme left and drop all digits beyond those two and add a zero (or zeros) when necessary to show the proper magnitude of the number. Now 346 "rounded off" to two places becomes 350. 78,600 "rounded off" to two places becomes 79,000. 1995 "rounded off" to three places becomes 2000, and so on.

### 6.9.1 BINARY "ROUNDING OFF"

The rules for binary numbers simplify greatly since a digit can be only "1" or "0". If the first digit to be dropped is a "1", then round "UP". If it is a "0", then round "DOWN". For example, 1.0101 "rounded off" to a three-place fractional is 1.011. "Rounding off" to a two-place fractional, we have: 1.01. Now suppose we have something like 1.01111. "Rounding off" to three places we have 1.100 because of the "carries". Likewise, "rounding off" to a four-place fractional, we have 1.1000 again because of the "carries". The presence of "carries" when "rounding off" binary numbers occurs much more often than with "rounding off" decimal numbers.

## 6.10 BINARY COMPLEMENT

The BINARY COMPLEMENT of a number is that binary number which has the "1's" and "0's" of its digits interchanged. Examples are:

6.10 BINARY COMPLEMENT (Continued)

BINARY NUMBER	BINARY COMPLEMENT
10111010011	01000101100
111	000
110110010	001001101
10110101000	01001010111
100001111	011110000

Note that the sum of a binary number and its complement is a binary number whose total digits are the same as the original number of digits and all "1's". Illustrating this with the above examples, we have:

10111010011	111	110110010	10110101000	100001111
<u>+01000101100</u>	<u>+000</u>	<u>+001001101</u>	<u>+01001010111</u>	<u>+011110000</u>
11111111111	111	111111111	11111111111	111111111

## 7. BINARY ELECTRONIC COMPUTER CIRCUITS

In order to build a digital electronic computer, we must convert logic FACTS and CONCLUSIONS into electronic circuits. The basic circuit for representing logic FACTS is the FLIP-FLOP. The electronic logic GATE uses the flip-flop "facts" (inputs) to determine a conclusion. In order to produce automatic computer operation, the PULSE GENERATOR is needed to command flip-flop operation.

The three circuits mentioned above (FLIP-FLOP, GATE, and PULSE GENERATOR) represent the very heart of all computers. It is possible to build giant computers with only these three basic circuits!

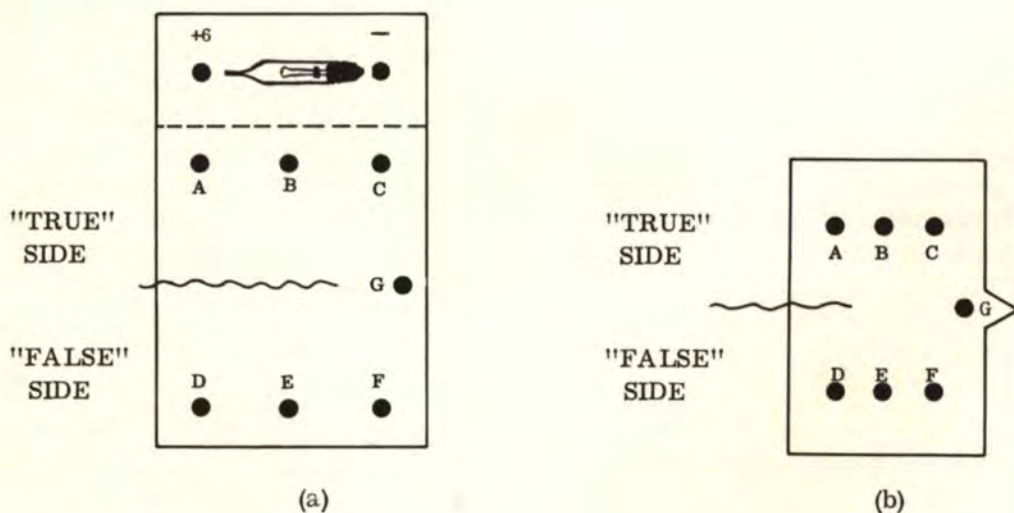
Electronic circuits shown in this section correspond to LIBE electronic circuits now on the market, such as the FF-1, AM-1, AND-1, OR-1, AO-1, and EOR-1. All LIBE circuits are built on printed circuit boards with connection pins protruding from the top side. Connections are made by placing "alligator-clip" wires over these pins. The two voltage pins are located at the top of each unit (the "positive" (+) pin is marked with a red dot and appears in the same position on all units). Power must be wired to each and every unit for operation.

### 7.1 THE FLIP-FLOP

The electronic FLIP-FLOP (the common name for a BISTABLE MULTIVIBRATOR) generates the logic FACTS for the digital computer. The flip-flop can serve as a "memory", perform addition and subtraction, count, and control other circuits.

This basic building block of all computers has millions of uses. There is no limit as to how many can be used in a system. It is possible to build the simplest binary counter or the most complicated computer depending on what is done and how many units are used. The wide range of wiring projects in this book illustrates this point.

Basically, the flip-flop is an electronic circuit which changes state from "ON" to "OFF" or "OFF" to "ON" at the proper pulse signal. One flip-flop can act as a pulse signal to the other, which can in turn trigger another, and so on.



## 7.1 THE FLIP-FLOP (Continued)

The figures above show the pin configuration of the LIBE FF-1 with its "readout" lamp on top (a), and the flip-flop symbol for comparison (b). The dots represent the pin connections (the two power pin connections are not shown in figure b) for wiring the flip-flop. The "tip" on the right-hand side represents the "trigger" input pin.

There are many ways that flip-flops can be wired, the simplest being to connect pin A to pin C and pin D to pin F (along with the proper 6-VDC connections to the voltage pins). A pulse signal wired into pin G (the "trigger" input) will cause the flip-flop to change state. Then either the "TRUE" output (pin A) or the "FALSE" output (pin D) of the driven flip-flop can be used to drive another flip-flop.

A flip-flop is defined to be "ON" when its readout light is on, and "OFF" when its readout light is off. The light is located on the "TRUE" (top) side of the flip-flop. The side without the light is called the "FALSE" side.

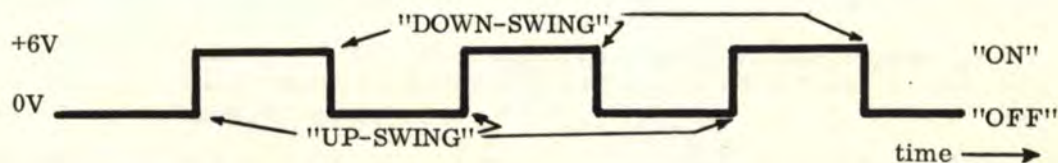
The voltage pins will not appear in any of the flip-flop symbols. They will always be on top and should be wired up to the proper voltages from the battery. Pin-identifying letters will not be shown after the initial explanation. Note that pins A, B, and C will always be in the top row. Pins D, E, and F will be in the bottom row. Pin G will be to the right center edge. The top half of the flip-flop is the "TRUE" side and the bottom half, the "FALSE" side. The light bulb indicator is always on the "TRUE" side (top) between the two voltage pins. The light bulb is also not shown on the flip-flop symbol.

To repeat again: In order to work, each flip-flop must have the proper voltage connections. "+6" must be connected to the pin above A and "-" to the terminal above C. The flip-flop pins are more accurately described as follows:

- A = "TRUE" DIRECTOR OUTPUT ("TRUE" OUTPUT)
- B = "RESET" INPUT
- C = "TRUE" FOLLOWER INPUT
- D = "FALSE" DIRECTOR OUTPUT ("FALSE" OUTPUT)
- E = "SET" INPUT
- F = "FALSE" FOLLOWER INPUT
- G = "TRIGGER" INPUT

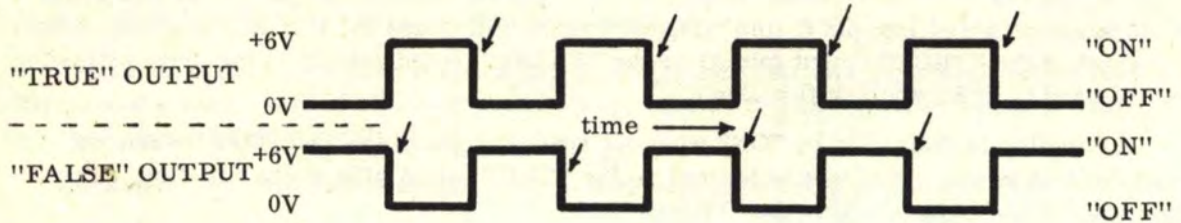
Pin G is where all pulse signals are wired in. Pins A and D are actual pulse signals and can be used to trigger another flip-flop. The FOLLOWER INPUTS (pins C and F) receive voltage pulse commands along with the "TRIGGER" INPUT (pin G) to operate the flip-flop. Pins B and E are used either to "enter" a binary number (turn the light on) or "cancel" (turn the light off). To "enter" a number, touch pin D to pin E. The flip-flop is now "set" (the number has been entered). To "cancel" a number, touch pins A and B together. The flip-flop is now "reset" (the number has been cancelled).

The electrical voltage output on the "TRUE" and "FALSE" OUTPUT pins is in the form of a SQUARE WAVE. This means that the voltage is either "ON" or "OFF" as shown below:



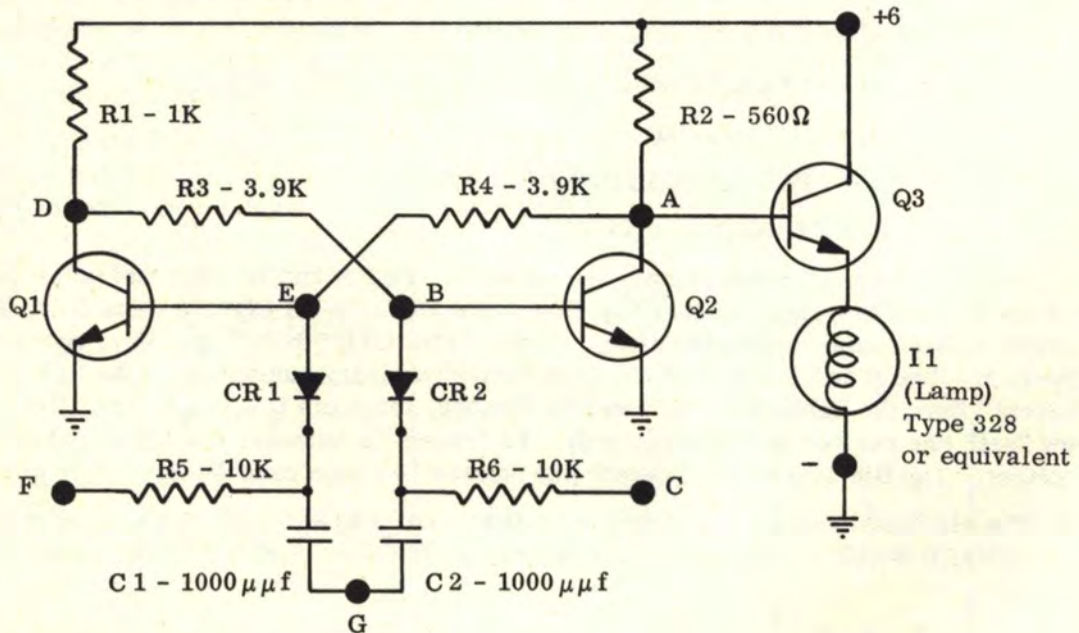
## 7.1 THE FLIP-FLOP (Continued)

Note that the voltage can either RISE from 0V to +6V, or FALL from +6V to 0V. There is no other voltage value in between 0 and +6. The RISE occurs during a voltage UP-SWING and the FALL occurs during the voltage DOWN-SWING. The TRIGGER INPUT is sensitive only to DOWN-SWING voltage changes and the flip-flop will change state on a pulse DOWN-SWING. The "TRUE" OUTPUT and "FALSE" OUTPUT of the flip-flop are always in opposite states:



When using the OUTPUTS of one flip-flop to trigger another flip-flop, it should be noted (from the arrows in the above figure) that the DOWN-SWINGS will occur at different times in relation to the "TRUE" readout light. The "TRUE" OUTPUT will produce a DOWN-SWING when the light goes off, but the "FALSE" output will produce a DOWN-SWING when the light comes on.

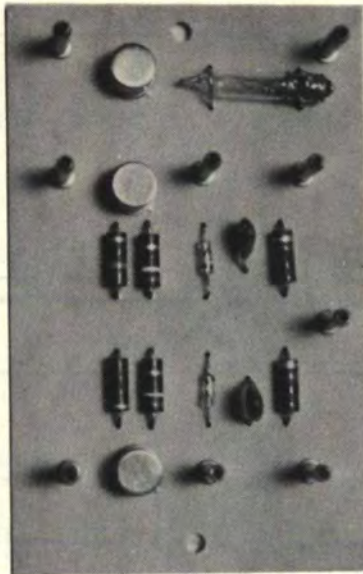
The following circuit is recommended for flip-flop construction and is used for the LIBE FF-1 units:



All resistance values  $-10, +30\%$

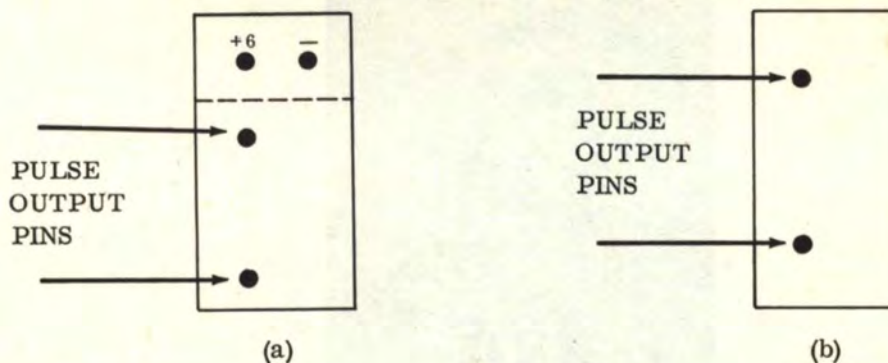
## 7.1 THE FLIP-FLOP (Continued)

The physical layout of the above circuit is on a 2 1/2" by 4" printed circuit board with top and bottom mounting holes as shown in the picture below.



## 7.2 THE PULSE GENERATOR

The PULSE GENERATOR (or ASTABLE MULTIVIBRATOR) is the commanding device that makes the flip-flops work automatically. Usually only one or two pulse generators are needed for specific computer projects (either fast pulse or slow pulse, depending upon application). Either of the two pulse output pins (not the voltage pins!) can be used for the pulse signal.



The figures above show the pin configuration of the LIBE AM-1 with the voltage pins on top (a), and the pulse generator symbol for comparison (b). The dots represent the output pins in figure (b).

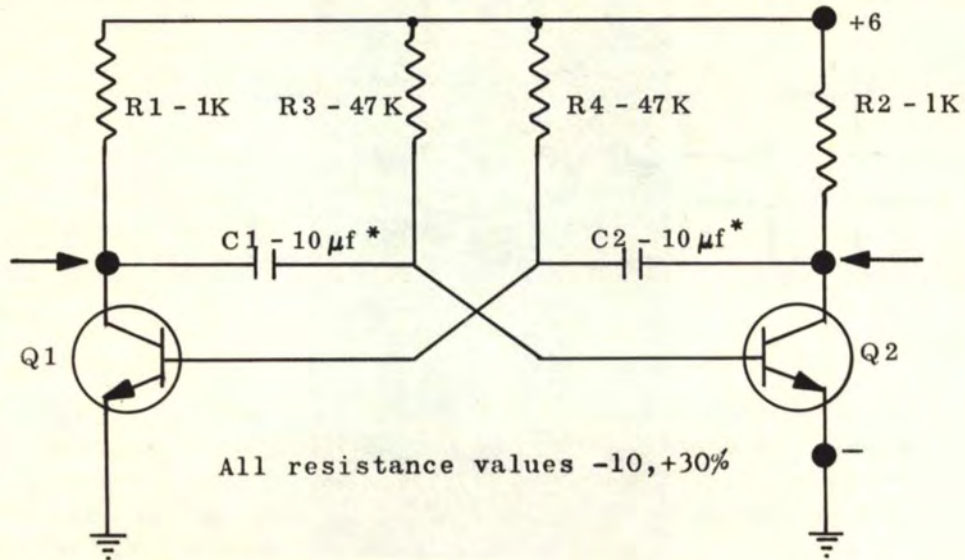
The pulse from either output pin can be wired directly to pin G of one or more flip-flops for automatic "trigger" operation.

The electrical voltage on the PULSE OUTPUT pins is also in the form of a SQUARE WAVE, but the difference is that the pulse generator produces this square wave auto-matically whereas the flip-flop must be driven by a pulse. The DOWN-SWINGS produced

7.2 THE PULSE GENERATOR (Continued)

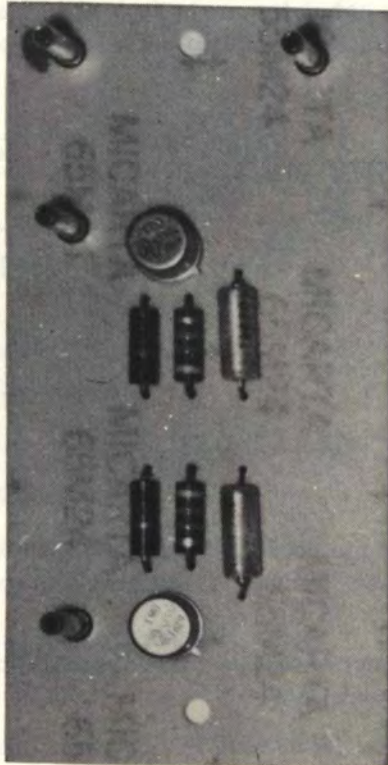
by the pulse generator will cause flip-flop triggering.

The following circuit is recommended for pulse generator construction and is used for the LIBE AM-1 units:



\*The  $10\ \mu\text{f}$  value will produce about 2 pulses per second. Lower value capacitors will produce faster pulse rate frequencies. Higher capacitance values will produce slower frequencies.

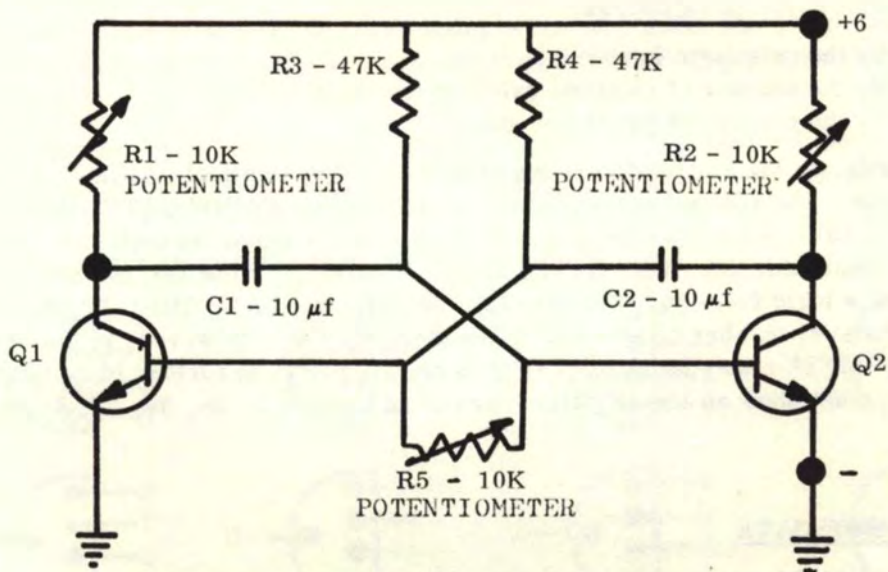
The physical layout of the above circuit is on a 2" x 4" printed circuit board with top and bottom mounting holes, as shown in the picture below.





## 7.2 THE PULSE GENERATOR (Continued)

A variable pulse generator circuit can be built by replacing R1 and R2 by adjustable resistors (potentiometers) of about 10K. Also, a 10K potentiometer can be added between the bases of the two transistors. The following schematic diagram shows a variable pulse generator circuit which will pulse at the rate of about 1 pps to 2000 pps by adjustment of any one or all three of the potentiometers.



All resistance values -10, +30%

### 7.3 LOGIC GATES

Up to now we have discussed flip-flops which generate logic FACTS. Now, using these facts, the LOGIC GATES generate CONCLUSIONS. In other words, logic gating is the control of flip-flop inputs (also "gate" flip-flop inputs) and outputs by additional electronic circuitry called a GATE. Each GATE has a specific logic function (such as "AND", "OR", "EOR", "SUM", etc.) and will produce flip-flop control determined by this logic function.

There are no limits to the kinds of electronic logic gates that can be built. Gates such as "AND", "OR", "NAND", "NOR", "EOR", "SUM", and "INVERTER" are but a very few. However, the most important two gate types which form the basis for all computer building logic are the "AND" and "OR".

Gate NOMENCLATURE is fairly easy. The general format for naming gates built on electronic circuit cards is as follows:

- (1) Identify the gate logic function.
- (2) Identify the number of identical gates on the card.
- (3) Identify the number of inputs per gate.

For example, in the LIBE AND-1, we have 2 separate identical "AND" gates with 4 inputs on each one. The correct nomenclature would then be: "AND" GATE, DUAL, 4-INPUT. The LIBE OR-1 has 2 separate identical "OR" gates with 4 inputs on each one. The correct nomenclature would then be: "OR" GATE, DUAL, 4-INPUT. Note that for step (1), we merely call out a logic function. For step (2), we call out DUAL, TRIPLE, QUADRUPLE, 5-, 6-, for whatever number of identical gates there may be. However, in the case of only one gate, this step may be skipped as it is not necessary to further identify a single gate. Step (3) must show an identification number of at least 2: 2-, 3-, 4-, 5-, 6-, etc.

#### 7.3.1 THE "AND" GATE

The single "AND" gate is represented by a semicircle as shown below. For discussion purposes, we will consider a gate with 4 inputs. Any inputs which are not used are merely left open. The LIBE AND-1 has two identical "AND" gates of 4 inputs each and they are on the right. The outputs are on the left. The top and bottom output pins are the "AND" output pins while the inner two pins are the "NAND" output pins for experimentation with negative logic. Inputs may be wired in from the "true" or "false" side of a flip-flop, or from the output of another gate.

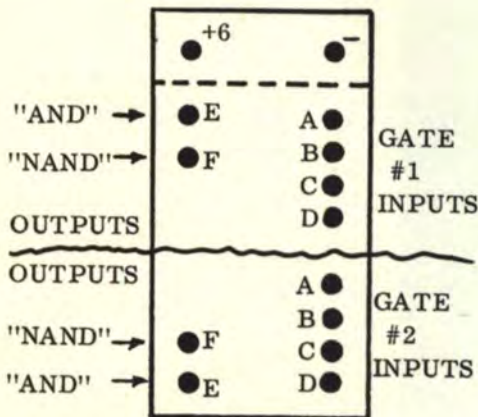


Figure (a)

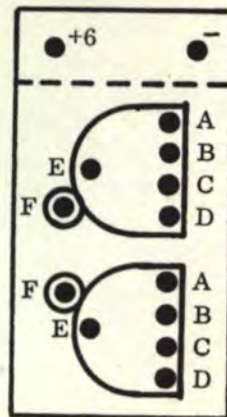


Figure (b)

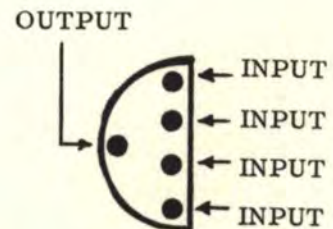


Figure (c)

### 7.3.1 THE "AND" GATE (Continued)

The two top power pin connections are made in the same manner as on the FF-1 flip-flop units.

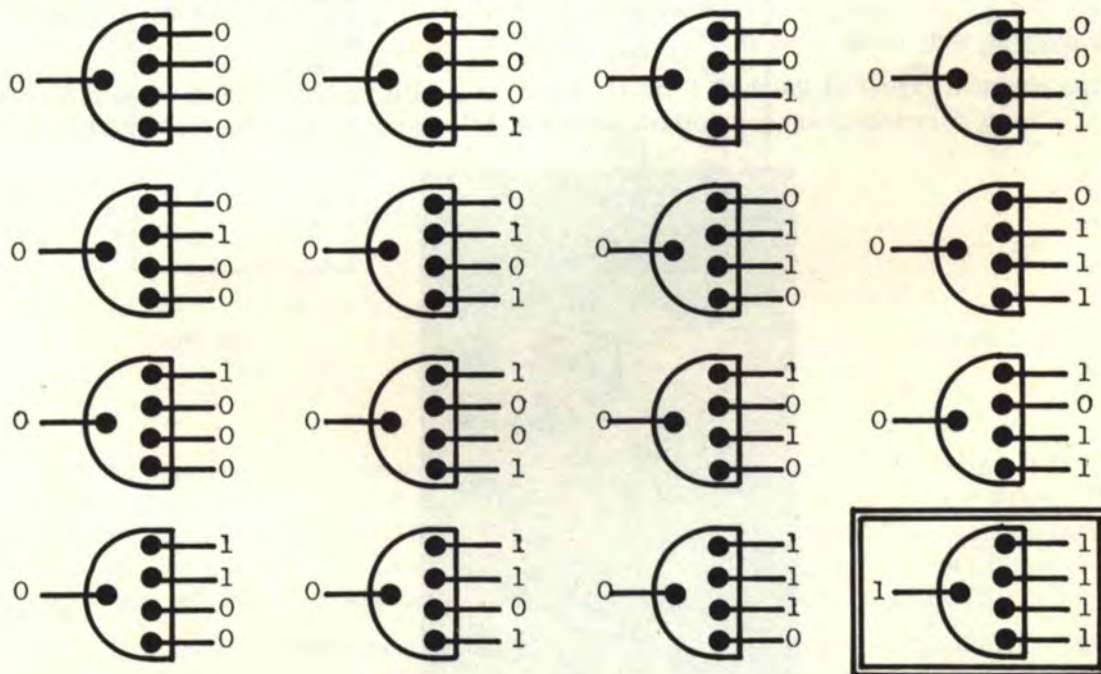
Pins A, B, C, and D are the 4 inputs of each gate (GATE INPUT PINS). The "E" PINS represent the "AND" OUTPUT. Note that these "E" PINS are the outside output pins and that the top gate (#1) has "E" on top while the bottom gate (#2) has "E" on the bottom. The "F" PINS, or inside "OUTPUT" pins are the "NAND" OUTPUTS.

The "AND" output of each respective gate is "ON" only when all its inputs are "ON", and "OFF" when any one of its inputs is "OFF". The "NAND" output of each respective gate is "OFF" only when all its inputs are "ON", and "ON" when any one of its inputs is "OFF".

Both the "AND" and "NAND" outputs can be used for "DOWN-SWING TRIGGERING" of flip-flops; and either output can be wired into pin "G" of a LIBE FF-1 flip-flop to trigger it. If only two or three "AND" inputs are needed, then the other "input" pins should be left open (no connections).

The simplified "gate" logic symbol (referring back to figure c) will be used to show all "AND" logic operations. The small circles, which are tangent to the logic diagrams in figure (b), represent the "NAND" outputs.

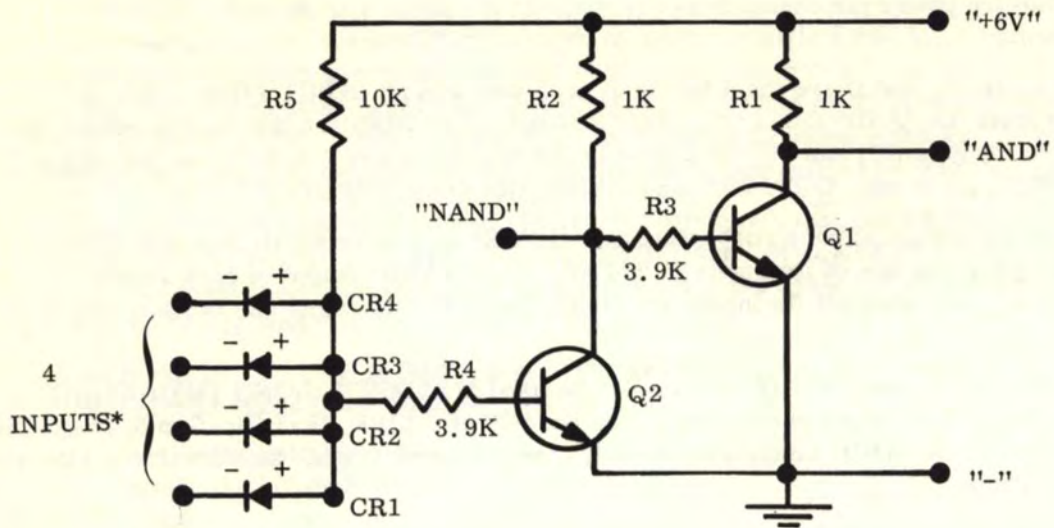
The following input possibilities exist for a 4-input "AND" gate:



Note that the last case is where all 4 inputs are a "1". This is the only case where the output is a "1". In all other cases, the output is a "0". To be more specific, the definition of an "AND" gate is as follows: IF ALL INPUTS, REGARDLESS OF HOW MANY, ARE "1" THEN THE OUTPUT IS A "1". IF ANY INPUT IS A "0", THEN THE OUTPUT IS A "0".

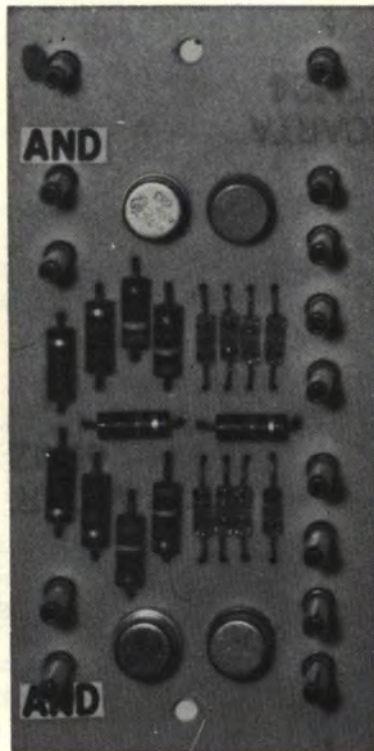
The following circuit is recommended for "AND" gate construction and is used on both the LIBE AND-1 and A0-1 logic gates.

## 7.3.1 THE "AND" GATE (Continued)



\*To increase the number of inputs, add extra diodes in addition to the four already shown. Be sure that all the anodes are connected to the same point. The input points will then be on the cathode ends of the diodes.

The physical layout of the dual 4-input "AND" gate with "NAND" output is on a 2" x 4" printed circuit board with top and bottom mounting holes as shown in the picture below.



### 7.3.2 THE "OR" GATE

The single "OR" gate is represented by a triangle as shown below. For discussion purposes, we will again consider a gate with 4 inputs. Any inputs, which are not used, are merely left open. The LIBE OR-1 has two identical "OR" gates of 4 inputs each and they are on the right. The outputs are on the left. The top and bottom pins are the "OR" output pins while the inner two pins are the "NOR" output pins for experimentation with negative logic. Inputs may be wired in from the "true" or "false" side of a flip-flop, or from the output of another gate.

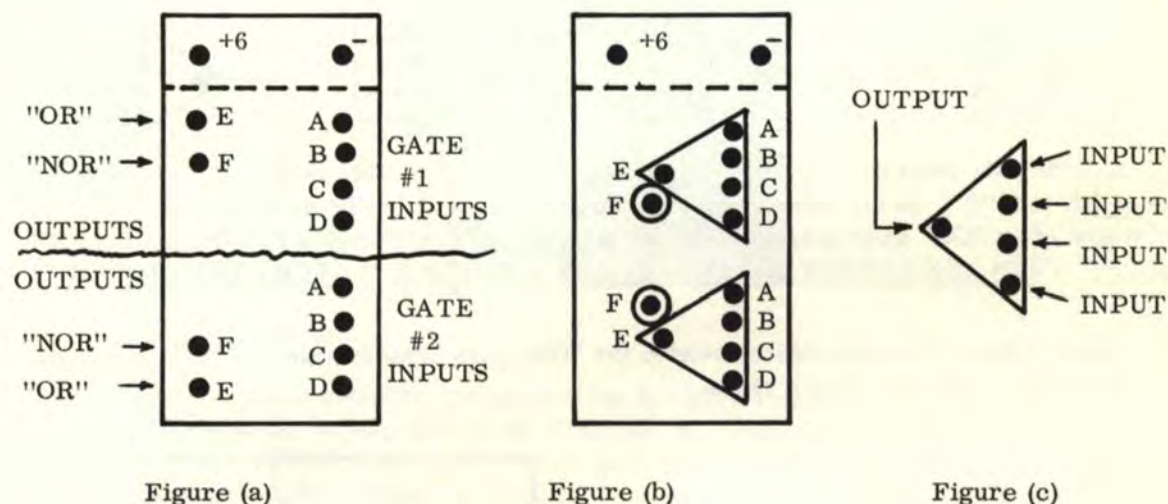


Figure (a)

Figure (b)

Figure (c)

The two top power pin connections are made in the same manner as on the FF-1 flip-flop units. Pins A, B, C, and D are the 4 inputs of each gate (GATE INPUT PINS). The "E" pins represent the "OR" OUTPUT. Note that these "E" PINS are the outside output pins and that the top gate (#1) has "E" on top while the bottom gate (#2) has "E" on the bottom. The "F" PINS, or inside "OUTPUT" pins, are the "NOR" OUTPUTS.

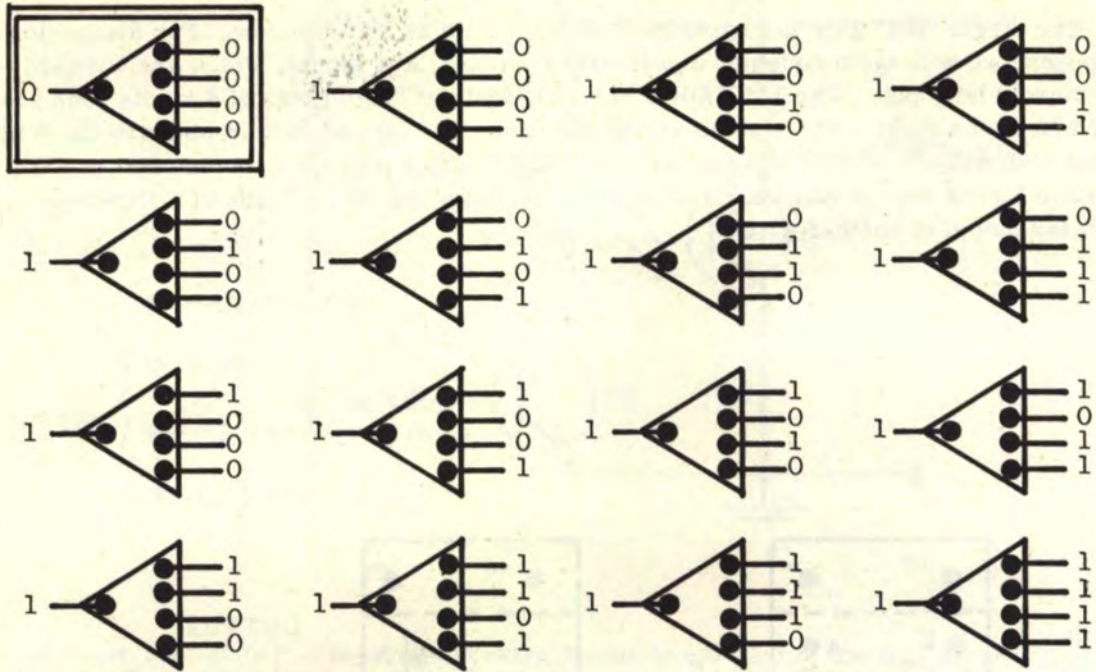
The "OR" output of each gate is "ON" whenever one or more of its inputs is "ON", and "OFF" only when all of its inputs are "OFF". The "NOR" output of each respective gate is "OFF" whenever one or more of its inputs is "ON", and "ON" only when all of its inputs are "OFF".

Both "OR" and "NOR" outputs can be used for "DOWN-SWING TRIGGERING" of flip-flops; and either output can be wired into pin "G" of a LIBE FF-1 flip-flop to trigger it. If only two or three "OR" inputs are needed, then the other input pins should be left open (no connections).

The simplified "gate" logic symbol (referring back to figure c) will be used to show all "OR" logic operations. The small circles, which are tangent to the logic diagrams in figure (b), represent the "NOR" outputs.

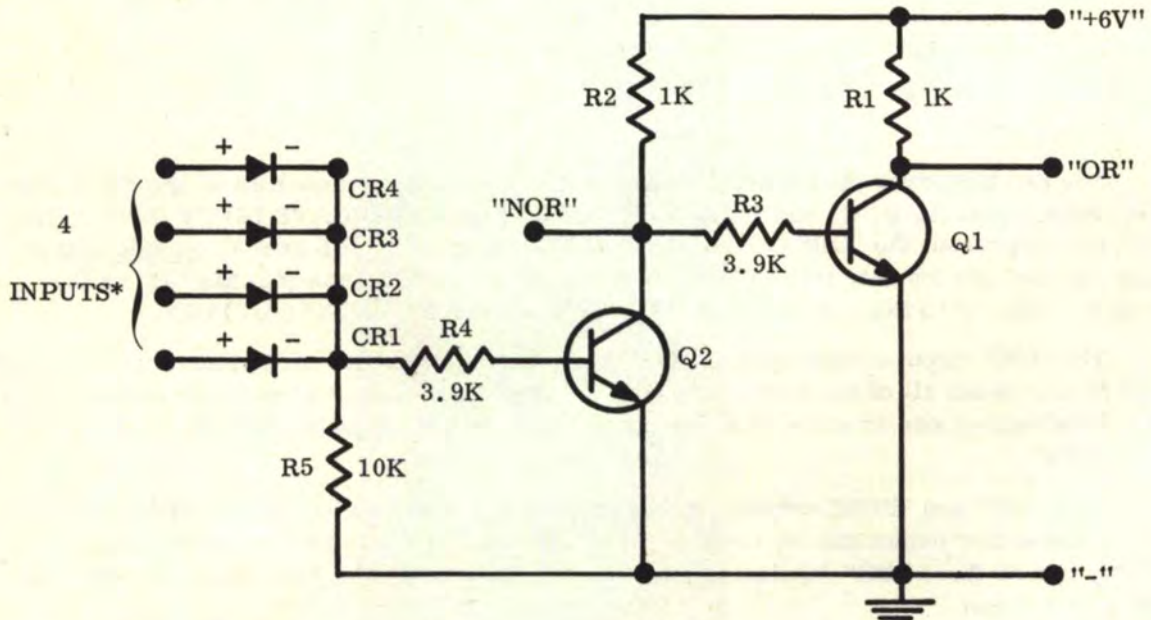
7.3.2 THE "OR" GATE (Continued)

The following input possibilities exist for a 4-input "OR" gate:



Note that the first case is where all 4 inputs are a "0". This is the only case where the output is a "0". In all other cases, the output is a "1". To be more specific, the definition of an "OR" gate is as follows: IF ALL INPUTS, REGARDLESS OF HOW MANY, ARE "0" THEN THE OUTPUT IS A "0". IF ANY INPUT IS A "1" THEN THE OUTPUT IS A "1".

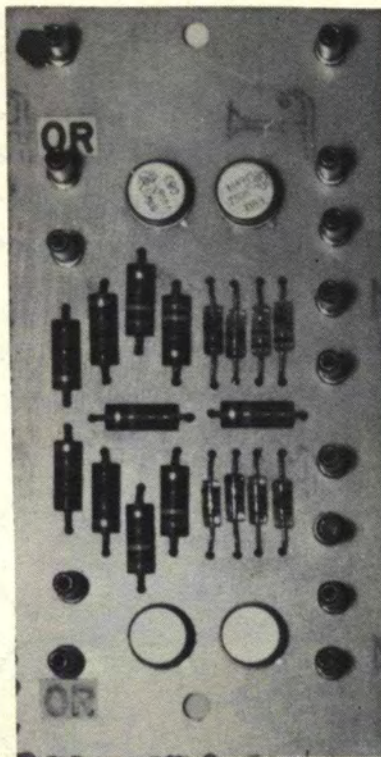
The following circuit is recommended for "OR" gate construction and is used on both the LIBE OR-1 AND A0-1 logic gates.



\*To increase the number of inputs, add extra diodes in addition to the four already shown. Be sure that all the cathodes are connected to the same point. The input points will then be on the anode ends of the diodes.

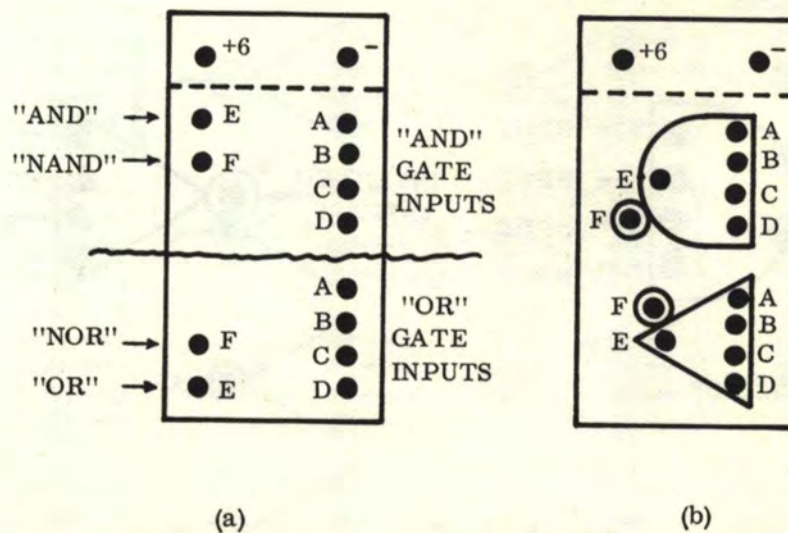
### 7.3.2 THE "OR" GATE (Continued)

The physical layout of the dual 4-input "OR" gate with "NOR" output is on a 2" x 4" printed circuit board with top and bottom mounting holes as shown in the picture below.



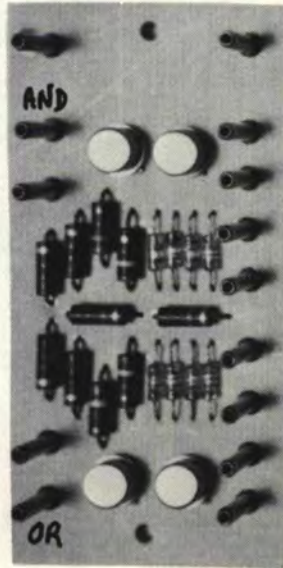
### 7.3.3 THE "AND" & "OR" GATE

The LIBE A0-1 card contains two gates (like the AND-1 and OR-1) but the top gate is an "AND" GATE and the bottom gate is an "OR" GATE. Both gates again have 4 inputs. Power connections, INPUT PINS and OUTPUT pins are similar to both the AND-1 and OR-1 configurations. The figures below can be compared to those of the AND-1 and OR-1.



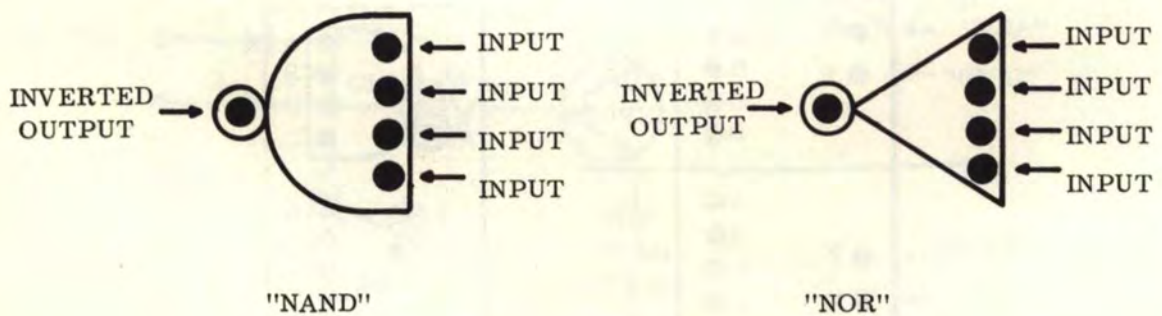
### 7.3.3 THE "AND" & "OR" GATE (Continued)

The physical layout of this gate is on a 2" x 4" printed circuit board with top and bottom mounting holes as shown in the picture below.



### 7.3.4 NEGATIVE "AND" & "OR" GATES ("NAND" & "NOR")

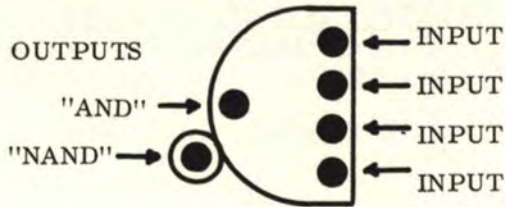
As was mentioned earlier, "NAND" and "NOR" are shortened expressions for "NOT AND" and "NOT OR". In other words, by putting the letter "N" before "AND" or "OR" means we have a gate with the outputs inverted. An inverted output is a "1" instead of a "0", or a "0" instead of a "1", (i. e., replace "1" by "0" and "0" by "1"). The electronic gate symbols for "NAND" and "NOR" are the same as the corresponding "AND" and "OR" except that a small circle on the output indicates that the output is inverted. Both "NAND" and "NOR" gates are shown below:



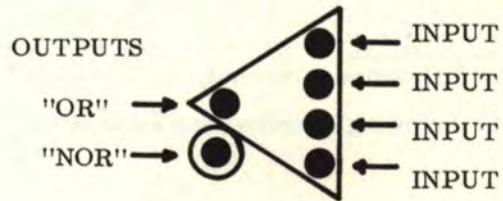


7.3.4 NEGATIVE "AND" & "OR" GATES ("NAND" & "NOR") (Continued)

The LIBE "OR-1", "AND-1", and "A0-1" gates have one regular and one inverted output. That is, the "AND" gate also has a "NAND" output and the "OR" gate also has a "NOR" output. If both the regular output and inverted outputs are used, then the "invert" circle is added to the gates as follows:

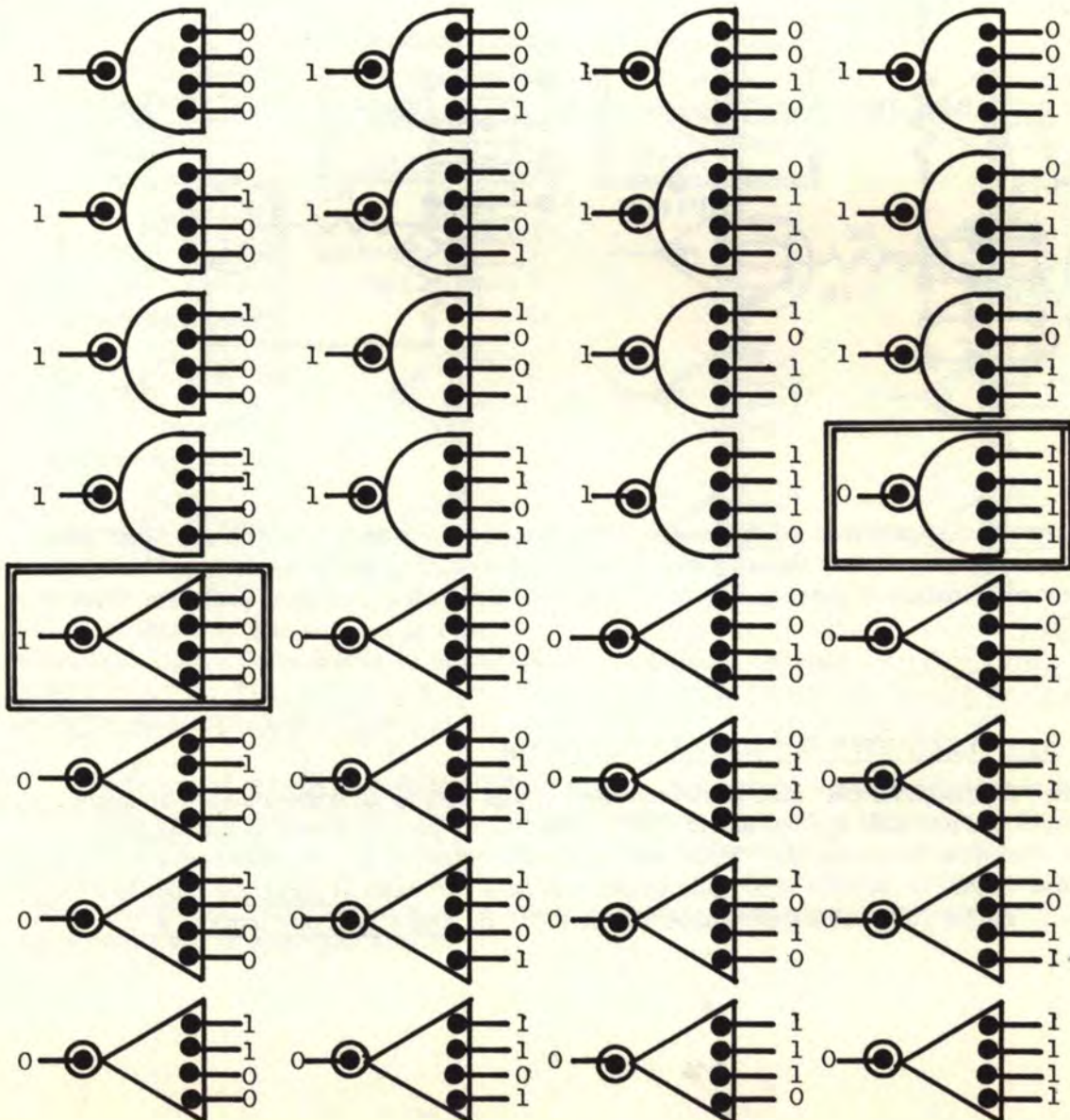


"AND" GATE WITH "NAND" OUTPUT



"OR" GATE WITH "NOR" OUTPUT

Again, considering all the 4 possible inputs for a "NAND" and "NOR" gate, the following possibilities exist:

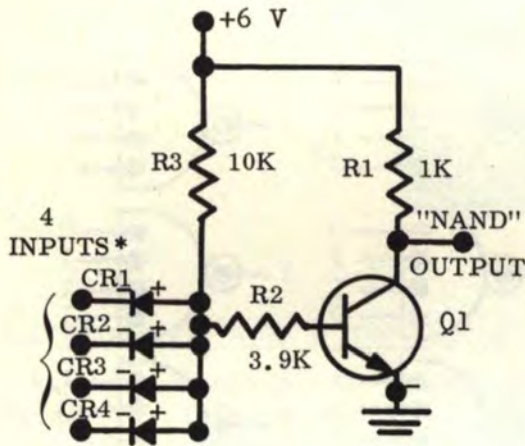


### 7.3.4 NEGATIVE "AND" & "OR" GATES ("NAND" & "NOR") (Continued)

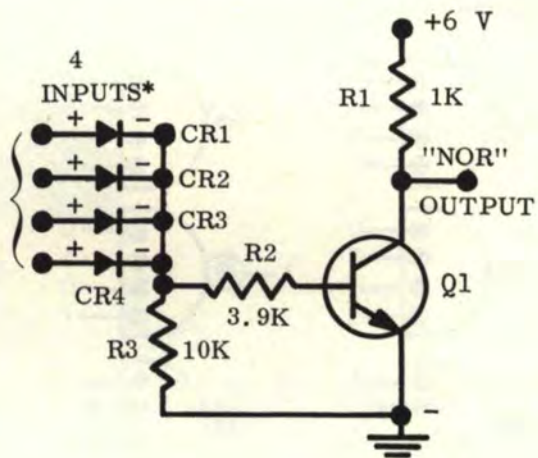
Note the last case for the "NAND" gate and the first case for the "NOR" gate. Specific definitions are as follows: IF ALL INPUTS, REGARDLESS OF HOW MANY, OF A "NAND" GATE ARE "1" THEN THE OUTPUT IS A "0". IF ANY INPUT IS A "0", THEN THE OUTPUT IS A "1". IF ALL INPUTS, REGARDLESS OF HOW MANY, OF A "NOR" GATE ARE "0" THEN THE OUTPUT IS A "1". IF ANY INPUT IS A "1", THEN THE OUTPUT IS A "0". Compare these definitions with those for an "AND" and "OR" gate and note the difference in the outputs.

The following circuits may be used for "NAND" and "NOR" gates:

"NAND" CIRCUIT



"NOR" CIRCUIT

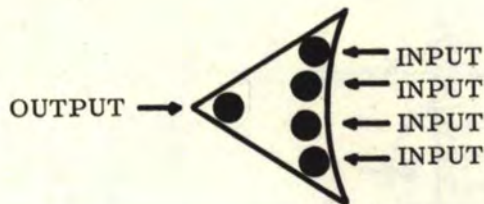


\*The number of inputs may be increased in the same manner as the "AND" and "OR" gates. The physical layout of this negative gate is also in a 2" x 4" printed circuit as a dual gate with any combination of gates (i. e., dual "NAND", dual "NOR", or dual gate with "NAND" & "NOR").

### 7.3.5 THE "EXCLUSIVE OR" GATE ("EOR" GATE)

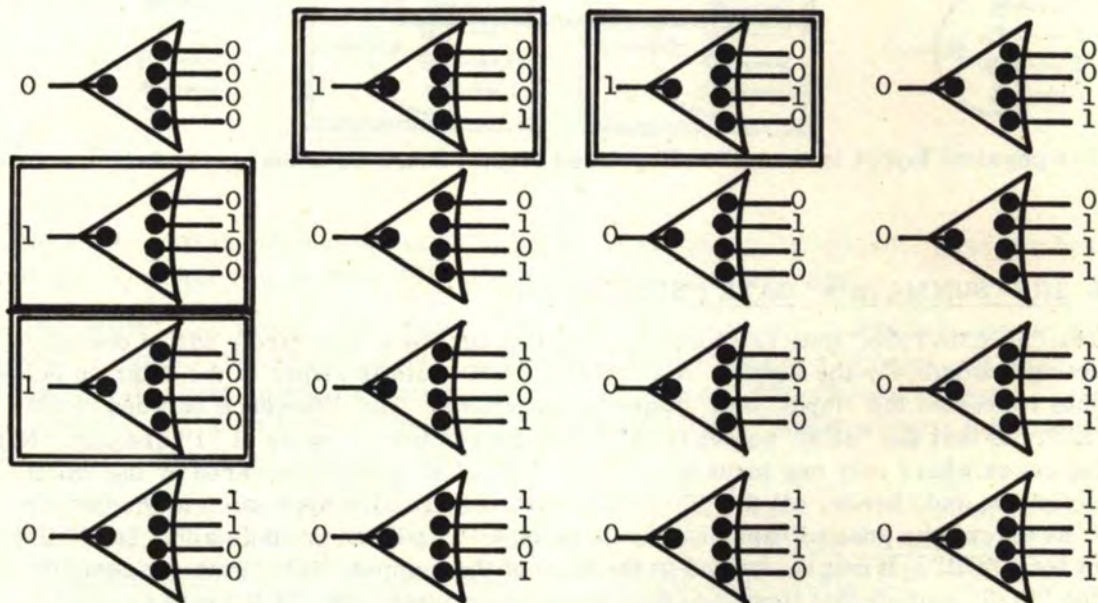
The "EXCLUSIVE OR" gate is represented in this text by an edge-standing triangle with a concave arc side for the inputs. A 4-input "EOR" gate is shown in the diagram below. The dots represent the "input" and "output" connections. The difference between "OR" and "EOR" is that the "OR" gate output will be a "1" when at least one input is a "1". However, the "EXCLUSIVE OR" output will be a "1" if, and only if one input is a "1".

## 7.3.5 THE "EXCLUSIVE OR" GATE ("EOR" GATE) (Continued)



An "EOR" gate of any number of inputs may be constructed by the sole use of "AND" and "OR" gates. This will be discussed in section 10.

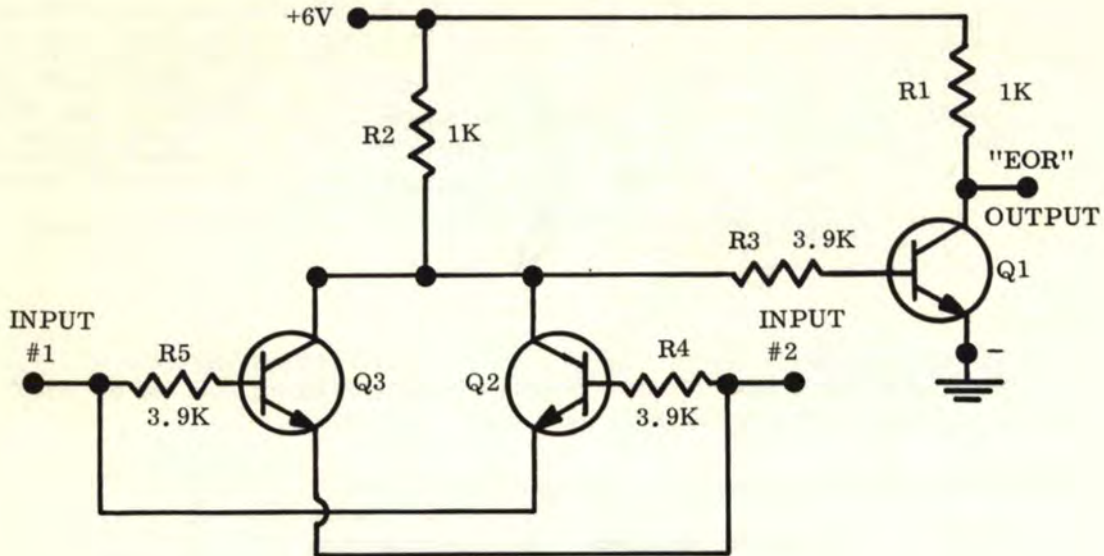
The following possibilities exist for a 4-input "EOR" gate:



Note that only where there is a single "1" input, the "EOR" output is a "1". To be more specific, the definition of an "EOR" gate is as follows: IF ONE AND ONLY ONE INPUT IS A "1", THE OUTPUT IS A "1". IF ANY OTHER INPUT IS A "1" OR IF ALL INPUTS, REGARDLESS OF HOW MANY, ARE "0" THEN THE OUTPUT IS A "0".

The following circuit is used for "EOR" gates, but a severe limitation is that it produces sharp electronic "spikes" (false "trigger" pulses) when any of its inputs go from "0" to "1" or from "1" to "0". Hence, it is only useful for direct display outputs and not for "triggering". The circuitry for "EOR" gates of more than 2 inputs gets very complicated. These multi-input "EOR" gates operate much more efficiently when "AND" and "OR" equivalent logic is used (see section 10).

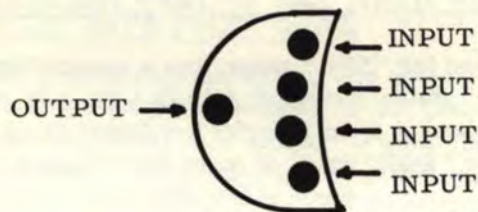
### 7.3.5 THE "EXCLUSIVE OR" GATE ("EOR" GATE) (Continued)



The physical layout is on a 2" x 4" printed circuit board as a dual gate, 2-input.

### 7.3.6 THE "SUMMATION" GATE ("SUM" GATE)

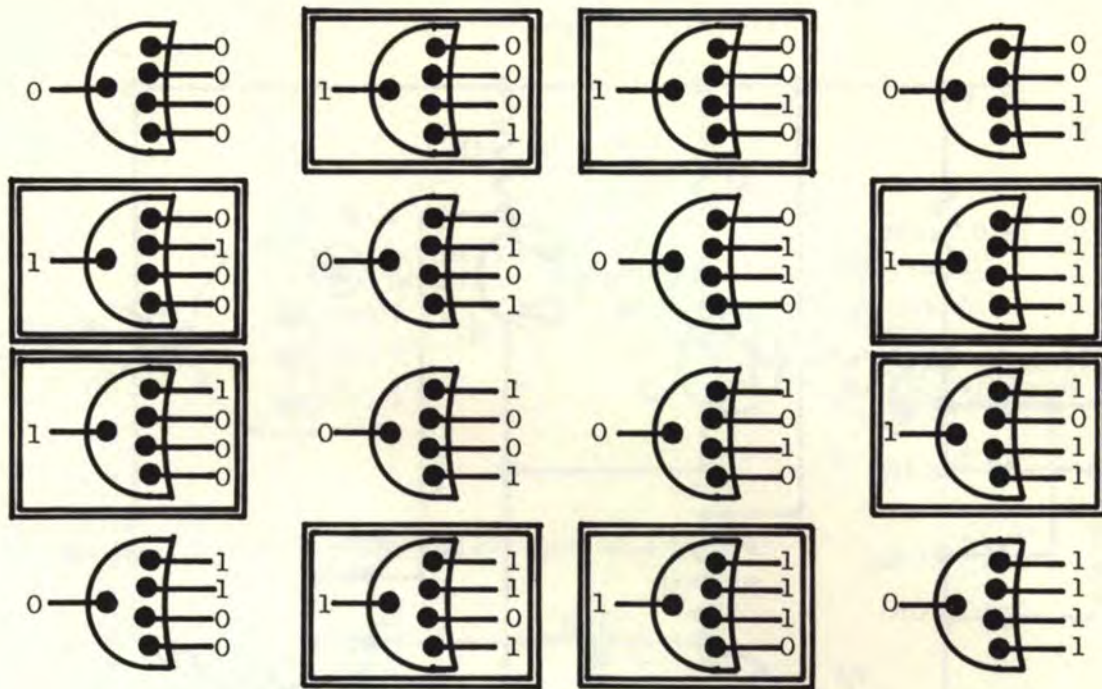
The "SUMMATION" gate is represented in this text by a semicircle with a concave connecting arc side for the inputs. A 4-INPUT "SUM" gate is shown in the diagram below. The dots represent the "input" and "output" connections. The difference between "SUM" and "EOR" is that the "SUM" output is "1" when the number of inputs of "1" are odd. Note that the cases where only one input is a "1" (in the "EOR" gate) is covered by the "SUM" gate definition and, hence, all the "EOR" possibilities are also applicable to the "SUM" gate. However, the possibilities of 3 inputs being a "1" will show up as a "1" for "SUM", but not for "EOR". It can be noticed in the case of the 4-input "SUM" gate (by comparison with the "EOR" output) that there are four more cases where the "SUM" gate is a "1".



A "SUM" gate of any number of inputs may be constructed by the sole use of "AND" and "OR" gates. This will be discussed in section 10.

The following possibilities exist for a 4-input "SUM" gate:

## 7.3.6 THE "SUMMATION" GATE ("SUM" GATE) (Continued)



Note that only an odd number of "1" inputs will produce a "1" output. Since we have 4 possibilities, either one or three "1" inputs will produce a "1" output.

The "SUM" gate is also known as a "HALF ADDER" and can, in addition, be considered a "PARITY" gate. "PARITY" as used in this text, is the condition of being odd or even. That is, it will sense an odd or even number of "1" inputs. If there are no inputs, 2 inputs, or 4 inputs of "1", then this is considered "EVEN PARITY". If there is one input or three inputs of "1", then this is considered "ODD PARITY".

In other words, the "SUM" gate will sense even or odd parity of "1" inputs. To be more specific, the definition of a "SUM" gate is as follows: IF THE PARITY OF ALL "1" INPUTS, REGARDLESS OF HOW MANY, IS ODD, THEN THE OUTPUT IS A "1". IF THE PARITY OF ALL "1" INPUTS IS EVEN, THEN THE OUTPUT IS A "0".

This can be verified by adding up any combination of binary "1's", or "1's" and "0's", noting only the SUM and ignoring any "carries".

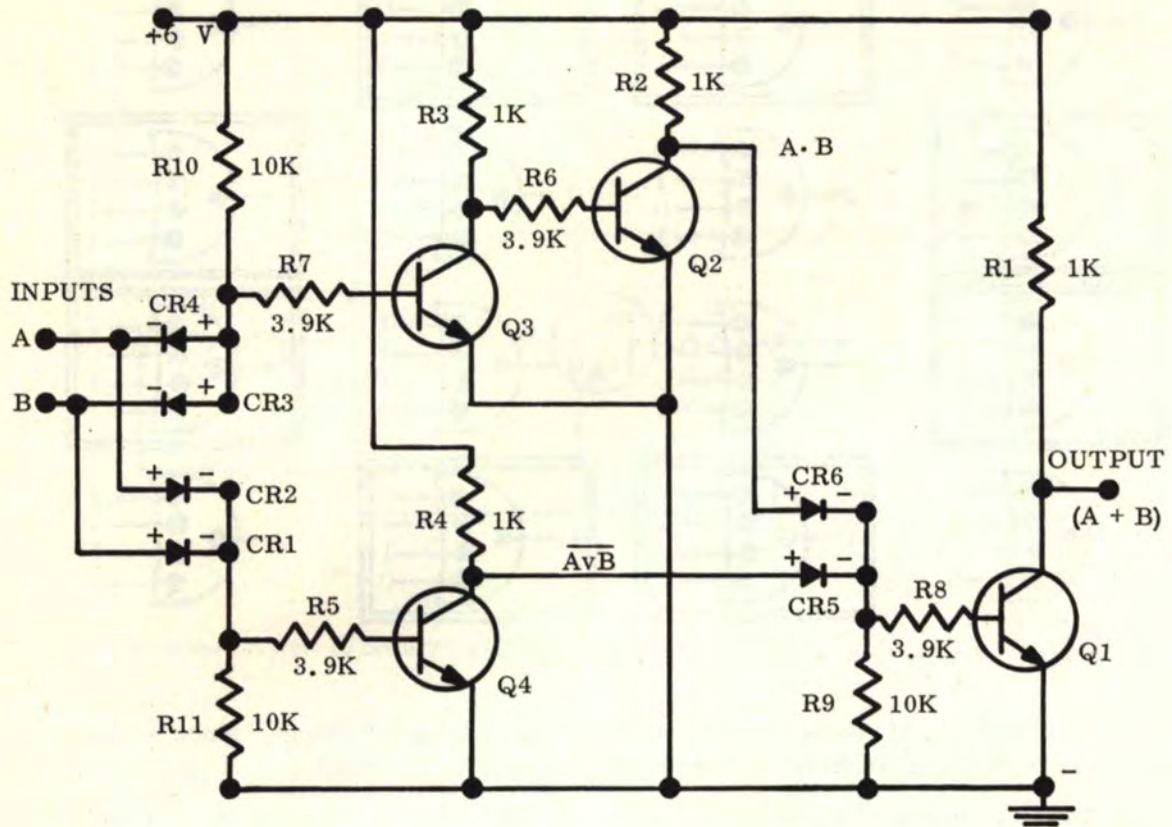
Examples:

$$\begin{array}{ll}
 1 + 1 = 0 & \text{(ignore "carry" 1)} \\
 1 + 1 + 1 = 1 & \text{(ignore "carry" 1)} \\
 1 + 1 + 0 + 0 + 1 + 1 + 1 = 1 & \text{(ignore "carry" 10)}
 \end{array}$$

Note that a 2-input "SUM" gate is exactly the same as a 2-input "EOR" gate. This means that for two inputs only, the "SUM" and "EOR" gates may be used interchangeably.

The following circuit is suggested for a 2-input "SUM" gate. The outputs of this circuit are free from electronic "spikes" and can be used for flip-flop triggering or direct display outputs. The circuitry for "SUM" gates of more than 2 inputs gets very complicated. These multi-input "SUM" gates operate much more efficiently when "AND" and "OR" equivalent logic is used (see section 10).

## 7.3.6 THE "SUMMATION" GATE ("SUM" GATE) (Continued)



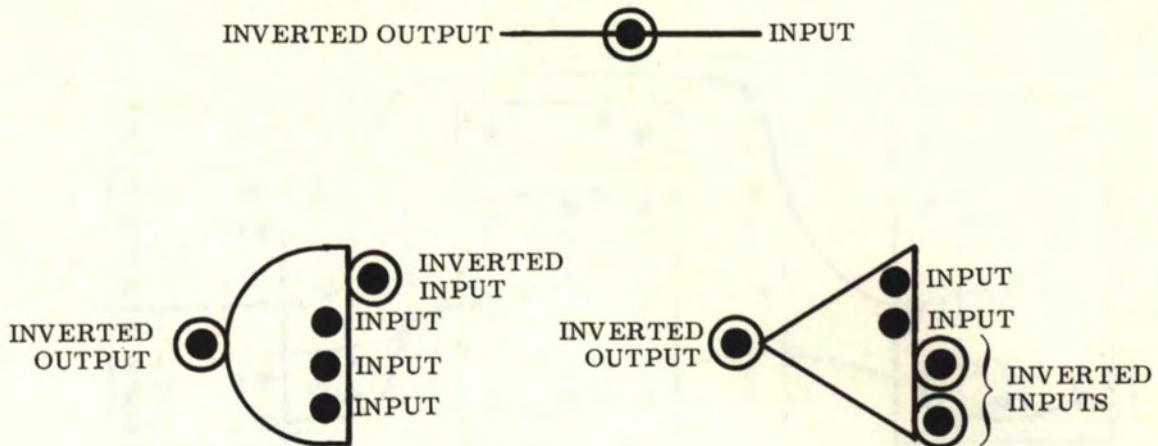
The physical layout is for a single 2-input "SUM" gate on a 2" x 4" printed circuit board.

## 7.3.7 THE "INVERTER" GATE ("NOT" GATE)

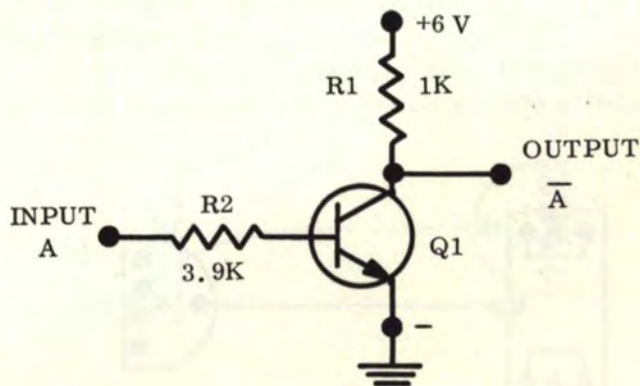
The function of the "INVERTER" gate is to change the output into the opposite (or "NOT") of what the input is. The "INVERTER" gate has only one input. By this definition, the inverted gates "NAND" and "NOR" operated with single inputs will act as "INVERTER" gates.

These "INVERTER" gates will be used very little (or not at all) because inverted outputs are available from the "FALSE" sides of flip-flop inputs, and from the "NAND" and "NOR" outputs of the "AND" and "OR" logic gates. The electronic gate symbol for "NOT" is a small circle in front of an INPUT or OUTPUT position as shown below. The dot within the small circle represents the pin connections which would be made.

### 7.3.7 THE "INVERTER" GATE ("NOT" GATE) (Continued)



The basic circuit for the "INVERTER" gate is as follows:

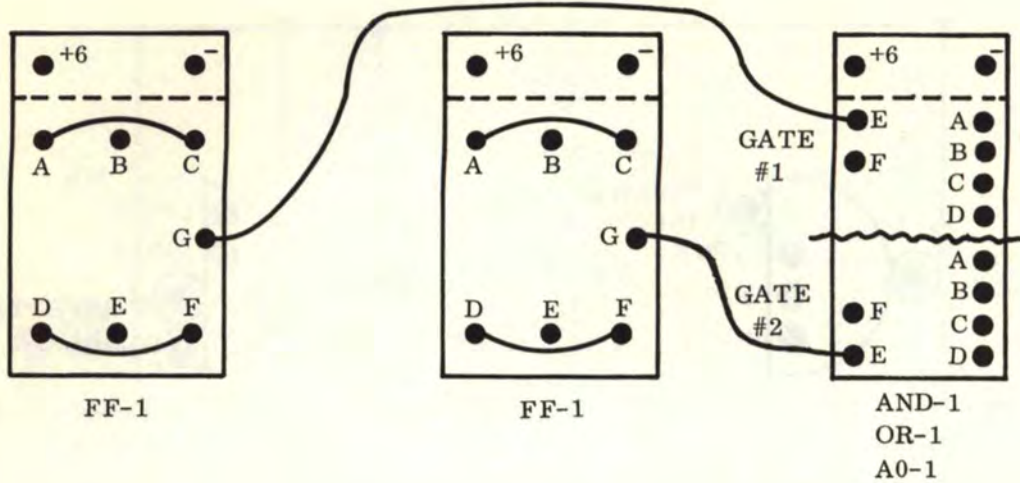


The physical layout consists of four of this single-input inverter on a 2" x 4" printed circuit board.

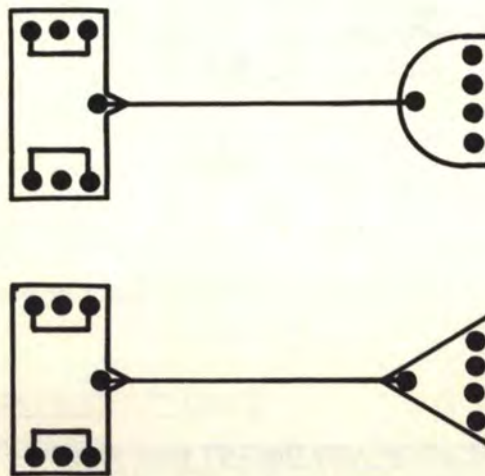
### 7.4 GATE OUTPUT "TRIGGERING" AND DIRECT DISPLAY

To produce flip-flop "triggering" by gate outputs, just wire the outputs directly to the flip-flop "trigger" point (pin "G" of the LIBE FF-1). Only one gate output may be used as a "trigger". The basic wiring for "triggering" is shown below:

7.4 GATE OUTPUT "TRIGGERING" AND DIRECT DISPLAY (Continued)



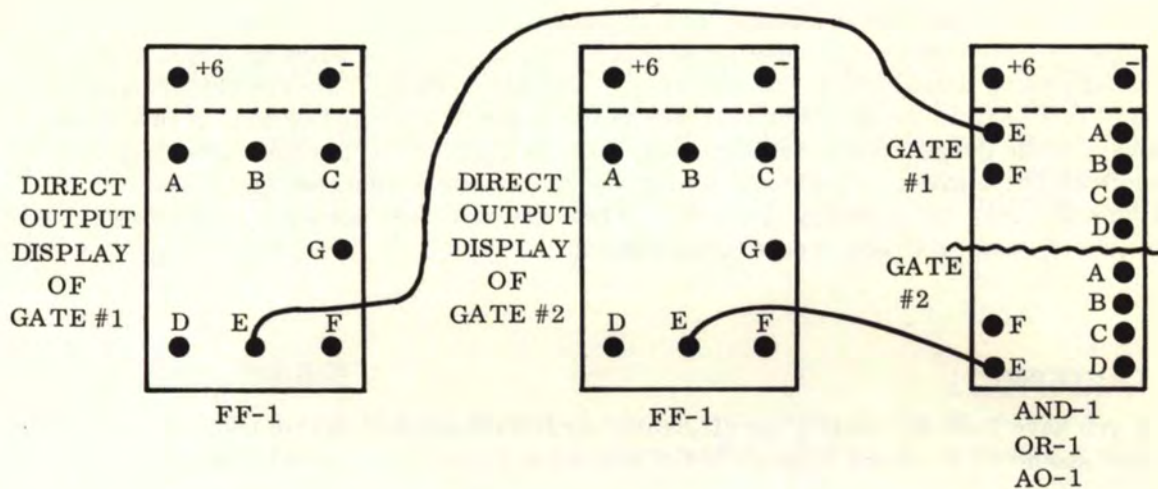
In this configuration, the left FF-1 is controlled by gate #1 output and the right FF-1 is controlled by the gate #2 output. Note that FF-1 pins "A" and "C" and "D" and "F" must be wired together for "triggering". The gate outputs can also be used to "trigger" a shift register or several other FF-1 flip-flops simultaneously. If the above gate were an A0-1, the logic representation of the above diagram would be as follows:



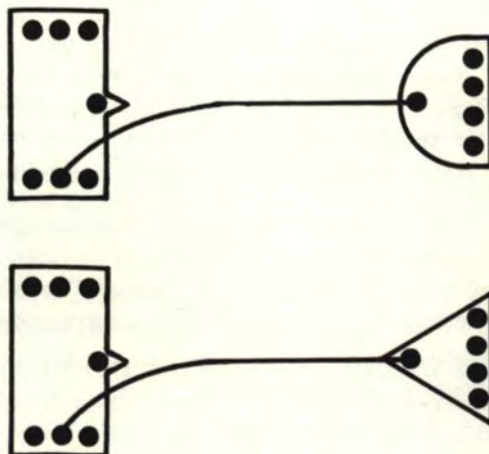
If a gate output is wired directly into pin "E" of an FF-1 flip-flop, the flip-flop light will display the DIRECT OUTPUT of the logic gate. This applies to all types of logic gate outputs. In this case, the flip-flop is not "triggered" but is driven directly by the gate output. No other flip-flop connections (except power) are necessary. Note the DIRECT OUTPUT DISPLAY configuration below:



## 7.4 GATE OUTPUT "TRIGGERING" AND DIRECT DISPLAY (Continued)



Assuming again that the gate in the example above is an A0-1, the logic representation of the above diagram would be as follows:



A gate is not designed to be used both as a "trigger" to a flip-flop and a direct output flip-flop display at the same time.

## 8. ELECTRONIC COMPUTER CIRCUIT OPERATION

Now that we know computer logic and the computer electronic circuits which make up this logic, we are now ready to use these basic building blocks to build up computer wiring projects by using these appropriate blocks over and over again. This chapter bridges the realm of theory from the previous chapters to the world of true applications of this theory. However, other things have to be discussed, such as CAUTIONS, physical mounting and connections, explanation of logic and wiring diagrams, power sources, and potential problems that may occur during operation. Also, discussion is necessary on "senses" and "commands", and "set" and "reset" capabilities.

### 8.1 CAUTIONS!!!!

If you have built (or bought) the electronic circuits described in this book, the following cautions should be observed in order not to cause unnecessary failure of your units.

1. DO NOT EXPERIMENT ON METAL SURFACES. The reason should be obvious. Conducting metal can short out printed circuit board traces or other exposed connections on the units.
2. BE CAREFUL OF THE "HOT" +6 VOLTAGE WIRES. If "+6" voltage hits pins A, B, D, or E on the flip-flop, either pulse generator output pin, or any gate output pin, immediate destruction of the units will result. This is also true if "+6" directly touches some other parts on these circuits.
3. AVOID CARELESS WIRING. Be sure that all voltage wires are properly connected to only the proper voltage pins. Sometimes, however, there may be direct voltage connections only to pins C and F of a flip-flop. These are the only two pins on any of the circuits (other than the voltage pins) that can accept direct voltage. If and when voltage connections to pins C and F are required, be sure that these are the only connections made to these pins!
4. DO NOT USE EXCESS VOLTAGE. These units were designed to run on 6 volts for maximum efficiency. However, satisfactory operation can be obtained with 8 volts, but this shortens the life of the lamps on the flip-flops slightly. The circuits can be operated up to 20 volts, but voltages in excess of 9 volts are not recommended.
5. REMOVE POWER WHEN WIRING OR CHANGING CONNECTIONS. All it takes is one slip of a hot "+6" wire to destroy your units. Other wires accidentally touching hot "+6" voltage connections will yield the same destructive results!
6. DO NOT POUND, DROP, OR CAUSE PHYSICAL SHOCK TO EITHER MOUNTED BANKS OF UNITS OR INDIVIDUAL UNITS. This may result in broken lamp filaments, or severing of the very fine microconnections inside the individual transistors and diodes which comprise these circuits. Avoid pounding in nails to mount your circuits. Use tacks or screws. Above all, be careful that the units do not drop on the floor.

### 8.2 PHYSICAL LAYOUT, MOUNTING, AND CONNECTIONS

The individual flip-flop, pulse generator, dual 4-input "OR" gate, dual 4-input "AND" gate, and the 4-input "AND" and "OR" gate are commercially available as FF-1, AM-1, OR-1, AND-1, and AO-1, respectively, from LIBE COMPANY. The FF-1 is built on a 2 1/2" x 4" printed circuit board with top and bottom mounting holes while the AM-1, OR-1, AND-1, and AO-1 are on 2" x 4" printed circuit boards with top and bottom mounting holes.

## 8.2 PHYSICAL LAYOUT, MOUNTING, AND CONNECTIONS (Continued)

If the reader wants to build his own circuits, it is suggested that these measurements be maintained when using printed circuit boards. The pin (or terminal) input and output locations should also be consistent with those of the commercially available units.

The beginning experimenter can build his own circuit on matrix board (vectorboard) with low cost surplus parts. These matrix boards should be designed to be screwed into a wooden mounting board. We will leave the method of building the units up to the experimenter's own preferences. However, the connection points (pins or terminals) on each similar unit should be consistent from one unit to another. It is also suggested that, when building logic gates, use four (4) inputs for each gate.

### 8.2.1 MOUNTING BOARDS

A wooden board covered with an attractive coat of spray paint makes an excellent mounting and display board. The simplest mounting method consists of screwing the units directly on the board and making wiring connections from the front. However, if permanent wiring is desired, holes can be drilled in the board (remove units from board first) so that wiring can be done from the back. As a starter, a 16" x 32" piece of plywood about 1/2 to 3/4 inch thick is suggested. All units should be mounted with the power pins on top. The flip-flop light should also be at the top of the unit. Be sure to mount all units before doing any wiring.

### 8.2.2 WIRES

You will need many, many wires! A fair estimate is five wires per every electronic unit used. If temporary experimental wires are desired, small alligator clips soldered on each wire end are recommended. Suggested wire lengths are: 8 inches, 16 inches, 24 inches, and 32 inches. If a large number of wire lengths are to be made up at one time the following percentages are estimated for the quantity of wires needed of the above four lengths: 80% - 8 inches, 15% - 16 inches, 3% - 24 inches, and 2% - 32 inches. Usually, 8-inch and 16-inch lengths will suffice. Wire lengths can also be made up as needed. If permanent soldered wire connections are desired, then the wires only need be stripped at each end and made ready for soldering. Otherwise, small alligator clips should be mounted and soldered onto each end. Sometimes it is feasible to solder on only all the power wire connections. By using alligator clips, multiple connections can be made at a single point by the clip-on-clip method.

The DIODE WIRE is a wire with a diode spliced into the middle of it. Usually the 8-inch wire is best for the DIODE WIRE.

### 8.2.3 LABELS

After completing a wiring project, it always helps when a project is properly labeled--especially when explaining it to someone else, or for demonstration purposes. The following labels are suggested:

1. Label each flip-flop (above the light) as "1", "2", "4", "8", "16", etc. to define what number it represents. These numbers can then be added up when the lights are "on" and ignored when the flip-flop lights are "off".

## 8.2.3 LABELS (Continued)

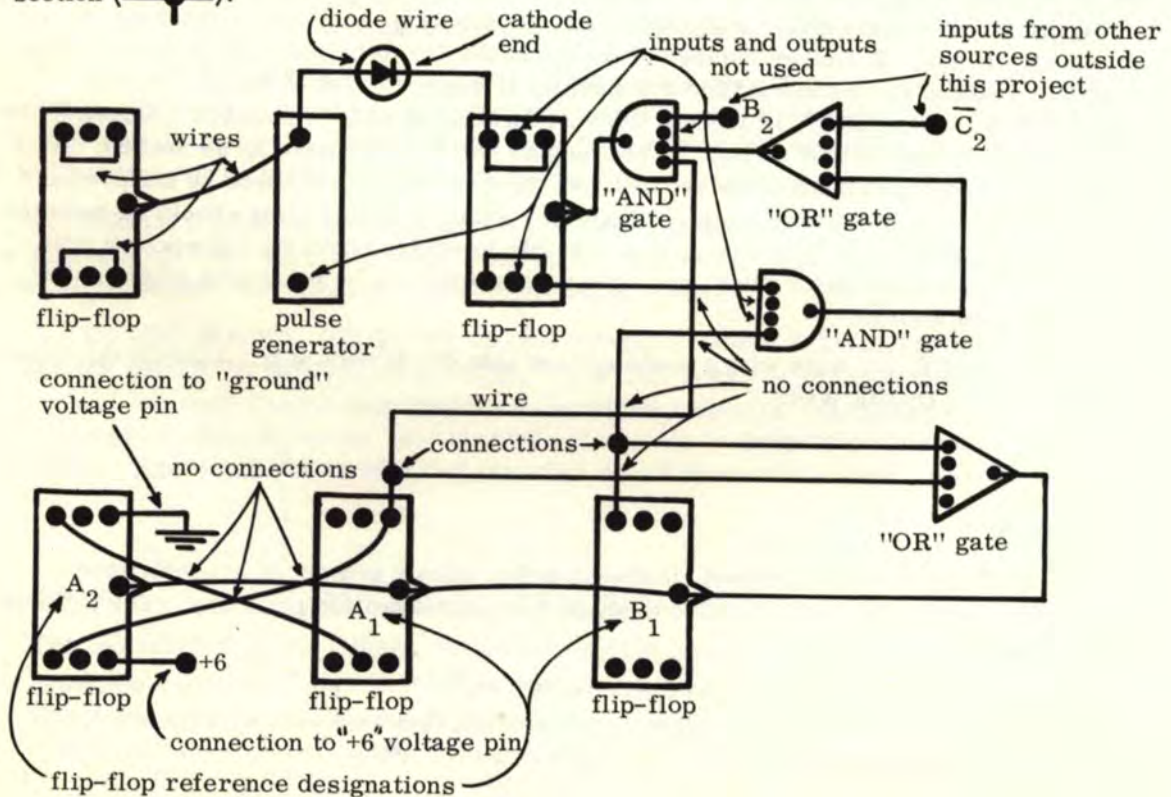
2. Label each flip-flop register such as "ADDEND REGISTER", "ACCUMULATOR REGISTER", etc. These are shown in the respective logic diagrams for the wiring projects.
3. Label other functions such as "MULTIPLICAND CONTROL", "HALT COMMAND", "SAMPLE-AND-HOLD LOGIC", etc., as necessary. These are also shown in the respective logic diagrams.

## 8.3 LOGIC DIAGRAMS AND WIRING

The projects in the next two chapters are all illustrated by the use of logic diagrams. The symbols for the flip-flop and pulse generator are explained in the previous chapter. The gates are shown individually and not in the "dual" configuration. The gate symbols are also explained in the previous chapter. All dots within the symbols represent points of possible connection. All lines, whether curved, straight, or "cornered", represent WIRE CONNECTIONS. No power connections are shown in the logic diagrams. However, power connections must be made on each and every unit individually in order for it to operate. The logic diagrams, in effect, explain what to do only after the power connections have been made. The DIODE WIRE is represented as follows:



Now let us look at a sample logic diagram and explain each point in the figure. In the wiring, all lines which cross (—) are not connected unless there is a dot at the intersection (—).

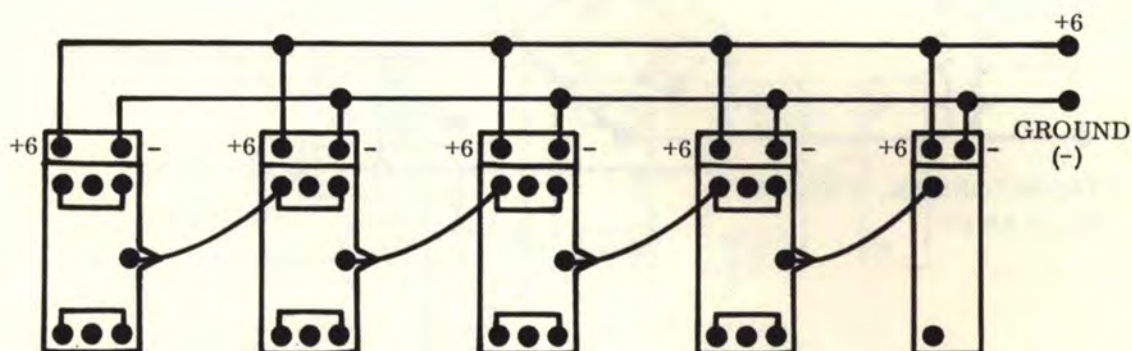


### 8.3 LOGIC DIAGRAMS AND WIRING (Continued)

The flip-flops are shown with lamp on top. For flip-flop  $A_2$ , the top half is the " $A_2$ " half and the bottom half is the " $\bar{A}_2$ " half. The "gate" symbols may be shown oriented in any direction, but the configurations for the flip-flop and pulse generator are only shown with the output pins facing left and the input pins facing right. Gates will be shown as often as possible with output pins facing left and input pins facing right, but sometimes it is necessary to reverse orientations or face the symbols at  $90^\circ$  angles in order to simplify wiring connections. If using commercial LIBE FF-1, AM-1, AO-1, AND-1, or OR-1, then the outputs will face always left and inputs right when the power pins are mounted on top. The "+6" power pins are marked with a red dot.

#### 8.3.1 POWER PIN CONNECTIONS

Again we remind the reader that all power pin connections must be made on each and every unit. If the power pins were shown in the logic diagrams, they would always be on top. Now let us include the power pins in a modified logic diagram and show how the connections are made:



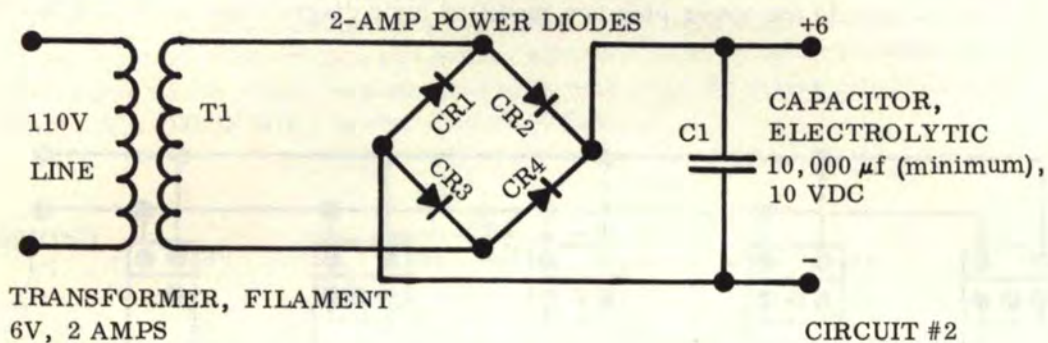
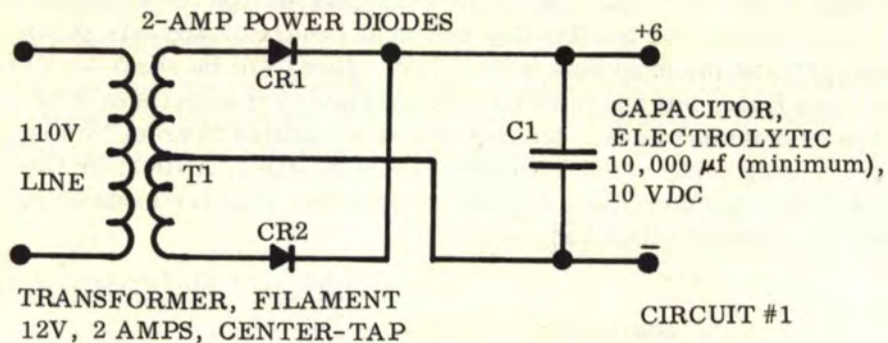
It should be apparent that including power wiring in logic diagram will only further confuse the complicated diagram.

### 8.4 POWER SOURCES

A large 6-volt battery is recommended (preferably a double-size lantern type) for projects which require 10 flip-flops or less. These batteries are easily obtained in hardware, variety, or general merchandise stores. The "off" state flip-flop current, pulse generator current, and gate current are negligible (about .006 amp, with .036 watt power consumption except for the gates which draw .012 amp, with .072 watt power consumption for each gate). However, the "on" state flip-flop current is very high (about .125 amp, .750 watt power consumption for each unit) because of the lamp display.

Flip-flops in the "on" state will cause a noticeable drain on the battery. When the battery is being overloaded, all the "on" flip-flops will dim considerably when another flip-flop turns "on".

A 6-volt, 2 to 3 amp D.C. power supply is ideal. More than one power supply can be used for larger projects. The following "brute force" power supply circuits are adequate for use with the computer circuits and can easily be built up from surplus parts.

8.4 POWER SOURCES (Continued)

Both of the above circuits represent unregulated power supplies and therefore actually put out more than 6 volts. The output is about 8 volts, but they are simple, easy to build, and will do the job.

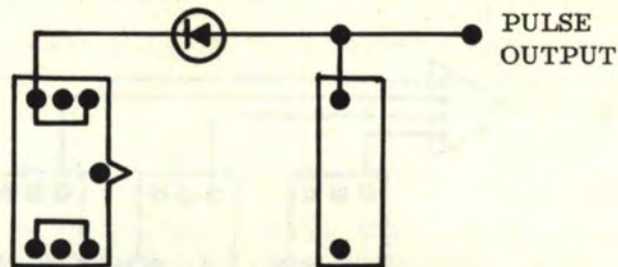
8.5 SENSES AND COMMANDS

The uses of "SENSES" and "COMMANDS" will be employed widely in the next two chapters on wiring projects. Let us first define the difference between the two. The "COMMAND" causes a computer operation to take place such as a flip-flop change of state, an addition, subtraction, or a "halt". A "SENSE" controls the execution of a command and is used only in the advanced projects. For instance, we can sense whether a register contains a number or does not (digit sense) or whether a flip-flop is a "1" or a "0".

The most important command comes from the pulse generator and causes the other circuits in turn to generate their own commands.

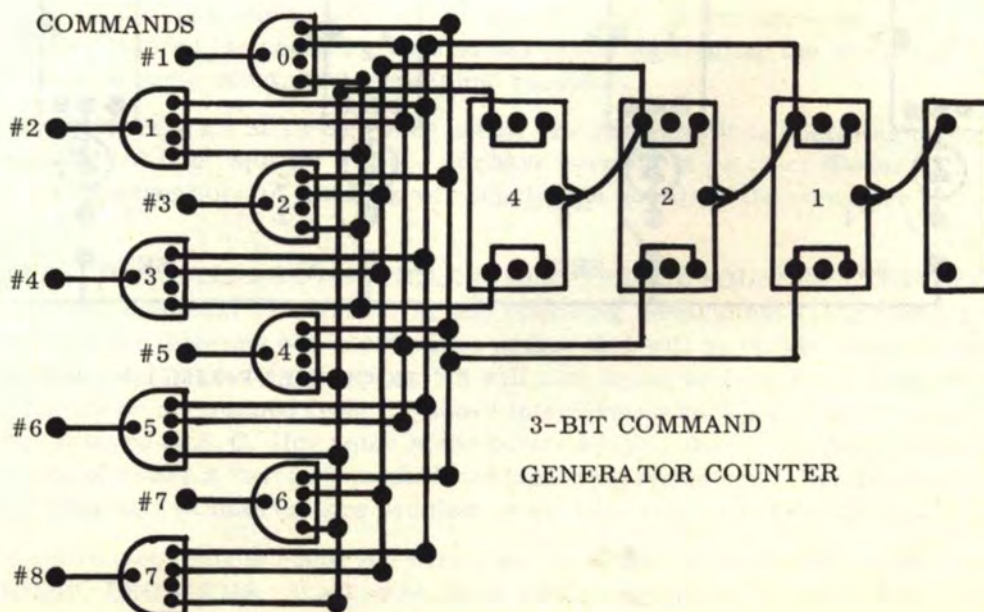
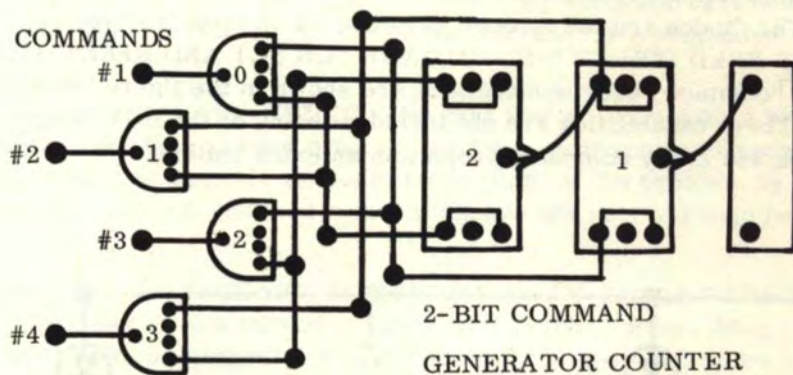
The next most important command is the "halt" command which is needed to stop an operation. The logic diagram for the "halt" command is as follows:

## 8.5 SENSES AND COMMANDS (Continued)



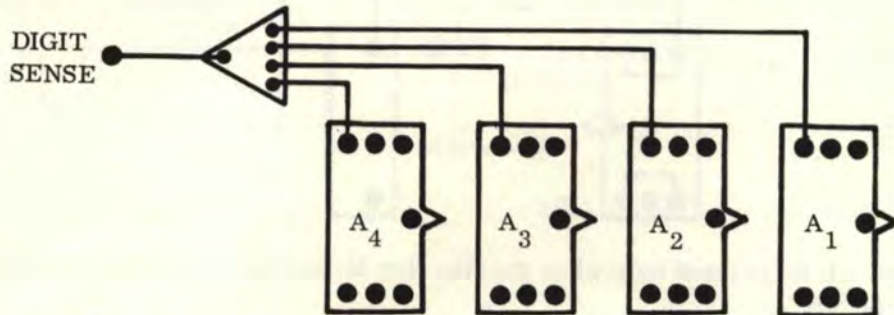
The pulse will be present only when the flip-flop is "on" and will be suppressed when the flip-flop is "off".

Two command generator counters are shown below for reference only. They are composed of "AND" gated binary counters and will generate as many different commands as the counter can count to. A 2-bit counter will generate 4 commands and a 3-bit counter will generate 8 commands. These commands can be controlled by senses if necessary.



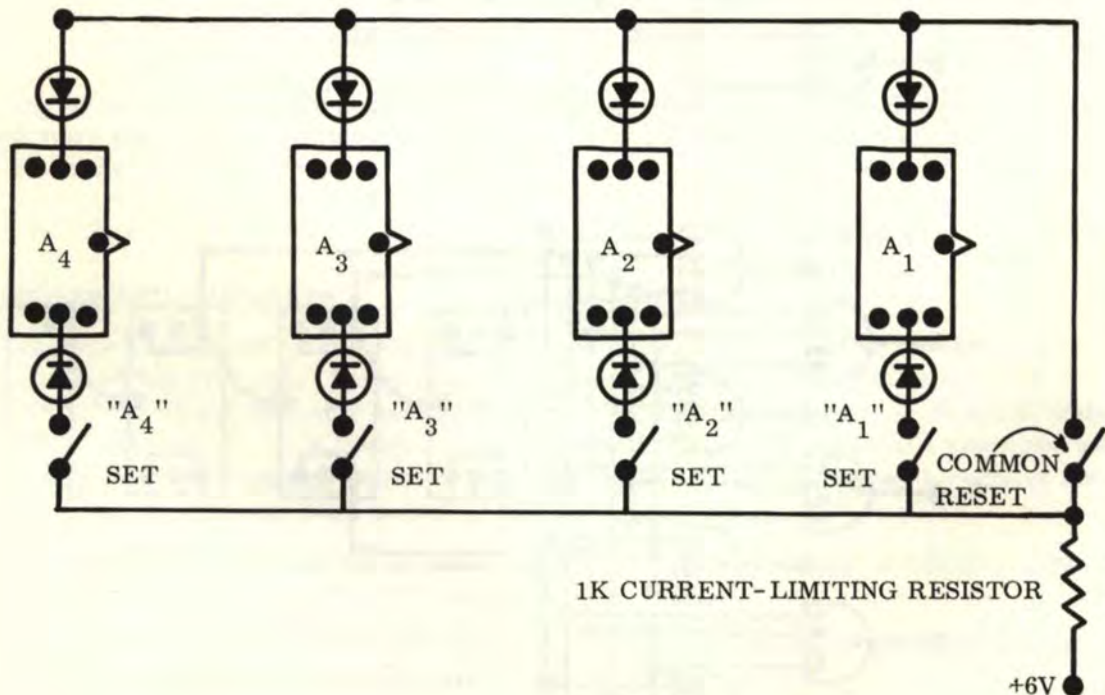
### 8.5 SENSES AND COMMANDS (Continued)

The "sense" most often used is the output of a flip-flop. This flip-flop output can then be used to control a command. The next most often used "sense" is the digit sense (to sense a number in a register). This consists of only one "OR" gate whose inputs are fed by the "true" outputs of all flip-flops in a register. The logic diagram for a 4-bit digit sense is shown below:



### 8.6 SET AND RESET

It is possible to add a "SET" and "RESET" capability to all projects in the next two chapters as an extra feature by using diode wires (or diodes, if soldered in). Diodes are connected to the bottom flip-flop center pin (pin E) for "SET" and to the top center pin (pin B) for reset. The diodes are fed through switches which feed "+6" voltage through a 1K resistor (DO NOT FEED DIRECT "+6" VOLTAGE FOR SET AND RESET!!!!). The individual "set" and "common reset" connections are shown in the figure below for one flip-flop register. These capabilities are not included in any of the logic diagrams, but may be added to each and every computer project as an extra feature.





## 8.6 SET AND RESET (Continued)

All switches are "normally-open" miniature pushbutton switches. The "set" capability is not recommended for a "DOWN" counter register because of the fact that a "set" of one flip-flop will always trigger the next flip-flop over (to the left). However, there are no problems with the "set" for "UP" counters and shift registers. The "common reset" will work on all types of registers (i. e. , "UP" and "DOWN" counters and shift registers).

## 8.7 INTRINSIC PROBLEMS AND DEBUGGING

You have carefully wired up a logic diagram project, but for some reason the project will not work. There may be extraneous pulses or improper triggering. A bit will not shift on a shift register or a counter may not count properly. A gated logic circuit might work improperly or might not work at all. When these symptoms appear, the reader should refer to this section for help. This means that we must go through a "DEBUGGING" stage. Usually there will be no problems with the simple counters, adders, and shift registers. However, on more complex projects, these problems may show up.

"DEBUGGING" comes from the slang word "bug" which means "problem" (you have probably heard the phrase "get the 'bugs' out" several times). Well, in electronics we have many, many "bugs" that may appear and we may spend several hours "troubleshooting" to get them out. When problems do occur, the following is a suggested order of steps to be followed:

1. **DOUBLE CHECK ALL WIRING.** Be sure that all connections have been made properly, that connections are not missing, and that extraneous connections have not been made. Sometimes it is possible to locate the vicinity of the problem by watching the unit operate. Be sure also that isolated connections are not shorting together--especially when using alligator clips.

2. **REMOVE AND CHECK OUT INDIVIDUAL UNITS IN THE VICINITY OF THE PROBLEM.** You may have a burned out gate or flip-flop. If you think you have a marginal unit which is not working properly, replace it with a similar unit to be sure. Be sure that there are no cold solder joints and that the connecting pins (or terminals) are making proper contact with the back of the printed circuit board.

3. **CHECK THE ALLIGATOR CLIP WIRES (when used).** There is the possibility of a cold solder joint where each clip is soldered to the wire end. If wiring connections have been permanently soldered in, check for cold solder joints by wiggling the wire(s) in question. If there is some doubt, then reheat and resolder.

4. **CHECK THE POWER SUPPLY (when used).** Be sure that it is operating properly and is not throwing out line "spikes" or does not have burned out rectifier diodes or a burned out filtering capacitor. A faulty power supply will not allow the computer units to work properly.

5. **BE SURE THE UNITS ARE NOT NEAR INTERFERENCE SOURCES.** Electrical appliances, electric blankets, refrigerators, and operating electric hand drills or electric saws are notorious for throwing out interference spikes that will be picked up by the flip-flops. An electric arc or heavy-current switch will also cause problems. A distance of about 20 feet should be maintained from the above interference sources if there appears to be a problem. If there is A. C. line noise in the power supply, the power supply should be "isolated" by use of either a "variac" variable voltage transformer or a line isolation transformer. This kind of interference problem is usually rare, but it does exist.

If all the above steps fail to debug the wiring project, then we have entered the nebulous area of **INTRINSIC PROBLEMS**. If all units check out properly individually and all other possibilities of wiring errors, cold solder joints, and interference have been eliminated,

## 8.7 INTRINSIC PROBLEMS AND DEBUGGING (Continued)

but the wiring project will still not function properly, we have an INTRINSIC PROBLEM. In other words, the units should function, but do not. We can only offer some suggestions for debugging such problems. Sometimes nothing will help and a wiring project may have to be redesigned or rebuilt with new units. If an intrinsic problem appears, proceed with the following steps:

1. WIRE IN MULTIPLE POWER CONNECTIONS. This should be done especially in the area where the problem seems to be. At times, when power has to travel through too many wires, a small resistance is created. Since the flip-flops change state in about 1 microsecond, this may cause a very small instantaneous power drain along the power wires. This power drain or "pull-down" gets worse as the distance increases from the main power connections and sometimes the flip-flops are sensitive enough to detect this. If the probes of an oscilloscope were placed across the power leads a considerable distance from the main power connections, these "pull-down" periods can be observed as definite "spikes" during operation. Other times, inductive and/or coupling effects may be the cause. Try wiring every third or fourth flip-flop with direct redundant power connections. Sometimes only one additional set of power connections may help. An alternative is to try making the single power connections at different points. Also, a third alternative is to wire in DECOUPLING CAPACITORS of at least 1 microfarad directly across the power pins in the area where the problem is occurring.

2. INTERCHANGE SIMILAR UNITS. Sometimes two units will not work properly with each other, but will perform normally elsewhere in the wiring project. Remove and replace any suspected marginal units. Sometimes this marginal interaction or accumulative tolerance is the cause of the problem.

3. ELIMINATE LOGIC GATES WHENEVER POSSIBLE. When a flip-flop signal passes through more than three logic gates, sometimes there is a slow-down of a flip-flop pulse and it will not "fall" fast enough to trigger a flip-flop. Other times some peculiar combinations of "AND" and "OR" gates will produce extraneous pulses. Sometimes merely changing gates around (or changing "AND" and "OR" logic around) will remedy the problem.

4. AVOID DRIVING TOO MANY OTHER UNITS WITH A SINGLE GATE OR FLIP-FLOP. The circuits in this book should be capable of driving at least 10 other units. If it is absolutely necessary to drive more than 10 units with a single gate or flip-flop, the driver unit should be modified by changing the voltage dropping resistors (usually R1 and R2) from 1K to 500  $\Omega$ .

5. SHIFT REGISTER UNITS MAY NEED TO BE MODIFIED. If bits fail to transfer properly in a shift register even after switching flip-flop units around, the flip-flop units may be modified as follows: Change R2 from 560 ohms to 200 ohms and/or decrease R5 and R6 from 10K to 4.7K. This modification will clear up most shift register intrinsic problems--especially when the shift register flip-flops are driving several logic gates.

Finding remedies for intrinsic problems is a very important phase in the manufacture of all large commercial electronic computers. This is also part of the "fun" of building your own computer and should give you some insight to problems encountered in the commercial computer industry. Again, when problems occur, refer back to this section!

## 8.8 CARE AND REPAIR OF UNITS

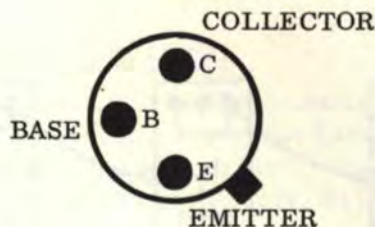
When soldering permanent connections to pins, be sure that any "swedged" or "friction-contact" pins or terminals are also soldered to the trace of the printed circuit board (when applicable). This will prevent intermittent connections due to soldering heat. Also, when making repairs or modifications on units on printed circuit boards, avoid

## 8.8 CARE AND REPAIR OF UNITS (Continued)

overheating copper traces. Avoid pulling or twisting parts so hard that traces will be either pulled loose or broken.

An ohmmeter, when used on the "X10" scale, makes a very effective trouble-shooting device. About 99.5% of all failures of units can be attributed to transistors, diodes, or lamps. In many cases, the blackened color or broken filament of a burned out lamp is obvious. However, when in doubt, an ohmmeter will register "short" for a good lamp and "open" for a bad one. The diode will conduct in only one direction. If a diode failure is suspected, place the ohmmeter leads across the diode in one direction and then reverse the leads and place across the diode in the other direction. A good diode will show a resistance of about 1000 ohms (silicon junction) or 200-500 ohms (germanium junction) in one direction, and as "open" in the other direction. A "leaky" diode will show some resistance in both directions. A "shorted" diode will show up as "short" in both directions, and an "open" diode will show up as "open" in both directions. Replace all "leaky", "shorted", and "open" diodes.

Transistors may also be checked with an ohmmeter. The transistor consists of a "collector", "base", and "emitter". The most common configuration for transistors, with leads facing out from the paper, is as shown below:



While the results of transistor checks on an ohmmeter are more uncertain than those of a lamp or diode, they still give a reasonable indication of a good or bad part. We start the check by holding an ohmmeter lead on the BASE and checking for resistance between B & C and B & E. Reverse the ohmmeter leads and check again for resistance between B & C and B & E. Then check for resistance between C & E and reverse ohmmeter leads and check again for resistance between C & E. A good transistor will show a junction resistance (about 1000 ohms for silicon or about 200-500 ohms for germanium) between B & C and B & E in one direction (with same lead on B), and as "open" between B & C and B & E in the other direction. The resistance between C & E in both directions should be "open"; however, in some germanium transistors a resistance of about 10K in one direction is acceptable. A "shorted" transistor will show up as "short" in at least one measurement. An "open" transistor will show up as "open" in all measurements. A "leaky" transistor will show some resistance in both directions of the B & C and B & E measurements and/or resistance in both directions of the C & E measurements. If the C & E resistance is low in one direction (less than 10,000 ohms), then the part should be changed. Replace all "open", "shorted", and leaky transistors,

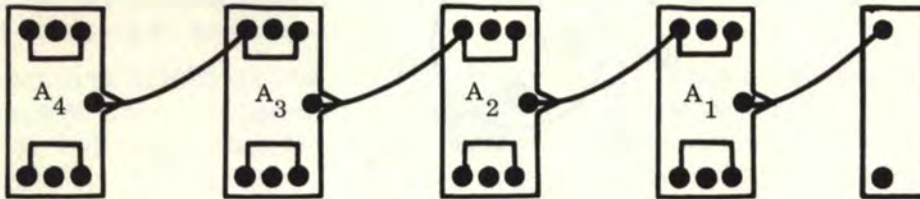
Do not operate repaired units immediately after soldering. Let the soldered junctions cool down for at least two (2) minutes.

## 9. BASIC NON-GATED COMPUTER PROJECTS

The projects in this section show all basic flip-flop operations without the use of any logic gating. These projects are fairly simple and should be wired up first before attempting the more difficult ones in the next section. Explanations are shown using four flip-flops for each register. However, there is no limit to the number of flip-flops that can be used. Note the wiring of the middle flip-flops. Extra flip-flops can be wired into the middle in the same manner. As an example, it is possible to have a 10-bit "up" counter by wiring in 6 extra flip-flops. Again, be sure that proper connections are first made to all voltage pins to provide power to each unit. Suggested parts lists are included only for projects in this section. The flip-flops are labeled with designations such as "A<sub>4</sub>", "A<sub>3</sub>", "A<sub>2</sub>", "A<sub>1</sub>", etc., for reference with projects in the next section.

### 9.1 THE BINARY "UP" COUNTER

The binary "up" counter will count upward in order from 1 through 15 (for the 4-bit example shown below), and then reset (clear) itself and start over again. The logic diagram for a 4-bit binary "up" counter is as follows:



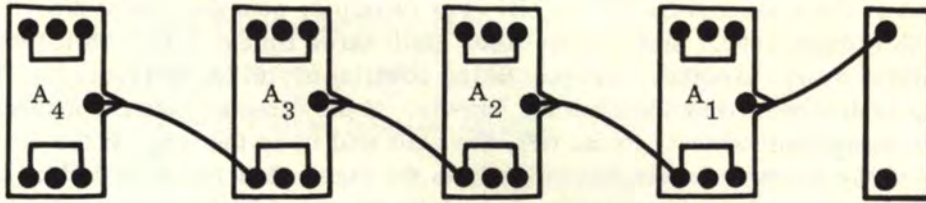
Suggested parts list:

- 4 Flip-Flops (FF-1)
- 1 Pulse Generator (AM-1)
- 22 8" wires (clip-on wires)

### 9.2 THE BINARY "DOWN" COUNTER

The binary "down" counter will count downward (backward) in order starting from 15 (for the 4-bit example shown below) down to 0 and then start over. The logic diagram for a 4-bit "down" counter is as follows:

### 9.2 THE BINARY "DOWN" COUNTER (Continued)

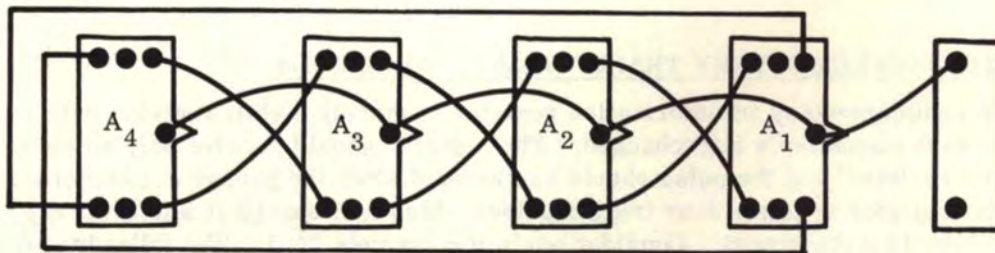


Suggested parts list:

- 4 Flip-Flops (FF-1)
- 1 Pulse Generator (AM-1)
- 22 8" wires (clip-on wires)

### 9.3 THE BINARY SHIFT REGISTER (LEFT)

The binary shift register (left) will shift any binary number entered to the left and do an "end-around" shift from the last flip-flop to the first. For example, consider the 4-bit number 0001. Shift once: 0010. Shift twice: 0100. Shift three times: 1000. Shift four times: 0001 (starts over). Another example: 0011, 0110, 1100, 1001, 0011... . The pulse generator controls the shift rate. A fast pulse will cause a fast shift, while a slow pulse will cause a slow shift rate. The logic diagram for a 4-bit binary shift register (left) is as follows:

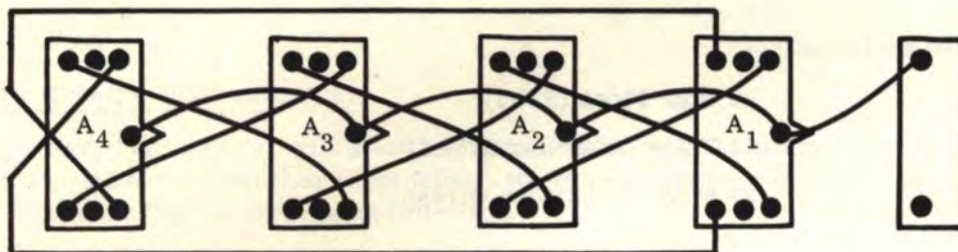


Suggested parts list:

- 4 Flip-Flops (FF-1)
- 1 Pulse Generator (AM-1)
- 20 8" wires (clip-on wires)
- 2 16" wires (clip-on wires)

#### 9.4 THE BINARY SHIFT REGISTER (RIGHT)

The binary shift register (right) will do essentially the same as the "left" shift, but will shift to the right instead of the left. For example, consider again the 4-bit number 0001. Shift once: 1000. Shift twice: 0100. Shift three times: 0010. Shift four times: 0001 (starts over). Another example: 0011, 1001, 1100, 0110, 0011.... . The direction of shifts is determined by the director outputs. If the director outputs connect to the follower inputs immediately to the left, the shift will be to the left. If the director outputs connect to the follower inputs immediately to the right, then the shift will be to the right (as in this case). The logic diagram for a 4-bit binary shift register (right) is as follows:

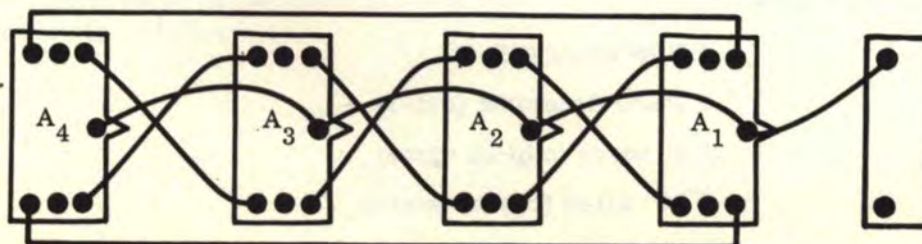


Suggested parts list:

- 4 Flip-Flops (FF-1)
- 1 Pulse Generator (AM-1)
- 20 8" wires (clip-on wires)
- 2 16" wires (clip-on wires)

#### 9.5 THE COMPLEMENTARY TRANSFORMATION REGISTER

The complementary transformation register is merely a shift register with the "end-around" shift connections interchanged. The register should receive only as many pulses as there are "bits" and the pulse should be removed after the proper number of triggers. The 4-bit register requires four trigger pulses which will change (transform) any binary number into its complement. Consider again the example 0001. The following series of shifts will take place with each number inverting at position A1 in a modified left-shift register: 0001, 0011, 0111, 1111, 1110. We start at 0001 and four pulses later we have 1110. The logic diagram for a 4-bit complementary transformation register is as follows:



## 9.5 THE COMPLEMENTARY TRANSFORMATION REGISTER (Continued)

Suggested parts list:

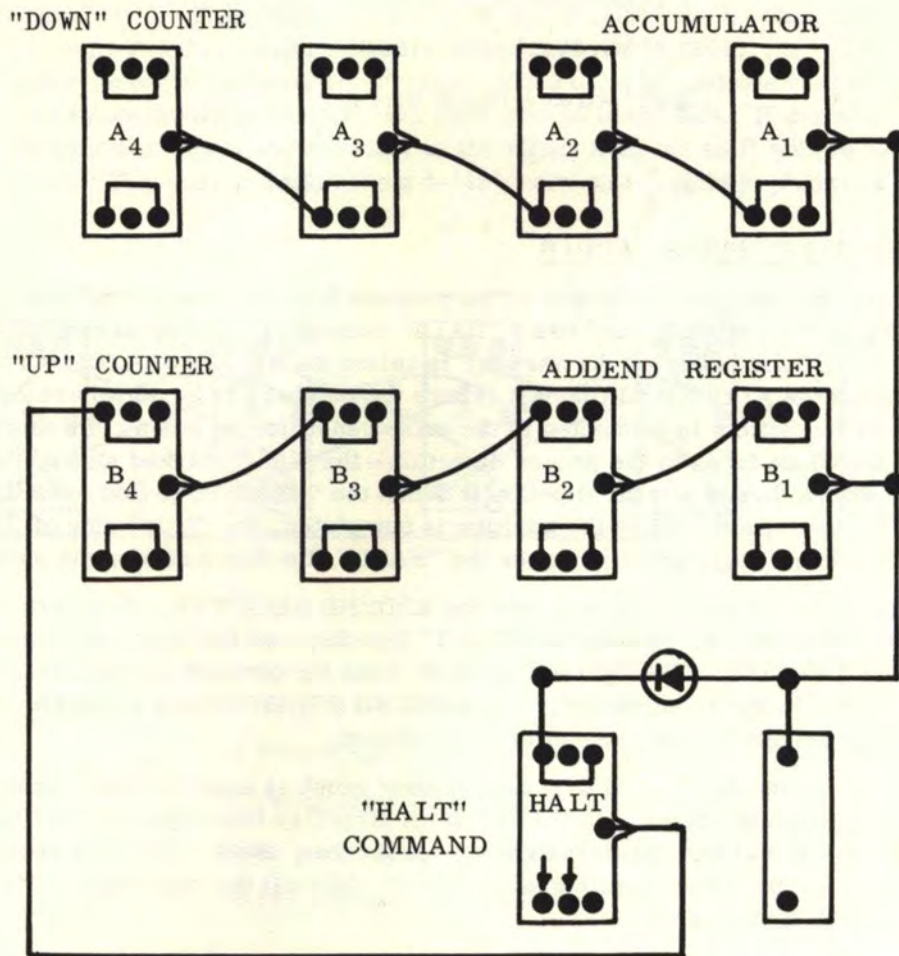
- 4 Flip-Flops (FF-1)
- 1 Pulse Generator (AM-1)
- 20 8" wires (clip-on wires)
- 2 16" wires (clip-on wires)

## 9.6 THE NON-GATED BINARY ADDER

In designing this simple binary adder, we combine both the binary "up" and "down" counters (to form two registers) and use a "HALT" command flip-flop to control the pulse. The top register, which indicates the answer, is called the ACCUMULATOR. The bottom register is called the ADDEND REGISTER (where the numbers to be added are entered). The flip-flop on the bottom is connected to the pulse generator by means of a diode wire (be sure that the diode faces in the proper direction—the "line" marked side of the diode is negative and goes directly to the flip-flop). When the "HALT" flip-flop is in the "0" state, the pulse is stopped. When the addition is completed, the "true" side of the last flip-flop in the addend register will trigger the "HALT" flip-flop and stop the pulse.

The number to be added is entered into the ADDEND REGISTER. Then touch the two pins together (shown with arrows) on the "HALT" flip-flop and the light will come on and start the addition process. The light will turn off when the process is completed and the answer will appear in the accumulator. The ADDEND REGISTER has automatically cleared and the next number can be entered for addition.

To clear the accumulator, add to it that number which is equal to its complement + 1. That is, starting from the right, enter a "1" in the first flip-flop opposite the flip-flop in the ACCUMULATOR that has the first light on. After that, enter "1's" only opposite zeros in the ACCUMULATOR. The resulting addition will clear all the registers. The logic diagram for a 4-bit adder is as follows:

9.6 THE NON-GATED BINARY ADDER (Continued)

Suggested parts list:

- 9 Flip-Flops (FF-1)
- 1 Pulse Generator (AM-1)
- 45 8" wires (flip-on wires)
- 1 16" wire (clip-on wire)
- 1 diode wire (clip-on wire)

9.7 THE NON-GATED BINARY SUBTRACTOR

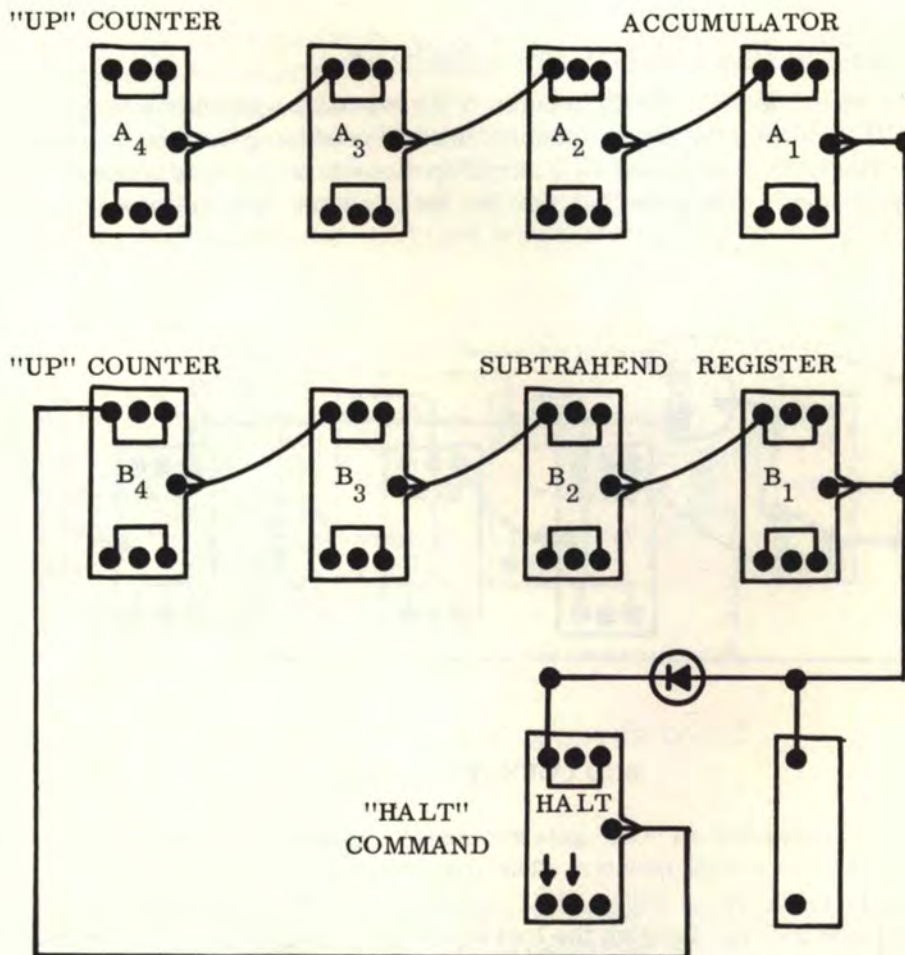
The principle of the subtractor is almost the same as that of the adder, except that both upper and lower registers are "up" counters. The bottom register is now called the SUBTRAHEND REGISTER.



### 9.7 THE NON-GATED BINARY SUBTRACTOR (Continued)

To subtract, enter a number in the ACCUMULATOR. Then enter the number to be subtracted in the SUBTRAHEND REGISTER. Touch the two pins together (shown with arrows) in the "HALT" flip-flop and the light will come on and start the subtraction process. The light will turn off when the subtraction is completed and the answer will appear in the ACCUMULATOR. The SUBTRAHEND REGISTER has automatically cleared and another number can be entered for subtraction.

To clear the ACCUMULATOR, subtract the number that is in it. That is, enter that same number in the SUBTRAHEND REGISTER. The resulting subtraction will clear all the registers. The logic diagram for a 4-bit subtracter is as follows:



#### Suggested parts list:

- 9 Flip-Flops (FF-1)
- 1 Pulse Generator (AM-1)
- 45 8" wires (clip-on wires)
- 1 16" wire (clip-on wire)
- 1 diode wire (clip-on wire)

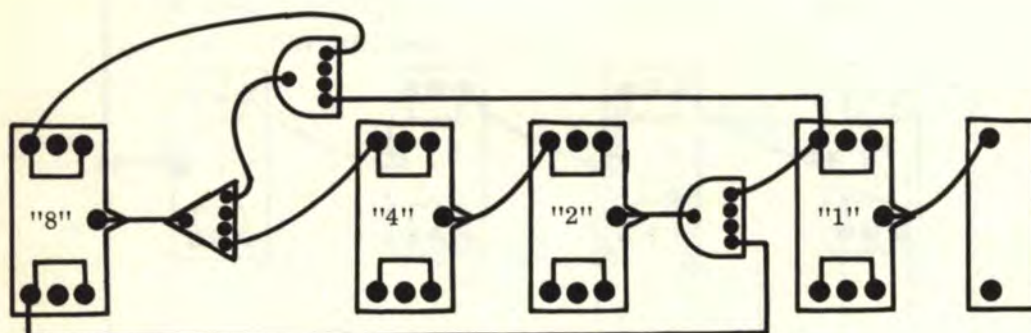
## 10. ADVANCED COMPUTER PROJECTS

The projects shown in this chapter are not meant to be easy, but to show how various operations are performed within a computer. The arithmetic processes of addition, subtraction, multiplication, division, and square rooting are shown here. There are also various other projects such as the BCD counter, time machine and electronic dice.

All projects in this chapter contain logic gating in one form or another, using "AND" and "OR" gates, whereas the previous chapter did not. It is up to the experimenter to determine his own parts lists and mounting displays.

### 10.1 THE BINARY CODED DECIMAL (BCD) COUNTER

The binary coded decimal (BCD) counter is a very basic application to get a feel for logic gating. This counter is quite commonly used in producing decimal number conversions from binary. The BCD counter uses 4 binary flip-flops to count to 10 (instead of 16 for a normal binary counter). The gate logic for the BCD counter is as follows:



BCD COUNTER LOGIC

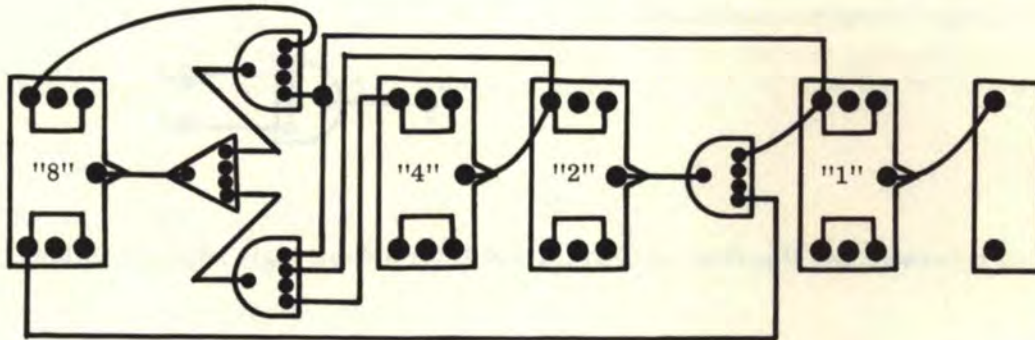
Only 2 "AND" gates and an "OR" gate are needed (in the above version) to convert a basic 4-bit counter into a BCD counter. The counting sequence will be: 0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111, 1000, 1001, and reset to 0000. Other binary coded logic counters can be obtained by changing the location of the logic gates. Further experimentation is left to the reader.

Let us analyze the logic involved in the BCD counter. First of all, we must start the count with all flip-flops reset to 0000. Note that the "FALSE" output of the "8" flip-flop is now "ON" since the flip-flop is a "0". When the "1" flip-flop comes on with a pulse from the AM-1 pulse generator, we have the first "AND" gate with two inputs that are "ON". The "AND" gate is now "ON". The next pulse turns the "1" flip-flop off. This, in turn, will turn off the "AND" gate and transmit a "DOWN-SWING TRIGGER" signal to the "2" flip-flop. We now have 0010. The next pulse turns the "1" flip-flop on again for 0011. Again the "AND" gate is "ON". The next pulse will turn the "1" flip-flop off again and the "AND" gate will once more be "OFF". This transmits another "DOWN-SWING TRIGGER" to the "2" flip-flop which will turn off and trigger the "4" flip-flop which will come on. We now have 0100. The "4" flip-flop is a direct input to the "OR" gate which is now "ON" with one input "ON".

### 10.1 THE BINARY CODED DECIMAL (BCD) COUNTER (Continued)

Again we repeat the above sequences for 0101, 0110, and 0111. The next pulse will reset the "1", "2", and "4" flip-flops to "0". When the "4" flip-flop is reset to "0", the "OR" gate turns off and produces a trigger to the "8" flip-flop which turns on for 1000. Now we have a different situation. The "8" flip-flop is now "ON" and the "FALSE" output to the first "AND" gate is now "OFF", which means that the first "AND" gate can not turn on while the "8" flip-flop is a "1". The next pulse will produce 1001. Now observe that the second "AND" gate has both inputs "ON" and is now turned on. This in turn produces an "ON" input to the "OR" gate which also turns on. The next pulse will reset the "1" flip-flop back to "0" which turns the second "AND" gate off and, in turn, turns off the "OR" gate to produce a trigger pulse which resets the "8" flip-flop back to "0" and now the whole BCD counter is reset to 0000.

There is one slight drawback to the simplified BCD counter logic discussed above. If an erroneous display such as 1010, 1011, 1100, 1101, 1110, or 1111 should occur, the "OR" gate will "hang up" and never turn off by pulsing the "1" flip-flop. As a result, only the "1" flip-flop will turn on and off without triggering any of the other flip-flops. To prevent this from happening, we must add a third "AND" gate as shown below. The resulting modified BCD counter will not "hang up" under any display combination, but merely continue counting, reset itself, and start over with a correct BCD counting sequence.



NON-"HANG-UP" BCD COUNTER LOGIC

### 10.2 THE LOGIC ADDER

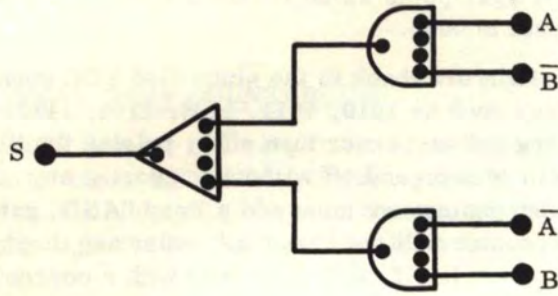
In logical addition, we have four numbers to consider. The first two are A and B, the two numbers to be added together. The third is called the "SUM" of A and B and is represented by "S". The fourth is called the "CARRY" generated by A and B when they are added together. Note that the only case for a single number sum to generate a "carry" is:  $1 + 1 = 0$ , carry 1. The other three combinations ( $1 + 0$ ,  $0 + 1$ , and  $0 + 0$ ) do not generate a "carry".

Using "AND" and "OR" gates, the logical "sum" S is represented as follows:

10.2 THE LOGIC ADDER (Continued)

$$S = A+B = (A \cdot \bar{B}) \vee (\bar{A} \cdot B)$$

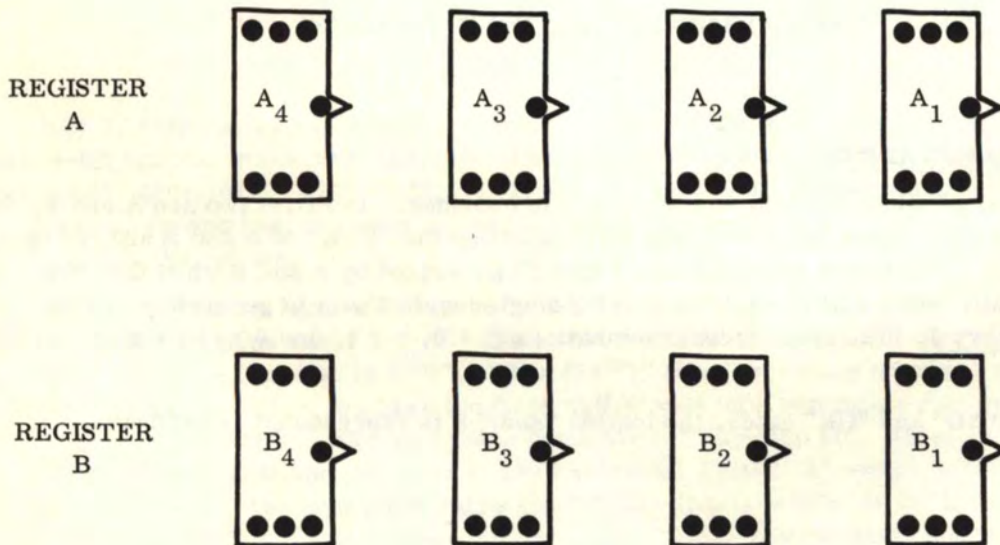
A  
+B  
—  
S



The "carry" is represented as follows:  $C = A \cdot B$

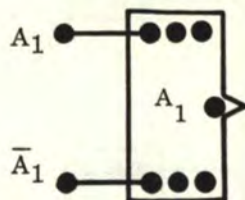
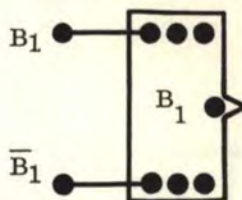


Let us represent two flip-flop registers A and B for a 4-bit logic adder as follows:



## 10.2 THE LOGIC ADDER (Continued)

The subscripts 1, 2, 3, and 4 indicate the flip-flop position in each register. Now let us consider  $A_1$  and  $B_1$ . Outputs " $A_1$ " and " $B_1$ " are generated by the "TRUE" side of flip-flops " $A_1$ ", and " $B_1$ ", respectively. Outputs " $\bar{A}_1$ " and " $\bar{B}_1$ " are generated by the "FALSE" sides of these respective flip-flops.

FLIP-FLOP " $A_1$ "FLIP-FLOP " $B_1$ "

In logical addition, flip-flops  $A_1$  and  $B_1$  in both registers A and B may be specially treated since they represent the least significant digit and no "carry" can be generated by any flip-flops to the right. In other words, the "carry" which we will call " $C_0$ ", is always "0" and the invert " $\bar{C}_0$ " is always "1". Therefore, using the general logic equation for the sum of three numbers A, B, and C, we may simplify both the "S" and the "C" for the first digit.

$$S_1 = A_1 + B_1 + C_0$$

$$S_1 = (A_1 \cdot B_1 \cdot C_0) \vee (A_1 \cdot \bar{B}_1 \cdot \bar{C}_0) \vee (\bar{A}_1 \cdot B_1 \cdot \bar{C}_0) \vee (\bar{A}_1 \cdot \bar{B}_1 \cdot C_0)$$

$$\text{Set } C_0 = 0 \text{ and } \bar{C}_0 = 1$$

$$S_1 = (A_1 \cdot B_1 \cdot 0) \vee (A_1 \cdot \bar{B}_1 \cdot 1) \vee (\bar{A}_1 \cdot B_1 \cdot 1) \vee (\bar{A}_1 \cdot \bar{B}_1 \cdot 0)$$

$$S_1 = (0) \vee (A_1 \cdot \bar{B}_1 \cdot 1) \vee (\bar{A}_1 \cdot B_1 \cdot 1) \vee (0)$$

$$S_1 = (A_1 \cdot \bar{B}_1 \cdot 1) \vee (\bar{A}_1 \cdot B_1 \cdot 1)$$

$$S_1 = (A_1 \cdot \bar{B}_1) \vee (\bar{A}_1 \cdot B_1)$$

$$C_1 = (A_1 \cdot B_1) \vee (A_1 \cdot C_0) \vee (B_1 \cdot C_0)$$

$$\text{Set } C_0 = 0$$

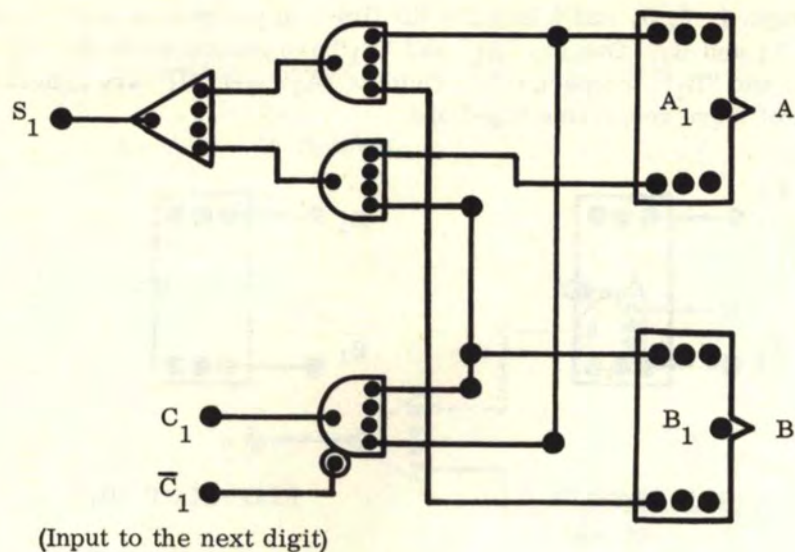
$$C_1 = (A_1 \cdot B_1) \vee (A_1 \cdot 0) \vee (B_1 \cdot 0)$$

$$C_1 = (A_1 \cdot B_1) \vee (0) \vee (0)$$

$$C_1 = A_1 \cdot B_1$$

From the calculations above, we can represent addition on the LEAST SIGNIFICANT DIGIT by the following logic:

## 10.2 THE LOGIC ADDER (Continued)



For intermediate positions such as  $A_2$  and  $B_2$ ,  $A_3$  and  $B_3$ , we have (using as an example  $A_2$  and  $B_2$ ) the following logic:

$$S_2 = A_2 + B_2 + C_1$$

$$S_2 = (A_2 \cdot B_2 \cdot C_1) \vee (A_2 \cdot \bar{B}_2 \cdot \bar{C}_1) \vee (\bar{A}_2 \cdot B_2 \cdot \bar{C}_1) \vee (\bar{A}_2 \cdot \bar{B}_2 \cdot C_1)$$

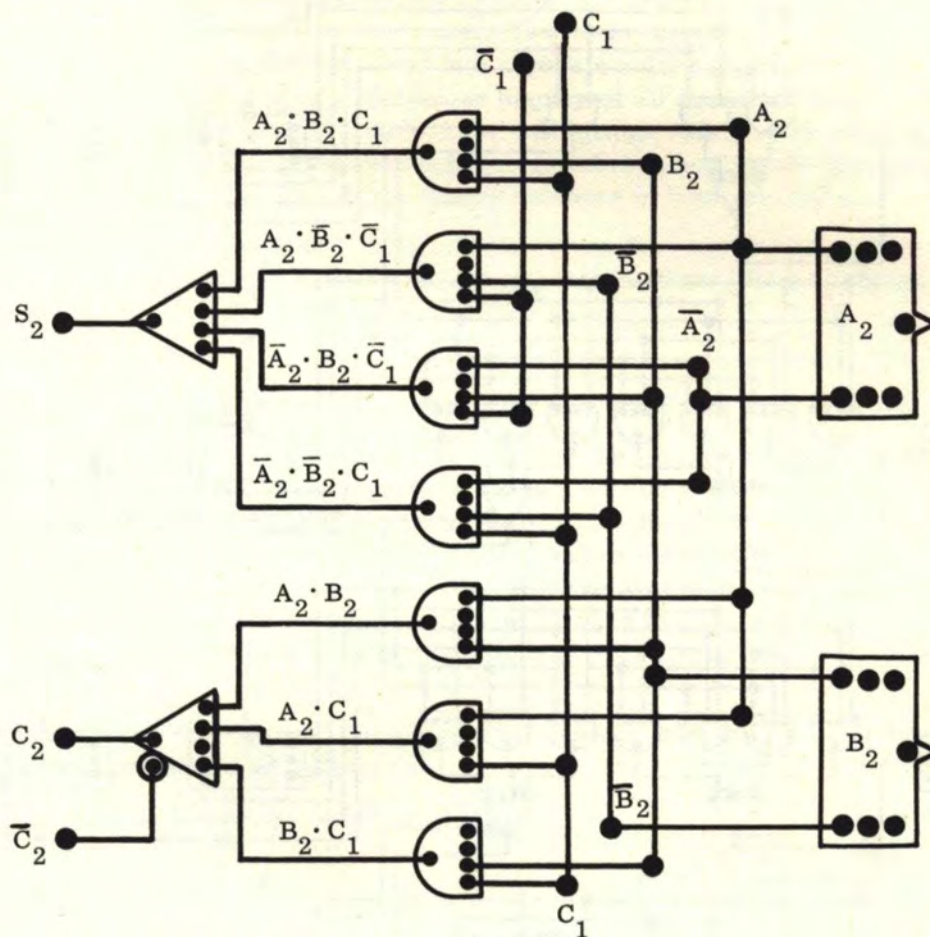
$$C_2 = (A_2 \cdot B_2) \vee (A_2 \cdot C_1) \vee (B_2 \cdot C_1)$$

The "TRUTH TABLE" for the intermediate positions  $A_2$  and  $B_2$  is as follows:

$A_2$	$B_2$	$C_1$	$S_2$	$C_2$
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

From the preceding equations, we can represent the intermediate digits by the following computer logic:

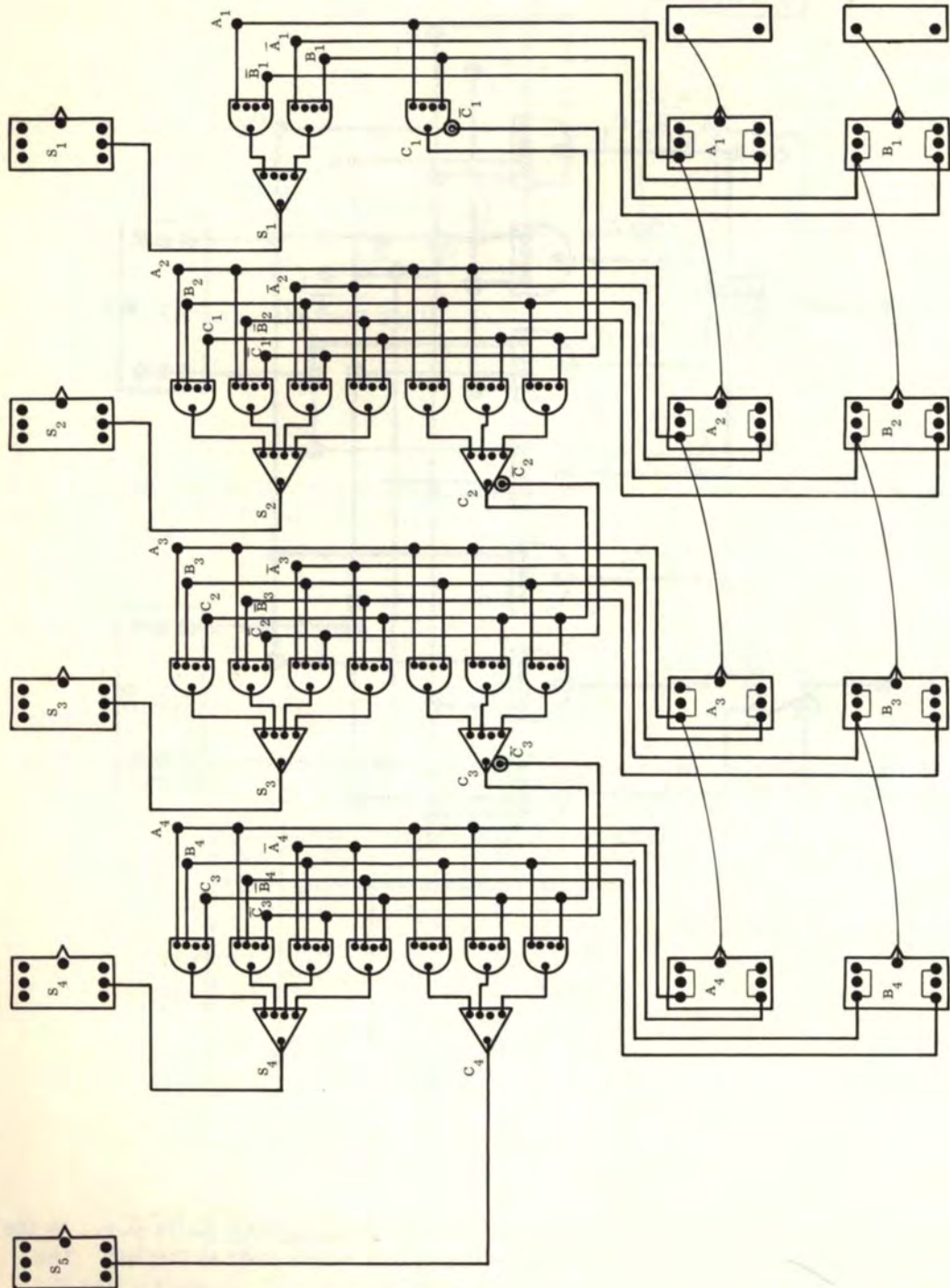
## 10.2 THE LOGIC ADDER (Continued)



Note that the "carries"  $C_1$  and  $\bar{C}_1$  are from the previous register digits (i. e. , to the right) and that " $C_2$ " and " $\bar{C}_2$ " are the inputs for the next position over to the left. The logic circuitry which generates only the "sum" and not the "carry" is called a HALF ADDER. Thus, the three "AND" gates and "OR" gate which generate  $S_2$  above are a HALF ADDER. The complete logic above which generates both the "sum" and "carry" is called a FULL ADDER.

10.2 THE LOGIC ADDER (Continued)

The complete 4-bit logic adder is shown in the figure below:



4-BIT LOGIC ADDER



## 10.2 THE LOGIC ADDER (Continued)

Note that the readouts consist only of flip-flops wired up in the "direct readout" configuration. The first four flip-flops  $S_1$ ,  $S_2$ ,  $S_3$ , and  $S_4$  read out the direct output of the final "OR" gates of  $S_1$ ,  $S_2$ ,  $S_3$ , and  $S_4$ . The last sum  $S_5$  is read out directly from carry  $C_4$ .

When the full logic adder is put into operation, the answer will appear in register S for any addition problem as soon as the binary digits are entered into registers A and B. There are no intermediate steps and the answer is always instantaneous. The flip-flops in registers A and B may be further wired into separate pulsed (or common-pulsed) "up" or "down" binary counters to give a continuous display of all the addition possibilities. Each time a register changes value, the answer will change value. The registers A and B may also be wired into separate shift registers to obtain a continuous display of the sum depending upon the shifted position of the binary numbers in both registers.

To make an adder larger than 4 bits, merely repeat the connections, logic, and circuitry as shown in positions 2 and 3 to add as many intermediate bits as necessary.

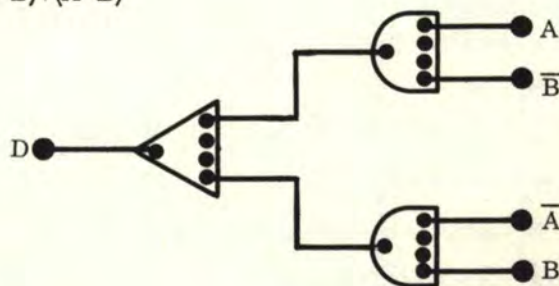
## 10.3 THE LOGIC SUBTRACTER

In logical subtraction, we also have four numbers to consider. The first is A, the "MINUEND" (the number from which another number will be subtracted). The second is B, the "SUBTRAHEND" (the number to be subtracted). The third is called the "DIFFERENCE" of A-B (the answer resulting from the subtraction) and is represented by "D". The fourth is called the "BORROW", represented by "W", which is generated when B is greater than A. Note that the only case for a single number difference to generate a "borrow" is:  $0 - 1 = 1$ , borrow 1. The other three combinations ( $0 - 0$ ,  $1 - 1$ , and  $1 - 0$ ) do not generate a "borrow".

Using "AND" and "OR" gates, the logical "difference" D is the same as the logical "sum" S and is represented as follows:

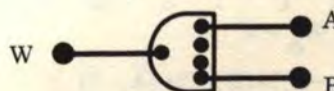
$$D = A - B = (A \cdot \bar{B}) \vee (\bar{A} \cdot B)$$

$$\begin{array}{r} A \\ -B \\ \hline D \end{array}$$



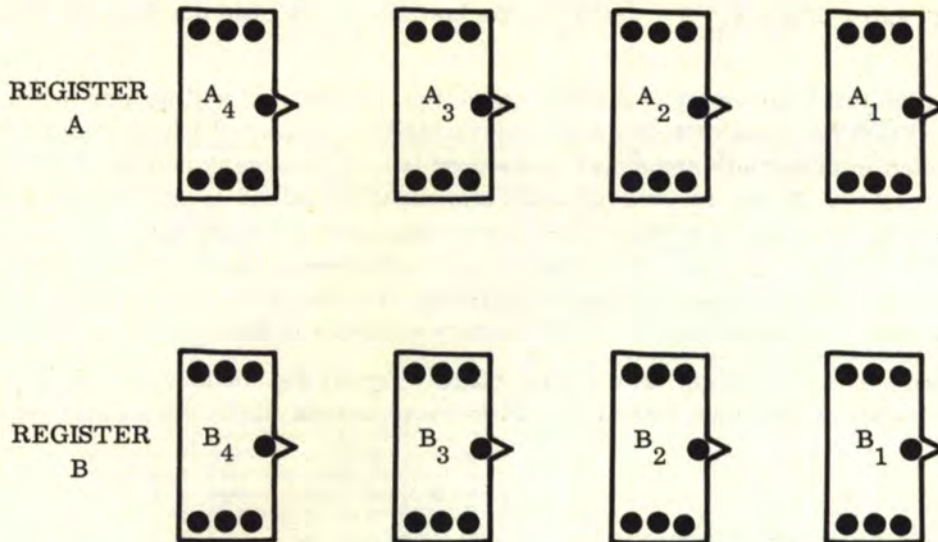
The "borrow" is represented as follows:

$$W = A \cdot B$$



## 10.3 THE LOGIC SUBTRACTER (Continued)

Let us represent registers A and B for a 4-bit logic subtracter as follows:



The subscripts 1, 2, 3, and 4 indicate the flip-flop position in each register. Now let us consider  $A_1$  and  $B_1$ . Outputs "A<sub>1</sub>" and "B<sub>1</sub>" are generated by the "TRUE" sides of flip-flops "A<sub>1</sub>" and "B<sub>1</sub>", respectively. Outputs " $\bar{A}_1$ " and " $\bar{B}_1$ " are generated by the "FALSE" sides of these respective flip-flops.

In logical subtraction, the binary number in register A must be greater than the binary number in register B. The "borrow" is treated much the same way as the "carry" in addition except that we will represent the "borrow" by the letter "W". Again, we will consider the least significant digits  $A_1$  and  $B_1$ . Note that no "borrow" can be generated by any flip-flops to the right. In other words, the initial "borrow" which we will call " $W_0$ ", is always "0" and the invert " $\bar{W}_0$ " is always "1". Therefore, using the general logic equation for the difference of three numbers A, B, and W, we may simplify both the "D" and the "W" for the first digit.

$$D_1 = (A_1 - B_1) - W_0$$

$$D_1 = (A_1 \cdot B_1 \cdot W_0) \vee (A_1 \cdot \bar{B}_1 \cdot \bar{W}_0) \vee (\bar{A}_1 \cdot B_1 \cdot \bar{W}_0) \vee (\bar{A}_1 \cdot \bar{B}_1 \cdot W_0)$$

$$\text{Set } W_0 = 0 \text{ and } \bar{W}_0 = 1$$

$$D_1 = (A_1 \cdot B_1 \cdot 0) \vee (A_1 \cdot \bar{B}_1 \cdot 1) \vee (\bar{A}_1 \cdot B_1 \cdot 1) \vee (\bar{A}_1 \cdot \bar{B}_1 \cdot 0)$$

$$D_1 = (0) \vee (A_1 \cdot \bar{B}_1 \cdot 1) \vee (\bar{A}_1 \cdot B_1 \cdot 1) \vee (0)$$

$$D_1 = (A_1 \cdot \bar{B}_1 \cdot 1) \vee (\bar{A}_1 \cdot B_1 \cdot 1)$$

$$D_1 = (A_1 \cdot \bar{B}_1) \vee (\bar{A}_1 \cdot B_1)$$

$$W_1 = (\bar{A}_1 \cdot B_1) \vee (B_1 \cdot W_0) \vee (\bar{A}_1 \cdot W_0)$$

$$\text{Set } W_0 = 0$$

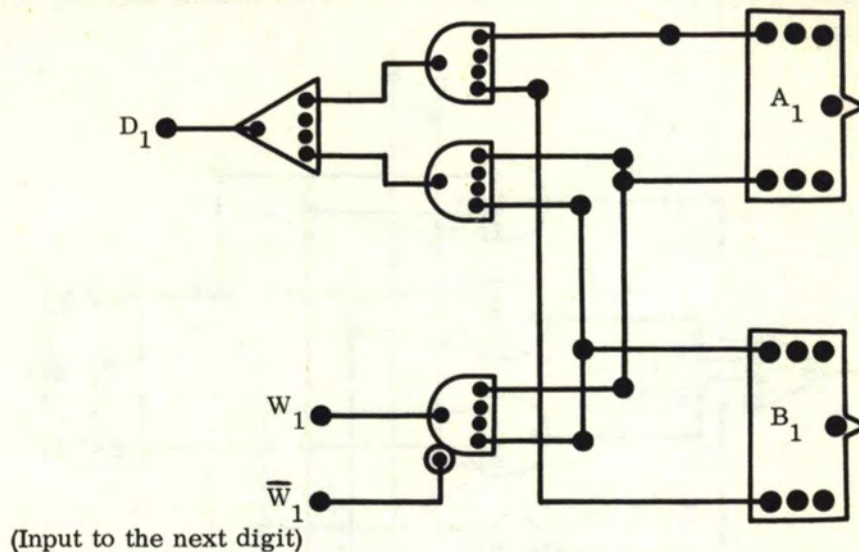
$$W_1 = (\bar{A}_1 \cdot B_1) \vee (B_1 \cdot 0) \vee (\bar{A}_1 \cdot 0)$$

$$W_1 = (\bar{A}_1 \cdot B_1) \vee (0) \vee (0)$$

$$W_1 = \bar{A}_1 \cdot B_1$$

## 10.3 THE LOGIC SUBTRACTER (Continued)

From the calculations above, we can represent subtraction on the LEAST SIGNIFICANT DIGIT by the following logic:



For intermediate positions such as  $A_2$  and  $B_2$ ,  $A_3$  and  $B_3$ , we have (using as an example  $A_2$  and  $B_2$ ) the following logic:

$$D_2 = (A_2 - B_2) - W_1$$

$$D_2 = (A_2 \cdot B_2 \cdot W_1) \vee (A_2 \cdot \bar{B}_2 \cdot \bar{W}_1) \vee (\bar{A}_2 \cdot B_2 \cdot \bar{W}_1) \vee (\bar{A}_2 \cdot \bar{B}_2 \cdot W_1)$$

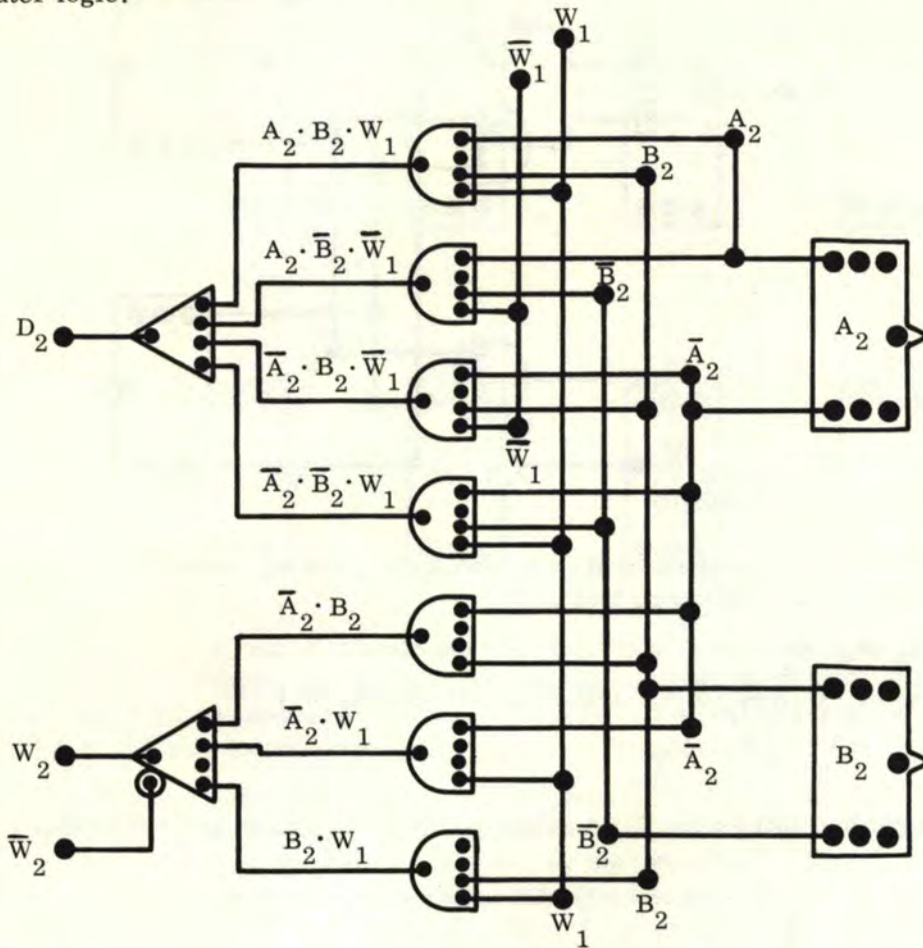
$$W_2 = (\bar{A}_2 \cdot B_2) \vee (B_2 \cdot W_1) \vee (\bar{A}_2 \cdot W_1)$$

The "TRUTH TABLE" for intermediate positions  $A_2$  and  $B_2$  is as follows:

$A_2$	$B_2$	$W_1$	$D_2$	$W_2$
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

## 10.3 THE LOGIC SUBTRACTER (Continued)

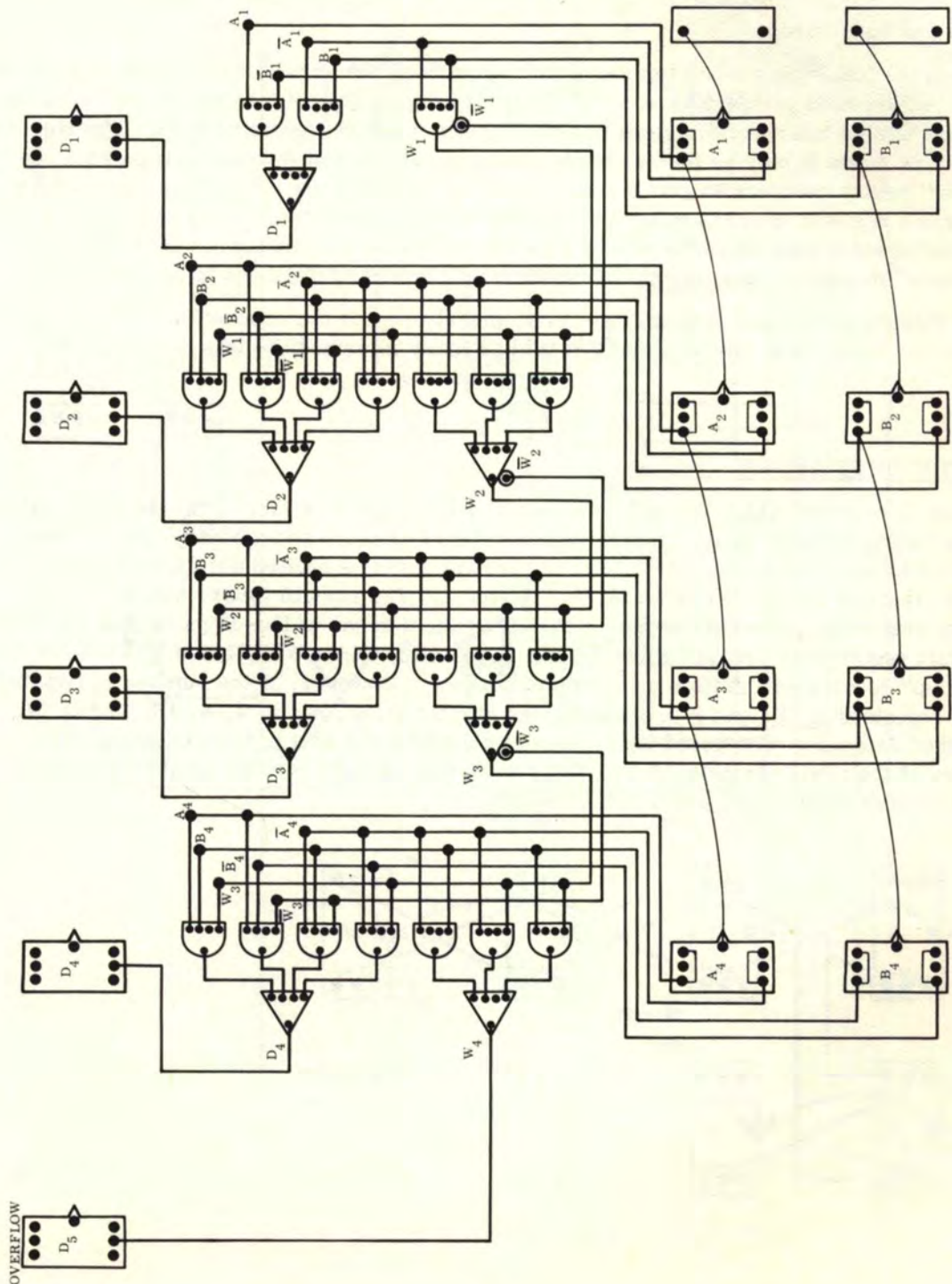
From the preceding equations, we can represent the intermediate digits by the following computer logic:



Note that the "borrows"  $W_1$  and  $\bar{W}_1$  are from the previous register digits (i. e., to the right) and that " $W_2$ " and " $\bar{W}_2$ " are the inputs for the next position over to the left. The logic circuitry which generates only the "difference" and not the "borrow" is called a HALF SUBTRACTER (which is logically the same as a HALF ADDER). Thus, the three "AND" gates and "OR" gate which generate  $D_2$  above are a HALF SUBTRACTER. The complete logic above which generates both the "difference" and "borrow" is called a FULL SUBTRACTER.

### 10.3 THE LOGIC SUBTRACTOR (Continued)

The complete 4-bit logic subtracter is shown in the figure below.



4-BIT LOGIC SUBTRACTER

### 10.3 THE LOGIC SUBTRACTER (Continued)

Note that the readouts consist only of flip-flops wired up in the "direct readout" configuration. The first four flip-flops  $D_1$ ,  $D_2$ ,  $D_3$ , and  $D_4$  read out the direct output of the final "OR" gates of  $D_1$ ,  $D_2$ ,  $D_3$ , and  $D_4$ . The last "difference"  $D_5$  is read out directly from the "borrow"  $W_4$ . This last "borrow" output to the extreme left of the register is called an "OVERFLOW". The "overflow" will come on only when an improper subtraction is performed (that is, when the number in register B is greater than the number in register A). This "overflow" may also be used as a control in more complicated operations such as division and square root.

When the full logic subtracter is put into operation, the answer will appear in register D to any subtraction problem as soon as the binary digits are entered into registers A and B. There are no intermediate steps and the answer is always instantaneous. The flip-flops in registers A and B may be further wired into separate pulsed (or common-pulsed) "up" or "down" binary counters to give a continuous display of all the subtraction possibilities. Each time a register changes value, the answer will change value. The registers A and B may also be wired into separate shift registers to obtain a continuous display of the "difference" depending upon the shifted position of the binary numbers in both registers.

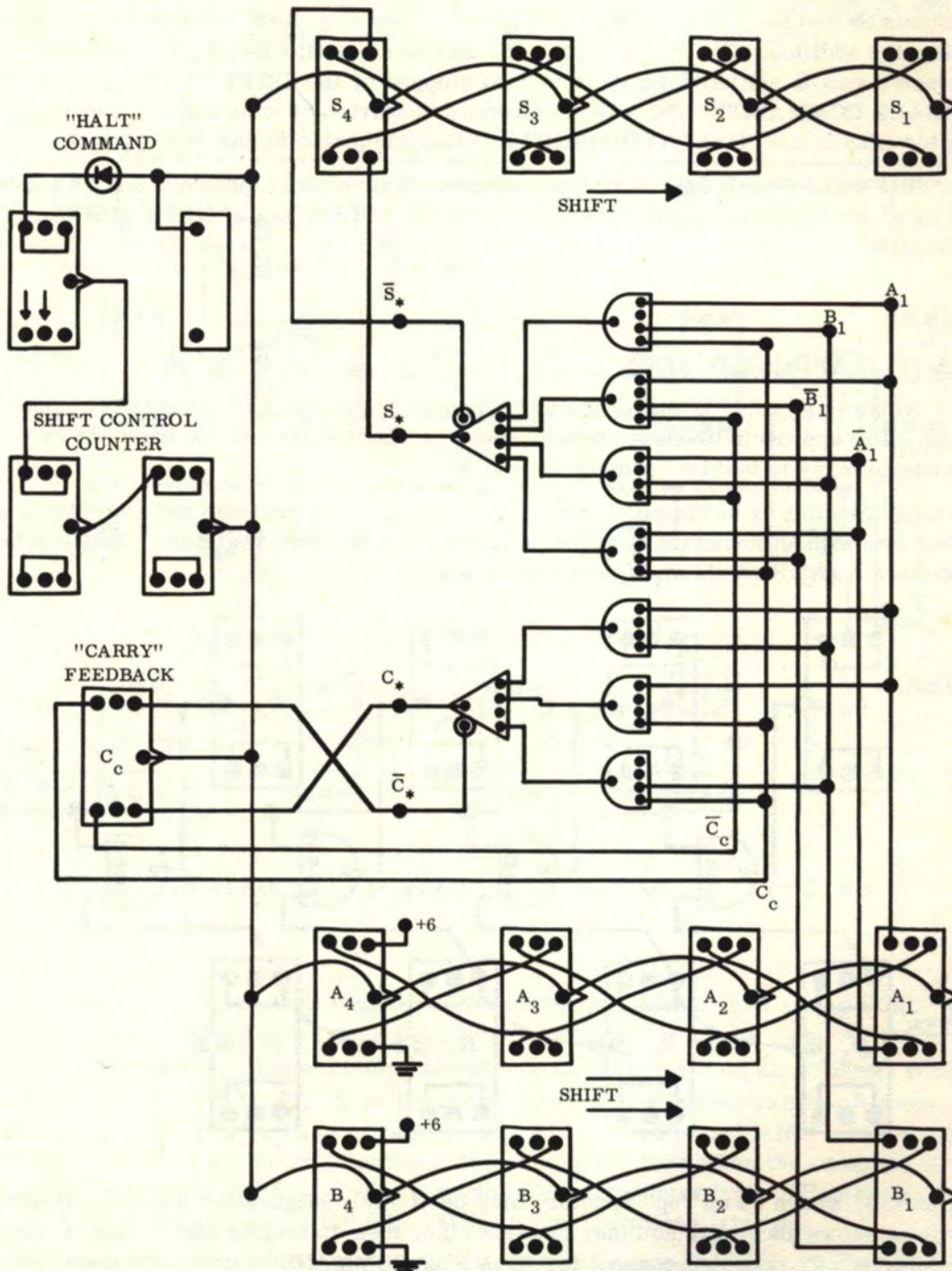
To make a subtracter larger than 4 bits, merely repeat the connections, logic, and circuitry as shown in positions 2 and 3 to add as many intermediate bits as necessary.

### 10.4 THE SHIFT ADDER

When it is necessary to keep the number of logic gates to a minimum, we can combine a single FULL ADDER logic circuit with three shift registers and a binary counter to perform the addition procedure. However, the answer must be shifted out from the registers A and B, through the full adder logic, to register S. The control binary counter must generate precisely the same number of pulses as the number of flip-flops in each register. If all shift registers A, B, and S have 4 flip-flops each, then a simple non-gated 2-bit binary "up" counter will suffice as a control counter. However, if the number of flip-flops in shift registers A, B, and S is a non-binary number (i. e. , not 2, 4, 8, 16, 32, or 64, etc.), then an appropriate gated logic counter must be used as a control counter. For example, if the registers have 10 flip-flops each, then a BCD counter must be used as a control counter to generate precisely 10 pulses.

## 10.4 THE SHIFT ADDER (Continued)

The diagram below shows the logic and wiring for a 4-bit shift adder.



## 10.4 THE SHIFT ADDER (Continued)

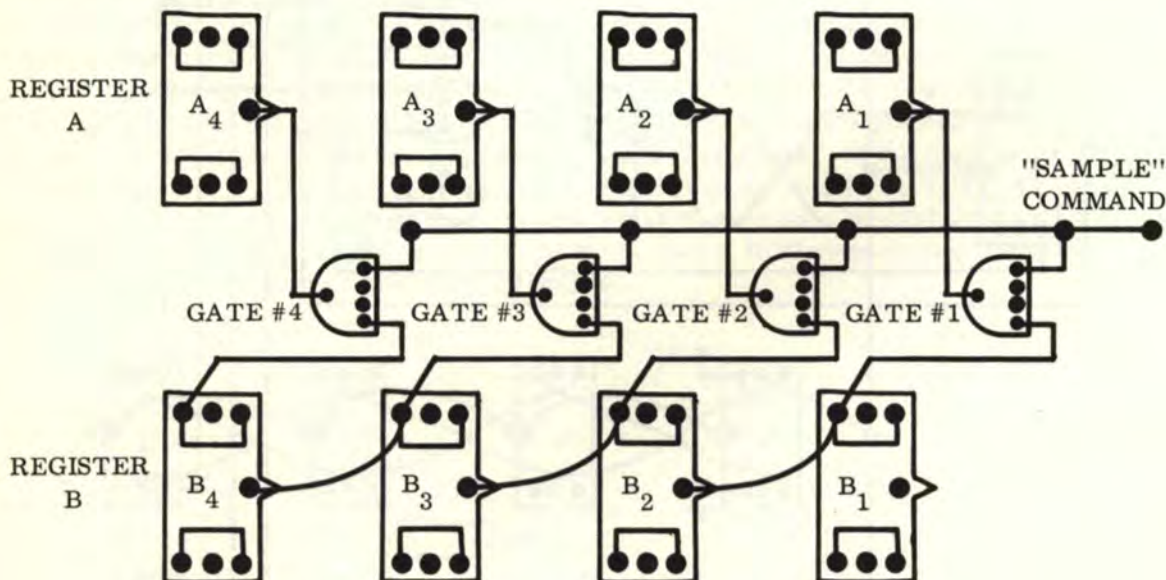
The first flip-flops to the right in the A and B registers are "sensed" by the full adder logic. Thus the inputs to the full adder are  $A_1$ ,  $\bar{A}_1$ ,  $B_1$ , and  $\bar{B}_1$ . The "carry" and inverted "carry"  $C^*$  and  $\bar{C}^*$  are generated from the flip-flop inputs. The "CARRY FEEDBACK" flip-flop  $C_c$  is directed by  $C^*$  and  $\bar{C}^*$  steering and generates the "carry" feedback outputs  $C_c$  and  $\bar{C}_c$ . These  $C_c$  and  $\bar{C}_c$  inputs are fed back into the full adder logic to generate the "sum" outputs  $S^*$  and  $\bar{S}^*$ . The "CARRY FEEDBACK" flip-flop must be reset to "0" before performing any addition. The "sum" outputs  $S^*$  and  $\bar{S}^*$  steer flip-flop  $S_4$  in register S and the successive sum is shifted down the line until stopped by the SHIFT CONTROL COUNTER and the "HALT COMMAND". The addition process is started by shorting together momentarily the two pins in the "HALT COMMAND" flip-flop (indicated by the two arrows).

The "SHIFT SUBTRACTER" is also possible and it is left as a problem for the reader to make the slight logic changes to convert the "SHIFT ADDER" into a "SHIFT SUBTRACTER" if he so wishes.

## 10.5 "SAMPLE-AND-HOLD" LOGIC

The "sample-and-hold" is one of the more important functions of advanced computer operations. The approach discussed here is that of a digital type and the logic shown is only one way in many to build a "sample-and-hold".

The basic function is to "sample" a number entered in one register and copy it into a second register without removing the basic number from the first register. Consider two 4-bit registers A and B as shown in the diagram below.



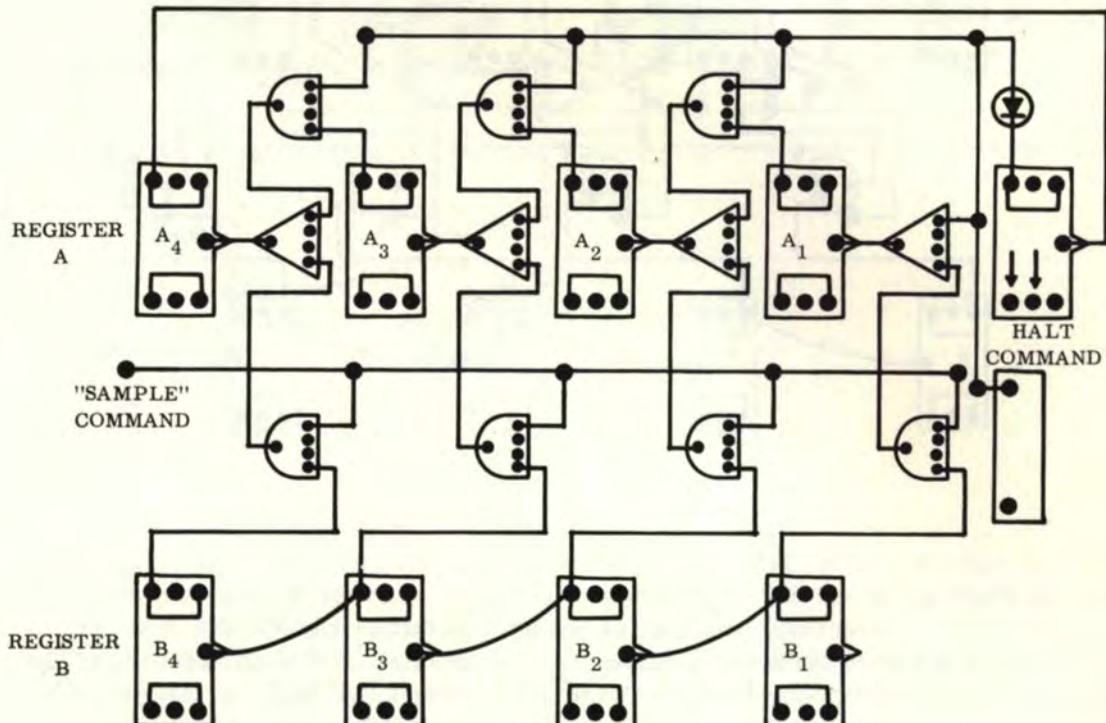
Register B, shown as an "up" counter, may be of any configuration desired. However, register A as shown above has no other function other than displaying the "sampled" number from register B. To operate, connect the "SAMPLE COMMAND" point to the output of an additional flip-flop. Each output in register B is now "AND"-ed with the outputs  $B_1, B_2, B_3$ , and  $B_4$ . Now, whatever number is present in register B will input to each respective "AND" gate where the number is a "1". The flip-flop which controls the "SAMPLE COMMAND" provides the second input to each "AND" gate. By setting this flip-flop to a "1", the gates are turned on where there is a "1" in the B register. By resetting this flip-flop, we turn off any "AND" gates that are "1", thus putting out a "trigger" pulse into each respective



### 10.5 "SAMPLE-AND-HOLD" LOGIC (Continued)

A register position where the "AND" gate was on. The number present in the B register is thus "sampled" into the A register and is held there until cancelled. As was mentioned earlier, there is no further function that can be performed with the A register as it stands now.

If we wish the A register to perform additional functions such as counting or shifting after sampling, more logic must be added as shown in the diagram below:



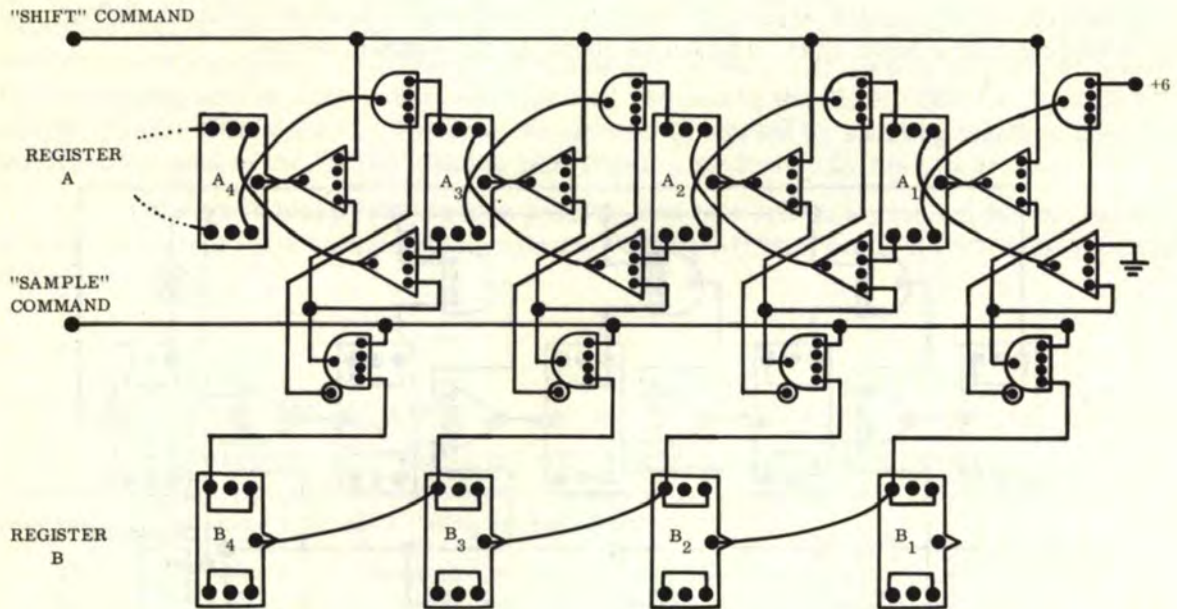
In this case, we have given register A the additional capability of acting as an "up" counter. By changing the gate inputs from the "true" sides of  $A_1$ ,  $A_2$ , and  $A_3$  to the "false" sides of  $A_1$ ,  $A_2$ , and  $A_3$ , respectively, register A will then operate as a "down" counter in addition to "sample-and-hold". Note the addition of the "OR" gates to allow for two separate inputs for each flip-flop in register A. Also, three more "AND" gates were added (on top) to "cut off" the counter interconnection during the time when the sampling is done. The HALT COMMAND flip-flop also acts as an effective counter "cut-off" control. This principle will be used in one of the MULTIPLIER configurations later on.


The number in register B will now be sampled into register A when the "SAMPLE" COMMAND is triggered by a flip-flop output. Then, by setting the HALT COMMAND flip-flop, the count will begin from the number that was "sampled" in. The count may be stopped at any time by resetting the HALT COMMAND flip-flop.

If register A is connected as a shift register, then the top "cut-off" "AND" gates are not needed, but the logic is more complicated and requires inverted outputs from the "AND" gate.

## 10.5 "SAMPLE-AND-HOLD" LOGIC (Continued)

The configuration is as follows:



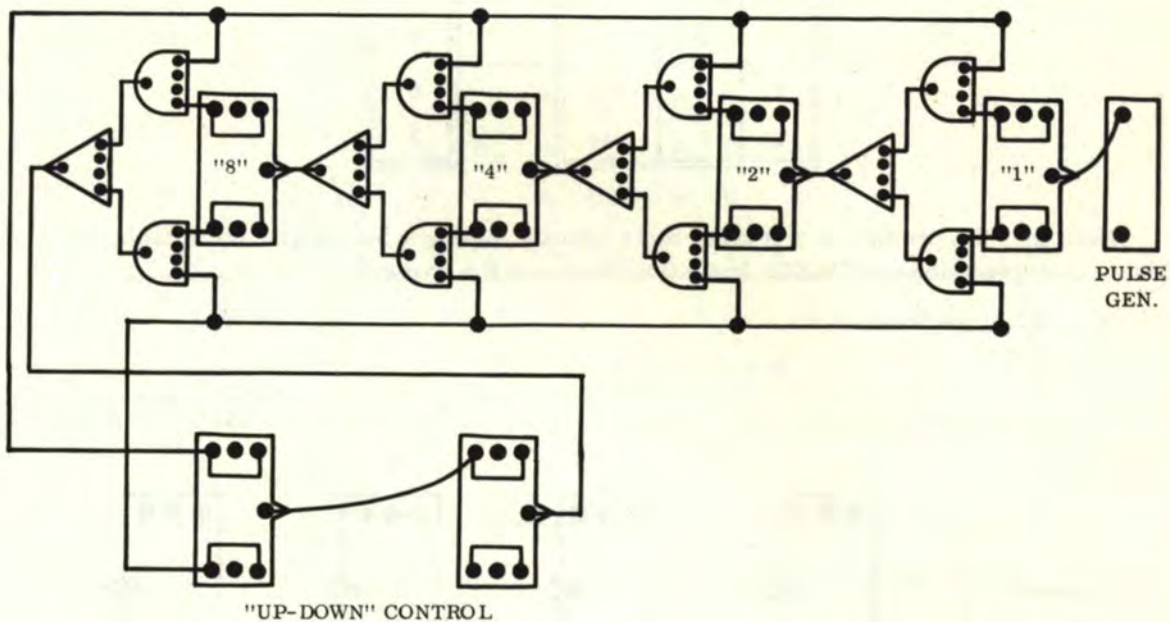
Now we have given register A the additional capability of acting as a shift register (left). The "shift" connections may also be changed so that a "right" shift is performed (no other wiring changes are necessary). Note the connections to "+6" and "ground" (  ). These must also be adjusted accordingly so that the number will shift out left (or right).

The number in register B will now be sampled into register A when the "SAMPLE" COMMAND is triggered by a flip-flop output. Then, by further triggering of the "SHIFT" COMMAND, the "sampled" number will shift out in the left direction. The A register can also be wired for "end-around" shifting in either direction.

## 10.6 GATED "UP-DOWN" COUNTER

The basic definition of a binary "UP-DOWN" COUNTER is a single register binary counter that can be controlled by some means so as to count "UP" or "DOWN" at the proper command. The basic 4-bit counter configuration shown here uses "AND" and "OR" logic with two flip-flops to control the "UP" and "DOWN" count.

## 10.6 GATED "UP-DOWN" COUNTER (Continued)



The "UP-DOWN" CONTROL flip-flops are wired up so that the free-running counter will count alternately "UP" and then switch automatically to "DOWN" after resetting from the "UP" count. The counter will then count "DOWN" to zero and then "overflow" (i. e. , all numbers in the counter will be "1's"). The "overflow" will switch the counter back to "UP" again and all the "1's" will reset. The "UP" count will again begin from zero and proceed over and over again as described above.

It is of interest to the reader to note that, as an automatic counter, it is not possible to go through all the "UP" counts and "DOWN" counts without allowing for the "overflow". The reason is that, as a "DOWN" counter, all the "AND" gates controlling the "FALSE" side of the flip-flops are "ON" when the binary numbers in the counter are all zeros. When the counter is switched to "UP" again, these "AND" gates will shut off and generate a "carry" which will turn on flip-flops when the counter should be reset. This condition does not exist when the "DOWN" count is in "overflow".

## 10.7 THE FULL-LOGIC BINARY MULTIPLIER

In logical multiplication, we have three main numbers to consider. The first two are A and B, the two numbers to be multiplied together (i. e. , the "MULTIPLICAND" and "MULTIPLIER"). The third is called the "PRODUCT" of A and B and is represented by "P". The "PRODUCT", however, must be broken down into individual cross-products which will be represented by double-subscript notations such as  $P_{13}$ ,  $P_{22}$ , etc.

The "times" sign (x) will be used in this discussion to refer to arithmetical multiplication. Now let us work out all the multiplication possibilities for a single-number product and present them in a truth table.

10.7 THE FULL-LOGIC BINARY MULTIPLIER (Continued)

A	B	A x B
0	0	0
0	1	0
1	0	0
1	1	1

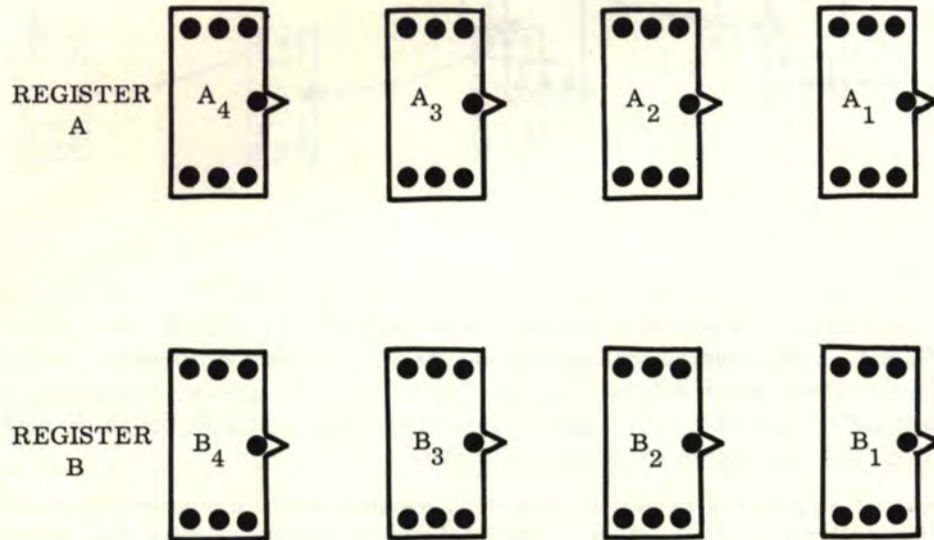
A · B
0
0
0
1

Note that the product  $A \times B$  truth table exactly matches the "AND" truth table for  $A \cdot B$ . Thus, multiplication and "AND" logic functions are the same.

Basically, we have:

$$P = A \times B = A \cdot B$$

Let us represent again two flip-flop registers A and B for a 4-bit full logic multiplier as follows:



The subscripts represent the flip-flop position in each register. The number in each register is then represented as  $A_4 A_3 A_2 A_1$  and  $B_4 B_3 B_2 B_1$ . Now let us multiply these two numbers together:

				$A_4$	$A_3$	$A_2$	$A_1$		
				x	$B_4$	$B_3$	$B_2$	$B_1$	
				$A_4 \times B_1$	$A_3 \times B_1$	$A_2 \times B_1$	$A_1 \times B_1$		
			$A_4 \times B_2$	$A_3 \times B_2$	$A_2 \times B_2$	$A_1 \times B_2$			
		$A_4 \times B_3$	$A_3 \times B_3$	$A_2 \times B_3$	$A_1 \times B_3$				
	$A_4 \times B_4$	$A_3 \times B_4$	$A_2 \times B_4$	$A_1 \times B_4$					
$P_8$	$P_7$	$P_6$	$P_5$	$P_4$	$P_3$	$P_2$	$P_1$		

10.7 THE FULL-LOGIC BINARY MULTIPLIER (Continued)

Using the definition that  $A \times B = A \cdot B$ , we have the following expression:

$$\begin{array}{r}
 \phantom{x} \phantom{A_4} \phantom{A_3} \phantom{A_2} A_1 \\
 \phantom{x} \phantom{A_4} \phantom{A_3} A_2 \phantom{A_1} \\
 \phantom{x} \phantom{A_4} A_3 \phantom{A_2} \phantom{A_1} \\
 \phantom{x} A_4 \phantom{A_3} \phantom{A_2} \phantom{A_1} \\
 \hline
 \phantom{x} \phantom{A_4} \phantom{A_3} \phantom{A_2} A_1 \cdot B_1 \phantom{A_1} \\
 \phantom{x} \phantom{A_4} \phantom{A_3} A_2 \cdot B_1 \phantom{A_2} \phantom{A_1} \\
 \phantom{x} \phantom{A_4} A_3 \cdot B_1 \phantom{A_3} \phantom{A_2} \phantom{A_1} \\
 \phantom{x} A_4 \cdot B_1 \phantom{A_4} \phantom{A_3} \phantom{A_2} \phantom{A_1} \\
 \hline
 \phantom{x} \phantom{A_4} \phantom{A_3} A_2 \cdot B_2 \phantom{A_2} \phantom{A_1} \\
 \phantom{x} \phantom{A_4} A_3 \cdot B_2 \phantom{A_3} \phantom{A_2} \phantom{A_1} \\
 \phantom{x} A_4 \cdot B_2 \phantom{A_4} \phantom{A_3} \phantom{A_2} \phantom{A_1} \\
 \hline
 \phantom{x} \phantom{A_4} \phantom{A_3} A_2 \cdot B_3 \phantom{A_2} \phantom{A_1} \\
 \phantom{x} \phantom{A_4} A_3 \cdot B_3 \phantom{A_3} \phantom{A_2} \phantom{A_1} \\
 \phantom{x} A_4 \cdot B_3 \phantom{A_4} \phantom{A_3} \phantom{A_2} \phantom{A_1} \\
 \hline
 \phantom{x} \phantom{A_4} \phantom{A_3} A_2 \cdot B_4 \phantom{A_2} \phantom{A_1} \\
 \phantom{x} \phantom{A_4} A_3 \cdot B_4 \phantom{A_3} \phantom{A_2} \phantom{A_1} \\
 \phantom{x} A_4 \cdot B_4 \phantom{A_4} \phantom{A_3} \phantom{A_2} \phantom{A_1} \\
 \hline
 P_8 \phantom{P_7} \phantom{P_6} \phantom{P_5} \phantom{P_4} \phantom{P_3} \phantom{P_2} \phantom{P_1} \\
 P_7 \phantom{P_6} \phantom{P_5} \phantom{P_4} \phantom{P_3} \phantom{P_2} \phantom{P_1} \\
 P_6 \phantom{P_5} \phantom{P_4} \phantom{P_3} \phantom{P_2} \phantom{P_1} \\
 P_5 \phantom{P_4} \phantom{P_3} \phantom{P_2} \phantom{P_1} \\
 P_4 \phantom{P_3} \phantom{P_2} \phantom{P_1} \\
 P_3 \phantom{P_2} \phantom{P_1} \\
 P_2 \phantom{P_1} \\
 P_1
 \end{array}$$

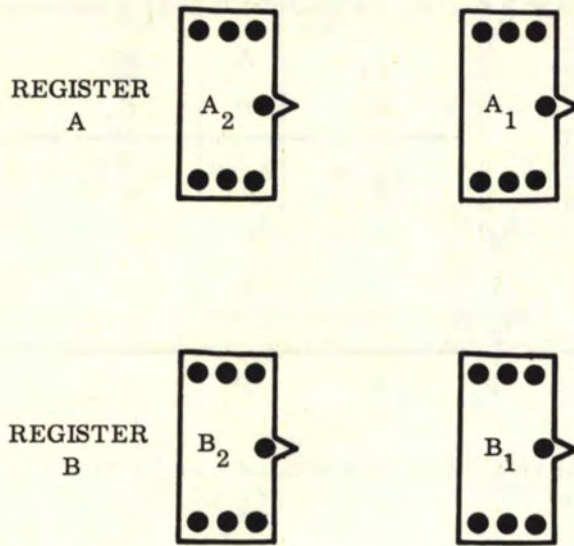
Now, using our "cross-product" subscript notation, we have:

$$\begin{array}{r}
 \phantom{x} \phantom{A_4} \phantom{A_3} \phantom{A_2} A_1 \\
 \phantom{x} \phantom{A_4} \phantom{A_3} A_2 \phantom{A_1} \\
 \phantom{x} \phantom{A_4} \phantom{A_3} A_3 \phantom{A_2} \phantom{A_1} \\
 \phantom{x} \phantom{A_4} A_4 \phantom{A_3} \phantom{A_2} \phantom{A_1} \\
 \hline
 \phantom{x} \phantom{A_4} \phantom{A_3} \phantom{A_2} P_{41} \phantom{A_1} \\
 \phantom{x} \phantom{A_4} \phantom{A_3} P_{31} \phantom{A_2} \phantom{A_1} \\
 \phantom{x} \phantom{A_4} P_{21} \phantom{A_3} \phantom{A_2} \phantom{A_1} \\
 \phantom{x} P_{11} \phantom{A_4} \phantom{A_3} \phantom{A_2} \phantom{A_1} \\
 \hline
 \phantom{x} \phantom{A_4} \phantom{A_3} P_{42} \phantom{A_2} \phantom{A_1} \\
 \phantom{x} \phantom{A_4} P_{32} \phantom{A_3} \phantom{A_2} \phantom{A_1} \\
 \phantom{x} P_{22} \phantom{A_4} \phantom{A_3} \phantom{A_2} \phantom{A_1} \\
 \phantom{x} P_{12} \phantom{A_4} \phantom{A_3} \phantom{A_2} \phantom{A_1} \\
 \hline
 \phantom{x} \phantom{A_4} \phantom{A_3} P_{43} \phantom{A_3} \phantom{A_2} \phantom{A_1} \\
 \phantom{x} \phantom{A_4} P_{33} \phantom{A_4} \phantom{A_3} \phantom{A_2} \phantom{A_1} \\
 \phantom{x} P_{23} \phantom{A_4} \phantom{A_3} \phantom{A_2} \phantom{A_1} \\
 \phantom{x} P_{13} \phantom{A_4} \phantom{A_3} \phantom{A_2} \phantom{A_1} \\
 \hline
 \phantom{x} \phantom{A_4} P_{44} \phantom{A_4} \phantom{A_3} \phantom{A_2} \phantom{A_1} \\
 \phantom{x} P_{34} \phantom{A_4} \phantom{A_3} \phantom{A_2} \phantom{A_1} \\
 \phantom{x} P_{24} \phantom{A_4} \phantom{A_3} \phantom{A_2} \phantom{A_1} \\
 \phantom{x} P_{14} \phantom{A_4} \phantom{A_3} \phantom{A_2} \phantom{A_1} \\
 \hline
 P_8 \phantom{P_7} \phantom{P_6} \phantom{P_5} \phantom{P_4} \phantom{P_3} \phantom{P_2} \phantom{P_1} \\
 P_7 \phantom{P_6} \phantom{P_5} \phantom{P_4} \phantom{P_3} \phantom{P_2} \phantom{P_1} \\
 P_6 \phantom{P_5} \phantom{P_4} \phantom{P_3} \phantom{P_2} \phantom{P_1} \\
 P_5 \phantom{P_4} \phantom{P_3} \phantom{P_2} \phantom{P_1} \\
 P_4 \phantom{P_3} \phantom{P_2} \phantom{P_1} \\
 P_3 \phantom{P_2} \phantom{P_1} \\
 P_2 \phantom{P_1} \\
 P_1
 \end{array}$$

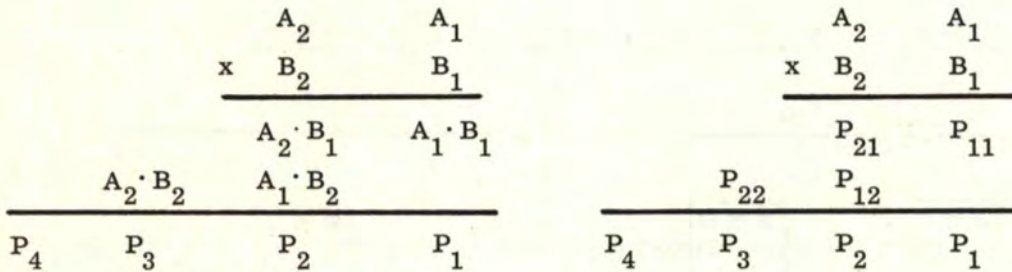
Note the equivalent cross-product expressions. Examples:  $P_{41} = A_4 \cdot B_1$ ,  $P_{22} = A_2 \cdot B_2$ , and  $P_{34} = A_3 \cdot B_4$ . Each cross-product is a separate "AND" expression. The bottom product numbers with single subscripts (i.e.,  $P_1$ ,  $P_2$ ,  $P_3$ , etc.) represent the binary product digits of the complete product. These product digits represent the sum of all respective cross-products in each column and the "carries" that may be generated by previous columns. Thus,  $P_1 = P_{11} = A_1 \cdot B_1$ .  $P_2 = P_{21} + P_{12}$  and  $C_2$  (the "carry") =  $P_{21} \cdot P_{12}$ . The expressions become very complicated. Note that the number of product digit positions must be exactly equal to the total number of digit positions in both the A and B registers. In this case, both registers have 4 digits each so  $4 + 4 = 8$  product digit positions.

We will continue this discussion using a 2-bit full logic multiplier for simplicity.

10.7 THE FULL-LOGIC BINARY MULTIPLIER (Continued)



Now let us start again with the cross-product definitions and set up the logic equations.



We can see that  $P_{11} = A_1 \cdot B_1$ ,  $P_{21} = A_2 \cdot B_1$ ,  $P_{12} = A_1 \cdot B_2$ , and  $P_{22} = A_2 \cdot B_2$ . We are now ready to determine the logic for the product digit positions  $P_4$ ,  $P_3$ ,  $P_2$ , and  $P_1$ .

$$P_1 = P_{11} = \boxed{A_1 \cdot B_1}$$

$$P_2 = P_{21} + P_{12} = (A_2 \cdot B_1) + (A_1 \cdot B_2)$$

Using the addition identity we have

$$P_{21} + P_{12} = (P_{21} \cdot \overline{P_{12}}) \vee (\overline{P_{21}} \cdot P_{12}) \tag{AI #1}$$

$$P_2 = [(A_2 \cdot B_1) \cdot \overline{(A_1 \cdot B_2)}] \vee [\overline{(A_2 \cdot B_1)} \cdot (A_1 \cdot B_2)] \tag{Substituting}$$

$$= [(A_2 \cdot B_1) \cdot (\overline{A_1} \vee \overline{B_2})] \vee [(\overline{A_2} \vee \overline{B_1}) \cdot (A_1 \cdot B_2)] \tag{DM #1}$$

$$= [(\overline{A_1} \cdot A_2 \cdot B_1) \vee (A_2 \cdot B_1 \cdot \overline{B_2})] \vee [(A_1 \cdot \overline{A_2} \cdot B_2) \vee (A_1 \cdot \overline{B_1} \cdot B_2)] \tag{DL #1}$$

$$P_2 = \boxed{(\overline{A_1} \cdot A_2 \cdot B_1) \vee (A_2 \cdot B_1 \cdot \overline{B_2}) \vee (A_1 \cdot \overline{A_2} \cdot B_2) \vee (A_1 \cdot \overline{B_1} \cdot B_2)} \tag{AL #2}$$

$$C_2 = P_{21} \cdot P_{12} = A_1 \cdot A_2 \cdot B_1 \cdot B_2$$

## 10.7 THE FULL-LOGIC BINARY MULTIPLIER (Continued)

$$\begin{aligned}
P_3 &= P_{22} + C_2 \\
&= (A_2 \cdot B_2) + C_2 && \text{(Substituting)} \\
&= [(A_2 \cdot B_2) \cdot \overline{C_2}] \vee [(A_2 \cdot B_2) \cdot C_2] && \text{(AI \#1)} \\
&= [(A_2 \cdot B_2) \cdot (\overline{A_1 \cdot A_2 \cdot B_1 \cdot B_2})] \vee [(A_2 \cdot B_2) \cdot (A_1 \cdot A_2 \cdot B_1 \cdot B_2)] && \text{(Substituting)} \\
&= [(A_2 \cdot B_2) \cdot (\overline{A_1} \vee \overline{A_2} \vee \overline{B_1} \vee \overline{B_2})] \vee [(A_2 \cdot B_2) \cdot (A_1 \cdot A_2 \cdot B_1 \cdot B_2)] && \text{(DM \#5)} \\
&= [(A_2 \cdot B_2 \cdot \overline{A_1}) \vee (A_2 \cdot B_2 \cdot \overline{A_2}) \vee (A_2 \cdot B_2 \cdot \overline{B_1}) \vee (A_2 \cdot B_2 \cdot \overline{B_2})] \vee \\
&\quad [(A_1 \cdot A_2 \cdot B_1 \cdot B_2 \cdot \overline{A_2}) \vee (A_1 \cdot A_2 \cdot B_1 \cdot B_2 \cdot \overline{B_2})] && \text{(DL \#1 \& AL \#2)} \\
&= [(A_2 \cdot B_2 \cdot \overline{A_1}) \vee (0) \vee (A_2 \cdot B_2 \cdot \overline{B_1}) \vee (0)] \vee [(0) \vee (0)] && \text{(FI \#2)} \\
&= [(A_2 \cdot B_2 \cdot \overline{A_1}) \vee (A_2 \cdot B_2 \cdot \overline{B_1})] && \text{(Simplifying)} \\
&= (A_2 \cdot B_2 \cdot \overline{A_1}) \vee (A_2 \cdot B_2 \cdot \overline{B_1}) && \text{(AL \#2)} \\
P_3 &= \boxed{(\overline{A_1} \cdot A_2 \cdot B_2) \vee (A_2 \cdot \overline{B_1} \cdot B_2)} && \text{(CL \#1)}
\end{aligned}$$

$$\begin{aligned}
C_3 &= P_{22} \cdot C_2 \\
&= A_2 \cdot B_2 \cdot C_2 && \text{(Substituting)} \\
&= A_2 \cdot B_2 \cdot A_1 \cdot A_2 \cdot B_1 \cdot B_2 && \text{(Substituting)} \\
&= A_1 \cdot A_2 \cdot B_1 \cdot B_2 && \text{(FI \#1)}
\end{aligned}$$

$$P_4 = C_3$$

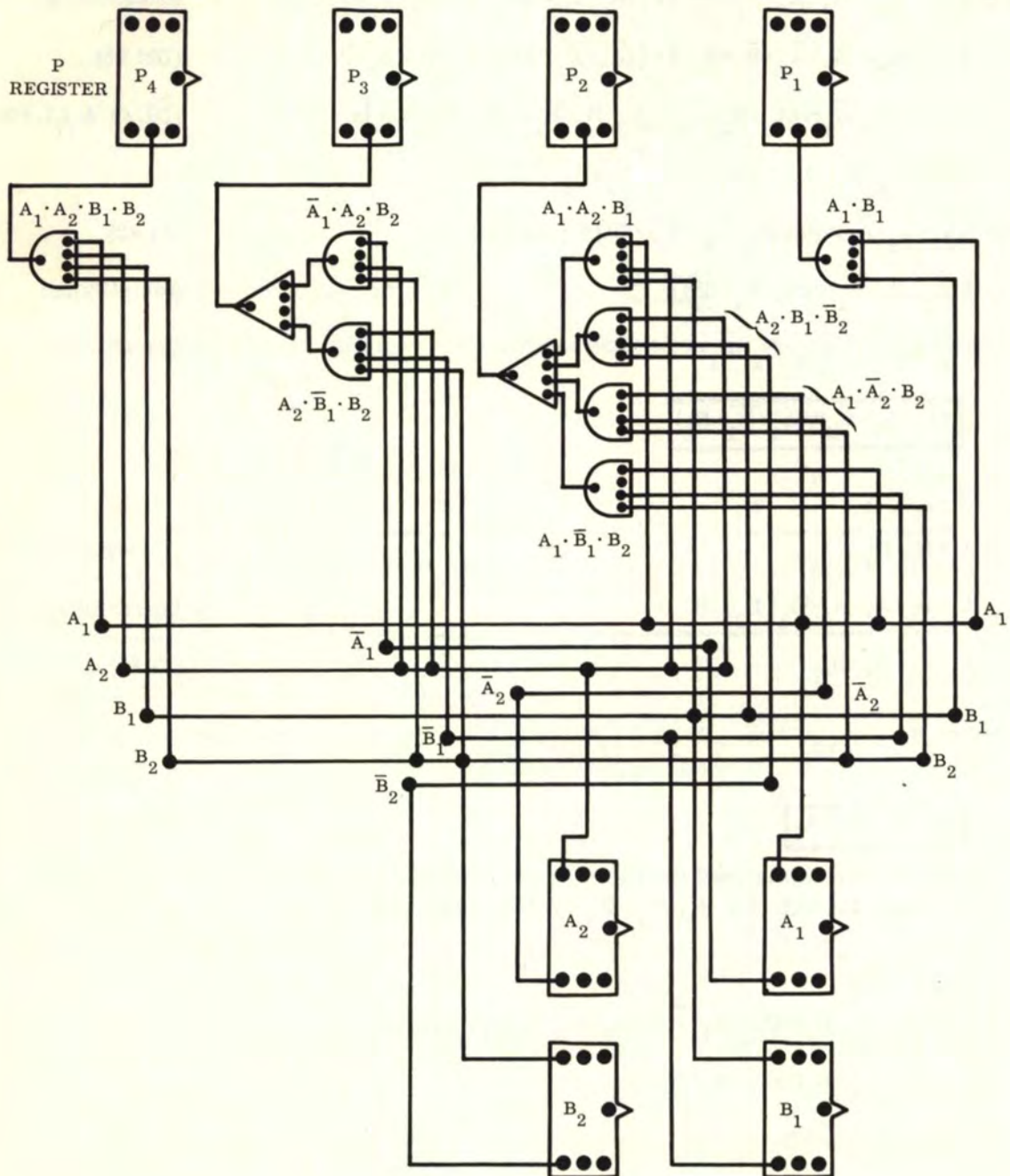
$$P_4 = \boxed{A_1 \cdot A_2 \cdot B_1 \cdot B_2} \quad \text{(Substituting)}$$

Thus we have determined the logic for the product digits of the 2-bit multiplier. Summarizing, the logic for  $P_1$ ,  $P_2$ ,  $P_3$ , and  $P_4$  is as follows:

$$\begin{aligned}
P_1 &= A_1 \cdot B_1 \\
P_2 &= (A_1 \cdot A_2 \cdot B_1) \vee (A_2 \cdot B_1 \cdot \overline{B_2}) \vee (A_1 \cdot \overline{A_2} \cdot B_2) \vee (A_1 \cdot \overline{B_1} \cdot B_2) \\
P_3 &= (\overline{A_1} \cdot A_2 \cdot B_2) \vee (A_2 \cdot \overline{B_1} \cdot B_2) \\
P_4 &= A_1 \cdot A_2 \cdot B_1 \cdot B_2
\end{aligned}$$

Using the above information, the configuration for the 2-bit full logic multiplier is as follows:

10.7 THE FULL-LOGIC BINARY MULTIPLIER (Continued)





## 10.7 THE FULL-LOGIC BINARY MULTIPLIER (Continued)

Note that the readouts for  $P_1$ ,  $P_2$ ,  $P_3$ , and  $P_4$  consist only of flip-flops wired up in the "direct readout" configuration.

When the full logic multiplier is put into operation, the answer will appear in register P for any multiplication problem as soon as the binary digits are entered into registers A and B. There are no intermediate steps and the answer is always instantaneous. The flip-flops in registers A and B may be further wired into separate pulsed (or common-pulsed) "up" or "down" binary counters to give a continuous display of all the multiplication possibilities. Each time a register changes value, the product will change value. The registers A and B may also be wired into separate shift registers.

It becomes very complicated to enlarge a multiplier of this kind to handle more than two binary digits in the A and B registers. As an example, let us summarize the very basic equations for the 3-bit and 4-bit configurations:

$$\begin{array}{r}
 \begin{array}{r}
 A_3 \quad A_2 \quad A_1 \\
 \times B_3 \quad B_2 \quad B_1 \\
 \hline
 P_{31} \quad P_{21} \quad P_{11} \\
 P_{32} \quad P_{22} \quad P_{12} \\
 P_{33} \quad P_{23} \quad P_{13} \\
 \hline
 P_6 \quad P_5 \quad P_4 \quad P_3 \quad P_2 \quad P_1
 \end{array}
 \qquad
 \begin{array}{r}
 A_4 \quad A_3 \quad A_2 \quad A_1 \\
 \times B_4 \quad B_3 \quad B_2 \quad B_1 \\
 \hline
 P_{41} \quad P_{31} \quad P_{21} \quad P_{11} \\
 P_{42} \quad P_{32} \quad P_{22} \quad P_{12} \\
 P_{43} \quad P_{33} \quad P_{23} \quad P_{13} \\
 P_{44} \quad P_{34} \quad P_{24} \quad P_{14} \\
 \hline
 P_8 \quad P_7 \quad P_6 \quad P_5 \quad P_4 \quad P_3 \quad P_2 \quad P_1
 \end{array}
 \end{array}$$

For 3-bit A and B registers, we have:

$$P_1 = P_{11} = A_1 \cdot B_1$$

$$P_2 = P_{21} + P_{12}$$

$$C_2 = P_{21} \cdot P_{12} = A_1 \cdot A_2 \cdot B_1 \cdot B_2$$

$$P_3 = P_{31} + P_{22} + P_{13} + C_2$$

$$C_3 = (P_{31} \cdot P_{22}) \vee (P_{31} \cdot P_{13}) \vee (P_{31} \cdot C_2) \vee (P_{22} \cdot P_{13}) \vee (P_{22} \cdot C_2) \vee (P_{13} \cdot C_2)$$

$$* C_{3A} = P_{31} \cdot P_{22} \cdot P_{13} \cdot C_2$$

$$P_4 = P_{32} + P_{23} + C_3$$

$$C_4 = (P_{32} \cdot P_{23}) \vee (P_{32} \cdot C_3) \vee (P_{23} \cdot C_3)$$

$$P_5 = P_{33} + C_4 + C_{3A}$$

$$C_5 = (P_{33} \cdot C_4) \vee (P_{33} \cdot C_{3A}) \vee (C_4 \cdot C_{3A})$$

$$P_6 = C_5$$

\* "Carry"  $C_{3A}$  is the second "carry" generated by  $P_{31}$ ,  $P_{22}$ ,  $P_{13}$ , and  $C_2$ . The  $C_{3A}$  occurs in a position two digits over to the left. Note how the "carries" become complicated!

## 10.7 THE FULL-LOGIC BINARY MULTIPLIER (Continued)

For 4-bit A and B registers, we have:

$$P_1 = P_{11} = A_1 \cdot B_1$$

$$P_2 = P_{21} + P_{12}$$

$$C_2 = P_{21} \cdot P_{12} = A_1 \cdot A_2 \cdot B_1 \cdot B_2$$

$$P_3 = P_{31} + P_{22} + P_{13} + C_2$$

$$C_3 = (P_{31} \cdot P_{22}) \vee (P_{31} \cdot P_{13}) \vee (P_{31} \cdot C_2) \vee (P_{22} \cdot P_{13}) \vee (P_{22} \cdot C_2) \vee (P_{13} \cdot C_2)$$

$$C_{3A} = P_{31} \cdot P_{22} \cdot P_{13} \cdot C_2$$

$$P_4 = P_{41} + P_{32} + P_{23} + P_{14} + C_3$$

$$C_4 = (P_{41} \cdot P_{32}) \vee (P_{41} \cdot P_{23}) \vee (P_{41} \cdot P_{14}) \vee (P_{41} \cdot C_3) \vee (P_{32} \cdot P_{23}) \vee (P_{32} \cdot P_{14}) \vee (P_{32} \cdot C_3) \vee (P_{23} \cdot P_{14}) \vee (P_{23} \cdot C_3) \vee (P_{14} \cdot C_3)$$

$$C_{4A} = (P_{41} \cdot P_{32} \cdot P_{23} \cdot P_{14}) \vee (P_{41} \cdot P_{23} \cdot P_{14} \cdot C_3) \vee (P_{41} \cdot P_{32} \cdot P_{14} \cdot C_3) \vee (P_{41} \cdot P_{32} \cdot P_{23} \cdot C_3) \vee (P_{32} \cdot P_{23} \cdot P_{14} \cdot C_3)$$

$$P_5 = P_{42} + P_{33} + P_{24} + C_{3A} + C_4$$

$$C_5 = (P_{42} \cdot P_{33}) \vee (P_{42} \cdot P_{24}) \vee (P_{42} \cdot C_{3A}) \vee (P_{42} \cdot C_4) \vee (P_{33} \cdot P_{24}) \vee (P_{33} \cdot C_{3A}) \vee (P_{33} \cdot C_4) \vee (P_{24} \cdot C_{3A}) \vee (P_{24} \cdot C_4) \vee (C_{3A} \cdot C_4)$$

$$C_{5A} = (P_{42} \cdot P_{33} \cdot P_{24} \cdot C_{3A}) \vee (P_{42} \cdot P_{24} \cdot C_{3A} \cdot C_4) \vee (P_{42} \cdot P_{33} \cdot C_{3A} \cdot C_4) \vee (P_{42} \cdot P_{33} \cdot P_{24} \cdot C_4) \vee (P_{33} \cdot P_{24} \cdot C_{3A} \cdot C_4)$$

$$P_6 = P_{43} + P_{34} + C_{4A} + C_5$$

$$C_6 = (P_{43} \cdot P_{34}) \vee (P_{43} \cdot C_{4A}) \vee (P_{43} \cdot C_5) \vee (P_{34} \cdot C_{4A}) \vee (P_{34} \cdot C_5) \vee (C_{4A} \cdot C_5)$$

$$C_{6A} = P_{43} \cdot P_{34} \cdot C_{4A} \cdot C_5$$

$$P_7 = P_{44} + C_{5A} + C_6$$

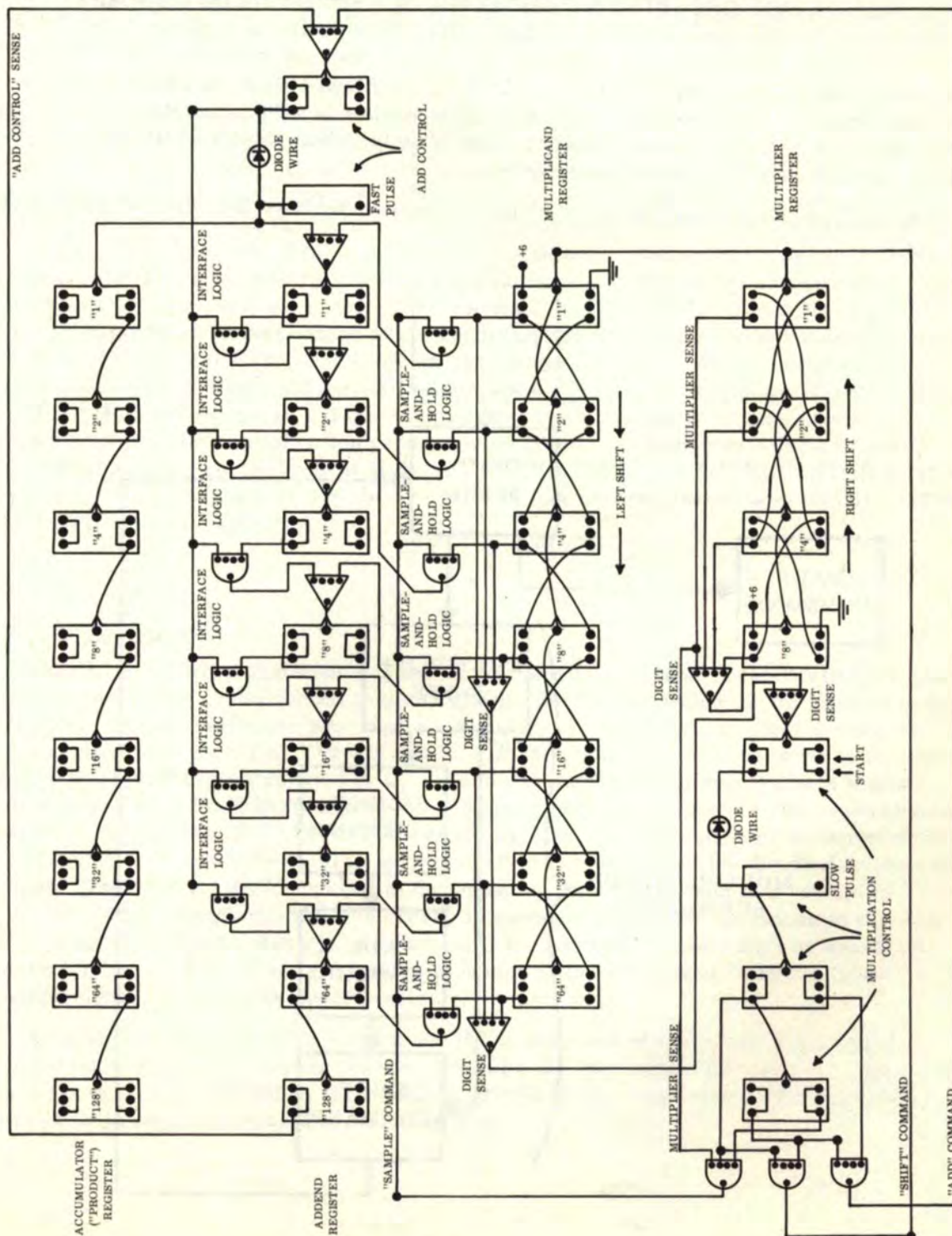
$$C_7 = (P_{44} \cdot C_{5A}) \vee (P_{44} \cdot C_6) \vee (C_{5A} \cdot C_6)$$

$$P_8 = C_{6A} + C_7 = C_{6A} \vee C_7$$

In order to build multipliers of larger size, we must use the cumulative addition method as shown in the next project.

### 10.8 THE CUMULATIVE-ADDITION MULTIPLIER

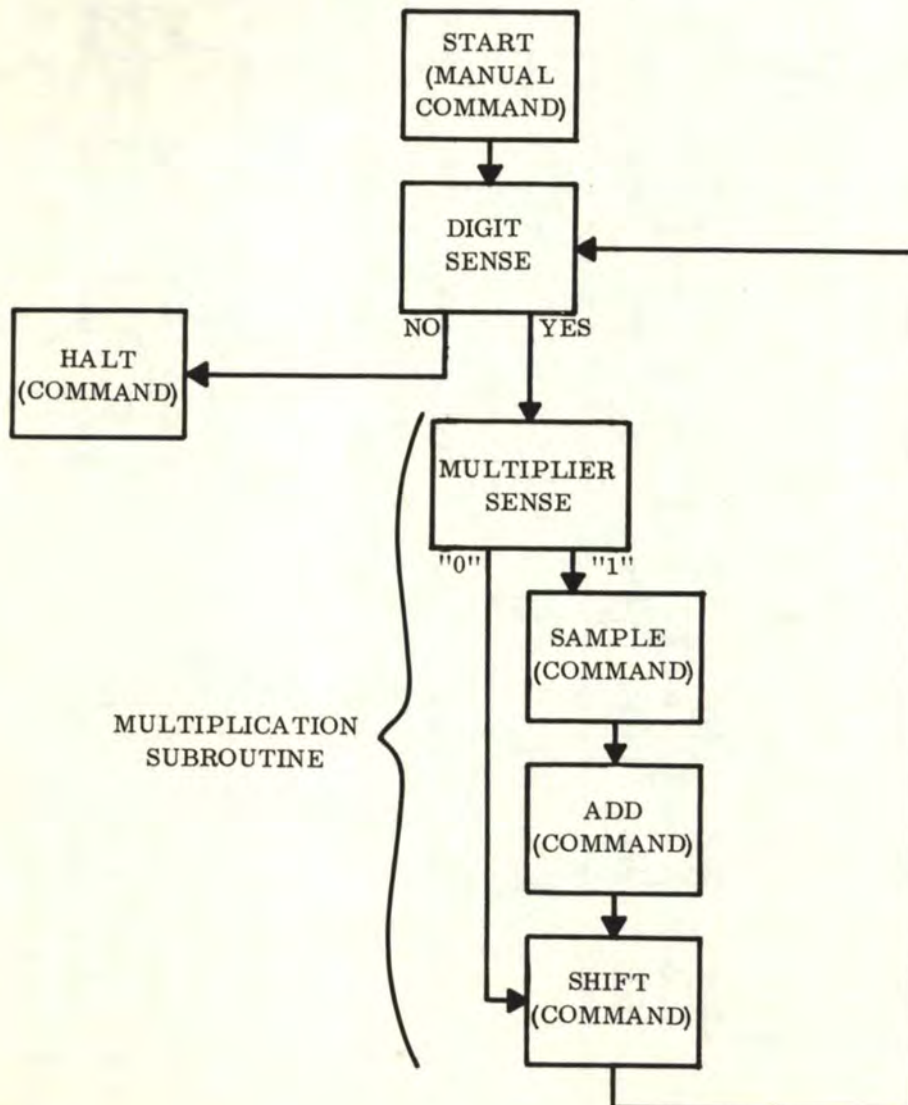
The cumulative-addition multiplier performs multiplication by successive addition rather than by full logic gating. Logic gates are used to control all the steps, but not for direct functional multiplication display. In effect, we will convert the non-gated adder described in section 9.6 into a multiplier by the addition of "sample-and-hold" logic. This project requires four registers. The 4-bit multiplier (with an 8-bit "product") will be described here. The logic diagram for this multiplier is as shown below.



## 10.8 THE CUMULATIVE-ADDITION MULTIPLIER (Continued)

The ACCUMULATOR (or PRODUCT) register is on top. This register is simply an 8-bit "DOWN" counter. The second register, the ADDEND register, is basically an 8-bit "UP" counter modified to accept "sampling" from the MULTIPLICAND register below for cumulative addition. The MULTIPLICAND register is a 7-bit left-shift register which will display a 4-bit number in four different positions (i. e. , the first or "set" position and three additional positions). The bottom MULTIPLIER register consists of a 4-bit right-shift register which will shift out a 4-bit number to the right. In addition, there is a "halt control" for the adder portion and a "halt control" for the multiplier portion. Only the extreme right digit will be "sensed" as this digit will control the successive addition process. Each time a binary number shifts into the extreme right position, it will be "sensed" by the logic circuitry. If it is a "1", then a "sample" and an addition will be performed, followed by a "shift" of both the MULTIPLIER and MULTIPLICAND registers to the next position. If the extreme right multiplier number is a "0", then there will be no "sample" and no "add"; but there will be a "shift" only in both the MULTIPLIER and MULTIPLICAND registers to the next position.

If we analyze this process by use of a "flow" diagram, we have the following processes:



## 10.8 THE CUMULATIVE-ADDITION MULTIPLIER (Continued)

The process begins with "START" which is accomplished by shorting together the two pins marked with arrows in the flip-flop in the MULTIPLICATION CONTROL. This starts the multiplication process in motion. The "digit sense" function is accomplished by connecting all the "true" outputs of the binary digits (bits) in the MULTIPLICAND and MULTIPLIER registers to an "OR" gate (or combination of "OR" gates). In other words, we must "OR" together all the "true" outputs mentioned above. In this configuration, when both registers "clear" (i. e., when the last number shifts out from either or both registers), the "OR" gate to the MULTIPLIER CONTROL will turn off and produce a "downswing" trigger which will stop the multiplication process. This "halt control" flip-flop will trigger precisely at the right time.

The SUBROUTINE is controlled by the 2-bit counter in the MULTIPLICATION CONTROL. This 2-bit counter is wired to cause three "AND" gates to turn on and off precisely at the right time to produce triggers which control the "SAMPLE", "SHIFT" and "ADD" commands. The "MULTIPLIER SENSE" is the "true" output of the last digit to the right in the MULTIPLIER register. This "MULTIPLIER SENSE" is wired into both the "AND" gates which control the "SAMPLE" and the "ADD". Thus, only when the "MULTIPLIER SENSE" is a "1", will a "SAMPLE" and "ADD" command be generated. There will always be a "SHIFT" command until the registers are "cleared". Note that there are two pulse generators--one in the ADD CONTROL and one in the MULTIPLICATION CONTROL. The ADD CONTROL pulse generator must be fast enough to complete the addition before another pulse is generated by the MULTIPLICATION CONTROL pulse generator. In the 4-bit multiplier, the ACCUMULATOR has 8 bits and therefore 256 "ADD" pulses must be generated before the next MULTIPLICATION pulse. If the ADD pulse is too slow, the final answer will be erroneous. After the multiplication process is complete, all registers will be "cleared" and the final answer will be present in the ACCUMULATOR. The ACCUMULATOR must be individually "cleared" digit by digit before entering the next two numbers in the MULTIPLIER and MULTIPLICAND registers.

## 10.9 THE DIVIDER

In division, we have four main numbers to consider. The first is the DIVIDEND, the number to be divided. The second is the DIVISOR, the number which the dividend is to be divided by. We will call these two numbers A and B, respectively. In other words, we have A divided by B ( $A \div B$ ). The QUOTIENT, or third number, is the answer to the problem  $A \div B$ . It will be represented by Q. The fourth number is an intermediate number which will be called the SUBTRAHEND for reference purposes (since we will be performing successive subtraction of the DIVISOR from the DIVIDEND). This number is represented by S. This intermediate number will be successively subtracted from the dividend to yield the quotient. The SUBTRAHEND number is dependent on the divisor, and is a "sampling" of the divisor (or its complement) which occurs in successive "shifted" positions to the right. Also, if and when the complement is subtracted, the number "1" (last digit on the right) must be further "sampled" and subtracted to complete the process of "addition by subtraction of complement" which will be described later.

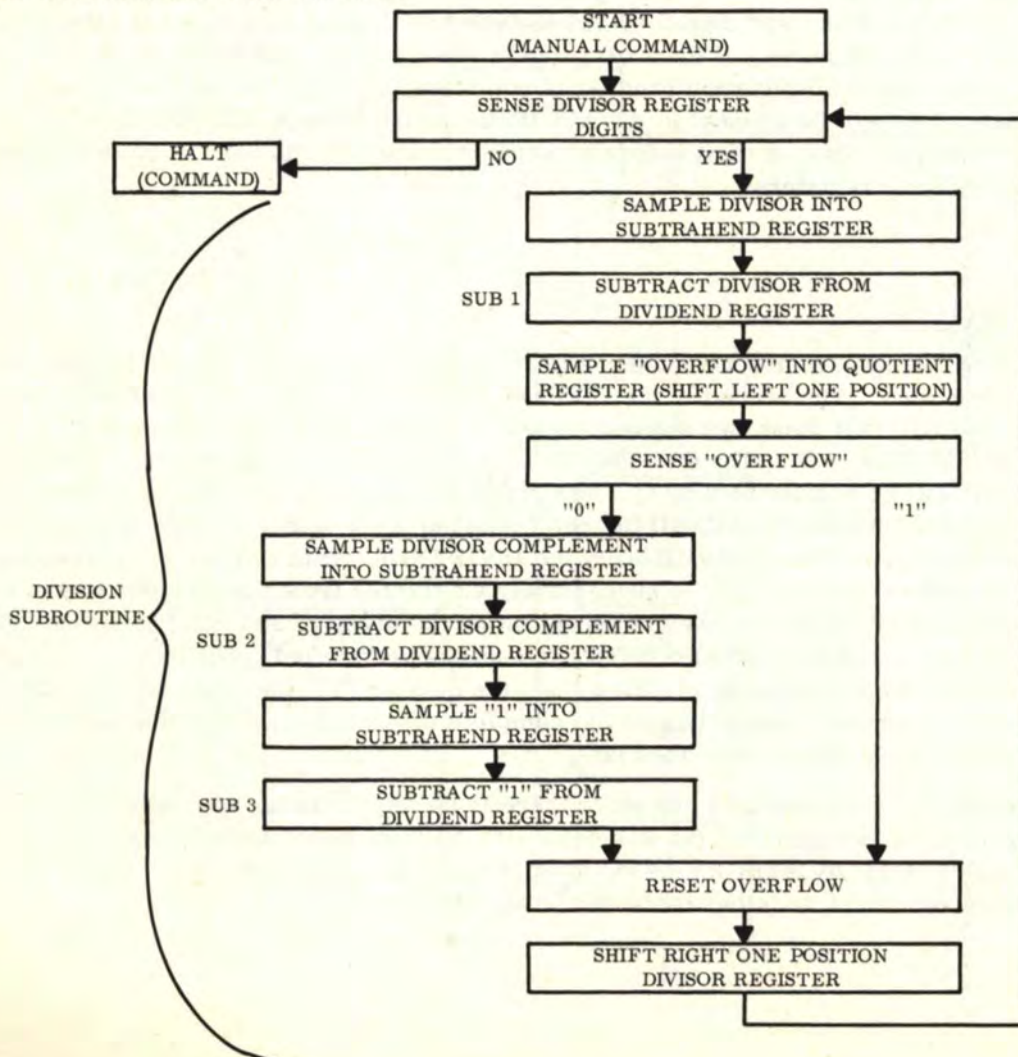
Division is a complicated process since there is no way of using only logic circuits to display a quotient of two numbers (as was done with the "full logic" multiplier). The primary reason for this is that DIVISION BY ZERO IS IMPOSSIBLE. Therefore, no "truth table" can be drawn up to represent division. Note the attempt as follows:

## 10.9 THE DIVIDER (Continued)

A	B	$A \div B$
0	0	(INDETERMINATE)
0	1	0
1	0	(INDETERMINATE)
1	1	1

Note that in the above example, two of the cases are indeterminate. Therefore, the "truth table" method is out. However, division can be accomplished by successive subtraction of the DIVISOR from the DIVIDEND and by shifting the DIVISOR one position to the right when it exceeds the DIVIDEND. An "OVERFLOW" sense is used to determine when the DIVIDEND is less than zero after subtraction. When this condition exists, we must add back the number which was subtracted and then shift the DIVISOR one position to the right. Since numbers can only be subtracted from the dividend, we must perform the equivalent of addition by first subtracting the divisor complement, and then by further subtracting the number "1" (last digit on the right). The QUOTIENT is entered in digit-by-digit and shifted left one position at a time until the process has been completed.

In order to convert the above processes to a computer function, the following "flow chart" indicates what must be done.

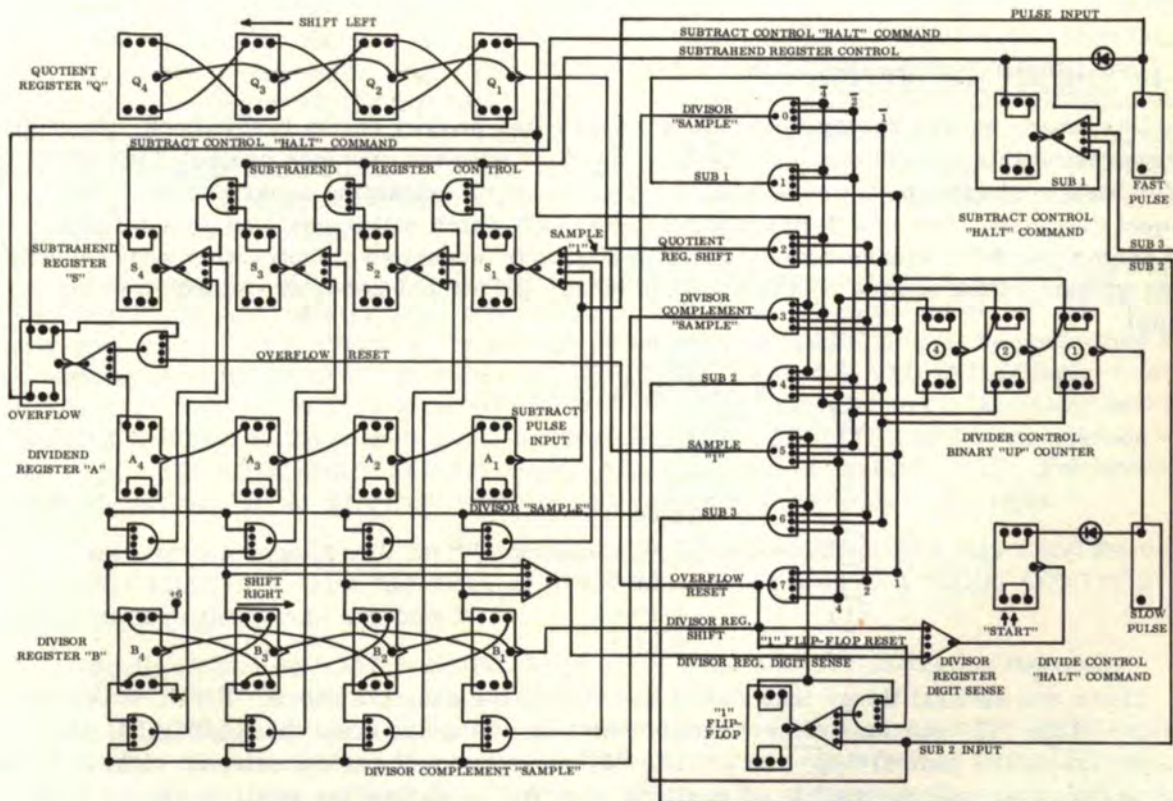


## 10.9 THE DIVIDER (Continued)

The process begins with "START" and branches out in the direction of the arrows. As long as the divisor contains at least one digit, the process will continue. The next "branch" occurs when the "overflow" is "0", indicating that the number subtracted from the dividend was too large. Therefore, we must initiate four additional steps to add back the number by "complement subtraction." If the "overflow" is "1", then the four steps above are not performed. The "overflow" is then reset, the divisor shifted right one position, and the entire process repeated until the divisor is depleted of all its "1" digits.

In planning these functions with flip-flops and computer logic, we make use of four registers: the DIVIDEND REGISTER "A", the DIVISOR REGISTER "B", the SUBTRAHEND REGISTER "S", and the QUOTIENT REGISTER "Q". The controls will consist of a SUBTRACT CONTROL "HALT" COMMAND with a fast pulse, a DIVIDE CONTROL "HALT" COMMAND with a slow pulse, a DIVIDE CONTROL 3-BIT BINARY "UP" COUNTER to control the "commands", and a "1" flip-flop to control the addition of the extra "1" in the extreme right digit to compensate for "complement subtraction" addition.

The following logic diagram indicates the set-up for a 4-BIT DIVIDER. In order to increase or decrease the number of binary digits (bits), add or remove the appropriate number of flip-flops in the center position and adjust wiring and gating accordingly.



### 10.9 THE DIVIDER (Continued)

As in the case of the "cumulative addition" multiplier, logic gates are used to control all the individual commands. However, since there are more commands involved, we must use a 3-bit "up" counter (instead of a 2-bit "up" counter as in the multiplier) to accomplish these additional steps. The preceding logic diagram uses two "SAMPLE-AND-HOLD" functions (to "sample" the divisor, or its complement as necessary, into the SUBTRAHEND register), an "OVERFLOW" sense, a "DIVIDE CONTROL" binary "up" counter, two "HALT" COMMANDS with their respective fast and slow pulses, and the "1" flip-flop. Note: the fast pulse must complete the subtraction before the next slow pulse triggers the control. Other functions designed into this divider are: DIVIDEND DIGIT SENSE LOGIC and SUBTRAHEND REGISTER CONTROL LOGIC.

After the project is wired up, division is accomplished as follows: Enter the DIVIDEND number in register A and the DIVISOR in register B. Both numbers should be entered FLUSH LEFT. Be sure that all other flip-flops have been reset. Then short together the two pins indicated by the arrows in the DIVIDE CONTROL "HALT" COMMAND. The division of the two numbers will be performed automatically and the process will stop after division is completed. If division is attempted by "0", then the process will automatically stop after the first complete cycle and a single "1" will show up in the extreme right in the QUOTIENT register. If two proper numbers have been divided, the answer will appear FLUSH LEFT in the QUOTIENT register and will contain four bits. For instance,  $1000 \div 1100$  will show up as 0101 in the QUOTIENT register. The binary point should be placed accordingly. In this example,  $1000 \div 1100 = .0101$ . If we had  $1000 \div 11.00$ , the answer would be 010.1.  $1000 \div 1.100 = 0101$ .

### 10.10 THE SQUARE ROOTER

The binary square rooter is the most complicated project shown in this book. In order to understand this project one must be fully familiar with the previous project: THE DIVIDER. Refer back to chapter 6, if necessary, for discussion on extracting square roots. The project described here is a 4-BIT SQUARE ROOTER which will operate on an 8-bit number and extract its 4-bit square root. Let us illustrate by successive subtraction what happens when we extract the square root of 3 (11) in binary (additional zeros are added on to make 8 bits).

$$\begin{array}{r}
 \textcircled{1} \quad \sqrt{11 \overset{00}{\wedge} \overset{00}{\wedge} \overset{00}{\wedge} 00} \\
 \quad \quad \underline{-01} \\
 \quad \quad 10 \ 00 \\
 \textcircled{1} \quad \quad \underline{-1 \ 01} \\
 \quad \quad \quad 11 \ 00 \\
 \textcircled{0} \quad \quad \quad \underline{-11 \ 01} \\
 \quad \quad \quad \quad 11 \ 00 \ 00 \\
 \textcircled{1} \quad \quad \quad \quad \underline{-1 \ 10 \ 01} \\
 \quad \quad \quad \quad \quad 1 \ 01 \ 11
 \end{array}$$

There are several things that should be noted in the example above. First, notice the position of the "01" portions of each number that is subtracted from the RADICAND (the number inside the radical " $\sqrt{\quad}$ "). The "01" always occurs at the extreme right and, in each successive "subtraction" level position, the "01" is shifted two positions to the right. We start the subtraction process with the "01" at the first level. If we can subtract, we enter a "1" (circled) to the left and perform the subtraction as shown above. Then we "bring down" the next two digits to the right. Next we shift the "01" twice to the right. Now, immediately to the left of the "01" we enter the circled number above which gives us "101". In this case, we can still subtract this "101" from the remainder "1000" (left after the first subtraction).



10.10 THE SQUARE ROOTER (Continued)

Therefore we enter a second "1" to the left of "101", perform the second subtraction, and "bring down" the next two digits. Again we shift "01" twice to the right. We now "tack on" to the left the two circled numbers above to obtain "1101". This time we can not subtract "1101" from the remainder "1100". Therefore we enter a "0" to the left of "1101" and circle it. Now we "bring down" the last two digits and again shift "01" twice to the right. We "tack on" to the left the three circled numbers above to yield "11001". Now we subtract "11001" from the remainder "110000" and enter a "1" to the left of "11001" and circle it. The circled numbers "1101" represent the 4-bit square root of the 8-bit number "11000000". Now, let us observe the positions of the numbers that were subtracted.

```

      01
     101
    1101
   11001

```

As can be seen, the extreme right "1" has shifted twice as we progress to each successive level downward. Note also that the extreme left digit shifts one position to the right as we progress to each successive level downward. Also, all other digits except the "01" represent the square root digits calculated for all the levels above by successive subtraction (or by attempted subtraction).

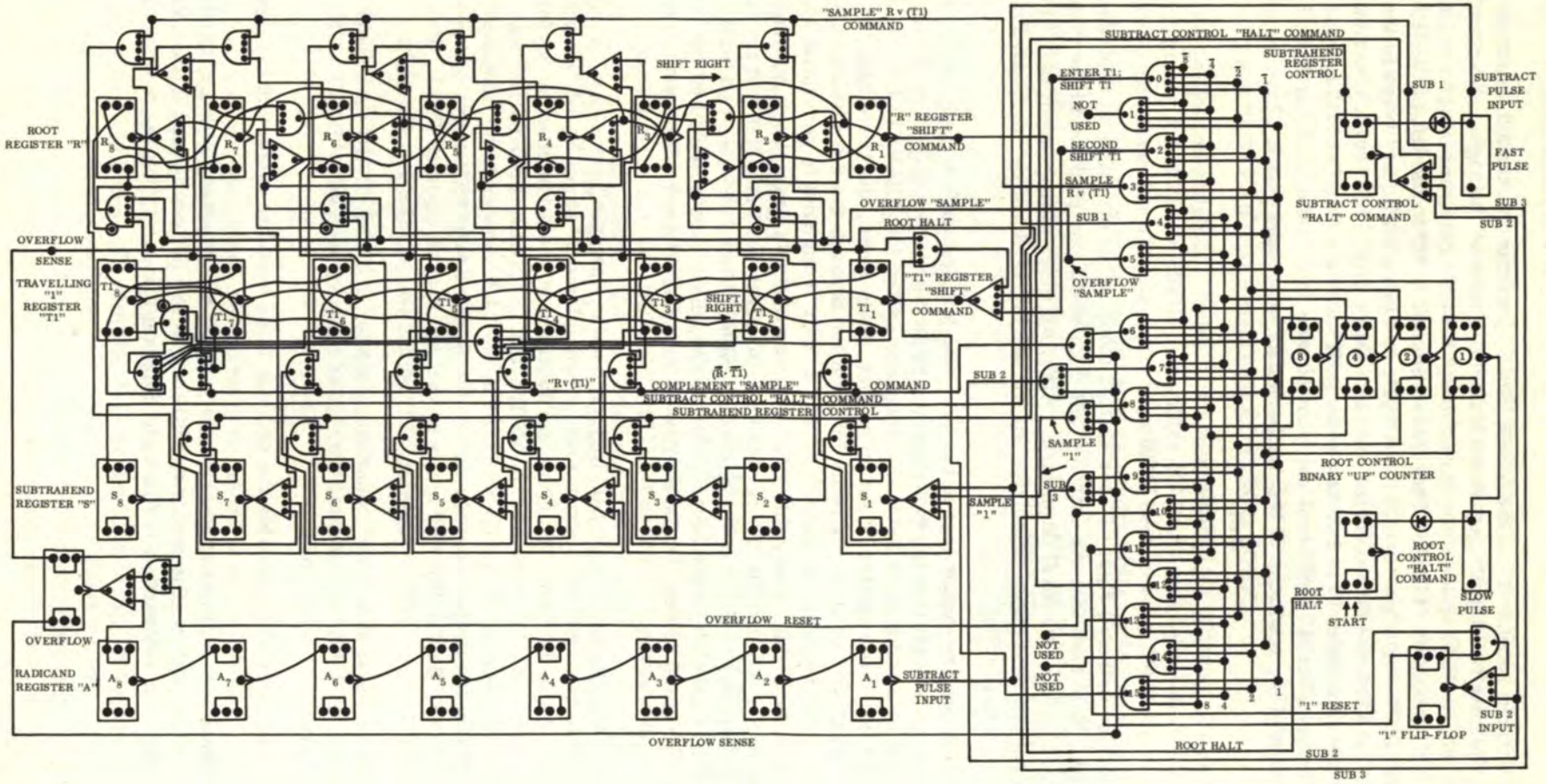
The key to the square rooter lies in the "01" which keeps shifting twice to the right before a successive subtraction is performed (or attempted). We make use of this travelling "01" as a control by setting up an 8-bit shift register called the TRAVELLING "1" ("T1") REGISTER. In fact, this register is set up so that, from reset, the travelling "1" will automatically be entered in at the extreme left, shifted through the 8 positions, then shifted out at the right. The register is "gated" so that a "halt" command is generated when the "1" disappears to the right. Also, this "T1" register controls "sampling" into the root register. The ROOT REGISTER "R" is also an 8-bit right-shift register and the calculated square root will appear in the last 4 digits to the right. There are also two more registers needed: the SUBTRAHEND REGISTER "S" for intermediate subtraction, and the RADICAND REGISTER "A" in which is entered the number to be square rooted. Both registers "S" and "A" are 8-bit "up" counters.

In addition, "SAMPLE-AND-HOLD" logic is used to "sample" registers "R" and "T1" simultaneously into register "S". Simultaneous "complement" sampling of "R" and "T1" is also performed. As in division, "complement subtraction" addition must be performed when necessary, and the extra "1" must also be subtracted. "SAMPLE-AND-HOLD" logic controlled by the "T1" register will allow the OVERFLOW in the "A" register to be "sampled" into the ROOT REGISTER in the proper position before the ROOT REGISTER shifts one position to the right. The ROOT REGISTER "R" shifts at half the rate of register "T1". For every shift of register "R" to the right, register "T1" shifts two positions to the right.

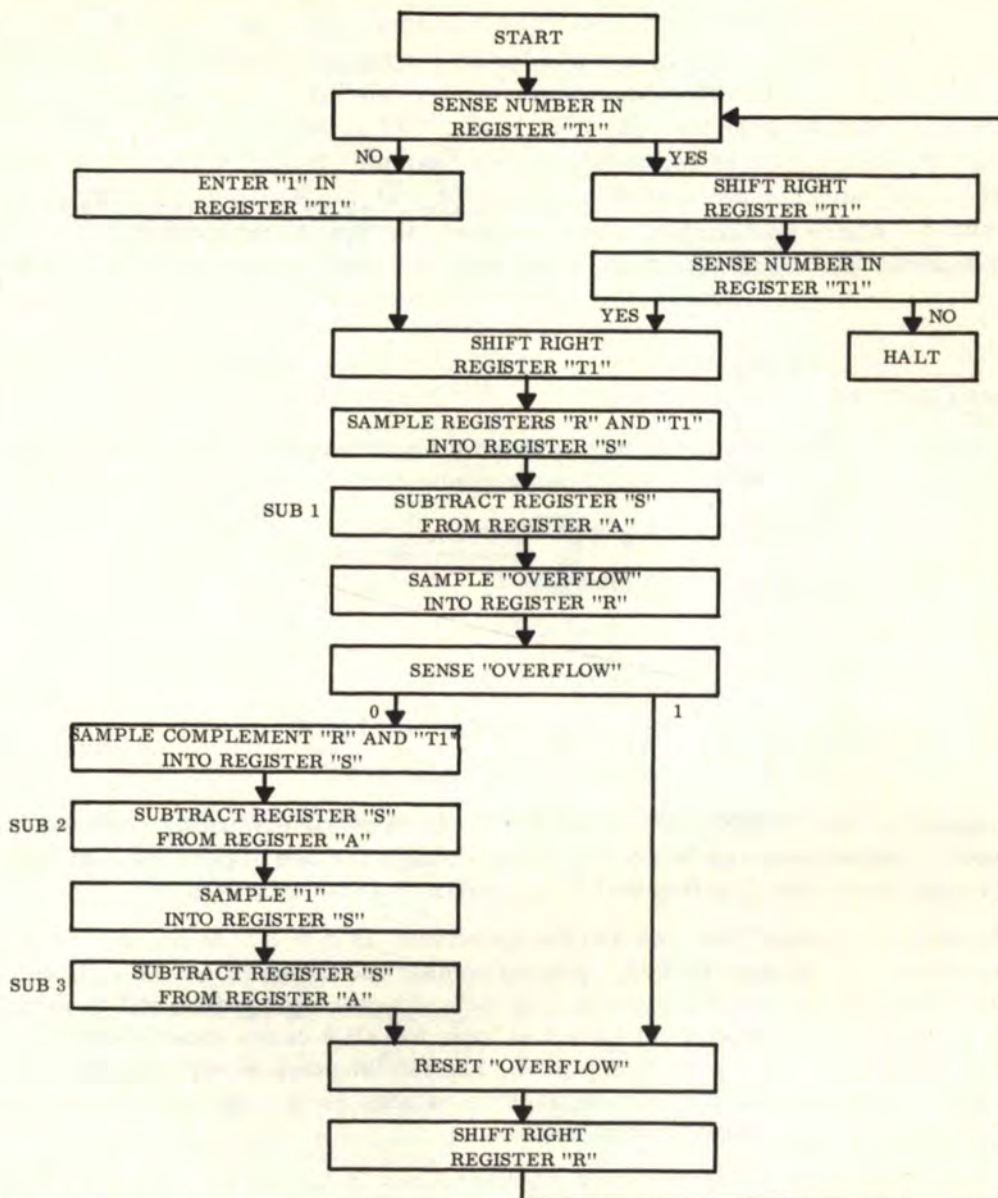
The controls consist of 2 "halt" commands (one for SUBTRACTION HALT and the other for ROOT HALT). A 4-bit "up" counter with gated logic is used as a "ROOT CONTROL" to generate the square root commands.

The flow chart on p.107 indicates the process to be used for extracting the square root.

Note that three subtractions are performed. They are labelled "SUB 1", "SUB 2" and "SUB 3", respectively. The last two subtractions are not performed if the OVERFLOW is a "1" after the first subtraction. The logic diagram for extracting the square root is as follows:



## 10.10 THE SQUARE ROOTER (Continued)



The ROOT CONTROL counter generates 16 commands, but only 13 are used. Commands "1", "13", and "14" should not be used, but the "AND" gates are shown for reference only. Command "1" must not be used so that commands "0" and "2" can properly double-shift the "T1" register. After the logic is connected up per the diagram, the root is extracted as follows: First, enter in the number to be rooted in register "A". The binary point reference is after every second binary digit in register "A". To enter  $\sqrt{1}$ , enter "1" in position  $A_7$ . For  $\sqrt{11}$ , enter "1" in  $A_8$  and  $A_7$ . For  $\sqrt{101.1101}$ , enter "1" in  $A_7$ ,  $A_5$ ,  $A_4$ ,  $A_3$ , and  $A_1$ . For  $\sqrt{10101001}$ , enter "1" in  $A_8$ ,  $A_6$ ,  $A_4$ , and  $A_1$ . The binary point is not calculated and must be mentally placed after the operation. All other registers must be reset before the square root can be extracted. Short together the proper pins (indicated by arrows) in the ROOT CONTROL "HALT" COMMAND. The rooting process will automatically start and stop when the rooting is finished. The square root then appears in the last four positions to the right ( $R_1$ ,  $R_2$ ,  $R_3$ , and  $R_4$ ) in the ROOT REGISTER.

The "FAST PULSE" in the SUBTRACT CONTROL must be fast enough to complete a subtraction by counting (256 pulses minimum) before the next slow pulse is generated in the ROOT CONTROL "SLOW PULSE". Also, note that in the "sampling" process, not all

10.10 THE SQUARE ROOTER (Continued)

positions need be "sampled" in the "R" and "T1" registers. Therefore, bits  $R_1$ ,  $R_3$ ,  $R_5$ , and  $R_7$  receive no "sample". Bits  $S_2$  and  $S_8$  also do not receive "sampling". In "sampling" the two registers "R" and "T1" simultaneously, we "sample" the equivalent of "R" v "T1" [that is, we "OR" each bit position:  $(R_1 \vee T1_1)$ ,  $(R_2 \vee T1_2)$ ,  $(R_3 \vee T1_3)$ ,  $(R_4 \vee T1_4)$ , etc.]. In "sampling" the complement of two registers, we "sample" the equivalent of " $\overline{R}$ ". "T1" [that is, we "AND" each complement bit position:  $(\overline{R_1} \cdot \overline{T1_1})$ ,  $(\overline{R_2} \cdot \overline{T1_2})$ ,  $(\overline{R_3} \cdot \overline{T1_3})$ ,  $(\overline{R_4} \cdot \overline{T1_4})$ , etc.]. Where no "sample" was necessary, the operation was simplified. The reader will discover that there are many, many logic concepts in this square rooter project.

10.11 COMPARATORS

A comparator performs a comparison between the numbers entered in two registers (which we will call "A" and "B"). The six comparisons that can be made are as follows:

1.  $A > B$  (A greater than B)
2.  $A < B$  (A less than B)
3.  $A = B$  (A equal to B)
4.  $A \neq B$  (A unequal to B)
5.  $A \geq B$  (A greater than or equal to B)
6.  $A \leq B$  (A less than or equal to B)

The comparator is a SENSE function and no calculations (or computing processes) need be performed. Comparisons can be accomplished through the use of pure logic gating and the result is read out on one flip-flop called the COMPARATOR READOUT.

In performing the comparison, we ask the questions: Is  $A > B$ ? Is  $A < B$ ? Is  $A = B$ ? etc. If the answer to a question is YES, then the comparator readout is a "1". If the answer is NO, then the comparator readout is a "0". Proper logic gating will be developed through use of truth tables. The development of logic for all 6 cases above should be fairly easy to follow because the reasoning in the English language is very similar to the developed logic. However, the larger comparators require many, many "AND" and "OR" gates, as can be seen in the following explanations.

Let us consider two single digit binary numbers A and B (digits may be either "0"s or "1"s). We can now set up a truth table for the 6 comparison functions as follows (this will be referred to as the main truth table):

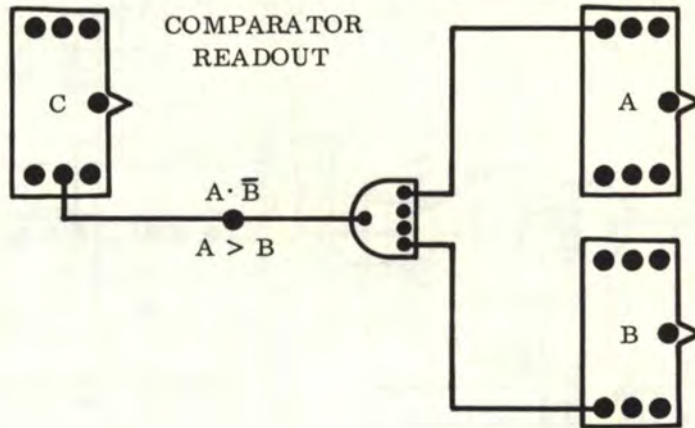
A	B	$A > B$	$A < B$	$A = B$	$A \neq B$	$A \geq B$	$A \leq B$
0	0	0	0	1	0	1	1
0	1	0	1	0	1	0	1
1	0	1	0	0	1	1	0
1	1	0	0	1	0	1	1
A	B	$A \cdot \overline{B}$	$\overline{A} \cdot B$	$(A \cdot B) \vee (\overline{A} \cdot \overline{B})$ $(A \vee \overline{B}) \cdot (\overline{A} \vee B)$	$(A \cdot \overline{B}) \vee (\overline{A} \cdot B)$ $(A \vee B) \cdot (\overline{A} \vee \overline{B})$	$A \vee \overline{B}$	$\overline{A} \vee B$

10.11 COMPARATORS (Continued)

The logic functions in the bottom row of the table have the exact same truth tables as the comparisons shown in the top row. Therefore the bottom row represents the equivalent "AND" and "OR" logic expressions for the comparisons and may be directly substituted in any logic equation.

10.11.1 THE "GREATER THAN" COMPARATOR

For single digit binary numbers A and B, we have already pointed out in the main truth table that  $(A > B) = (A \cdot \bar{B})$ . The logic diagram is as follows:



For 2-bit binary numbers A and B, we represent the digit positions as follows:

$$\begin{matrix} A_2 A_1 \\ B_2 B_1 \end{matrix}$$

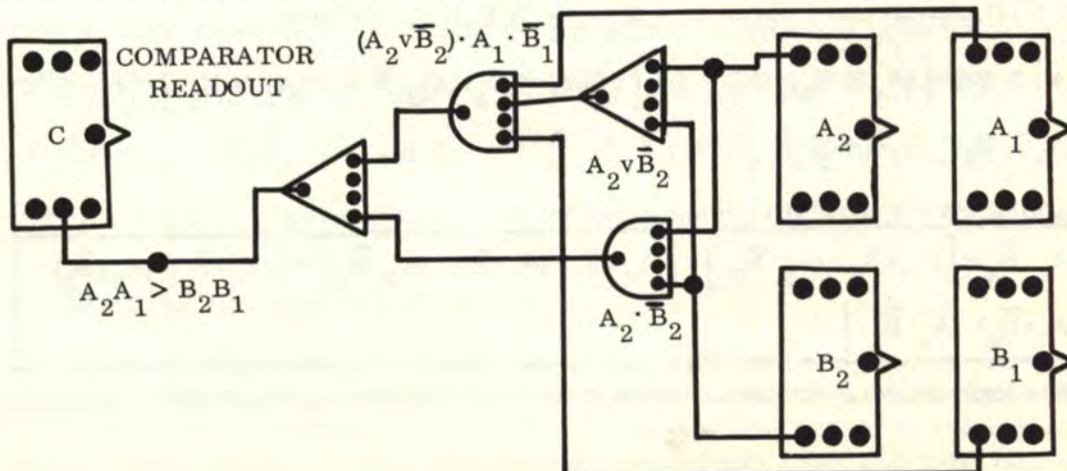
Now, the number  $A_2 A_1$  is greater than  $B_2 B_1$  ( $A_2 A_1 > B_2 B_1$ ) when  $(A_2 > B_2)$  or when  $(A_2 \geq B_2)$  and  $(A_1 > B_1)$ . This can be written logically as follows:

$$C = (A_2 > B_2) \vee [(A_2 \geq B_2) \cdot (A_1 > B_1)]$$

The letter "C" denotes "COMPARISON". We must now refer back to the main truth table and substitute "AND" and "OR" functions for  $(A_1 > B_1)$ ,  $(A_2 \geq B_2)$ , and  $(A_2 > B_2)$ . Substituting, we have:

$$C = (A_2 \cdot \bar{B}_2) \vee [(A_2 \vee \bar{B}_2) \cdot (A_1 \cdot \bar{B}_1)]$$

The logic diagram for the above expression for the 2-bit comparator is as follows:



## 10.11.1 THE "GREATER THAN" COMPARATOR (Continued)

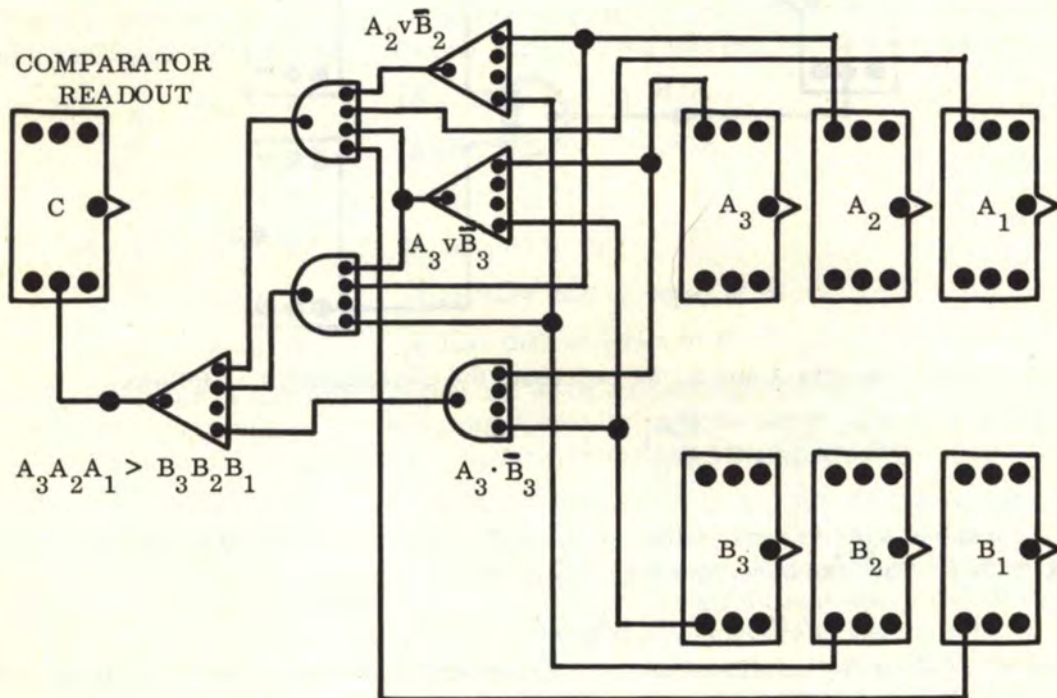
For a 3-bit comparator, where  $A_3A_2A_1 > B_3B_2B_1$ , this condition exists when  $(A_3 > B_3)$  or when  $(A_3 \geq B_3)$  and  $(A_2 > B_2)$  or when  $(A_3 \geq B_3)$  and  $(A_2 \geq B_2)$  and  $(A_1 > B_1)$ . This can be written logically as follows:

$$C = (A_3 > B_3) \vee [(A_3 \geq B_3) \cdot (A_2 > B_2)] \vee [(A_3 \geq B_3) \cdot (A_2 \geq B_2) \cdot (A_1 > B_1)]$$

Substituting "AND" and "OR" comparison equivalents, we have:

$$C = (A_3 \cdot \bar{B}_3) \vee [(A_3 \vee \bar{B}_3) \cdot (A_2 \cdot \bar{B}_2)] \vee [(A_3 \vee \bar{B}_3) \cdot (A_2 \vee \bar{B}_2) \cdot (A_1 \cdot \bar{B}_1)]$$

The logic diagram for the above expression for the 3-bit comparator is as follows:



For a 4-bit comparator, where  $A_4A_3A_2A_1 > B_4B_3B_2B_1$  we have:

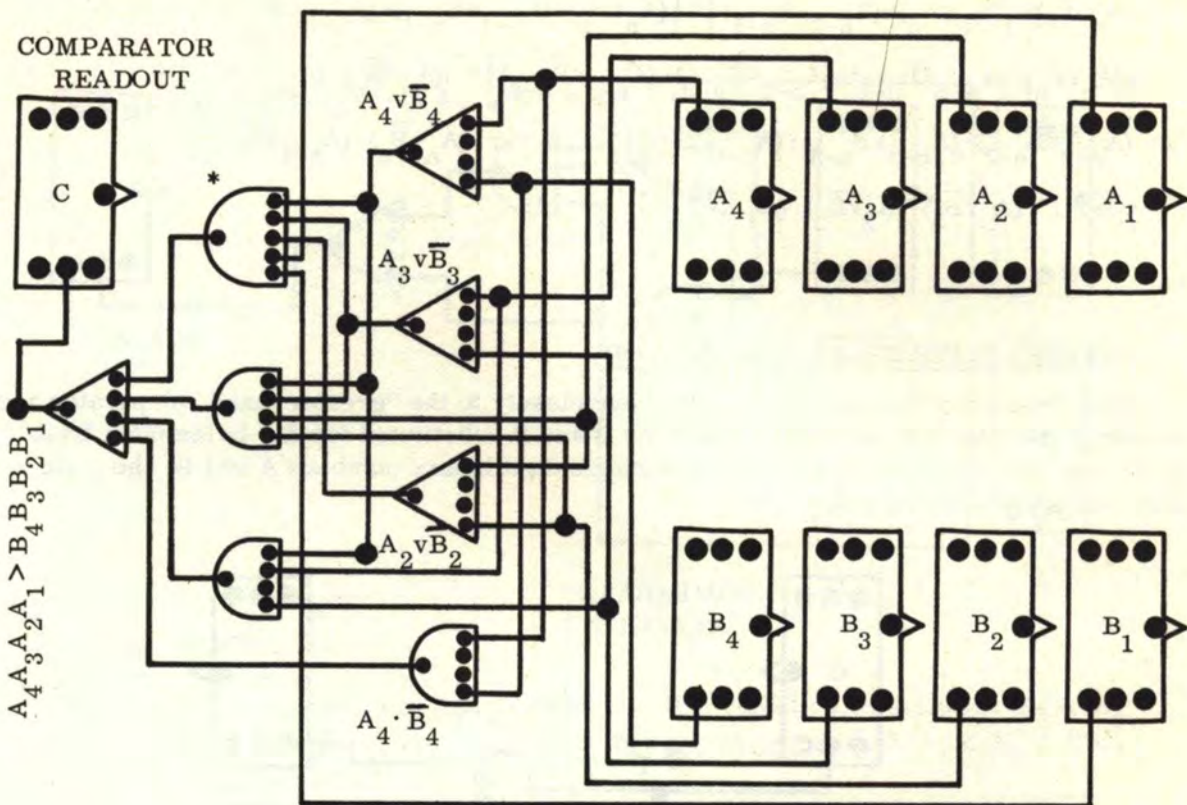
$$C = (A_4 > B_4) \vee [(A_4 \geq B_4) \cdot (A_3 > B_3)] \vee [(A_4 \geq B_4) \cdot (A_3 \geq B_3) \cdot (A_2 > B_2)] \vee [(A_4 \geq B_4) \cdot (A_3 \geq B_3) \cdot (A_2 \geq B_2) \cdot (A_1 > B_1)]$$

Substituting "AND" and "OR" comparison equivalents, we have:

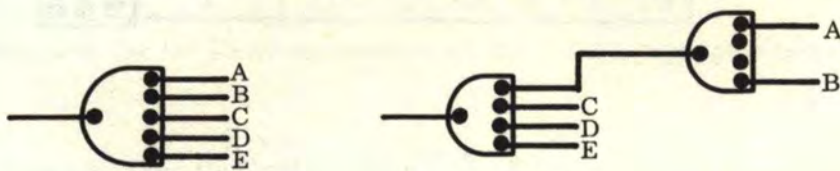
$$C = (A_4 \cdot \bar{B}_4) \vee [(A_4 \vee \bar{B}_4) \cdot (A_3 \cdot \bar{B}_3)] \vee [(A_4 \vee \bar{B}_4) \cdot (A_3 \vee \bar{B}_3) \cdot (A_2 \cdot \bar{B}_2)] \vee [(A_4 \vee \bar{B}_4) \cdot (A_3 \vee \bar{B}_3) \cdot (A_2 \vee \bar{B}_2) \cdot (A_1 \cdot \bar{B}_1)]$$

The logic diagram for the above expression for the 4-bit comparator is as follows:

10.11.1 THE "GREATER THAN" COMPARATOR (Continued)



\*Note: 5 or more inputs may be accomplished by "expanding" a 4-input gate (i. e., by using more than one gate) as follows:



5 INPUTS

5 INPUTS

The above example also applies to "OR" gates.

The general logic equation for an "n"-bit "greater than" comparator is as follows:

$$\begin{aligned}
 C = & (A_n > B_n) \vee [(A_n \geq B_n) \cdot (A_{n-1} > B_{n-1})] \vee [(A_n \geq B_n) \cdot (A_{n-1} \geq B_{n-1}) \cdot (A_{n-2} > B_{n-2})] \vee \\
 & [(A_n \geq B_n) \cdot (A_{n-1} \geq B_{n-1}) \cdot (A_{n-2} \geq B_{n-2}) \cdot (A_{n-3} > B_{n-3})] \vee [(A_n \geq B_n) \cdot (A_{n-1} \geq B_{n-1}) \cdot \\
 & (A_{n-2} \geq B_{n-2}) \cdot (A_{n-3} \geq B_{n-3}) \cdot (A_{n-4} > B_{n-4})] \vee \dots \vee [(A_n \geq B_n) \cdot (A_{n-1} \geq B_{n-1}) \cdot \\
 & \dots \cdot (A_3 \geq B_3) \cdot (A_2 \geq B_2) \cdot (A_1 > B_1)]
 \end{aligned}$$

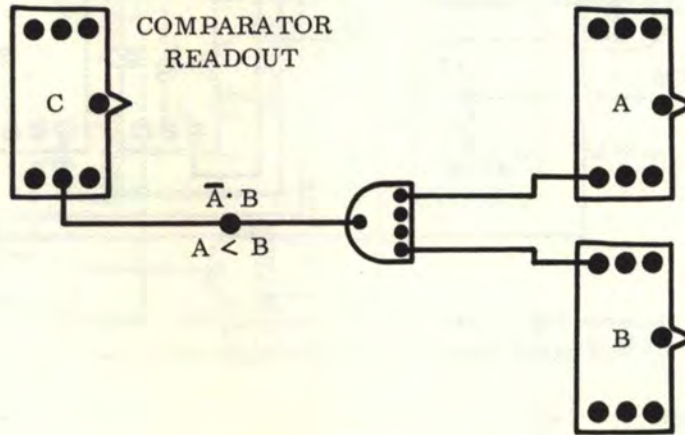
Note that the last term within each set of brackets is ">" while all other terms are "≥". Also, each successive set of brackets contains one more term than the previous set. The very first term in the expression is always  $A_n > B_n$ . The "terms" are those expressions within each set of parentheses. After substituting the equivalent "AND" and "OR" functions, the general equation becomes as follows:

10. 11. 1 THE "GREATER THAN" COMPARATOR (Continued)

$$C = (A_n \cdot \bar{B}_n) \vee [(A_n \vee \bar{B}_n) \cdot (A_{n-1} \cdot \bar{B}_{n-1})] \vee [(A_n \vee \bar{B}_n) \cdot (A_{n-1} \vee \bar{B}_{n-1}) \cdot (A_{n-2} \cdot \bar{B}_{n-2})] \vee [(A_n \vee \bar{B}_n) \cdot (A_{n-1} \vee \bar{B}_{n-1}) \cdot (A_{n-2} \vee \bar{B}_{n-2}) \cdot (A_{n-3} \cdot \bar{B}_{n-3})] \vee [(A_n \vee \bar{B}_n) \cdot (A_{n-1} \vee \bar{B}_{n-1}) \cdot (A_{n-2} \vee \bar{B}_{n-2}) \cdot (A_{n-3} \vee \bar{B}_{n-3}) \cdot (A_{n-4} \cdot \bar{B}_{n-4})] \vee \dots \vee [(A_n \vee \bar{B}_n) \cdot (A_{n-1} \vee \bar{B}_{n-1}) \cdot \dots \cdot (A_3 \vee \bar{B}_3) \cdot (A_2 \vee \bar{B}_2) \cdot (A_1 \cdot \bar{B}_1)]$$

10. 11. 2 THE "LESS THAN" COMPARATOR

The "less than" comparator follows very closely to the "greater than" comparator and would be identical if A were substituted for B and B substituted for A. In the main truth table, we see that  $(A < B) = (\bar{A} \cdot B)$ . For single digit binary numbers A and B, the logic diagram is as follows:



For 2-bit binary numbers A and B, we represent the digit positions as follows:

$$A_2 A_1$$

$$B_2 B_1$$

Now, the number  $A_2 A_1$  is less than  $B_2 B_1$  ( $A_2 A_1 < B_2 B_1$ ) when  $(A_2 < B_2)$  or when  $(A_2 \leq B_2)$  and  $(A_1 < B_1)$ . This can be written logically as follows:

$$C = (A_2 < B_2) \vee [(A_2 \leq B_2) \cdot (A_1 < B_1)]$$

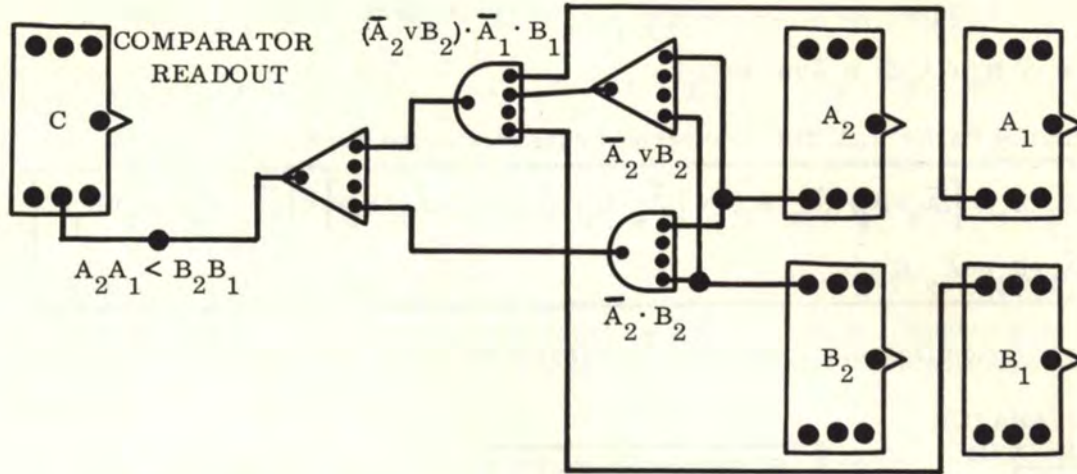
The letter "C" denotes "COMPARISON". We must now refer back to the main truth table and substitute "AND" and "OR" functions for  $(A_1 < B_1)$ ,  $(A_2 \leq B_2)$ , and  $(A_2 < B_2)$ . Substituting, we have:

$$C = (\bar{A}_2 \cdot B_2) \vee [(A_2 \vee B_2) \cdot (\bar{A}_1 \cdot B_1)]$$

The logic diagram for the above expression for the 2-bit comparator is as follows:



10.11.2 THE "LESS THAN" COMPARATOR (Continued)



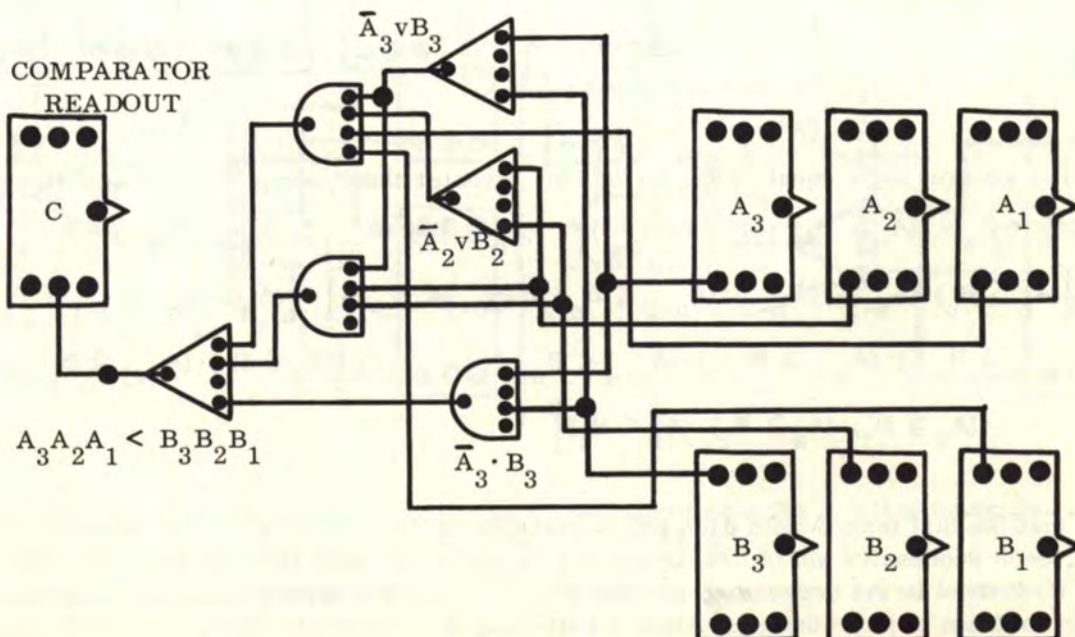
For a 3-bit comparator, where  $A_3A_2A_1 < B_3B_2B_1$ , this condition exists when  $(A_3 < B_3)$  or when  $(A_3 \leq B_3)$  and  $(A_2 < B_2)$  or when  $(A_3 \leq B_3)$  and  $(A_2 \leq B_2)$  and  $(A_1 < B_1)$ . This can be written logically as follows:

$$C = (A_3 < B_3) \vee [(A_3 \leq B_3) \cdot (A_2 < B_2)] \vee [(A_3 \leq B_3) \cdot (A_2 \leq B_2) \cdot (A_1 < B_1)]$$

Substituting "AND" and "OR" comparison equivalents, we have:

$$C = (\bar{A}_3 \cdot B_3) \vee [(\bar{A}_3 \vee B_3) \cdot (\bar{A}_2 \cdot B_2)] \vee [(\bar{A}_3 \vee B_3) \cdot (\bar{A}_2 \vee B_2) \cdot (\bar{A}_1 \cdot B_1)]$$

The logic diagram for the above expression for the 3-bit comparator is as follows:



## 10.11.2 THE "LESS THAN" COMPARATOR (Continued)

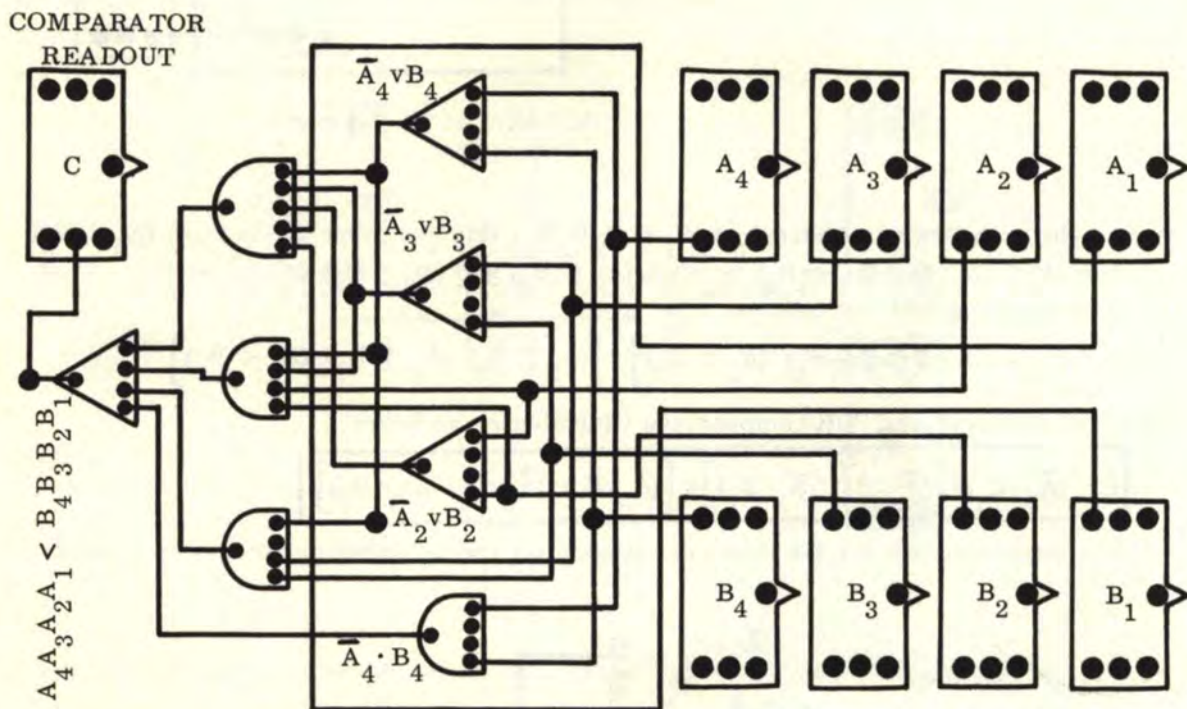
For a 4-bit comparator, where  $A_4 A_3 A_2 A_1 < B_4 B_3 B_2 B_1$  we have:

$$C = (A_4 < B_4) \vee \left[ (A_4 \leq B_4) \cdot (A_3 < B_3) \right] \vee \left[ (A_4 \leq B_4) \cdot (A_3 \leq B_3) \cdot (A_2 < B_2) \right] \vee \left[ (A_4 \leq B_4) \cdot (A_3 \leq B_3) \cdot (A_2 \leq B_2) \cdot (A_1 < B_1) \right]$$

Substituting "AND" and "OR" comparison equivalents, we have:

$$C = (\bar{A}_4 \cdot B_4) \vee \left[ (\bar{A}_4 \vee B_4) \cdot (\bar{A}_3 \cdot B_3) \right] \vee \left[ (\bar{A}_4 \vee B_4) \cdot (\bar{A}_3 \vee B_3) \cdot (\bar{A}_2 \cdot B_2) \right] \vee \left[ (\bar{A}_4 \vee B_4) \cdot (\bar{A}_3 \vee B_3) \cdot (\bar{A}_2 \vee B_2) \cdot (\bar{A}_1 \cdot B_1) \right]$$

The logic diagram for the above expression for the 4-bit comparator is as follows:



The general logic equation for an "n"-bit "greater than" comparator is as follows:

$$C = (A_n < B_n) \vee \left[ (A_n \leq B_n) \cdot (A_{n-1} < B_{n-1}) \right] \vee \left[ (A_n \leq B_n) \cdot (A_{n-1} \leq B_{n-1}) \cdot (A_{n-2} < B_{n-2}) \right] \vee \left[ (A_n \leq B_n) \cdot (A_{n-1} \leq B_{n-1}) \cdot (A_{n-2} \leq B_{n-2}) \cdot (A_{n-3} < B_{n-3}) \right] \vee \left[ (A_n \leq B_n) \cdot (A_{n-1} \leq B_{n-1}) \cdot (A_{n-2} \leq B_{n-2}) \cdot (A_{n-3} \leq B_{n-3}) \cdot (A_{n-4} < B_{n-4}) \right] \vee \dots \vee \left[ (A_n \leq B_n) \cdot (A_{n-1} \leq B_{n-1}) \cdot (A_{n-2} \leq B_{n-2}) \cdot (A_{n-3} \leq B_{n-3}) \cdot (A_{n-4} \leq B_{n-4}) \cdot (A_{n-5} < B_{n-5}) \right] \vee \dots \vee \left[ (A_n \leq B_n) \cdot (A_{n-1} \leq B_{n-1}) \cdot (A_{n-2} \leq B_{n-2}) \cdot (A_{n-3} \leq B_{n-3}) \cdot (A_{n-4} \leq B_{n-4}) \cdot (A_{n-5} \leq B_{n-5}) \cdot (A_{n-6} < B_{n-6}) \right]$$

Note that the last term within each set of brackets is "<" while all other terms are "≤". Also, each successive set of brackets contains one more term than the previous set. The very first term in the expression is always  $A < B$ . The "terms" are those expressions within each set of parentheses. After substituting the equivalent "AND" and "OR" functions, the general equation becomes as follows:

## 10.11.3 THE "EQUAL TO" COMPARATOR (Continued)

$$A_2A_1$$

$$B_2B_1$$

Now, the number  $A_2A_1$  is equal to  $B_2B_1$  ( $A_2A_1 = B_2B_1$ ) only when  $A_2 = B_2$  and  $A_1 = B_1$ . This can be written logically as follows:

$$C = (A_2 = B_2) \cdot (A_1 = B_1)$$

The letter "C" denotes "COMPARISON". We can now substitute the expression  $(A \vee \bar{B}) \cdot (\bar{A} \vee B)$  for  $A = B$  ( $A_1 = B_1$  and  $A_2 = B_2$ ) as follows:

$$C = (A_2 = B_2) \cdot (A_1 = B_1)$$

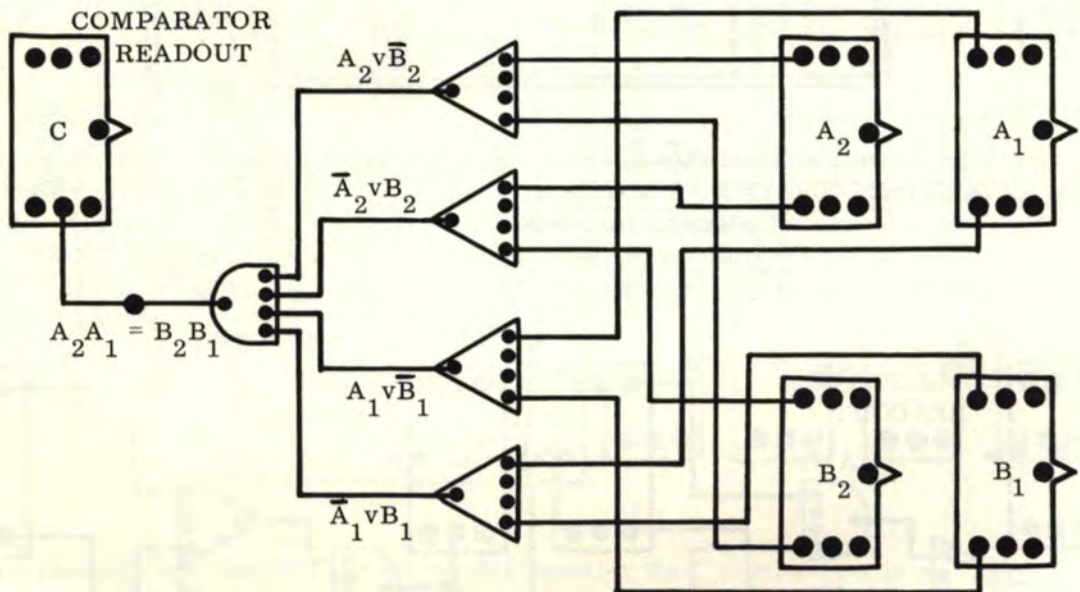
$$C = [(A_2 \vee \bar{B}_2) \cdot (\bar{A}_2 \vee B_2)] \cdot [(A_1 \vee \bar{B}_1) \cdot (\bar{A}_1 \vee B_1)]$$

$$C = (A_2 \vee \bar{B}_2) \cdot (\bar{A}_2 \vee B_2) \cdot (A_1 \vee \bar{B}_1) \cdot (\bar{A}_1 \vee B_1)$$

(ASSOCIATION)

(AL #1)

The logic diagram for the above expression for the 2-bit comparator is as follows:



For a 3-bit comparator, where  $A_3A_2A_1 = B_3B_2B_1$  we have:

$$A_3A_2A_1$$

$$B_3B_2B_1$$

Now, the number  $A_3A_2A_1$  is equal to  $B_3B_2B_1$  ( $A_3A_2A_1 = B_3B_2B_1$ ) only when  $A_3 = B_3$  and  $A_2 = B_2$  and  $A_1 = B_1$ . This can be written logically as follows:

$$C = (A_3 = B_3) \cdot (A_2 = B_2) \cdot (A_1 = B_1)$$

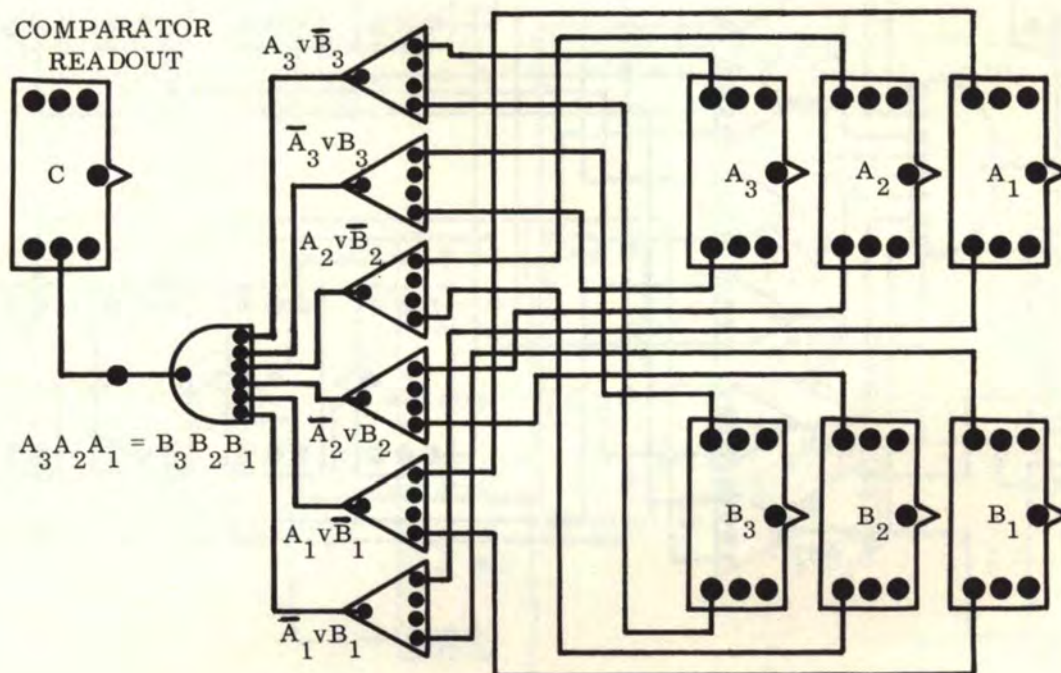
## 10.11.3 THE "EQUAL TO" COMPARATOR (Continued)

Substituting "AND" and "OR" functions, we have:

$$C = [(A_3 \bar{v} \bar{B}_3) \cdot (\bar{A}_3 v B_3)] \cdot [(A_2 \bar{v} \bar{B}_2) \cdot (\bar{A}_2 v B_2)] \cdot [(A_1 \bar{v} \bar{B}_1) \cdot (\bar{A}_1 v B_1)]$$

$$C = (A_3 \bar{v} \bar{B}_3) \cdot (\bar{A}_3 v B_3) \cdot (A_2 \bar{v} \bar{B}_2) \cdot (\bar{A}_2 v B_2) \cdot (A_1 \bar{v} \bar{B}_1) \cdot (\bar{A}_1 v B_1)$$

The logic diagram for the above expression for the 3-bit comparator is as follows:



For a 4-bit comparator, where  $A_4 A_3 A_2 A_1 = B_4 B_3 B_2 B_1$ , we have:

$$C = (A_4 = B_4) \cdot (A_3 = B_3) \cdot (A_2 = B_2) \cdot (A_1 = B_1)$$

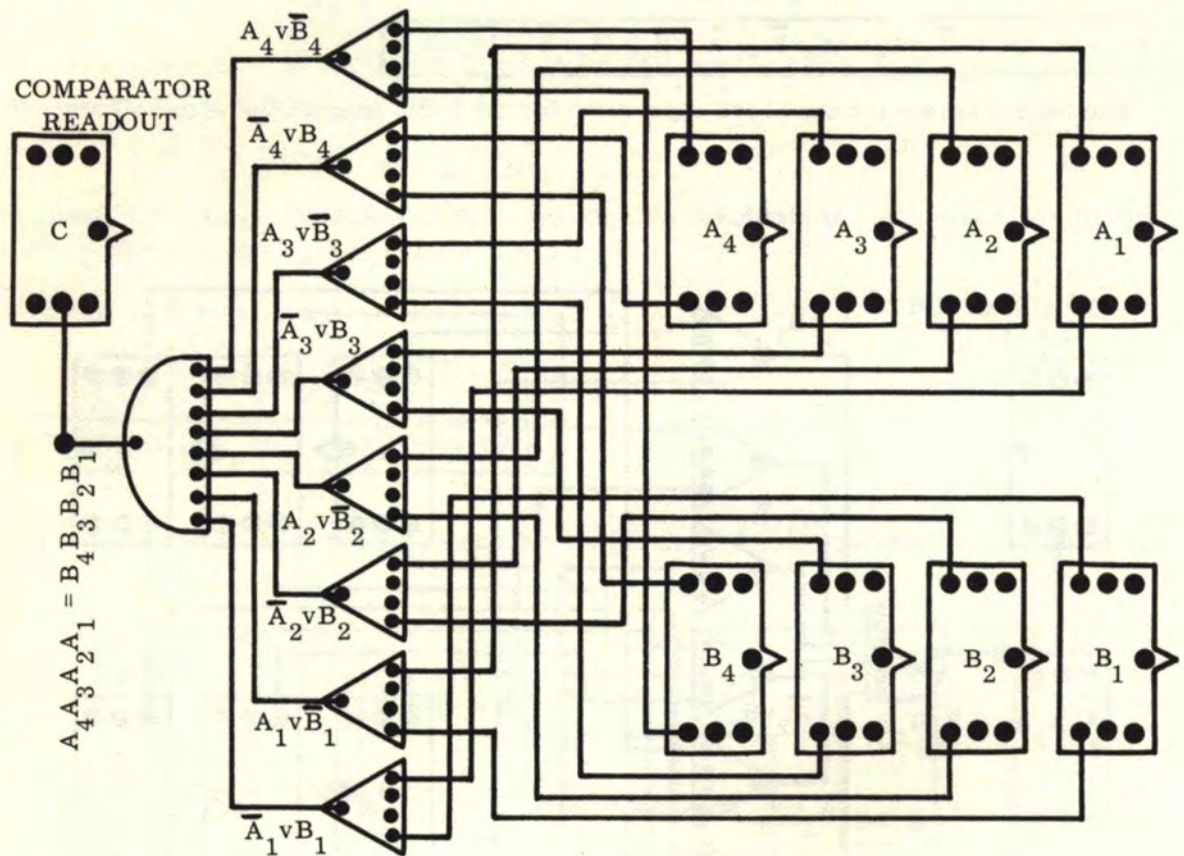
Substituting "AND" and "OR" functions, we have:

$$C = [(A_4 \bar{v} \bar{B}_4) \cdot (\bar{A}_4 v B_4)] \cdot [(A_3 \bar{v} \bar{B}_3) \cdot (\bar{A}_3 v B_3)] \cdot [(A_2 \bar{v} \bar{B}_2) \cdot (\bar{A}_2 v B_2)] \cdot [(A_1 \bar{v} \bar{B}_1) \cdot (\bar{A}_1 v B_1)]$$

$$C = (A_4 \bar{v} \bar{B}_4) \cdot (\bar{A}_4 v B_4) \cdot (A_3 \bar{v} \bar{B}_3) \cdot (\bar{A}_3 v B_3) \cdot (A_2 \bar{v} \bar{B}_2) \cdot (\bar{A}_2 v B_2) \cdot (A_1 \bar{v} \bar{B}_1) \cdot (\bar{A}_1 v B_1)$$

The logic diagram for the above expression for the 4-bit comparator is as follows:

## 10.11.3 THE "EQUAL TO" COMPARATOR (Continued)



The general equation for an "n"-bit "equal to" comparator is as follows:

$$C = (A_n = B_n) \cdot (A_{n-1} = B_{n-1}) \cdot (A_{n-2} = B_{n-2}) \cdot (A_{n-3} = B_{n-3}) \cdot \dots \cdot (A_2 = B_2) \cdot (A_1 = B_1)$$

The transformed "AND" and "OR" logic equation, after substitution, is:

$$C = (A_n v \bar{B}_n) \cdot (\bar{A}_n v B_n) \cdot (A_{n-1} v \bar{B}_{n-1}) \cdot (\bar{A}_{n-1} v B_{n-1}) \cdot (A_{n-2} v \bar{B}_{n-2}) \cdot (\bar{A}_{n-2} v B_{n-2}) \cdot (A_{n-3} v \bar{B}_{n-3}) \cdot (\bar{A}_{n-3} v B_{n-3}) \cdot \dots \cdot (A_2 v \bar{B}_2) \cdot (\bar{A}_2 v B_2) \cdot (A_1 v \bar{B}_1) \cdot (\bar{A}_1 v B_1)$$

The order and transformation of terms is very straightforward for the "equal to" comparator.

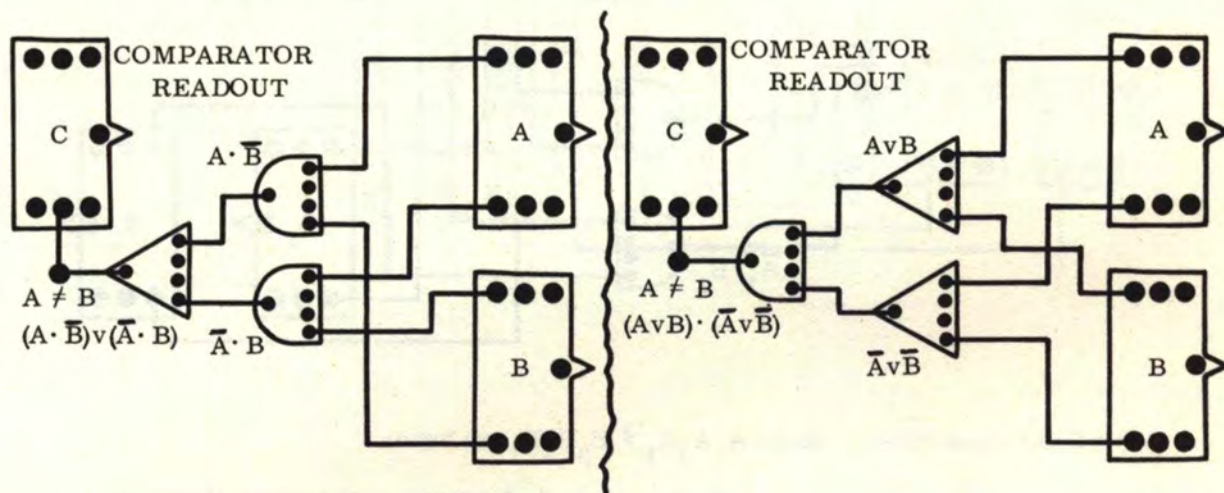
## 10.11.4 THE "UNEQUAL TO" COMPARATOR

Let us again consider the two single digit binary numbers A and B. Now we want to sense when A is unequal to B, or  $A \neq B$ . We will again set up a one-bit truth table (see also main truth table) for  $A \neq B$  as follows:

## 10.11.4 THE "UNEQUAL TO" COMPARATOR (Continued)

A	B	$A \neq B$	$(A \cdot \bar{B}) \vee (\bar{A} \cdot B)$	$(A \vee B) \cdot (\bar{A} \vee \bar{B})$
0	0	0	0	0
0	1	1	1	1
1	0	1	1	1
1	1	0	0	0

The last two entries  $(A \cdot \bar{B}) \vee (\bar{A} \cdot B)$  and  $(A \vee B) \cdot (\bar{A} \vee \bar{B})$  represent logic functions which have the same truth table as  $A \neq B$  and therefore all three expressions are equivalent.  $A \neq B$  may be represented either by  $(A \cdot \bar{B}) \vee (\bar{A} \cdot B)$ , or by  $(A \vee B) \cdot (\bar{A} \vee \bar{B})$ . Note that these are the exact same logic expressions for  $A + B$ ,  $A - B$ , and  $A \nabla B$ . If we have only one flip-flop for A and one flip-flop for B, the comparison is accomplished as follows:



Either one of the preceding logic diagrams will determine the  $A \neq B$  comparison. Further discussion will be with the expression  $(A \cdot \bar{B}) \vee (\bar{A} \cdot B)$ .

For 2-bit binary numbers A and B, we represent the digit positions as follows:

$$A_2 A_1$$

$$B_2 B_1$$

Now, the number  $A_2 A_1$  is unequal to  $B_2 B_1$  ( $A_2 A_1 \neq B_2 B_1$ ) when  $A_2 \neq B_2$  or  $A_1 \neq B_1$ . This can be written logically as follows:

$$C = (A_2 \neq B_2) \vee (A_1 \neq B_1)$$

The letter "C" denotes "COMPARISON". We can now substitute the expression  $(A \cdot \bar{B}) \vee (\bar{A} \cdot B)$  for  $A \neq B$  ( $A_1 \neq B_1$  and  $A_2 \neq B_2$ ) as follows:

## 10.11.4 THE "UNEQUAL TO" COMPARATOR (Continued)

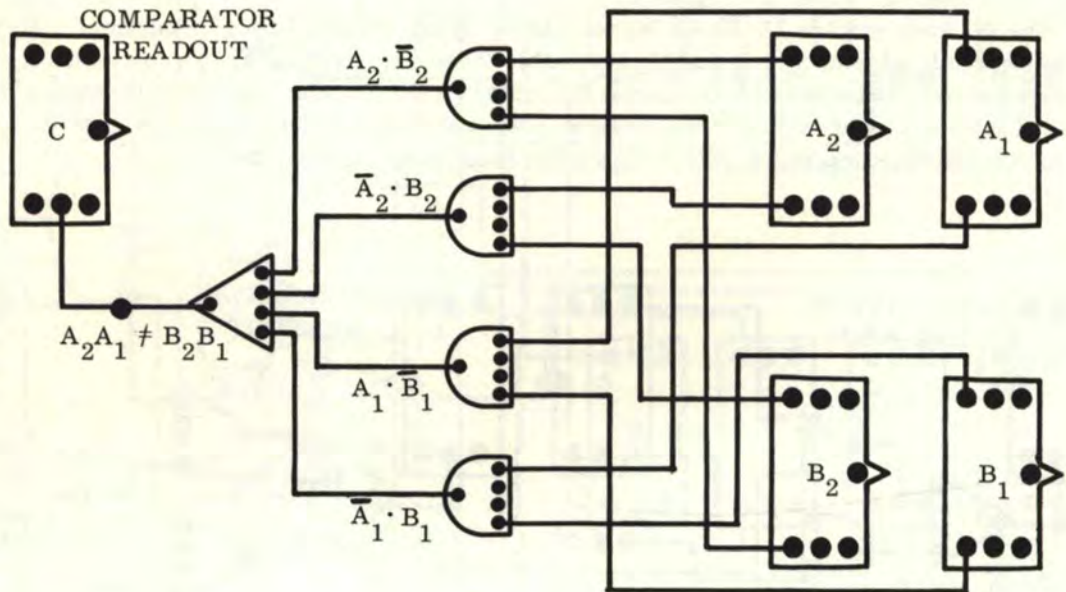
$$C = (A_2 \neq B_2) \vee (A_1 \neq B_1)$$

$$C = [(A_2 \cdot \bar{B}_2) \vee (\bar{A}_2 \cdot B_2)] \vee [(A_1 \cdot \bar{B}_1) \vee (\bar{A}_1 \cdot B_1)]$$

$$C = (A_2 \cdot \bar{B}_2) \vee (\bar{A}_2 \cdot B_2) \vee (A_1 \cdot \bar{B}_1) \vee (\bar{A}_1 \cdot B_1)$$

(ASSOCIATION: AL #2)

The logic diagram for the above expression for the 2-bit comparator is as follows:



For a 3-bit comparator, where  $A_3A_2A_1 \neq B_3B_2B_1$  we have:

$$\begin{array}{c} A_3A_2A_1 \\ B_3B_2B_1 \end{array}$$

Now, the number  $A_3A_2A_1$  is unequal to  $B_3B_2B_1$  ( $A_3A_2A_1 \neq B_3B_2B_1$ ) when  $A_3 \neq B_3$  or  $A_2 \neq B_2$  or  $A_1 \neq B_1$ . This can be written logically as follows:

$$C = (A_3 \neq B_3) \vee (A_2 \neq B_2) \vee (A_1 \neq B_1)$$

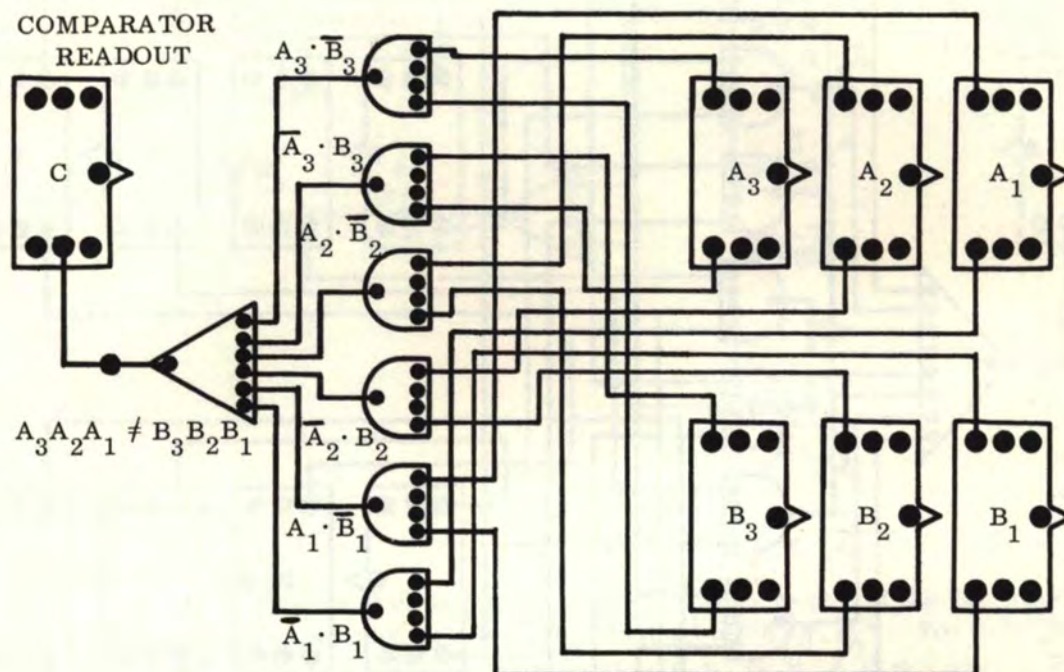
Substituting "AND" and "OR" functions, we have:

$$C = [(A_3 \cdot \bar{B}_3) \vee (\bar{A}_3 \cdot B_3)] \vee [(A_2 \cdot \bar{B}_2) \vee (\bar{A}_2 \cdot B_2)] \vee [(A_1 \cdot \bar{B}_1) \vee (\bar{A}_1 \cdot B_1)]$$

$$C = (A_3 \cdot \bar{B}_3) \vee (\bar{A}_3 \cdot B_3) \vee (A_2 \cdot \bar{B}_2) \vee (\bar{A}_2 \cdot B_2) \vee (A_1 \cdot \bar{B}_1) \vee (\bar{A}_1 \cdot B_1)$$

The logic diagram for the above expression for the 3-bit comparator is as follows:

## 10.11.4 THE "UNEQUAL TO" COMPARATOR (Continued)



For a 4-bit comparator, where  $A_4A_3A_2A_1 \neq B_4B_3B_2B_1$  we have:

$$C = (A_4 \neq B_4) \vee (A_3 \neq B_3) \vee (A_2 \neq B_2) \vee (A_1 \neq B_1)$$

Substituting "AND" and "OR" functions, we have:

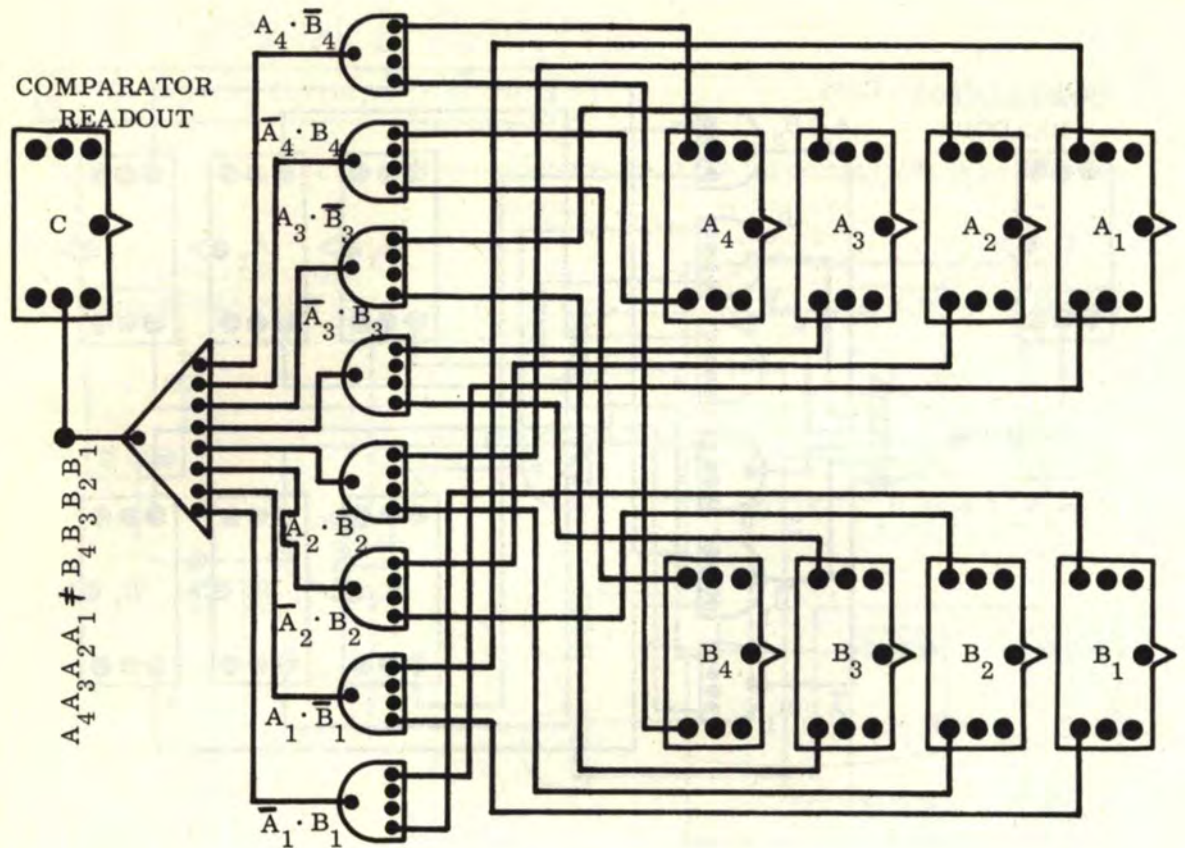
$$C = [(A_4 \cdot \bar{B}_4) \vee (\bar{A}_4 \cdot B_4)] \vee [(A_3 \cdot \bar{B}_3) \vee (\bar{A}_3 \cdot B_3)] \vee [(A_2 \cdot \bar{B}_2) \vee (\bar{A}_2 \cdot B_2)] \vee [(A_1 \cdot \bar{B}_1) \vee (\bar{A}_1 \cdot B_1)]$$

$$C = (A_4 \cdot \bar{B}_4) \vee (\bar{A}_4 \cdot B_4) \vee (A_3 \cdot \bar{B}_3) \vee (\bar{A}_3 \cdot B_3) \vee (A_2 \cdot \bar{B}_2) \vee (\bar{A}_2 \cdot B_2) \vee (A_1 \cdot \bar{B}_1) \vee (\bar{A}_1 \cdot B_1)$$

The logic diagram for the above expression for the 4-bit comparator is as follows:



## 10.11.4 THE "UNEQUAL TO" COMPARATOR (Continued)



The general equation for an "n"-bit "unequal to" comparator is as follows:

$$C = (A_n \neq B_n) \vee (A_{n-1} \neq B_{n-1}) \vee (A_{n-2} \neq B_{n-2}) \vee (A_{n-3} \neq B_{n-3}) \vee \dots \vee (A_2 \neq B_2) \vee (A_1 \neq B_1)$$

The transformed "AND" and "OR" logic equation, after substitution, is:

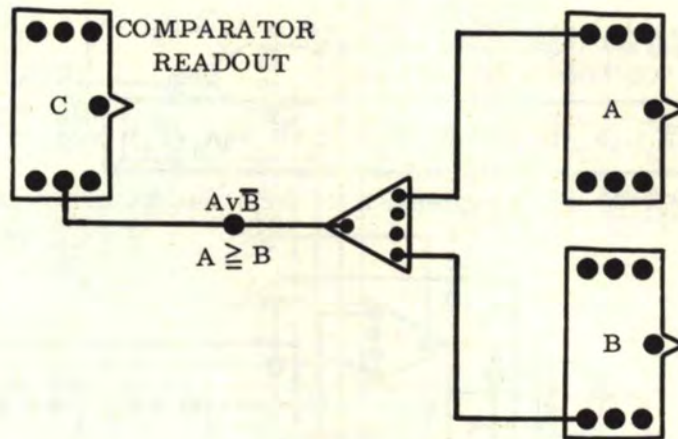
$$C = (A_n \cdot \bar{B}_n) \vee (\bar{A}_n \cdot B_n) \vee (A_{n-1} \cdot \bar{B}_{n-1}) \vee (\bar{A}_{n-1} \cdot B_{n-1}) \vee (A_{n-2} \cdot \bar{B}_{n-2}) \vee (\bar{A}_{n-2} \cdot B_{n-2}) \vee (A_{n-3} \cdot \bar{B}_{n-3}) \vee (\bar{A}_{n-3} \cdot B_{n-3}) \vee \dots \vee (A_2 \cdot \bar{B}_2) \vee (\bar{A}_2 \cdot B_2) \vee (A_1 \cdot \bar{B}_1) \vee (\bar{A}_1 \cdot B_1)$$

The order and transformation of terms, like the "equal to" comparator, is also very straightforward for the "unequal to" comparator.

## 10.11.5 THE "GREATER THAN OR EQUAL TO" COMPARATOR

For single digit binary numbers A and B, we want to sense when A is greater than or equal to B ( $A \geq B$ ). Referring back to the main truth table, we note that  $A \geq B$  is represented logically by  $(A \vee \bar{B})$ . The logic diagram is as follows:

## 10.11.5 THE "GREATER THAN OR EQUAL TO" COMPARATOR (Continued)



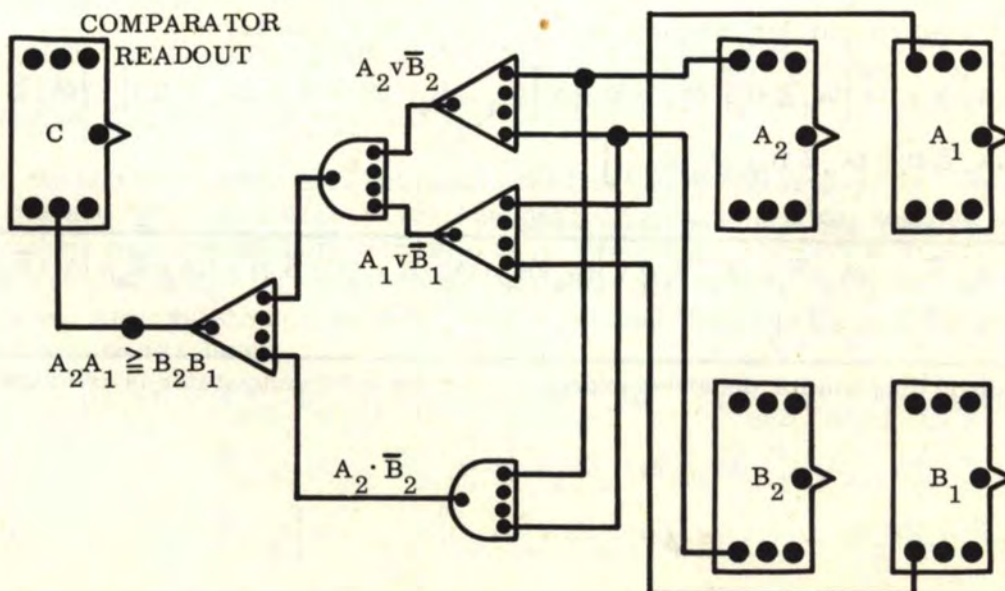
From this point on, the general discussion follows closely to that of the "greater than" comparator except that the last term is " $A_1 \geq B_1$ " instead of " $A_1 > B_1$ ". For 2-bit binary numbers, the number  $A_2A_1$  is greater than or equal to  $B_2B_1$  ( $A_2A_1 \geq B_2B_1$ ) when  $A_2 > B_2$  or when  $A_2 \geq B_2$  and  $A_1 \geq B_1$ . This can be written logically as follows:

$$C = (A_2 > B_2) \vee [(A_2 \geq B_2) \cdot (A_1 \geq B_1)]$$

Substituting "AND" and "OR" functions for  $A_2 > B_2$ ,  $A_2 \geq B_2$ , and  $A_1 \geq B_1$ , we have:

$$C = (A_2 \cdot \bar{B}_2) \vee [(A_2 \vee \bar{B}_2) \cdot (A_1 \vee \bar{B}_1)]$$

The logic diagram for the above expression for the 2-bit comparator is as follows:



## 10.11.5 THE "GREATER THAN OR EQUAL TO" COMPARATOR (Continued)

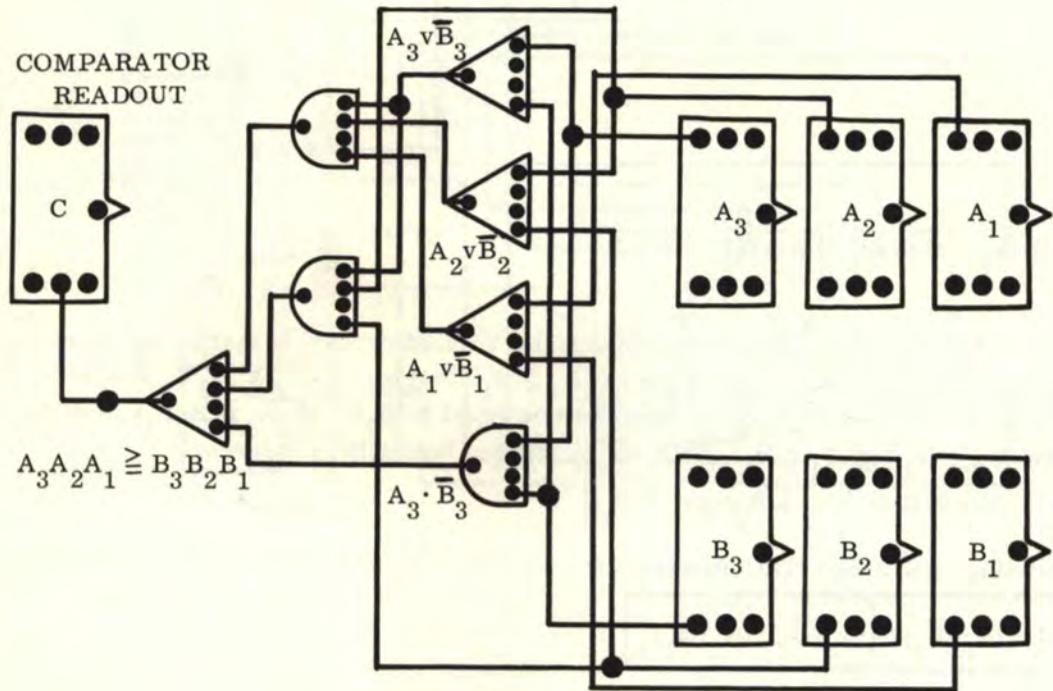
For a 3-bit comparator,  $A_3A_2A_1 \geq B_3B_2B_1$  when  $A_3 > B_3$  or when  $A_3 \geq B_3$  and  $A_2 > B_2$  or when  $A_3 \geq B_3$  and  $A_2 \geq B_2$  and  $A_1 \geq B_1$ . This can be written logically as follows:

$$C = (A_3 > B_3) \vee [(A_3 \geq B_3) \cdot (A_2 > B_2)] \vee [(A_3 \geq B_3) \cdot (A_2 \geq B_2) \cdot (A_1 \geq B_1)]$$

Substituting "AND" and "OR" functions, we have:

$$C = (A_3 \cdot \bar{B}_3) \vee [(A_3 \vee \bar{B}_3) \cdot (A_2 \cdot \bar{B}_2)] \vee [(A_3 \vee \bar{B}_3) \cdot (A_2 \vee \bar{B}_2) \cdot (A_1 \vee \bar{B}_1)]$$

The logic diagram for the above expression for the 3-bit comparator is as follows:



For a 4-bit comparator, where  $A_4A_3A_2A_1 \geq B_4B_3B_2B_1$ , we have:

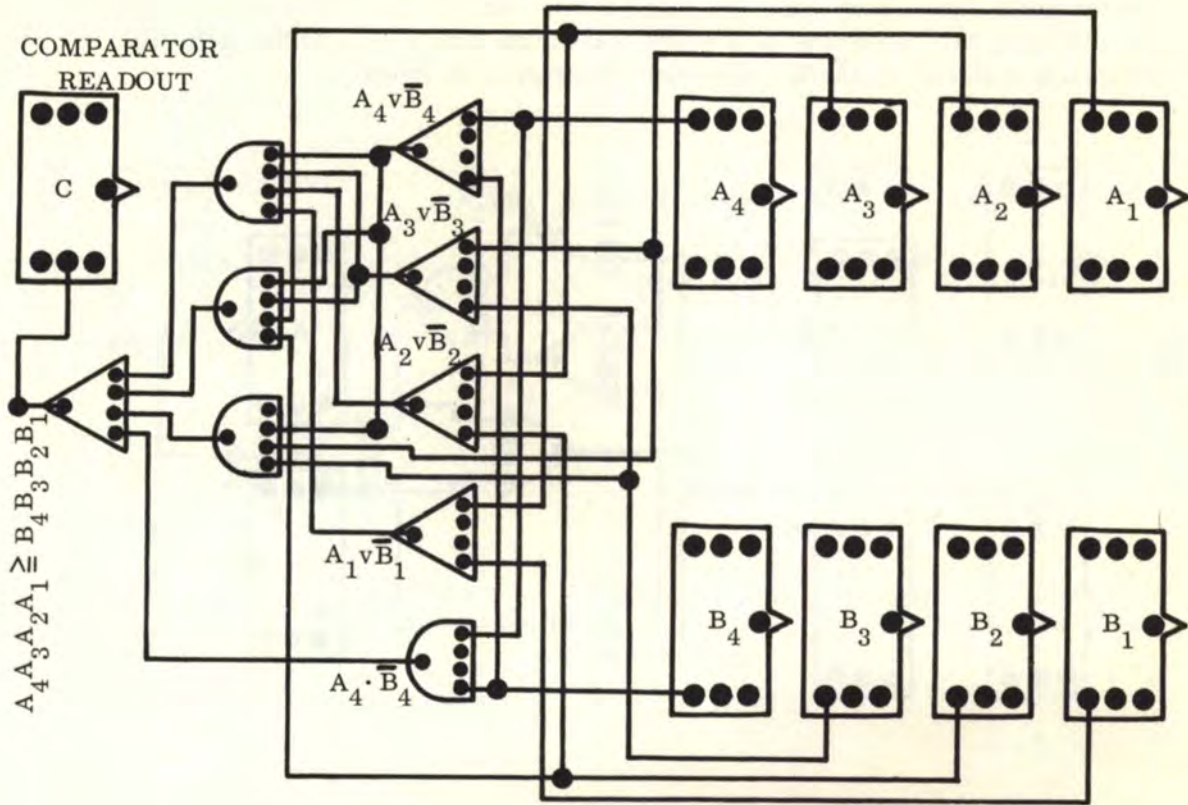
$$C = (A_4 > B_4) \vee [(A_4 \geq B_4) \cdot (A_3 > B_3)] \vee [(A_4 \geq B_4) \cdot (A_3 \geq B_3) \cdot (A_2 > B_2)] \vee [(A_4 \geq B_4) \cdot (A_3 \geq B_3) \cdot (A_2 \geq B_2) \cdot (A_1 \geq B_1)]$$

Substituting "AND" and "OR" functions, we have:

$$C = (A_4 \cdot \bar{B}_4) \vee [(A_4 \vee \bar{B}_4) \cdot (A_3 \cdot \bar{B}_3)] \vee [(A_4 \vee \bar{B}_4) \cdot (A_3 \vee \bar{B}_3) \cdot (A_2 \cdot \bar{B}_2)] \vee [(A_4 \vee \bar{B}_4) \cdot (A_3 \vee \bar{B}_3) \cdot (A_2 \vee \bar{B}_2) \cdot (A_1 \vee \bar{B}_1)]$$

The logic diagram for the above expression for the 4-bit comparator is as follows:

10.11.5 THE "GREATER THAN OR EQUAL TO" COMPARATOR (Continued)



The general logic equation of an "n"-bit "greater than or equal to" comparator is as follows:

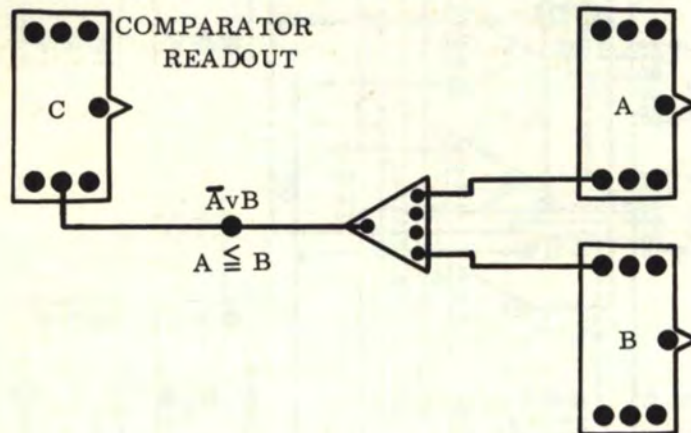
$$C = (A_n > B_n) \vee [(A_n \geq B_n) \cdot (A_{n-1} > B_{n-1})] \vee [(A_n \geq B_n) \cdot (A_{n-1} \geq B_{n-1}) \cdot (A_{n-2} > B_{n-2})] \vee [(A_n \geq B_n) \cdot (A_{n-1} \geq B_{n-1}) \cdot (A_{n-2} \geq B_{n-2}) \cdot (A_{n-3} > B_{n-3})] \vee [(A_n \geq B_n) \cdot (A_{n-1} \geq B_{n-1}) \cdot (A_{n-2} \geq B_{n-2}) \cdot (A_{n-3} \geq B_{n-3}) \cdot (A_{n-4} > B_{n-4})] \vee \dots \vee [(A_n \geq B_n) \cdot (A_{n-1} \geq B_{n-1}) \cdot \dots \cdot (A_3 \geq B_3) \cdot (A_2 \geq B_2) \cdot (A_1 \geq B_1)]$$

Note that the last term within each set (except the last set) of brackets is ">" while all other terms are "≥". The last set of brackets has all "≥" terms. Also, each successive set of brackets contains one more term than the previous set. The very first term in the expression is always  $A_n > B_n$ . The "terms" are those expressions within each set of parentheses. After substituting the equivalent "AND" and "OR" functions, the general equation becomes as follows:

$$C = (A_n \cdot \bar{B}_n) \vee [(A_n \vee \bar{B}_n) \cdot (A_{n-1} \cdot \bar{B}_{n-1})] \vee [(A_n \vee \bar{B}_n) \cdot (A_{n-1} \vee \bar{B}_{n-1}) \cdot (A_{n-2} \cdot \bar{B}_{n-2})] \vee [(A_n \vee \bar{B}_n) \cdot (A_{n-1} \vee \bar{B}_{n-1}) \cdot (A_{n-2} \vee \bar{B}_{n-2}) \cdot (A_{n-3} \cdot \bar{B}_{n-3})] \vee [(A_n \vee \bar{B}_n) \cdot (A_{n-1} \vee \bar{B}_{n-1}) \cdot (A_{n-2} \vee \bar{B}_{n-2}) \cdot (A_{n-3} \vee \bar{B}_{n-3}) \cdot (A_{n-4} \cdot \bar{B}_{n-4})] \vee \dots \vee [(A_n \vee \bar{B}_n) \cdot (A_{n-1} \vee \bar{B}_{n-1}) \cdot \dots \cdot (A_3 \vee \bar{B}_3) \cdot (A_2 \vee \bar{B}_2) \cdot (A_1 \vee \bar{B}_1)]$$

### 10.11.6 THE "LESS THAN OR EQUAL TO" COMPARATOR

For single digit binary numbers A and B, we want to sense when A is less than or equal to B ( $A \leq B$ ). Referring back to the main truth table, we note that  $A \leq B$  is represented logically by  $(\bar{A} \vee B)$ . The logic diagram is as follows:



From this point on, the general discussion follows closely to that of the "less than" comparator except that the last term is " $A_1 \leq B_1$ " instead of " $A_1 < B_1$ ". For 2-bit binary numbers, the number  $A_2A_1$  is less than or equal to  $B_2B_1$  ( $A_2A_1 \leq B_2B_1$ ) when  $A_2 < B_2$  or when  $A_2 \leq B_2$  and  $A_1 \leq B_1$ . This can be written logically as follows:

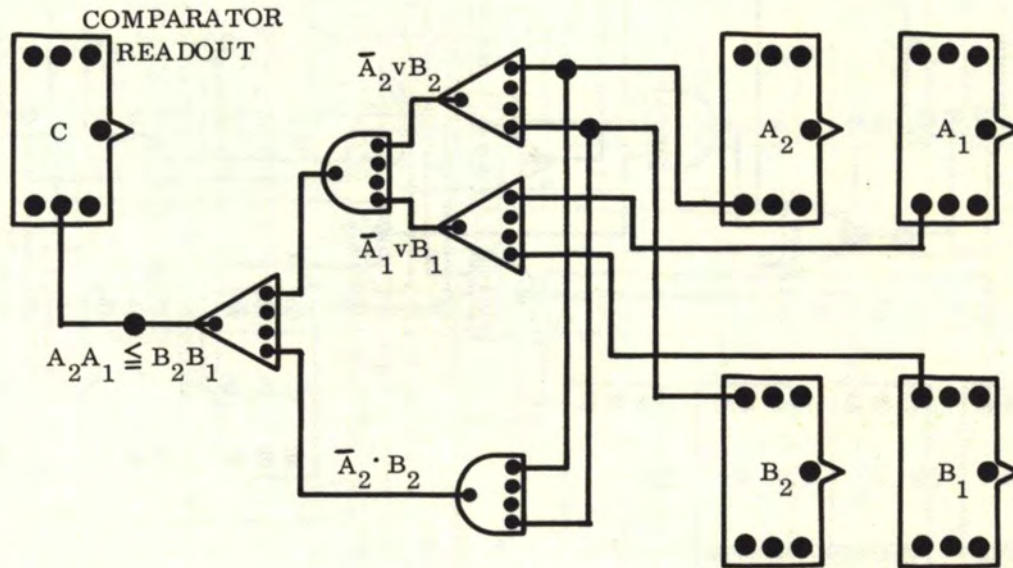
$$C = (A_2 < B_2) \vee [(A_2 \leq B_2) \cdot (A_1 \leq B_1)]$$

Substituting "AND" and "OR" functions for  $A_2 < B_2$ ,  $A_2 \leq B_2$ , and  $A_1 \leq B_1$ , we have:

$$C = (\bar{A}_2 \cdot B_2) \vee [(\bar{A}_2 \vee B_2) \cdot (\bar{A}_1 \vee B_1)]$$

The logic diagram for the above expression for the 2-bit comparator is as follows:

## 10.11.6 THE "LESS THAN OR EQUAL TO" COMPARATOR (Continued)



For a 3-bit comparator,  $A_3A_2A_1 \leq B_3B_2B_1$  when  $A_3 < B_3$  or when  $A_3 \leq B_3$  and  $A_2 < B_2$  or when  $A_3 \leq B_3$  and  $A_2 \leq B_2$  and  $A_1 \leq B_1$ . This can be written logically as follows:

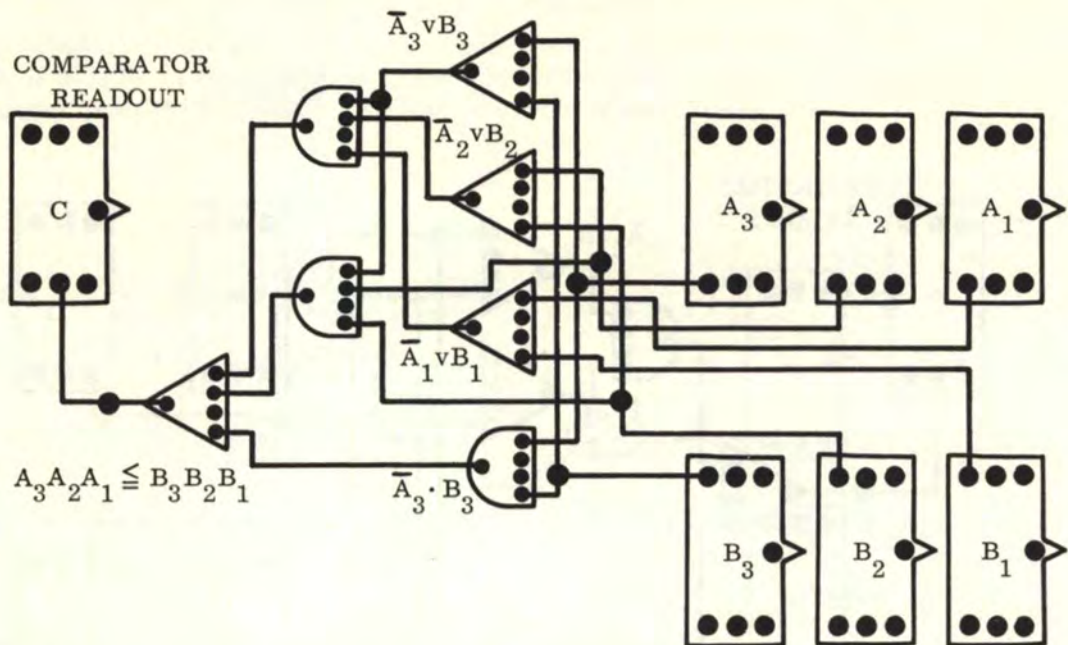
$$C = (A_3 < B_3) \vee [(A_3 \leq B_3) \cdot (A_2 < B_2)] \vee [(A_3 \leq B_3) \cdot (A_2 \leq B_2) \cdot (A_1 \leq B_1)]$$

Substituting "AND" and "OR" functions, we have:

$$C = (\bar{A}_3 \cdot B_3) \vee [(\bar{A}_3 \vee B_3) \cdot (\bar{A}_2 \cdot B_2)] \vee [(\bar{A}_3 \vee B_3) \cdot (\bar{A}_2 \vee B_2) \cdot (\bar{A}_1 \vee B_1)]$$

The logic diagram for the above expression for the 3-bit comparator is as follows:

## 10.11.6 THE "LESS THAN OR EQUAL TO" COMPARATOR (Continued)



For a 4-bit comparator, where  $A_4 A_3 A_2 A_1 \leq B_4 B_3 B_2 B_1$ , we have:

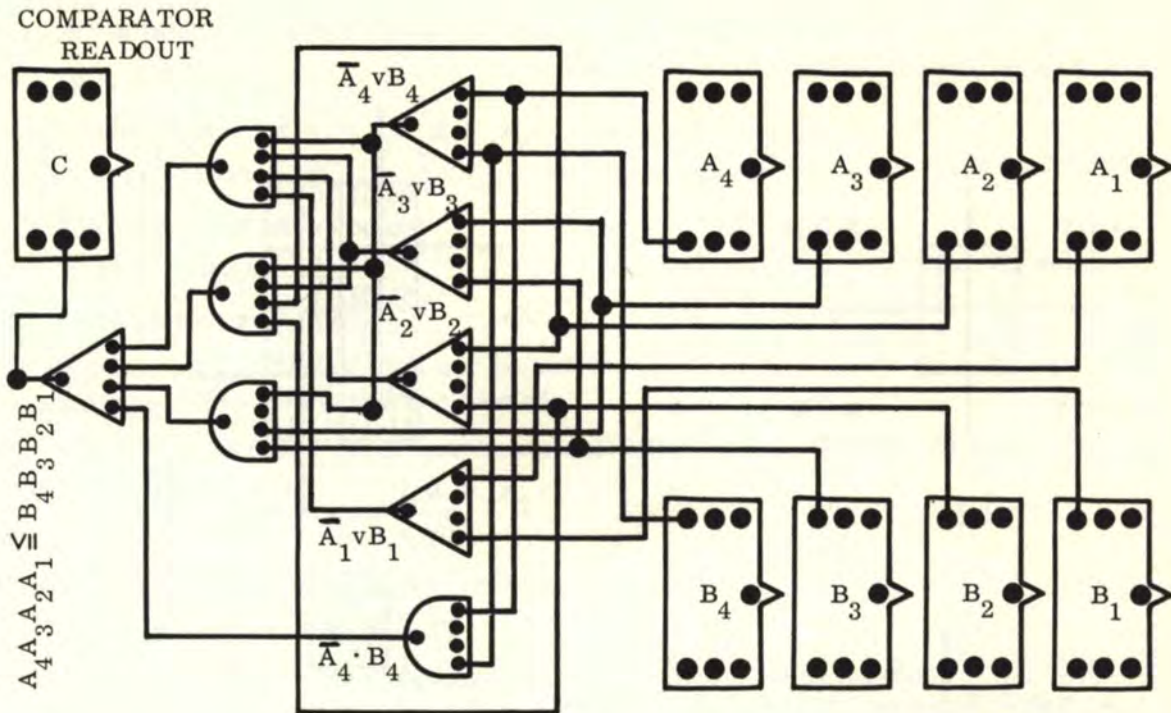
$$C = (A_4 < B_4) \vee [(A_4 \leq B_4) \cdot (A_3 < B_3)] \vee [(A_4 \leq B_4) \cdot (A_3 \leq B_3) \cdot (A_2 < B_2)] \vee [(A_4 \leq B_4) \cdot (A_3 \leq B_3) \cdot (A_2 \leq B_2) \cdot (A_1 \leq B_1)]$$

Substituting "AND" and "OR" functions, we have:

$$C = (\bar{A}_4 \cdot B_4) \vee [(\bar{A}_4 \vee B_4) \cdot (\bar{A}_3 \cdot B_3)] \vee [(\bar{A}_4 \vee B_4) \cdot (\bar{A}_3 \vee B_3) \cdot (\bar{A}_2 \cdot B_2)] \vee [(\bar{A}_4 \vee B_4) \cdot (\bar{A}_3 \vee B_3) \cdot (\bar{A}_2 \vee B_2) \cdot (\bar{A}_1 \vee B_1)]$$

The logic diagram for the above expression for the 4-bit comparator is as follows:

## 10.11.6 THE "LESS THAN OR EQUAL TO" COMPARATOR (Continued)



The general logic equation for an "n"-bit "less than or equal to" comparator is as follows:

$$\begin{aligned}
 C = & (A_n < B_n) v [(A_n \leq B_n) \cdot (A_{n-1} < B_{n-1})] v [(A_n \leq B_n) \cdot (A_{n-1} \leq B_{n-1}) \cdot (A_{n-2} < B_{n-2})] v \\
 & [(A_n \leq B_n) \cdot (A_{n-1} \leq B_{n-1}) \cdot (A_{n-2} \leq B_{n-2}) \cdot (A_{n-3} < B_{n-3})] v [(A_n \leq B_n) \cdot (A_{n-1} \leq B_{n-1}) \cdot \\
 & (A_{n-2} \leq B_{n-2}) \cdot (A_{n-3} \leq B_{n-3}) \cdot (A_{n-4} < B_{n-4})] v [(A_n \leq B_n) \cdot (A_{n-1} \leq B_{n-1}) \cdot \dots \cdot \\
 & \cdot (A_3 \leq B_3) \cdot (A_2 \leq B_2) \cdot (A_1 \leq B_1)]
 \end{aligned}$$

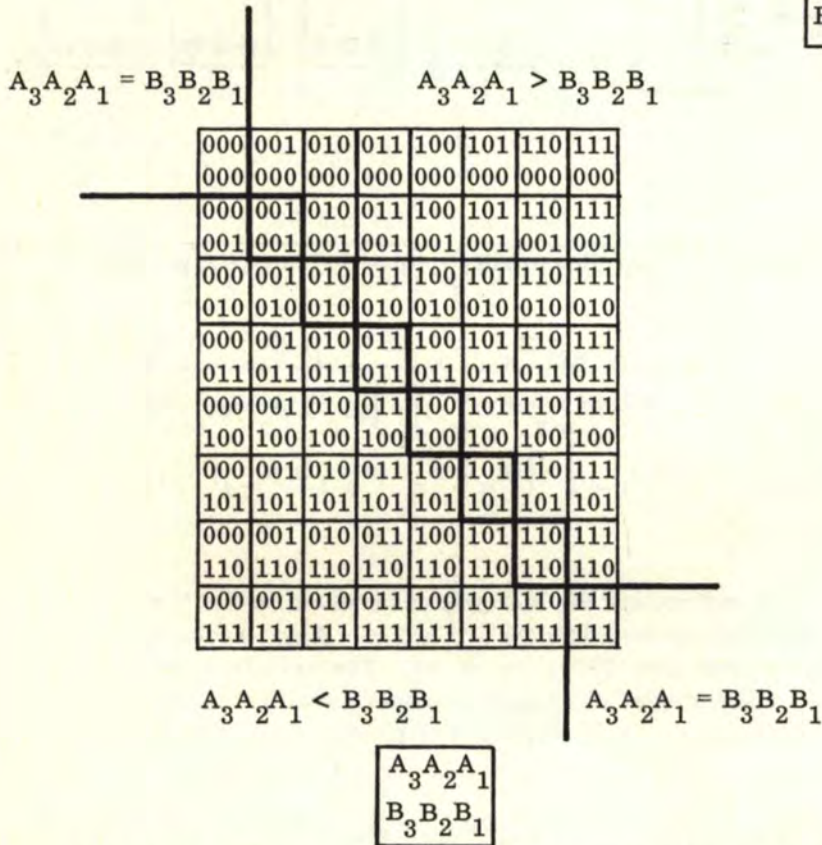
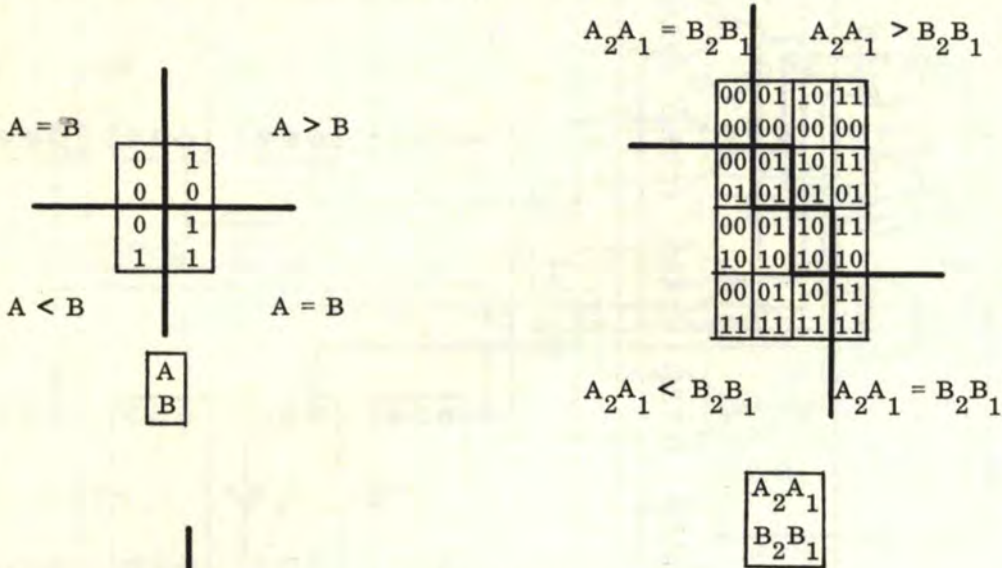
Note that the last term within each set (except the last set) of brackets is "<" while all other terms are " $\leq$ ". The last set of brackets has all " $\leq$ " terms. Also, each successive set of brackets contains one more term than the previous set. The very first term in the expression is always  $A_n < B_n$ . The "terms" are those expressions within each set of parentheses. After substituting the equivalent "AND" and "OR" functions, the general equation becomes as follows:

$$\begin{aligned}
 C = & (\bar{A}_n \cdot B_n) v [(\bar{A}_n v B_n) \cdot (\bar{A}_{n-1} \cdot B_{n-1})] v [(\bar{A}_n v B_n) \cdot (\bar{A}_{n-1} v B_{n-1}) \cdot (\bar{A}_{n-2} \cdot B_{n-2})] v [(\bar{A}_n v B_n) \cdot \\
 & (\bar{A}_{n-1} v B_{n-1}) \cdot (\bar{A}_{n-2} v B_{n-2}) \cdot (\bar{A}_{n-3} \cdot B_{n-3})] v [(\bar{A}_n v B_n) \cdot (\bar{A}_{n-1} v B_{n-1}) \cdot (\bar{A}_{n-2} v B_{n-2}) \cdot \\
 & (\bar{A}_{n-3} v B_{n-3}) \cdot (\bar{A}_{n-4} \cdot B_{n-4})] v \dots v [(\bar{A}_n v B_n) \cdot (\bar{A}_{n-1} v B_{n-1}) \cdot \dots \cdot (\bar{A}_3 v B_3) \cdot (\bar{A}_2 v B_2) \cdot \\
 & (\bar{A}_1 v B_1)]
 \end{aligned}$$



10.11.7 GENERAL COMPARISON

This section interrelates all the six individual comparators with respect to each other. We now introduce COMPARISON DIAGRAMS for one-bit, 2-bit, and 3-bit numbers. These blocks show all the possibilities for comparison.



Note that the "A = B" region is the upper-left-lower-right diagonal while the "A > B" region is to the upper right of the diagonal and the "A < B" region is to the lower left of the diagonal. The above diagrams include all possibilities of comparison. If we pick one cell at random from each diagram, this cell may represent A = B, or A > B, or A < B. Thus we can represent the total possibilities of comparison (C<sub>T</sub>) as follows:

$$C_T = (A = B) \vee (A > B) \vee (A < B)$$

10.11.7 GENERAL COMPARISON (Continued)

In order to determine a region of comparison, we must set at least one of the "OR"-ed comparison terms equal to zero. The term set equal to zero will then drop out. The single deletion possibilities are:

$$C = (A = B) \vee (A > B) \qquad (A < B) \text{ deleted}$$

This is the same as  $A \geq B$ .

$$C = (A = B) \vee (A < B) \qquad (A > B) \text{ deleted}$$

This is the same as  $A \leq B$ .

$$C = (A > B) \vee (A < B) \qquad (A = B) \text{ deleted}$$

This is the same as  $A \neq B$ .

If we delete any two terms, this will isolate the  $(A = B)$ ,  $(A > B)$ , or  $(A < B)$  regions depending on which two terms are deleted.

Also, we can delete all comparison possibilities leaving nothing. However, we can negate  $C_T$  as follows:

$$\overline{C_T} = \overline{(A = B) \vee (A > B) \vee (A < B)}$$

Using DeMorgan's Theorem, we obtain:

$$\overline{C_T} = \overline{(A = B)} \cdot \overline{(A > B)} \cdot \overline{(A < B)}$$

Now, in order to determine a comparison, we must again set at least one of the "AND"-ed comparison terms equal to zero. Then the term set equal to "0" becomes "0" or "1" and drops out when "AND"-ed with the rest of the expression. The single deletion possibilities are:

$$\overline{C} = \overline{(A = B)} \cdot \overline{(A > B)} \qquad (A < B) \text{ deleted}$$

Now let us substitute the "AND" and "OR" comparison functions and simplify the expression.

$$\begin{aligned} \overline{C} &= \overline{[(A \vee \overline{B}) \cdot (\overline{A} \vee B)] \cdot (A \vee \overline{B})} \\ &= \overline{[(A \vee \overline{B}) \vee (\overline{A} \vee B)] \cdot (\overline{A} \cdot B)} \\ &= \overline{[(\overline{A} \cdot B) \vee (A \cdot \overline{B})] \cdot (\overline{A} \cdot B)} \\ &= \overline{(\overline{A} \cdot B)} \end{aligned}$$

$$\boxed{\overline{C} = A < B}$$

Substituting back the comparison expression. Now, we go back to

$\overline{C} = \overline{(A = B)} \cdot \overline{(A > B)}$  and negate again:

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

$$C = (A = B) \vee (A > B) \quad \text{Using DeMorgan's Theorem.}$$

$$\boxed{C = (A \geq B)}$$

This means that if we wire up a comparator for  $A \geq B$ , we also have an  $A < B$  comparator by reading out  $\overline{C}$  instead of  $C$ .

The other possibilities are:

$$\overline{C} = \overline{(A = B)} \cdot \overline{(A < B)} \qquad (A > B) \text{ deleted}$$

$$\overline{C} = \overline{(A > B)} \cdot \overline{(A < B)} \qquad (A = B) \text{ deleted}$$

Also, deleting two terms, we have:

10.11.7 GENERAL COMPARISON (Continued)

$$\begin{aligned} \overline{C} &= \overline{(A = B)} && (A > B) \text{ and } (A < B) \text{ deleted} \\ \overline{C} &= \overline{(A > B)} && (A = B) \text{ and } (A < B) \text{ deleted} \\ \overline{C} &= \overline{(A < B)} && (A = B) \text{ and } (A > B) \text{ deleted} \end{aligned}$$

By substituting and working out the "AND" and "OR" logic for the above expressions, we can show that we have a different comparison by reading out  $\overline{C}$  instead of C. The "C" and " $\overline{C}$ " comparators are as follows:

$$\begin{aligned} \text{If } C &= (A > B) && \text{then } \overline{C} = (A \leq B) \\ \text{If } C &= (A < B) && \text{then } \overline{C} = (A \geq B) \\ \text{If } C &= (A = B) && \text{then } \overline{C} = (A \neq B) \\ \text{If } C &= (A \neq B) && \text{then } \overline{C} = (A = B) \\ \text{If } C &= (A \geq B) && \text{then } \overline{C} = (A < B) \\ \text{If } C &= (A \leq B) && \text{then } \overline{C} = (A > B) \end{aligned}$$

Thus, if we build one comparator, we have actually built two!!!

The comparison diagram may be used to prove comparison identities in the same manner as a truth table. We will use the 2-bit comparison diagram for the discussion. The quantity  $A_1$  is represented by marking all cells with a dot "•" that have an  $A_1$  (i. e.,  $A_1 = 1$ ). The numbers are not shown here again, but should be referred back to the main comparison diagram.  $A_1$  is represented as follows:

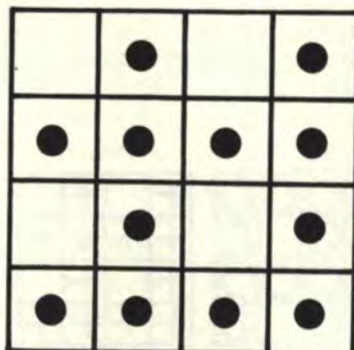
	•		•
	•		•
	•		•
	•		•

Refer back to the main comparison diagram and note that each cell with  $A_1 = 1$  is dotted. If we represent  $A_1 \cdot B_1$ , we must dot all cells with  $A_1 = B_1 = 1$ . The  $A_1$  and  $B_1$  must be identical in all dotted cells.

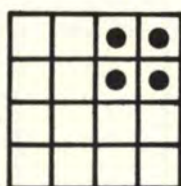
	•		•
	•		•

10.11.7 GENERAL COMPARISON (Continued)

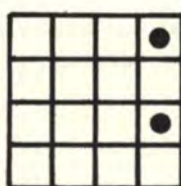
To represent  $A_1 \vee B_1$ , we must dot all cells with  $A_1 = 1$  and all cells with  $B_1 = 1$ , but in this case we look for either one case or the other, or both.



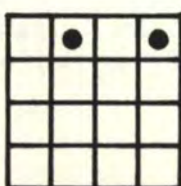
The following examples show cell plots of various comparison expressions:



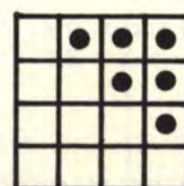
$$A_2 \cdot \bar{B}_2$$



$$A_1 \cdot A_2 \cdot \bar{B}_1$$



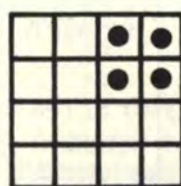
$$\bar{B}_2 \cdot A_1 \cdot \bar{B}_1$$



$$(A_2 \cdot \bar{B}_2) \vee (A_1 \cdot A_2 \cdot \bar{B}_1) \vee (\bar{B}_2 \cdot A_1 \cdot \bar{B}_1)$$

When we "OR" two or more comparison diagrams, we merely superimpose the diagrams. The diagram to the right shows the "OR"-ed representation of the first three. Note that the  $A > B$  region is filled. When a comparison region is filled, and the remaining cells are open, the "OR"-ed expression represents an identical comparison expression as is the case for the above example of  $A > B$ . Note that for an equivalent comparison expression, all cells in a comparison region must be filled and the remaining cells must be open. If this condition does not exist, then the expression does not represent a comparison identity.

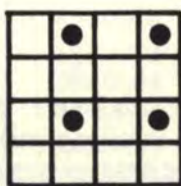
Consider the examples  $(A_2 \cdot \bar{B}_2)$ ,  $(\bar{A}_2 \cdot B_2)$ ,  $(A_1 \cdot \bar{B}_1)$ ,  $(\bar{A}_1 \cdot B_1)$  and  $(A_2 \cdot \bar{B}_2) \vee (\bar{A}_2 \cdot B_2) \vee (A_1 \cdot \bar{B}_1) \vee (\bar{A}_1 \cdot B_1)$  as shown below.



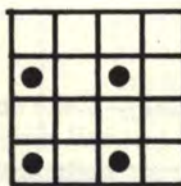
$$A_2 \cdot \bar{B}_2$$



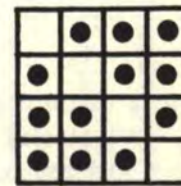
$$\bar{A}_2 \cdot B_2$$



$$A_1 \cdot \bar{B}_1$$



$$\bar{A}_1 \cdot B_1$$

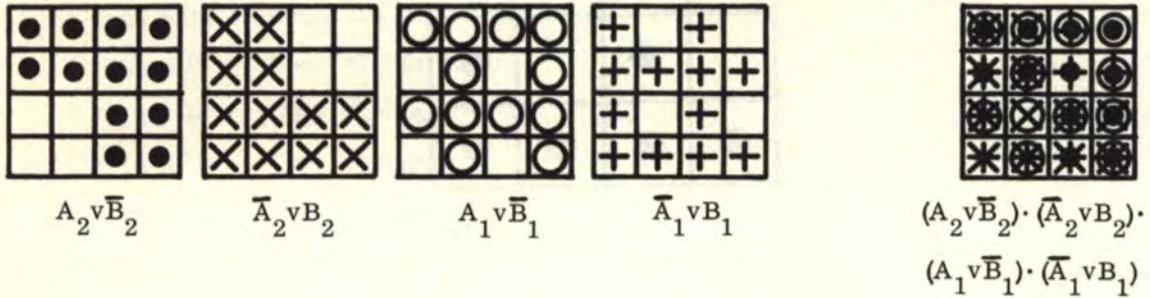


$$(A_2 \cdot \bar{B}_2) \vee (\bar{A}_2 \cdot B_2) \vee (A_1 \cdot \bar{B}_1) \vee (\bar{A}_1 \cdot B_1)$$

10.11.7 GENERAL COMPARISON (Continued)

The "OR"-ed function to the right has the regions filled which represent  $A > B$  or  $A < B$ . This is the same as  $A_2A_1 \neq B_2B_1$  or  $A \neq B$  for which the expression shown is a true identity.

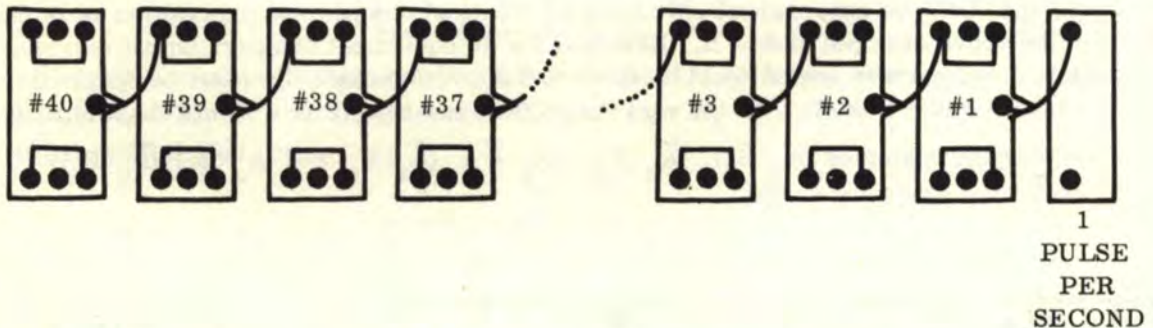
If we want to superimpose cells by "AND"-ing, the process becomes more complicated and we must use more than a dot. For instance, if we represent one expression by dots, the other by "x"s, another by "o"s and another by "+"s, then we can only count a superimposed cell as filled when all different symbols are superimposed on each other. Consider  $(A_2 \vee \bar{B}_2)$ ,  $(\bar{A}_2 \vee B_2)$ ,  $(A_1 \vee \bar{B}_1)$ ,  $(\bar{A}_1 \vee B_1)$ , and  $(A_2 \vee \bar{B}_2) \cdot (\bar{A}_2 \vee B_2) \cdot (A_1 \vee \bar{B}_1) \cdot (\bar{A}_1 \vee B_1)$ . The comparison diagrams are as follows:



The "AND"-ed function to the right has only the top-left-bottom-right diagonal which contains all 4 superimposed symbols. This is the "=" diagonal and hence this represents  $A_2A_1 = B_2B_1$  or  $A = B$  for which the expression shown is a true identity.

10.12 THE TIME MACHINE

Now we are ready for a fantastic adventure in time--not in the sense of going back in time, but in the sense of keeping and determining time. In order to get an idea of the vast capability that flip-flops have for determining time, let us set up a 40-bit binary "up" counter and trigger the first flip-flop at a rate of one pulse per second.



How long would it take the last light to come on? An hour? A day? Maybe a year? The true, but fantastic answer is that you will never live long enough to see it come on!!!! Neither will your grandchildren--nor their grandchildren. Neither will 200 successive generations of your descendants!!!! Let us assume that we started the counter going on zero hour, Monday, January 1, 1968--the year this book was written. Light #40 will then come "on" on Sunday, May 2, 19388 A.D. !!! The following table shows the light numbers and the almost unbelievable dates they will come on!

10.12 THE TIME MACHINE (Continued)

LIGHT #	DAYS	TIME "ON" HRS:MIN:SEC	DATE "ON" MO/DAY/YR	DAY OF WEEK
1		00:00:01	JAN 1, 1968	M
2		00:00:02	JAN 1, 1968	M
3		00:00:04	JAN 1, 1968	M
4		00:00:08	JAN 1, 1968	M
5		00:00:16	JAN 1, 1968	M
6		00:00:32	JAN 1, 1968	M
7		00:01:04	JAN 1, 1968	M
8		00:02:08	JAN 1, 1968	M
9		00:04:16	JAN 1, 1968	M
10		00:08:32	JAN 1, 1968	M
11		00:17:04	JAN 1, 1968	M
12		00:34:08	JAN 1, 1968	M
13		01:08:16	JAN 1, 1968	M
14		02:16:32	JAN 1, 1968	M
15		04:33:04	JAN 1, 1968	M
16		09:06:08	JAN 1, 1968	M
17		18:12:16	JAN 1, 1968	M
18	1 d	12:24:32	JAN 2, 1968	T
19	3 d	00:49:04	JAN 4, 1968	TH
20	6 d	01:38:08	JAN 7, 1968	SN
21	12 d	03:16:16	JAN 13, 1968	S
22	24 d	06:32:32	JAN 25, 1968	TH
23	48 d	13:05:04	FEB 18, 1968	SN
24	97 d	02:10:08	APR 7, 1968	SN
25	194 d	04:20:16	JUL 13, 1968	S
26	388 d	08:40:32	JAN 23, 1969	TH
27	776 d	17:21:04	FEB 15, 1970	SN
28	1553 d	10:42:08	APR 2, 1972	SN
29	3106 d	21:24:16	JUL 3, 1976	S
30	6213 d	18:48:32	JAN 4, 1985	F
31	12427 d	13:37:04	JAN 9, 2002	W
32	24855 d	03:14:08	JAN 19, 2036	S
33	49710 d	06:28:16	FEB 5, 2104	T
34	99420 d	12:56:32	MAR 11, 2240	W
35	198841 d	01:53:04	MAY 21, 2512	S
36	397682 d	03:46:08	OCT 9, 3056	TH
37	795364 d	07:32:16	JUL 18, 4145	SN
38	1590728 d	15:04:32	JAN 31, 6323	T
39	3181457 d	06:09:04	MAR 3, 10678	F
40	6362914 d	12:18:08	MAY 2, 19388	SN
(RESET)	12725828 d	00:36:16	AUG 31, 36808	TH

10.12 THE TIME MACHINE (Continued)

Note that the "action" begins after the eighteenth light. The counter will reset on Thursday, August 31, 36808 at 12:36:16 A. M. !! The calculations of the dates in the table become complicated after 2000 A. D. , and we must consider "leap year" corrections in order to pinpoint the correct date and day of the week. Just thinking about these dates shows us how small a speck we are in the sands of time!!!

Now let us take a closer look at what a YEAR really is. The astronomical books define a year as 365d 05h 48m 46s (365 days, 5 hours, 48 minutes, and 46 seconds). If we were to have only 365 days in every year, the total days for 4 years would be 1460 days. However, if we consider the extra 5h 48m 46s, four years will produce a loss of 23h 15m 4s (almost 24h). If we add an extra day to every fourth year (to that year which is divisible by 4), we will compensate for the lost 23h 15m 4s, but we will also have an extra 44m 56s too much. Therefore, every four years will contain 1461 days but will gain 44m 56s on what the true calendar year should be. Let us consider an interval of 400 years. This will contain 100 4-year totals of 1461 days. However, the 44m 56s, when multiplied by 100, amounts to 74h 53m 20s (slightly more than 3 days or 72 hours). Therefore, we must subtract 3 days every 400 years and can easily do this by deleting 3 leap years from centuries not divisible by 400 and by retaining the leap year for centuries that are divisible by 400. (Note that in our era, 1700, 1800, and 1900 were not leap years, but the year 2000 will be a leap year.)

The above discussion defines the present Gregorian calendar with a first-order correction (leap year every fourth year) and second-order correction (leap century every fourth century). The old Julian calendar did not use the second-order "century" correction and has included too many leap years in the century years. Now, we must look still further into the present Gregorian calendar if we are to calculate dates such as 36,808 A. D. in the previous table. We must now determine a third-order correction. In doing this, we will assume that the 365d 5h 48m 46s year length remains unchanged. In reality, scientists estimate that the day length will gain about one second each century due to slowdown of the earth's rotation and that the earth's orbital slowdown will cause further effects; but we will not include these estimates in our calculations. Now let us again consider an interval of 400 years and call it a "QUAD" century. After the Gregorian calendar corrections for every "QUAD", there is still an excess of 2h 53m 20s from the true calendar date. An interval of 216 "QUADS" will yield 624h 00m 00s too many, which must be deleted. Now, 624 hours equal 26 days exactly. We must now delete 26 days every 216 "QUADS". To do this, we will delete leap years every eighth "QUAD" starting with 5200 A. D. for a third-order correction. However,  $216 \div 8$  equals 27 days deleted in 216 "QUADS", or one day too many deleted. This can be compensated by not deleting a leap year every twenty-seventh interval starting with 88,400 A. D. This completes all the corrections necessary for this scheme and the cycle may be repeated over and over again as far as we might go--even to 1,000,000 A. D. !!!

Briefly summarized, our calendar scheme is as follows:

Every 4 years: LEAP YEAR (years divisible by 4)

Every 4th century: LEAP YEAR ("QUAD" century years divisible by 400)

Every 8th "QUAD" century interval (starting with 5200): LEAP YEAR only every 27th interval

The days of the week may be calculated by referring to the perpetual calendar in any almanac which shows the dates, years, and days of the week for a 400-year interval. Then we have all the information necessary when we note that if we add or subtract 400 years to each year shown, the months and days of the week will be identical. For instance, a calendar for 1968 would be the same as the calendar for 2368, 2768, and 3168. However, if we pass any "interval" "QUAD" century years such as 5200, 8400, etc., we must subtract

10.12 THE TIME MACHINE (Continued)

one weekday for each "interval". For instance, January 1, 1968 was on Monday. Such would be the case for January 1, 2368 or January 1, 2768 or January 1, 3168. The years 3568, 3968, 4368, 4768 and 5168 would also have January 1 on Monday. However, 5568 would have January 1 on Sunday because 5200 is not a leap year. Likewise, January 1, 9768 will be on Saturday because 5200 and 8400 are not leap years. The table following shows the "QUAD" century leap years from 2000 A. D. to 101,600 A. D. The dates for the "ON" times of the last bits of the 40-bit binary "up" counter were determined from this table.



10.12 THE TIME MACHINE (Continued)**"QUAD" CENTURY YEARS**

("L" = Leap Year; "\*\*\*" = Normal Year; "##" = 27th

Interval Leap Year) (1 year = 365d 05h 48m 46s)

2000 L	22000 L	42000 L	62000 L	82000 **25
2400 L	22400 L	42400 L	62400 L	82400 L
2800 L	22800 L	42800 L	62800 **19	82800 L
3200 L	23200 L	43200 L	63200 L	83200 L
3600 L	23600 L	43600 **13	63600 L	83600 L
4000 L	24000 L	44000 L	64000 L	84000 L
4400 L	24400 **7	44400 L	64400 L	84400 L
4800 L	24800 L	44800 L	64800 L	84800 L
5200 **1	25200 L	45200 L	65200 L	85200 **26
5600 L	25600 L	45600 L	65600 L	85600 L
6000 L	26000 L	46000 L	66000 **20	86000 L
6400 L	26400 L	46400 L	66400 L	86400 L
6800 L	26800 L	46800 **14	66800 L	86800 L
7200 L	27200 L	47200 L	67200 L	87200 L
7600 L	27600 **8	47600 L	67600 L	87600 L
8000 L	28000 L	48000 L	68000 L	88000 L
8400 **2	28400 L	48400 L	68400 L	88400 L ##
8800 L	28800 L	48800 L	68800 L	88800 L
9200 L	29200 L	49200 L	69200 **21	89200 L
9600 L	29600 L	49600 L	69600 L	89600 L
10000 L	30000 L	50000 **15	70000 L	90000 L
10400 L	30400 L	50400 L	70400 L	90400 L
10800 L	30800 **9	50800 L	70800 L	90800 L
11200 L	31200 L	51200 L	71200 L	91200 L
11600 **3	31600 L	51600 L	71600 L	91600 **1
12000 L	32000 L	52000 L	72000 L	92000 L
12400 L	32400 L	52400 L	72400 **22	92400 L
12800 L	32800 L	52800 L	72800 L	92800 L
13200 L	33200 L	53200 **16	73200 L	93200 L
13600 L	33600 L	53600 L	73600 L	93600 L
14000 L	34000 **10	54000 L	74000 L	94000 L
14400 L	34400 L	54400 L	74400 L	94400 L
14800 **4	34800 L	54800 L	74800 L	94800 **2
15200 L	35200 L	55200 L	75200 L	95200 L
15600 L	35600 L	55600 L	75600 **23	95600 L
16000 L	36000 L	56000 L	76000 L	96000 L
16400 L	36400 L	56400 **17	76400 L	96400 L
16800 L	36800 L	56800 L	76800 L	96800 L
17200 L	37200 **11	57200 L	77200 L	97200 L
17600 L	37600 L	57600 L	77600 L	97600 L
18000 **5	38000 L	58000 L	78000 L	98000 **3
18400 L	38400 L	58400 L	78400 L	98400 L
18800 L	38800 L	58800 L	78800 **24	98800 L
19200 L	39200 L	59200 L	79200 L	99200 L
19600 L	39600 L	59600 **18	79600 L	99600 L
20000 L	40000 L	60000 L	80000 L	100000 L
20400 L	40400 **12	60400 L	80400 L	100400 L
20800 L	40800 L	60800 L	80800 L	100800 L
21200 **6	41200 L	61200 L	81200 L	101200 **4
21600 L	41600 L	61600 L	81600 L	101600 L

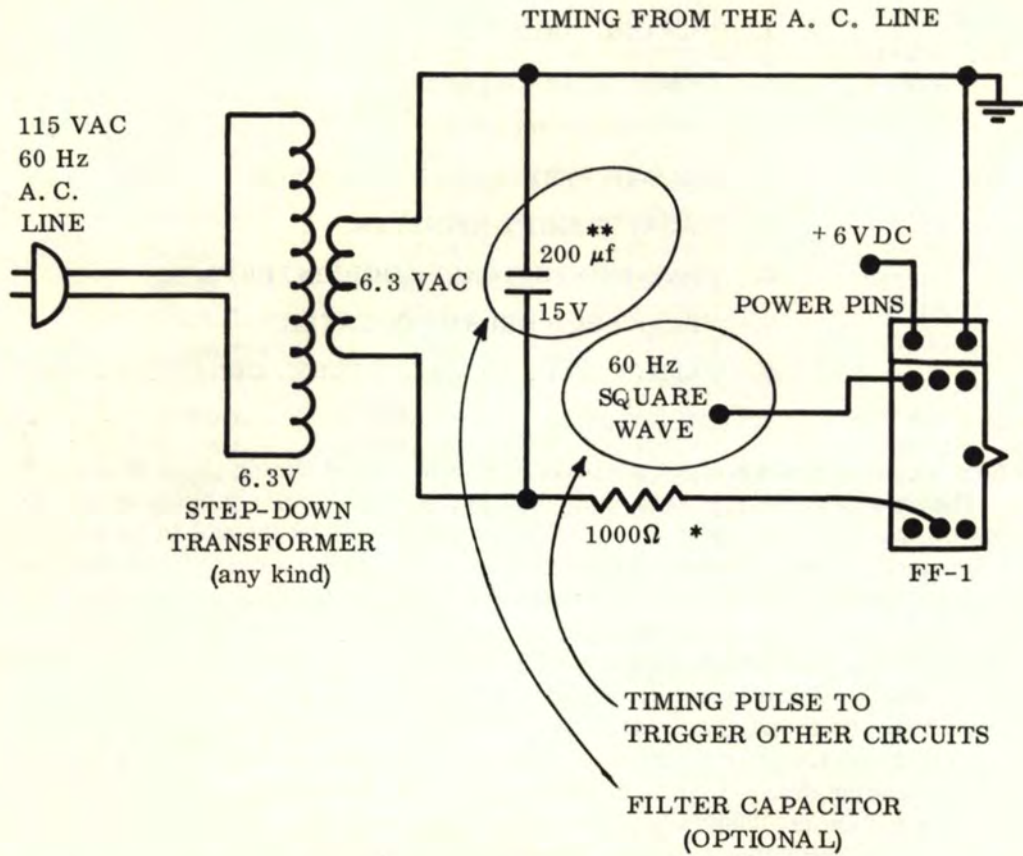
### 10.12.1 THE TIME MACHINE CLOCK

Let us now return to our humble mid-1900's and determine how to build a gated logic time clock. In fact we can build a very accurate time clock that will automatically give the time, exact day of the month, month, and year. The clock can also sense leap years, leap centuries, and leap twenty-seventh 8-"QUAD" intervals if necessary. The discussion here is confined to a clock that will determine leap years and leap centuries. In order to do this, we will break down the clock as follows:

1. "SECOND" COUNTER
2. "MINUTE" COUNTER
3. "HOUR" COUNTER
4. "DAY-OF-THE-MONTH" COUNTER
5. "MONTH" SHIFT REGISTER
6. "DAY-MONTH-YEAR" LOGIC INTERFACE
7. "YEAR" BCD DECADE COUNTER
8. "LEAP-YEAR, LEAP-CENTURY, CENTURY" LOGIC INTERFACE

Before we can operate a clock properly, we need a good timing pulse of one pulse per second. The pulse accuracy is very important and will not properly operate the clock if variations can occur. For instance, an accuracy of 1% may be thought to be good, but a pulse of 1 second  $\pm 1\%$  may gain or lose as much as 15 minutes a day!!! An accuracy of 1 second  $\pm .1\%$  will gain or lose about 1.5 minutes a day, or about 45 minutes a month!!! A pulse of 1 second  $\pm .01\%$  may gain or lose 4.5 minutes a month which represents the accuracy of a cheap wristwatch. A pulse of 1 second  $\pm .001\%$  may gain or lose .45 minutes a month or about 5.4 minutes a year. A pulse of 1 second  $\pm .0001\%$  may gain or lose .54 minute, or about 33 seconds per year, which represents a high-priced precision wristwatch. A wall clock timed from the A. C. line has an accuracy better than  $\pm .000001\%$ . The most accurate timing device to date is the atomic clock and has an accuracy on the order of one part in ten billion ( $\pm .000000001$  seconds or  $\pm .00000001\%$ ). It should be very apparent that an ordinary timing pulse generator circuit is not accurate enough to drive a clock! However, a very accurate timing pulse from the A. C. line (see following diagram) is more than sufficient for this project. The 60-cycle A. C. must be converted to a square wave by use of a flip-flop and then be divided by 60 (i. e. , be divided by 10 and then by 6) to provide the 1-second pulse needed for this clock. A "second" or "minute" counter described in this section may be used to convert the 60-cycle sine wave to the required 1 cycle per second square wave.

10.12.1 THE TIME MACHINE CLOCK (Continued)



\* VALUES OF 500 $\Omega$  TO 1000 $\Omega$  MAY BE USED.

\*\* VALUES OF 50  $\mu$ f TO 300  $\mu$ f MAY BE USED.

ADJUST ABOVE VALUES IF FILTERING IS UNSATISFACTORY  
WITH VALUES SHOWN IN MAIN CIRCUIT.

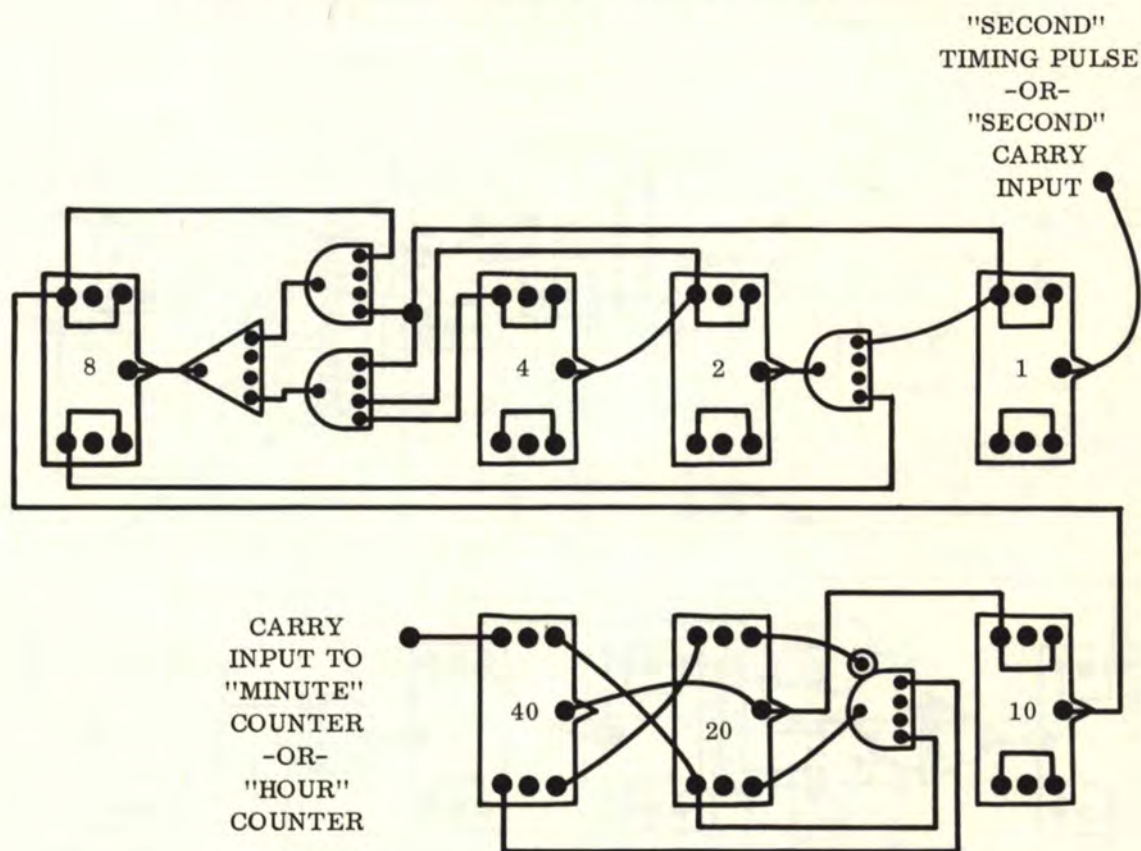
10.12.1 THE TIME MACHINE CLOCK (Continued)

Let us now proceed to build the clock. The timing pulse is fed directly into the first counter or the "second" counter. The "second" counter is a 2-stage BCD counter which is gated to count to 59 in BCD

$$\begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 1 \\ 1 \end{pmatrix}$$

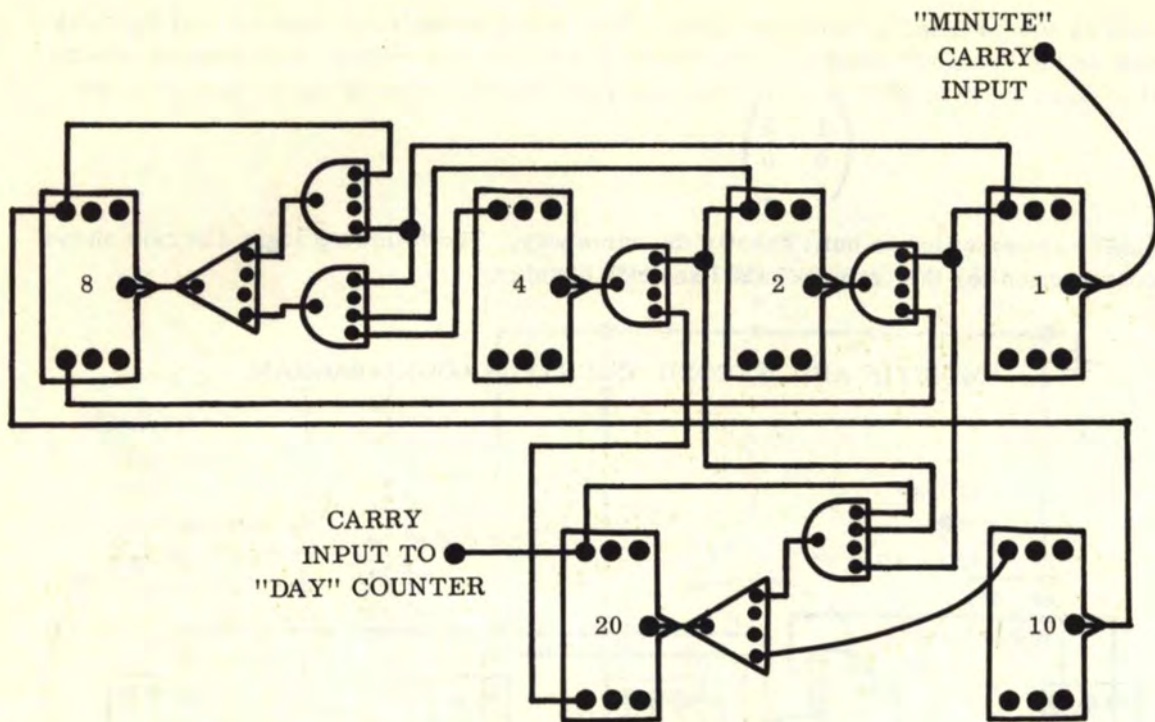
"minute" counter which is built exactly the same way. The following logic diagram shows the construction for the "minute" and "second" counter.

"MINUTE" AND "SECOND" COUNTERS LOGIC DIAGRAM

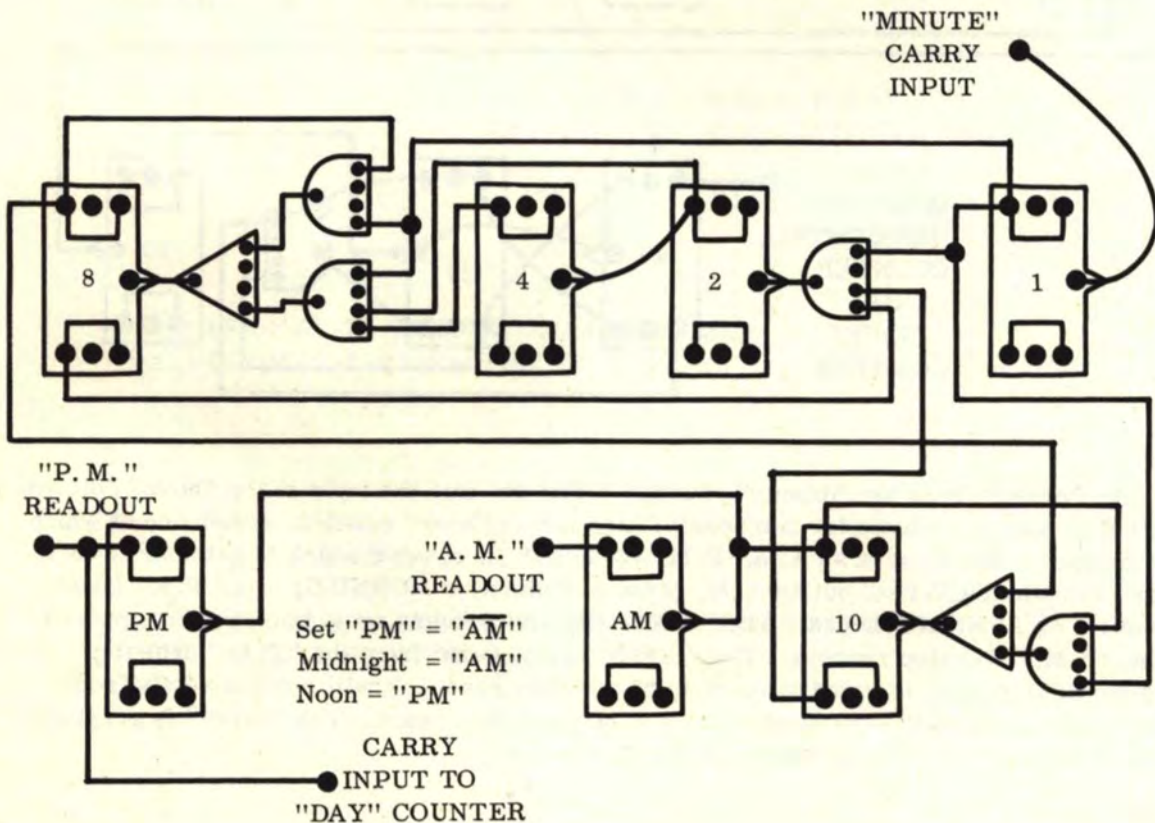


The "carry" from the "minute" counter is then fed into the input to the "hour" counter. Logic diagrams are shown for two possibilities for an "hour" counter, either one of which may be used. The first is an A.M. - P.M. 12-hour BCD counter which is gated to count to 11 and reset and will read out as A.M. (Ante Meridiem or MORNING) or as P.M. (Post Meridiem or EVENING) hours. The "A.M." flip-flop readout must be set to the opposite of the "P.M." flip-flop readout. The "carry" is generated from the "P.M." flip-flop. The 24-hour BCD counter is gated to count to 23 and then reset. It will count directly from "zero" hour (midnight) to 23 hours (11:00 P.M.) and then reset. The "carry" is generated by the last stage. The logic diagrams are as follows:

10.12.1 THE TIME MACHINE CLOCK (Continued)



24-HOUR COUNTER LOGIC DIAGRAM

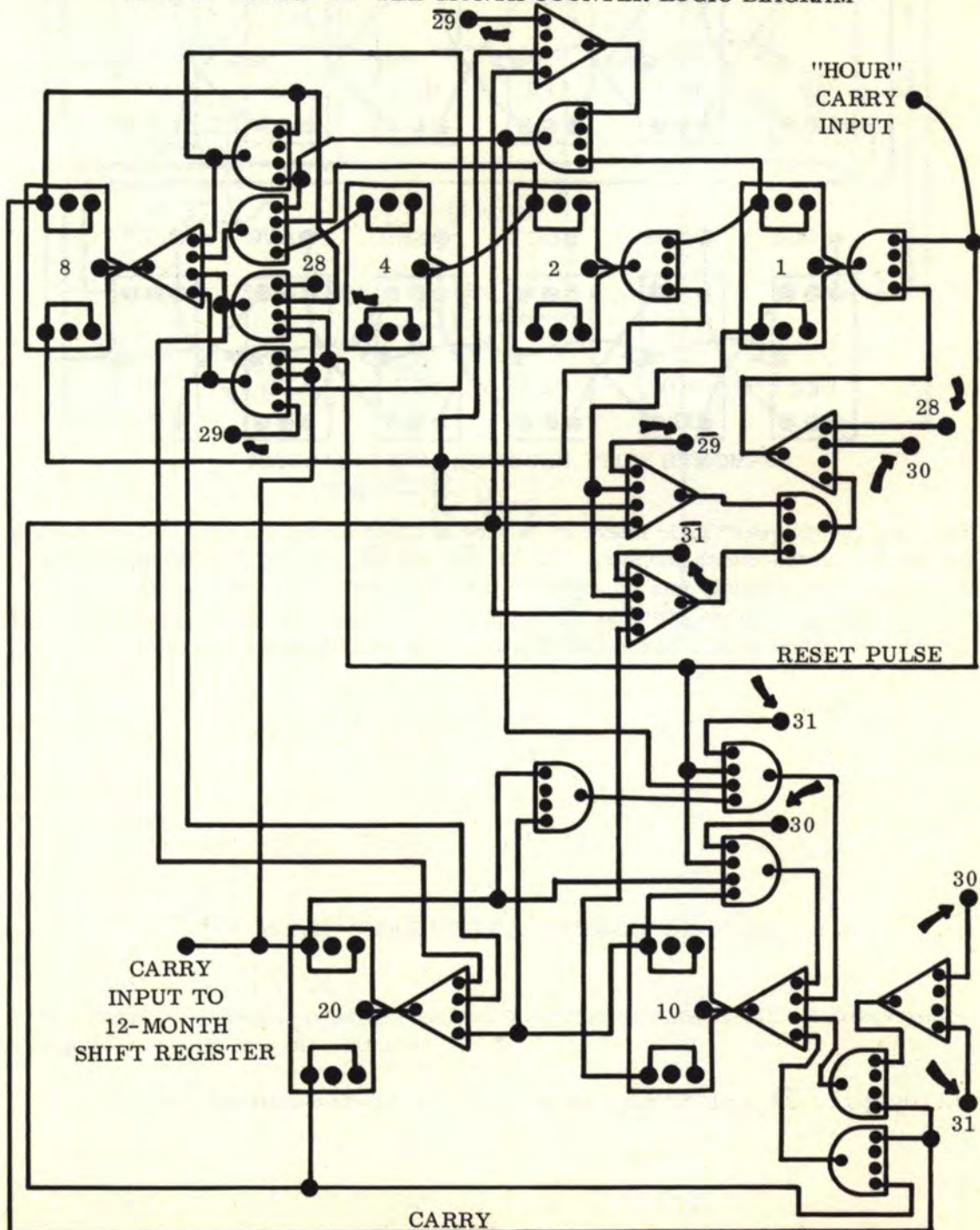


12-HOUR A.M.-P.M. COUNTER LOGIC DIAGRAM

10.12.1 THE TIME MACHINE CLOCK (Continued)

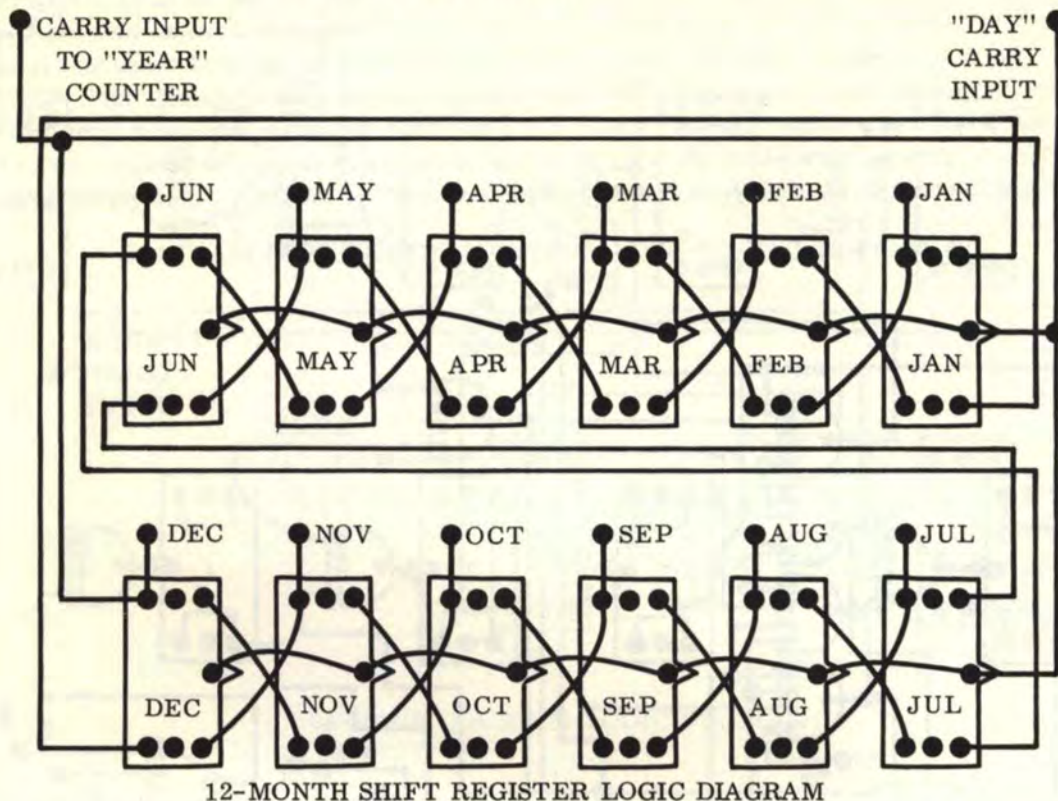
Now we come to the most complicated of all the counters, the 28-29-30-31 day-of-the-month BCD counter. This counter will sense "month" and "leap year" inputs and count to the proper number of days, depending on the month and whether or not we have a leap year. The "hour" "carry" is used to drive the day-of-the-month counter and is fed into the first stage. The "day-of-the-month" or "day" "carry" is generated at the last stage and is used to change the month when triggered. The open inputs to some gates labeled "28", "29", "30", "29", etc., are those "senses" which are generated by day-month-year logic interface. Those "senses" are wired directly into this counter and cause the decision as to how many days to which to count before resetting and changing the month. The logic diagram is as follows:

28-29-30-31 DAY-OF-THE-MONTH COUNTER LOGIC DIAGRAM



## 10.12.1 THE TIME MACHINE CLOCK (Continued)

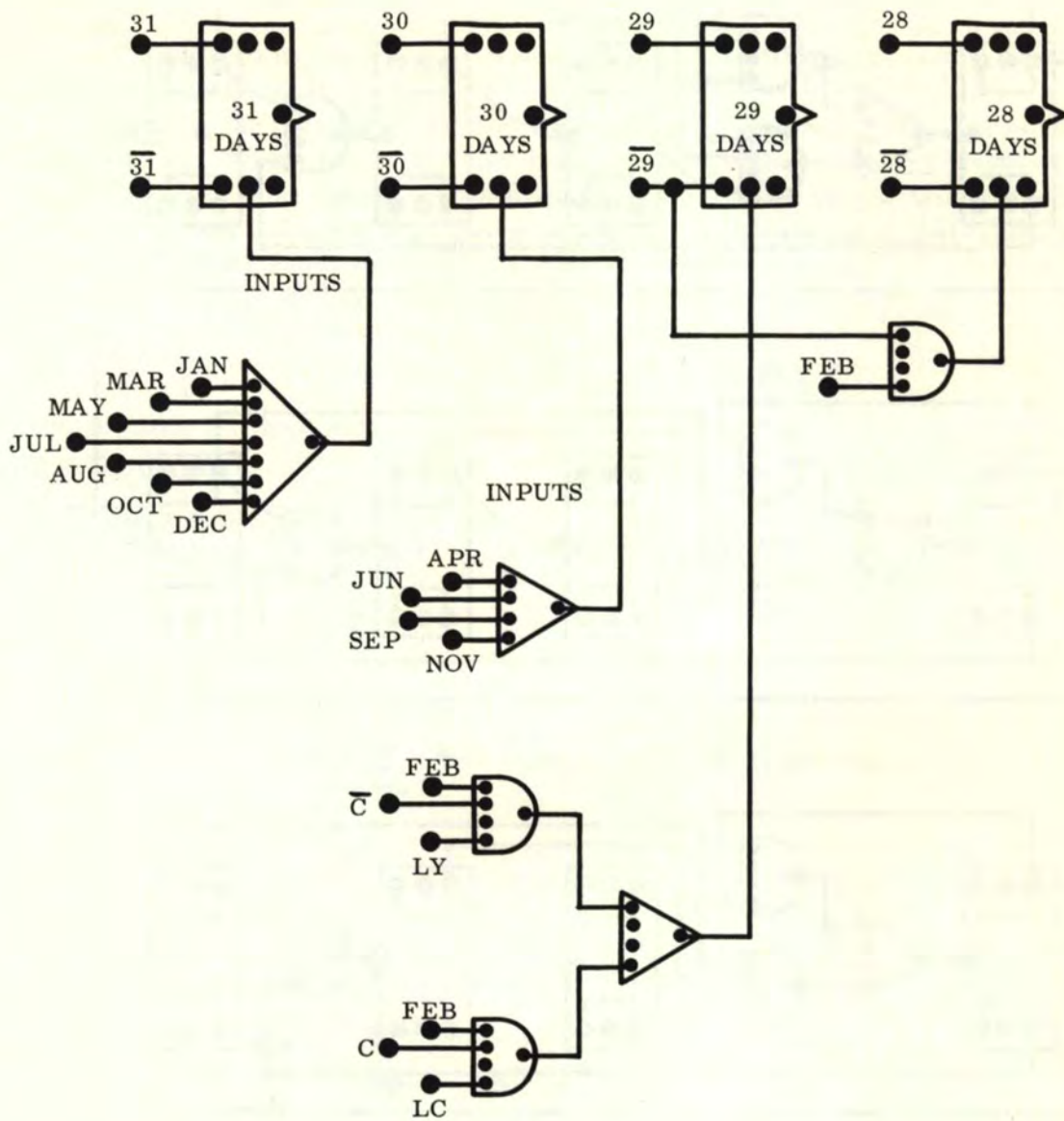
The 12-month shift register is very straightforward and needs no further explanation. It is triggered by the day-of-the-month "carry" and generates a "carry" from "DEC" to the "year" counter. The logic diagram is as follows:



The "day-month-year" logic interface is used to generate the decision-making information for the day-of-the-month counter. The  $\overline{28}$ ,  $\overline{29}$ ,  $\overline{30}$ ,  $\overline{31}$  outputs are shown and are wired directly into the positions shown in the "day-of-the-month" counter.\* The "month" interface inputs are also labeled with the name of the appropriate month. The LY, LC, C, and  $\overline{C}$  inputs will be discussed later. The logic diagram is as follows:

\*NOTE: Outputs  $\overline{28}$  and  $\overline{30}$  not used in "day-of-the-month" counter.

10.12.1 THE TIME MACHINE CLOCK (Continued)

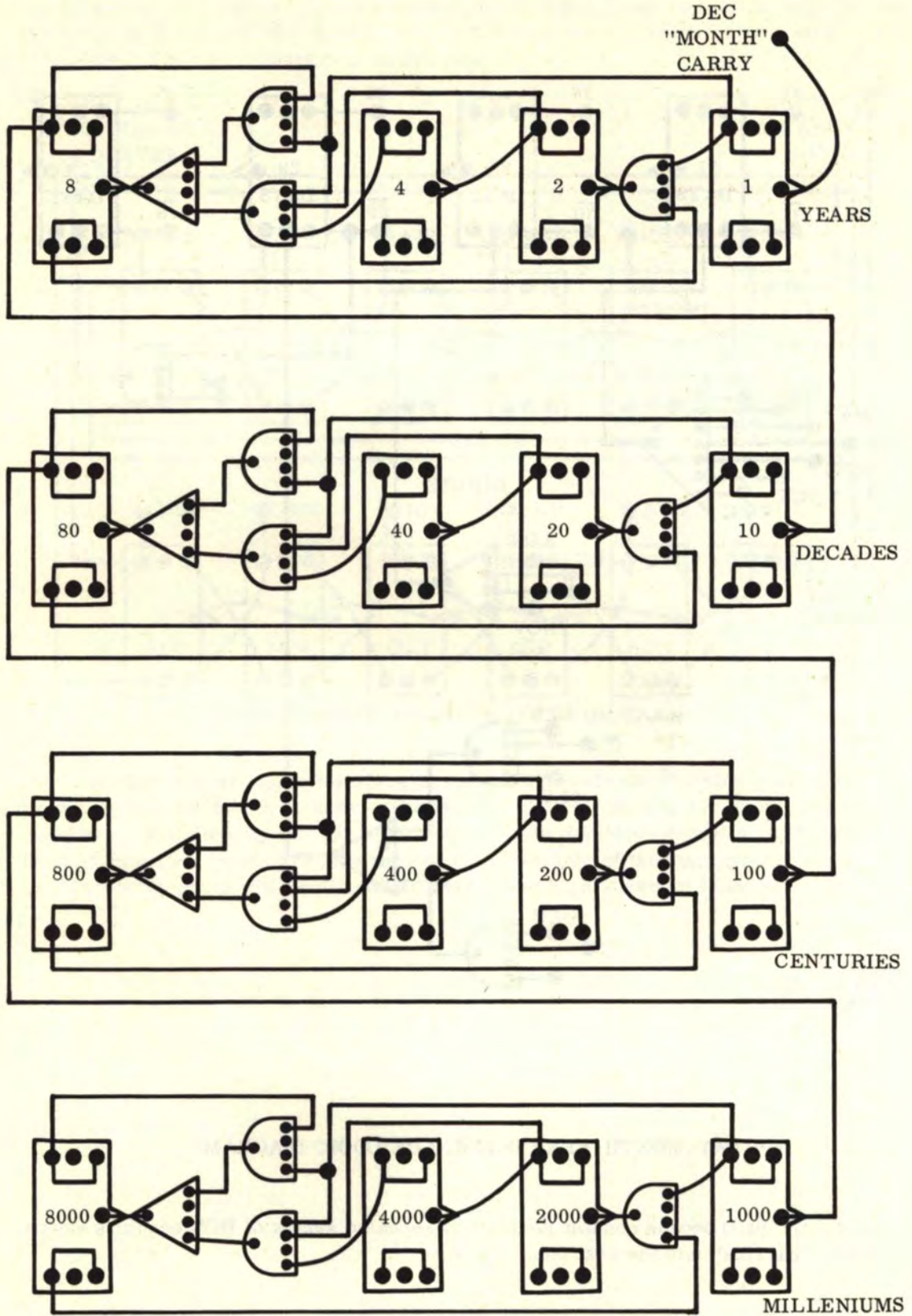


DAY-MONTH-YEAR INTERFACE LOGIC DIAGRAM

The "year" BCD decade counter consists of repeated stages of BCD counters where one counter "carries" into the next one.



10.12.1 THE TIME MACHINE CLOCK (Continued)



YEAR BCD DECADE COUNTER LOGIC DIAGRAM

10.12.1 THE TIME MACHINE CLOCK (Continued)

Now we come to the final logic interface—the "leap year (LY), leap century (LC), and century year (C)" logic interface. The inputs to this logic are shown with the numbers assigned the flip-flops in the BCD "year" decade counter. Note that we also need a "non-century" output ( $\bar{C}$ ). Essentially, we must test divisibility by 4, 100, and 400. A year is divisible by 4 if the first flip-flop in the "decade year" stage is "1" and the second "year" flip-flop is "on" and the first "year" flip-flop is "1" , or if the first "decade year" flip-flop is "off" and the first two "year" flip-flops are "off". Logically, this is shown by  $(\bar{10} \cdot \bar{2} \cdot \bar{1}) \vee (10 \cdot 2 \cdot \bar{1})$ . We have a century year if all flip-flops in the "year" and "decade year" are "off". Logically, this is shown by  $(\bar{80} \cdot \bar{40} \cdot \bar{20} \cdot \bar{10} \cdot \bar{8} \cdot \bar{4} \cdot \bar{2} \cdot \bar{1})$ . We have a "QUAD" century year (divisible by 400) if the first flip-flop in "millenium" is "1" and the second "century" flip-flop is "on" and the first "century" flip-flop is "1" ; or, if the first "millenium" flip-flop is "off" and the first two "century" flip-flops are "off"; and all "year" and "decade year" flip-flops must also be "off". Logically, this is shown by:

$$\left[ (\bar{1000} \cdot \bar{200} \cdot \bar{100}) \vee (1000 \cdot 200 \cdot \bar{100}) \right] \cdot (\bar{80} \cdot \bar{40} \cdot \bar{20} \cdot \bar{10} \cdot \bar{8} \cdot \bar{4} \cdot \bar{2} \cdot \bar{1})$$

The logic diagrams for the interface are as follows:

$$(\bar{10} \cdot \bar{2} \cdot \bar{1}) \vee (10 \cdot 2 \cdot \bar{1})$$

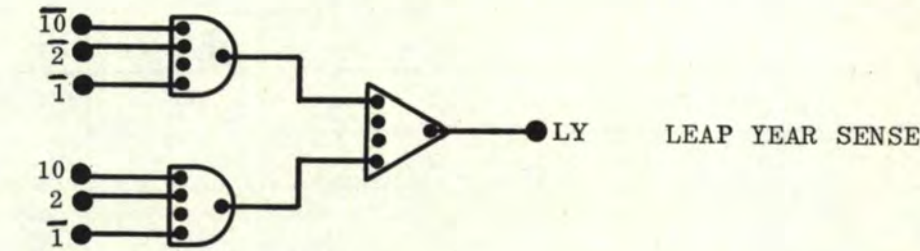
DIVISIBLE BY 4

$$(\bar{80} \cdot \bar{40} \cdot \bar{20} \cdot \bar{10} \cdot \bar{8} \cdot \bar{4} \cdot \bar{2} \cdot \bar{1})$$

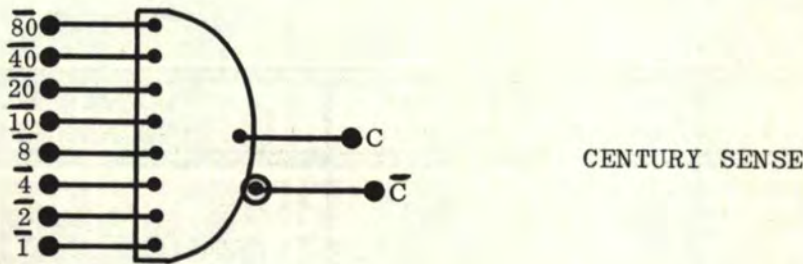
DIVISIBLE BY 100

$$\left[ (\bar{1000} \cdot \bar{200} \cdot \bar{100}) \vee (1000 \cdot 200 \cdot \bar{100}) \right] \cdot (\bar{80} \cdot \bar{40} \cdot \bar{20} \cdot \bar{10} \cdot \bar{8} \cdot \bar{4} \cdot \bar{2} \cdot \bar{1})$$

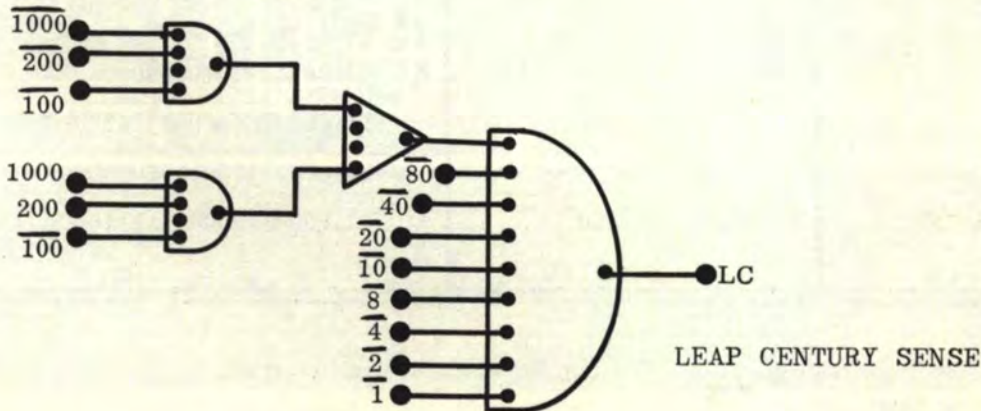
DIVISIBLE BY 400



LEAP YEAR SENSE



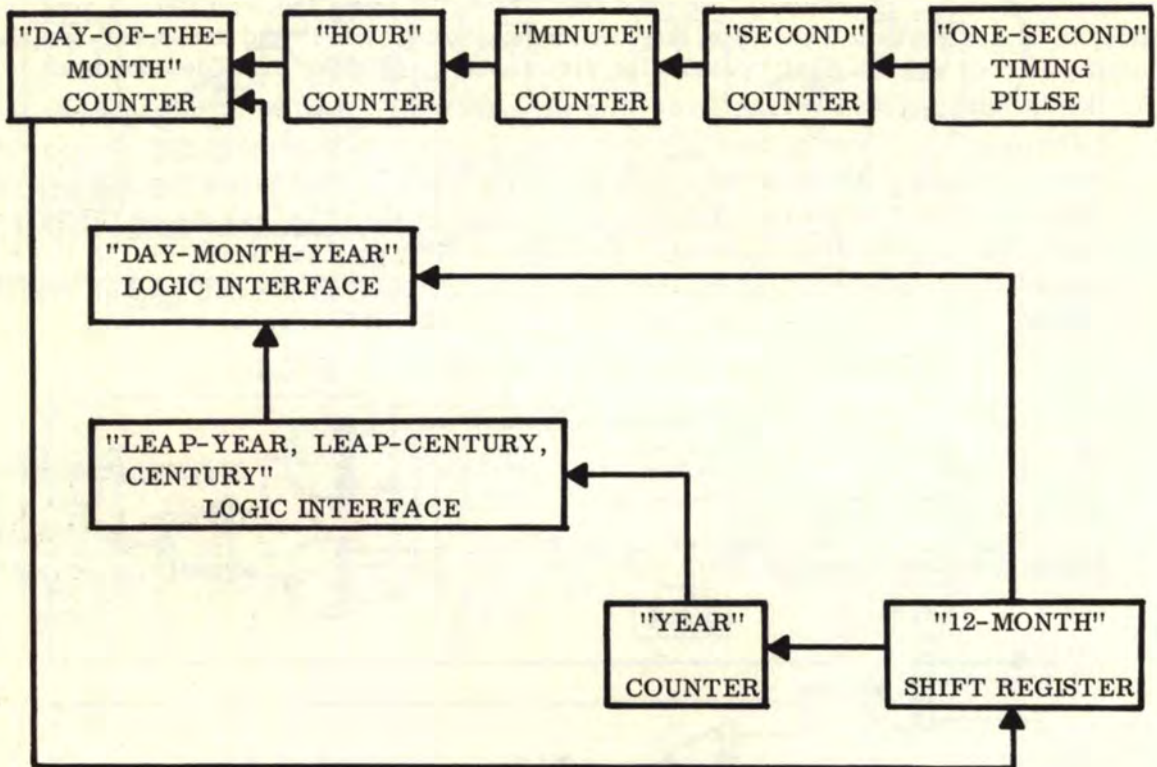
CENTURY SENSE



LEAP CENTURY SENSE

10.12.1 THE TIME MACHINE CLOCK (Continued)

Now that we have figured out how to build the specific parts of the clock, we will again illustrate in a graphic summary how they are put together.



We could also combine the clock and the 40-bit "up" counter, trigger them simultaneously, set the clock to the proper date, and the clock will then automatically read out the date for the binary display. If this is done, the following binary conversions for years, hours, and minutes into seconds will be very helpful.

UNIT	CONVERSION TO SECONDS (DECIMAL)	CONVERSION TO SECONDS (BINARY)
MINUTE	60	111100
HOURL	3,600	111000010000
DAY	86,400	10101000110000000
WEEK	604,800	10010011101010000000
YEAR	31,536,000	1111000010011001110000000
LEAP YEAR	31,622,400	1111000101000010100000000
4-YEAR CYCLE (1461 DAYS)	126,230,400	111100001100001111110000000

A suggested light configuration for a combined clock and 40-bit "up" counter is shown on the following page.

10.12.1 THE TIME MACHINE CLOCK (Continued)

"WE ARE BUT A SMALL GRAIN IN THE SANDS OF TIME" --- LIBE

●40 MAY 2, 19388 12:18:08 SUNDAY  RESET AUG 31, 36808 00:36:16 THURSDAY	●39 MAR 3, 10678 06:09:04 FRIDAY	●38 JAN 31, 6323 15:04:32 TUESDAY	●37 JUL 18, 4145 07:32:16 SUNDAY	●36 OCT 9, 3056 03:46:08 THURSDAY	●35 MAY 21, 2512 01:53:04 SATURDAY	●34 MAR 11, 2240 12:56:32 WEDNESDAY	●33 FEB 5, 2104 06:28:16 TUESDAY
●32 JAN 19, 2036 03:14:08 SATURDAY	●31 JAN 9, 2002 13:37:04 WEDNESDAY	●30 JAN 4, 1985 18:48:32 FRIDAY	●29 JUL 3, 1976 21:24:16 SATURDAY	●28 APR 2, 1972 10:42:08 SUNDAY	●27 FEB 15, 1970 17:21:04 SUNDAY	●26 JAN 23, 1969 08:40:32 THURSDAY	●25 JUL 13, 1968 04:20:16 SATURDAY
●24 APR 7, 1968 02:10:08 SUNDAY	●23 FEB 18, 1968 13:05:04 SUNDAY	●22 JAN 25, 1968 06:32:32 THURSDAY	●21 JAN 13, 1968 03:16:16 SATURDAY	●20 JAN 7, 1968 01:38:08 SUNDAY	●19 JAN 4, 1968 00:49:04 THURSDAY	●18 JAN 2, 1968 12:24:32 TUESDAY	●17 JAN 1, 1968 18:12:16 MONDAY
●16 JAN 1, 1968 09:06:08 MONDAY	●15 JAN 1, 1968 04:33:04 MONDAY	●14 JAN 1, 1968 02:16:32 MONDAY	●13 JAN 1, 1968 01:08:16 MONDAY	●12 JAN 1, 1968 00:34:08 MONDAY	●11 JAN 1, 1968 00:17:04 MONDAY	●10 JAN 1, 1968 00:08:32 MONDAY	●9 JAN 1, 1968 00:04:16 MONDAY
●8 JAN 1, 1968 00:02:08 MONDAY	●7 JAN 1, 1968 00:01:04 MONDAY	●6 JAN 1, 1968 00:00:32 MONDAY	●5 JAN 1, 1968 00:00:16 MONDAY	●4 JAN 1, 1968 00:00:08 MONDAY	●3 JAN 1, 1968 00:00:04 MONDAY	●2 JAN 1, 1968 00:00:02 MONDAY	●1 JAN 1, 1968 00:00:01 MONDAY

# THE TIME MACHINE

LEAP YEAR ●8 ●8 ●8 ●8 ●8 ●8	DEC ●8	AUG ●8	APR ●8	P.M. ●8	●8	●8	●8	●8
LEAP CENTURY ●4 ●4 ●4 ●4 ●4 ●4	NOV ●4	JUL ●4	MAR ●4	A.M. ●4	●4	●4	●4	●4
●2 ●2 ●2 ●2 ●2 ●2	OCT ●2	JUN ●2	FEB ●2	●2	●2	●2	●2	●2
●1 ●1 ●1 ●1 ●1 ●1	SEP ●1	MAY ●1	JAN ●1	●1	●1	●1	●1	●1
100,000 10,000 1,000 100 10 1	DAY OF MONTH		HOURS		MINUTES		SECONDS	
YEARS A.D.			MONTH		TIME CONTROL PANEL			

THIS EXHIBIT WAS DONATED BY LIBE COMPANY TO SHOW THE BASIC USE OF BINARY ELECTRONIC COMPUTER CIRCUITS FOR KEEPING TIME, AND TO FAMILIARIZE PEOPLE WITH THE BINARY NUMBER SYSTEM. THE CLOCK WAS STARTED AT 00:00:00 A.M. 1/1/68. THE 40-BIT ["BIT"-BINARY DIGIT] "UP" COUNTER ABOVE (BIT LIGHTS NUMBERED) SHOWS THE DATE AND TIME EACH LIGHT WILL FIRST COME ON. THE FIRST LIGHT IS TRIGGERED ONCE EACH SECOND. EACH SUCCESSIVE LIGHT TAKES TWICE THE TIME OF THE PRECEDING LIGHT TO COMPLETE A CYCLE AND WILL TURN ON ONLY WHEN THE PRECEDING LIGHT GOES OFF. NOTE THAT THE LAST LIGHTS TAKE A LONG LONG TIME TO COME ON! THE LIGHT PATTERN WILL NOT REPEAT ITSELF FOR NEARLY 35,000 YEARS! ONE CAN COME BACK AGAIN AND AGAIN TO OBSERVE THE CHANGING LIGHT PATTERN THROUGHOUT THE YEARS! -- THE TIME CONTROL PANEL CLOCK SHOWS THE CORRESPONDING YEAR, MONTH, DAY, HOUR, MINUTE, AND SECOND IN GREENWICH MEAN TIME (GMT OR "Z" TIME). IN ORDER TO TRANSLATE THE BINARY TIME CONTROL LIGHTS BACK TO OUR OWN NUMBER SYSTEM, NOTE THE NUMBERS BESIDE EACH LIGHT. FOR EACH LIGHT COLUMN, ADD UP ALL NUMBERS WHOSE CORRESPONDING LIGHTS ARE ON. A COLUMN WITH ALL LIGHTS OFF REPRESENTS A "0". BE SURE TO DROP BY AND OBSERVE THE ZERO-HOUR DATE CHANGES.

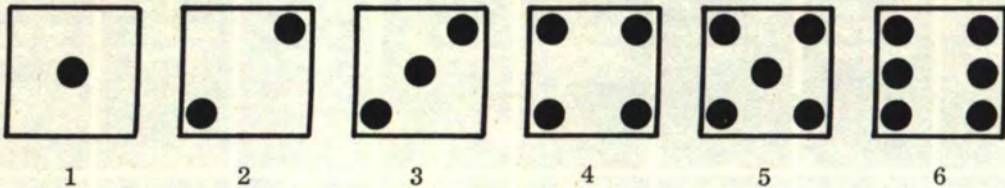
NOTE: The above is the front face of an exhibit which was donated to a San Francisco Bay Area science museum.

### 10.12.1 THE TIME MACHINE CLOCK (Continued)

In leaving this discussion on time, we leave one of the most fascinating subjects of nature. The material in this chapter represents only a brief introduction. We leave it to the imagination of the reader to explore the full depth of the time machine.

### 10.13 ELECTRONIC DICE

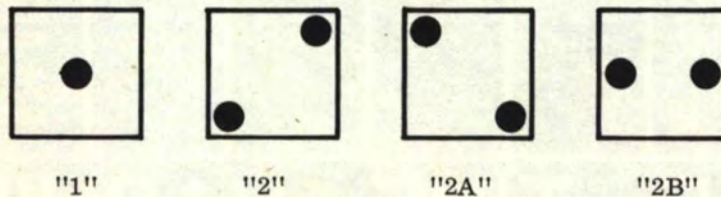
In order to set up electronic dice, we first note the die faces:



Now, if we superimpose the dots of the above die faces, we form a single "face" of seven dots as follows:



We can pick out any combination of dots from single face above to represent a die "face" display representing 1, 2, 3, 4, 5, or 6. In order to set the dots up as a counter, we will use the center dot for the "1" dot, the upper-right-lower-left diagonal pair for the "2" dots, the upper-left-lower-right diagonal for the "2A" dots, and the center side pair for the "2B" dots. They are represented as follows:



Using the above 4 representations, we can see that the die displays occur as follows:

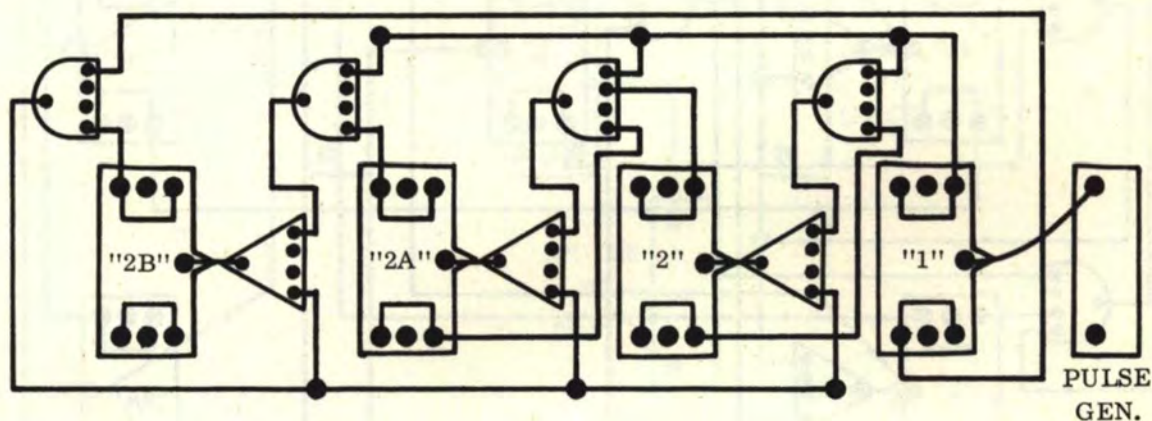
- 1 = "1"
- 2 = "2"
- 3 = "2" and "1"
- 4 = "2" and "2A"
- 5 = "2" and "2A" and "1"
- 6 = "2" and "2A" and "2B"

We now set up a truth table as follows:

10.13 ELECTRONIC DICE (Continued)

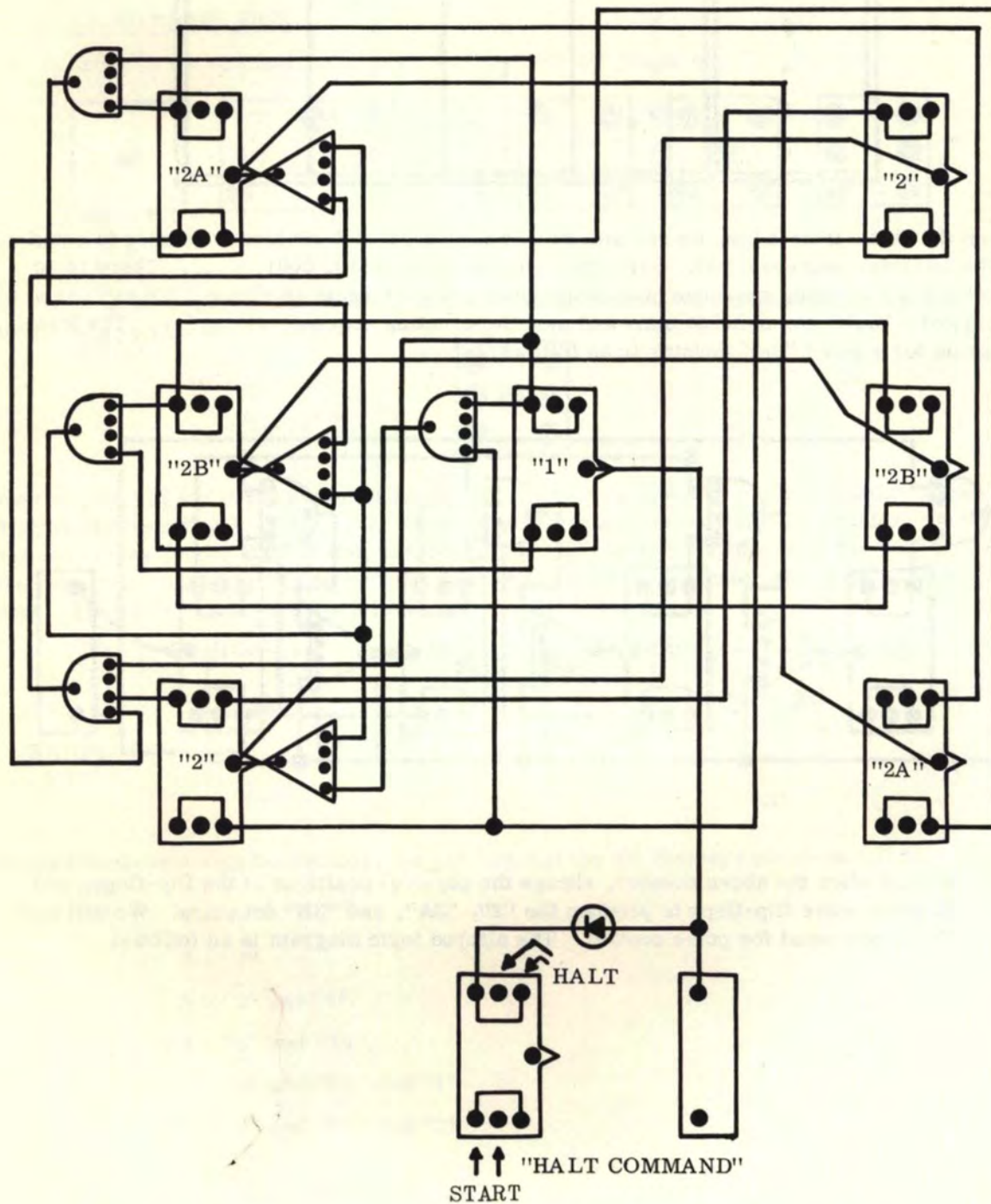
DIE DISPLAY	"2B" FACE	"2A" FACE	"2" FACE	"1" FACE
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	1	0
5	0	1	1	1
6	1	1	1	0

From the above truth table, we can see that we need a gated 4-bit binary counter to count in the following sequence: 0001, 0010, 0011, 0110, 0111, 1110, 0001..... There is no reset and the counting sequence must occur over and over again as shown. We can use a pulse and a "halt" command to start and stop the counting sequence at random. The logic diagram for a gated "die" counter is as follows:



Now we must alter the above counter, change the physical positions of the flip-flops, and then add three more flip-flops to produce the "2", "2A", and "2B" dot pairs. We will also add a "halt" command for pulse control. The altered logic diagram is as follows:

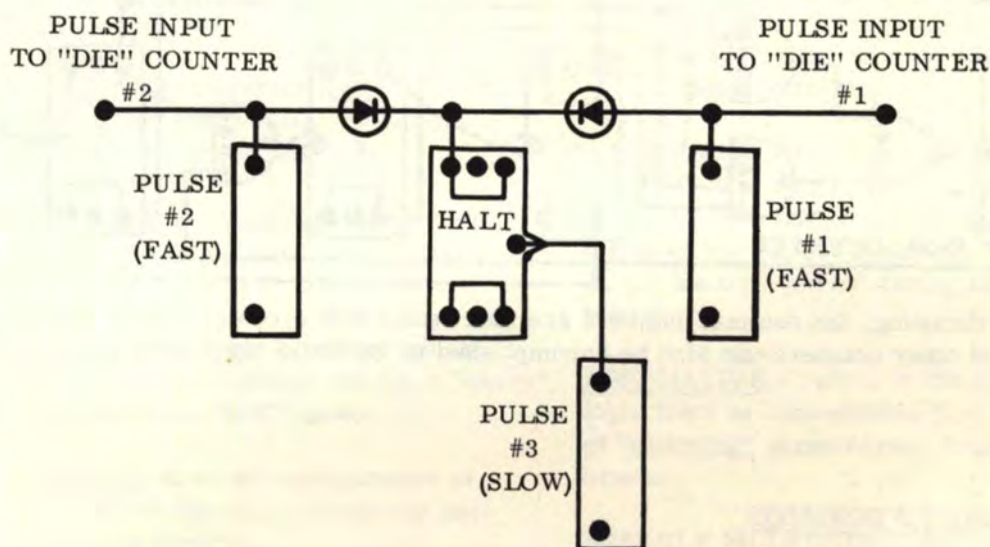
10.13 ELECTRONIC DICE (Continued)



### 10.13 ELECTRONIC DICE (Continued)

The counter will now read out directly like a die face when the pulse is stopped. Thus we can turn on the die counter by shorting together the two points indicated by arrows on the bottom of the "HALT" flip-flop, and stop the counter by shorting together the two points indicated by arrows at the top of the "HALT" flip-flop.

Now, if we were to build up two "die" counters, we could set up an automatic gating command using 3 pulse generators. Pulse #1 would control die #1 and pulse #2 would control die #2. Both pulse #1 and #2 should be fast pulses of greater than 100 pulses per second. Pulse #3 would control the "HALT" flip-flop and should be as slow as possible (less than one pulse per second, preferably 1 pulse every 5 or 6 seconds) so the die face displays can be perceived during the time the pulse has stopped. Then the counter will automatically start up again and once more stop at random die face displays. For two "die" counters, the pulse should be set up as follows:



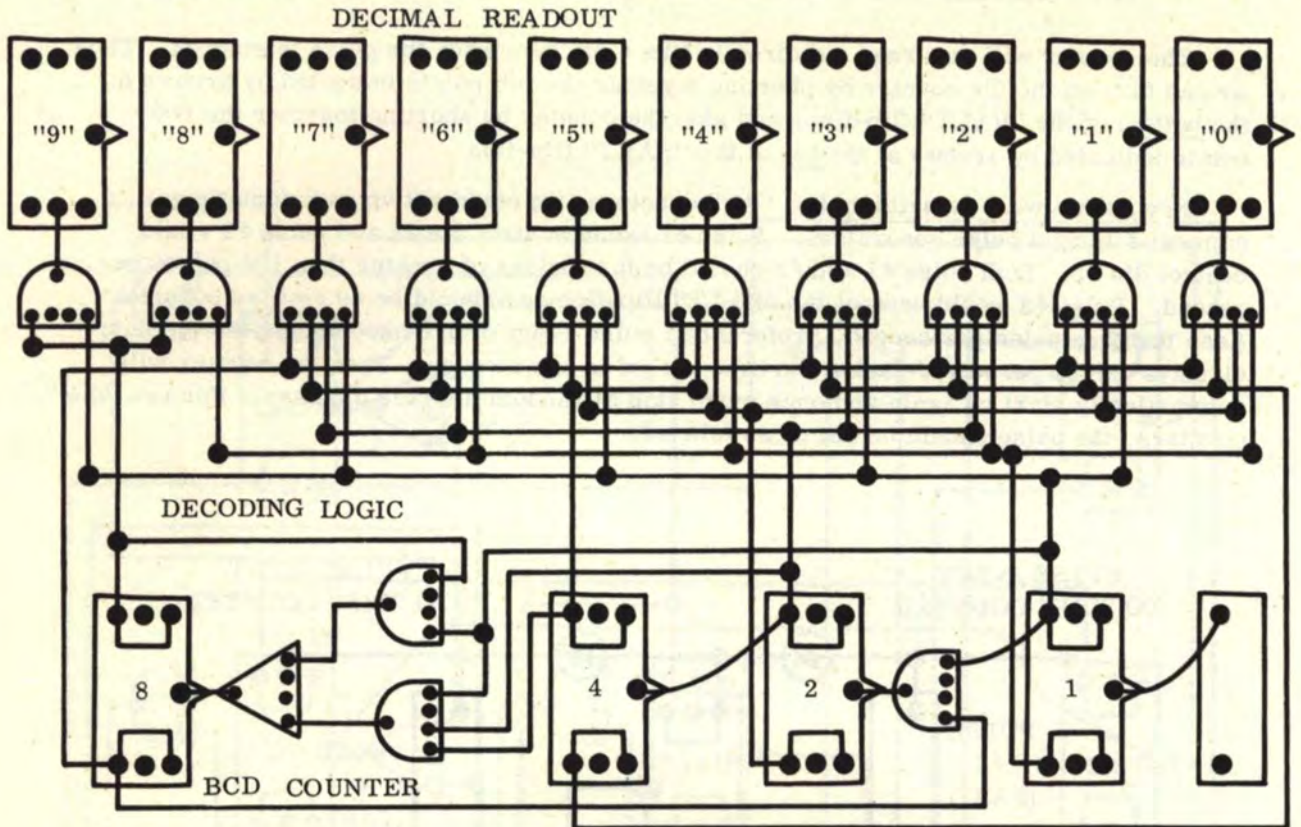
The two completed "die" counters now form a pair of electronic dice which will automatically "gate" count and stop at random displays. Happy gambling!

### 10.14 THE BINARY-TO-DECIMAL DECODER

The binary-to-decimal decoder will decode a 4-bit binary number from a BCD (binary coded decimal counter) and display any of 10 decimal positions from 0 to 9. The decoding can be done with "AND" gates as follows:



## 10.14 THE BINARY-TO-DECIMAL DECODER (Continued)



After decoding, the decimal numbers are then read out in a row of 10 flip-flops. Decoding of other counters can also be accomplished in the same manner by use of "AND" gates.

11. UNLIMITED HORIZONS

Now that the workings of the computer have been explained, the imagination is now ready to take over. One can now design with microelectronic integrated circuits using these same principles. The direction one can go from here is unlimited. Perhaps a large computer? Perhaps a special computer that will do something no other computers will do! The scope and horizons are unlimited!!!

## 12. GLOSSARY

ACCUMULATOR - the register (row of flip-flops) that displays an answer such as in the case of the "adder" and "subtractor" wiring projects.

ADD CONTROL - the "halt" command which is used to control (or stop) the addition process in an "adder" configuration.

ADDEND - the number which will be added to another number (in the process of addition).

ADDEND REGISTER - the register (row of flip-flops) in which the number to be added is entered (in the "adder" computer project).

ADDER - a wired configuration of flip-flops and/or logic gates which will perform the arithmetic process of addition.

ADDER, FULL - the complete logic circuitry which generates both a "sum" and a "carry" output.

ADDER, HALF - the logic circuitry which generates only a "sum" output and not a "carry" output; (also called a "SUM" gate).

ADDER, LOGIC - a wired configuration of logic gates and/or flip-flops which will perform addition (in binary).

ADDER, NON-GATED BINARY - a wired configuration of only flip-flops that will perform addition (i. e., that will allow a binary number to be added in one register and the answer to be displayed in a second register).

ADDER, SHIFT - a wired configuration of flip-flops and logic gates that consists of one full adder and three shift registers. Two shift registers shift the two numbers to be added through the full adder and the answer is shifted out in the third shift register.

ADDITION IDENTITY - a logic expression which defines a "sum", a "carry", or both for two or more binary numbers.

"AND" GATE - an electronic circuit which forms a logic gate (represented by a semicircle) whose output is a "1" only when all of its inputs are "1". The gate output is "0" for all other combinations of inputs.

"AND" LOGIC OPERATION - the logic operation denoted by the "dot" ( $\cdot$ ); the operation which answers the question: "Are all the facts 'TRUE'?" or "Are all inputs '1'?"

ANODE - the lead on a diode which receives positive (+) voltage, as opposed to the other lead, the cathode, which receives negative voltage.

ANTE MERIDIEM - before noon (abbreviated A. M.); denotes a time period of half a day starting from zero hour (midnight) up to, but not including, noon hour (i. e., the time period during the morning hours).

ASSOCIATIVE - refers to the grouping of logic facts or expressions (i. e., the use of "grouping" parentheses, brackets, or braces).

ASTABLE MULTIVIBRATOR - a pulse generator.

AUGEND - the number to which another number will be added.

BAR, OVERHEAD - ( $\overline{\quad}$ ) used to group together specific parts of "FALSE" logic expressions; used to denote a "FALSE" logic (i. e., to negate) logic fact.

BASE - the lead on a transistor which is used to control the flow of voltage and current through the transistor (see also COLLECTOR and EMITTER).

BCD - the abbreviation for "binary coded decimal".

12. GLOSSARY (Continued)

BINARY ADDITION - obtaining the binary sum of two binary numbers.

BINARY CODED DECIMAL - any decimal number (or decimal readout) that is controlled by a logic-gated binary counter which will count from 1 through 9 and then reset itself.

BINARY CODED DECIMAL (BCD) COUNTER - a binary counter with logic gating which will count up to binary nine (1001) and then reset itself to 0000; the counter used to control decimal system digital displays.

BINARY COMPLEMENT - that binary number which has the "1's" and "0's" of its digits interchanged from those of the original binary number. The sum of a binary number and its complement is a third binary number whose digits are all "1's".

BINARY DIVISION - the process of dividing a first binary number by a second binary number and obtaining a binary quotient.

BINARY FRACTIONAL - all binary digits to the right of the binary point. Similar to the common "decimal" digits to the right of the decimal point, except that the digits are all in binary.

BINARY INPUT - a binary "1" or "0" which is fed to a logic gate input or flip-flop input in order to perform a logic function or flip-flop triggering operation; any pin on a flip-flop or logic gate which can receive a binary output signal.

BINARY MULTIPLICATION - the process of multiplying two binary numbers together and obtaining a binary product.

BINARY NUMBER - a digit which represents a power of 2 and can only be a "1" or a "0"; a number of two or more digits which represent powers of 2 and each digit is a "1" or a "0".

BINARY OUTPUT - any pin of a flip-flop or logic gate that generates a binary "1" or "0" voltage signal which results from one or more input signals. Any pin on a pulse generator which generates a pulse voltage signal.

BINARY POINT (·) - similar to a "decimal" point, except that it is used to denote a binary fractional.

BINARY SUBTRACTION - the process of subtracting two binary numbers to obtain a binary difference.

BINARY SYSTEM - the number system to the base 2. Consists only of "1's" and "0's". Starting to the right, we have the "1's" column, then the 2's column just left of it, then the 4's column, the 8's column (each column to the left represents a number double the value of that to the right of it). Binary "1" means the column number is used; binary "0" means the column number is not used.

BISTABLE MULTIVIBRATOR - a flip-flop.

BIT - this is the shortened form for the words BINARY DIGIT. A "bit" is a binary digit.

BOOLEAN APPROACH - to impose the condition that all logic statements, reasons, conclusions, facts, etc., are either "TRUE" or "FALSE".

BORROW - the binary "1" which is removed from the adjacent column to the left by the subtraction of binary "1" from binary "0", leaving "1" in the column which was subtracted.

BORROW FEEDBACK - a "borrow", or binary "1" which is removed from a column other than the adjacent column to the left; the "borrow" of the last binary number at the extreme left of a register which is fed back into the first binary number (extreme right) of the same register.

BRACES { } - symbols which denote logic grouping along with parentheses and brackets.

BRACKETS [ ] - symbols which denote logic grouping along with parentheses and braces.

12. GLOSSARY (Continued)

"BRING-DOWN" - those numbers which are in the dividend (in division) or in the radicand (under the square root radical) which are tacked onto the remainder in order for a further division process to be performed; the process of tacking on the numbers mentioned above.

CANCEL - to remove a binary "1" from a flip-flop; to turn the flip-flop readout light off by shorting or touching together pins A and B. The process of removing a binary number from a register.

CAPACITOR - a two-lead electronic device which is used to store up electronic voltage charge.

CARRY - the binary "1" generated in the next column to the left by the addition of two binary "1's", leaving "0" in the column which was added.

CARRY FEEDBACK - a "carry", or binary "1" which is generated in a column other than the next column to the left; the "carry" of the last binary number at the extreme left of a register which is fed back into the first binary number (extreme right) of the same register.

CATHODE - the lead on a diode which receives negative (-) voltage, as opposed to the other lead, the anode, which receives positive voltage.

CENTURY - a time interval of 100 years.

CENTURY, QUAD - a time interval of 400 years.

CENTURY YEAR - a year which ends in "00" (is divisible by 100) such as 1700, 1800, 1900, 2000, and 2100.

CLEAR - to remove a binary "1" from a flip-flop; same as "cancel"; to remove a binary number from a register.

COLLECTOR - the lead on a transistor which receives voltage and current input (see also BASE and EMITTER).

COMMAND GENERATOR - a binary counter which is used in conjunction with "AND" gates in order to generate a repetitive series of command pulses. An "n"-bit counter can generate up to  $2^n$  repetitive command pulses.

COMPARATOR - will compare two numbers "A" and "B" and will determine whether A is greater than, greater than or equal to, equal to, less than or equal to, less than, or unequal to B depending on the type of comparator.

COMPARATOR, BINARY - will perform the same function as a comparator, except that the numbers being compared are in binary. A logic configuration which will sense any number of comparison inputs and provide a single binary comparison output.

COMPARATOR, "EQUAL TO" - will compare two binary numbers "A" and "B". If  $A = B$ , then the comparator output will be a "1". The comparator output will be "0" in all other cases.

COMPARATOR, "GREATER THAN" - will compare two binary numbers "A" and "B". If A is greater than B, then the comparator output will be a "1". The comparator output will be "0" in all other cases.

COMPARATOR, "GREATER THAN OR EQUAL TO" - will compare two binary numbers "A" and "B". If A is greater than or equal to B, then the comparator output will be a "1". The comparator output will be "0" in all other cases.

COMPARATOR, "LESS THAN" - will compare two binary numbers "A" and "B". If A is less than B, then the comparator output will be a "1". The comparator output will be "0" in all other cases.

COMPARATOR, "LESS THAN OR EQUAL TO" - will compare two binary numbers "A" and "B". If A is less than or equal to B, then the comparator output will be a "1". The comparator output will be "0" in all other cases.

12. GLOSSARY (Continued)

COMPARATOR, "UNEQUAL TO" - will compare two binary numbers "A" and "B". If A is unequal to B, the comparator output will be a "1". The comparator output will be "0" in all other cases.

COMPARATOR READOUT - the output of either a gate or a flip-flop (the comparator output) in a comparator computer project which indicates that a comparison is valid if it is a "1" and invalid if it is a "0".

COMPARISON DIAGRAM - a pictorial diagram square which is divided into separate cells. Each cell represents a comparison possibility for comparison. The total number of cells represents the total number of comparison possibilities. The number of cells that will appear for comparing numbers of "n" digits is  $2^{2n}$ .

COMPLEMENT, BINARY - that binary number which has the "1's" and "0's" of its digits interchanged from those of the original binary number. The sum of a binary number and its complement is a third binary number whose digits are all "1's".

COMPLEMENTARY TRANSFORMATION REGISTER - a register (row of flip-flops) which will transform, by a computer process, any number (entered into the register) into its complement and display the complement in the same register.

CONCLUSION - a specific logical decision or evaluation reasoned out from a set of logical facts.

CONNECTIONS, POWER - those connections on electronic computer units which supply the operating voltage needed for each unit to function.

COUNTER - a system of wired flip-flops whose binary digits will continuously increase by "1" in numerical order, or decrease by "1" in reverse order with each pulse command.

COUNTER, BINARY CODED DECIMAL (BCD) - a binary counter with logic gating which will count up to binary "nine" (1001) and then reset itself to 0000. Consists of four binary digits; the counter used to control decimal system digital displays.

COUNTER, "DOWN" - a counter that starts from any specified number and decreases its value by 1 with each pulse signal input. In other words, it "counts backwards".

COUNTER, GATED "UP-DOWN" - a single-register binary counter that can be controlled by logic gating so as to count "UP" or "DOWN" at the proper command.

COUNTER, "UP-DOWN" - a binary counter wired with extra logic gates so that the free-running counter will count alternately "UP" and then switch automatically to "DOWN" after resetting from the "UP" count. It will switch automatically back to "UP" after the "borrow" overflow is generated.

CROSS-PRODUCT - those individual products which are obtained by multiplying the multiplicand by each separate digit in the multiplier. The sum of these cross-products yields the total multiplied product.

CURRENT-LIMITED VOLTAGE - voltage present when a resistor is placed in series with the power source (the value of 1,000 ohms should be used for most current limiting described in this text).

DAY - a time interval of exactly 24 hours.

DEBUG - to eliminate problems that may occur in the wiring or operation of computer circuits.

DECADE - a time interval of ten years.

## 12. GLOSSARY (Continued)

DECIMAL SYSTEM - the most commonly used number system throughout the world; our ordinary numbering system which employs the use of the digits 1, 2, 3, 4, 5, 6, 7, 8, 9, and 0.

DECODER, BINARY-TO-DECIMAL - a configuration of logic gating which will convert any binary number to a decimal number. Usually used with a binary coded decimal counter.

DECOUPLE - to block out any electronic noise or interference which is generated by nearby circuits, the power source (power supply), or by external noise sources.

DECOUPLING CAPACITORS - capacitors which are connected directly across the power pins of each electronic circuit (flip-flop, gate, or pulse generator) in order to block out electronic noise or interference.

DEMORGAN'S THEOREMS - the two basic theorems which define the relationship between "AND" and "OR" logic operations.

DENOMINATOR - the bottom part of a fraction.

DIAGRAMS, LOGIC - drawings which illustrate how flip-flops and logic gates must be wired up to perform specific computer functions; the use of symbols to represent flip-flop, logic gate, and pulse generator electronic circuits without drawing out the full electrical schematic each time. Dots on the symbols are used to represent pin connections. Lines which connect these dots from one symbol to another represent all wire connections (power connections are not shown).

DICE, ELECTRONIC - a gated logic circuit which will produce random die displays on an electronic die face. Two of these circuits are used to comprise a pair of electronic dice.

DIFFERENCE - the answer which results from the subtraction of two numbers; the binary digit which represents the subtraction of one binary digit from another binary digit, ignoring the "borrow".

DIGIT - a single number.

DIGIT, BINARY - a single binary number (a "1" or a "0").

DIGIT, LEAST SIGNIFICANT - the digit at the extreme right of any number.

DIGIT, MOST SIGNIFICANT - the digit at the extreme left of any number.

DIGIT SENSE - the use of an "OR" logic gate to determine whether or not a number is present in a register.

DIGIT SENSE LOGIC - a gated logic circuit (usually a multi-input "OR" gate) which will determine if any numbers (or digits) are present in a register.

DIGITAL - the representation of a number in discrete terms of "on" or "off". The flip-flop register is a digital representation of a binary number. This is contrasted with the term ANALOG which is a representation of a number in terms of variable (not "on" or "off") voltage outputs.

DIODE - a two-lead electronic device which is used to block the flow of electronic voltage and current in one direction (see also ANODE and CATHODE).

DIODE WIRE - a wire with a diode spliced in the middle. The main use for the diode wire is to stop (or halt) the pulse from the pulse generator.

DIRECT OUTPUT DISPLAY - the use of a flip-flop to read out the output of a logic gate by wiring that output of the logic gate directly into pin E of the flip-flop.

DIRECTOR - an output pulse or signal which is capable of driving one or more inputs (or input followers).

DISPLAY, DIRECT OUTPUT - the use of a flip-flop to read out the output of a logic gate by wiring that output of the logic gate directly into pin E of the flip-flop.

## 12. GLOSSARY (Continued)

**DISTRIBUTE** - to logically "multiply" a logic expression; to perform an indicated logic operation on a group of logic facts, one at a time, when such facts are grouped by parentheses and the logic operator is outside the parentheses. The parentheses are then removed after the "distribute" operation is performed.

**DISTRIBUTIVE LAW** - any definition that sets a rule for logic distribution operations.

**DIVIDE CONTROL** - a wired configuration of flip-flops with or without logic gates which generates pulse signals to perform the division process (in the "divider" computer project).

**DIVIDEND** - the number to be divided (in the arithmetical process of division).

**DIVIDEND REGISTER** - the register (row of flip-flops) in which is entered the number to be divided (in the "divider" computer project).

**DIVIDER, LOGIC** - a configuration of gated logic circuits and registers which will divide one number by another.

**DIVISOR** - the number by which a second number is divided (in the arithmetical process of division).

**DIVISOR REGISTER** - the register (row of flip-flops) in which is entered the number by which a second number is being divided (in the "divider" computer project).

**DOUBLING, BINARY** - the process of adding an extra "0" to the left of a given binary number. Adding on a "0" to the left of any binary number will always result in doubling that number.

**"DOWN" COUNTER** - a counter that starts from any specified number and decreases its value by 1 with each pulse signal input. In other words, it "counts backwards".

**DOWN-SWING, VOLTAGE** - the change of the output of a pulse generator, flip-flop, or logic gate from a specific voltage, to zero volts; the voltage change which will cause flip-flop triggering.

**EMITTER** - the lead on a transistor where the controlled voltage and current flow out (see also COLLECTOR and BASE).

**END-AROUND** - the transfer of a pulse command from the last flip-flop (extreme left) in a register to the first flip-flop (extreme right). The signal generated by the last flip-flop, which would normally control another flip-flop to the left (beyond the last one, if it were present), instead is used to control the first flip-flop.

**ENTER** - to place a binary "1" in a flip-flop (i. e. , turn the flip-flop light on) by touching pins D and E together. To place a binary number in a flip-flop register by changing the proper flip-flops to "1's".

**"EOR" GATE** - an "exclusive or" gate. An electronic circuit which forms a logic gate whose output is a "1" if, and only if, one of its binary inputs is a "1". The gate output is "0" for all other combinations of inputs.

**"EOR" LOGIC OPERATION** - exclusive "or" logic operation. The logic operation denoted by the "triangle" ( $\nabla$ ); the operation which answers the question: "Is there only one 'TRUE' fact?" or "Is only one input a '1'?"

**EQUALITY, LOGIC** - defines two or more logic expressions, equations, or facts to be equivalent or identical. This condition is denoted by the "equal" sign (=).

**EXCLUSIVE OR** - (see "EOR".)

**FACT, LOGIC** - a statement (which may be true or false) which is used to reason out a conclusion.

**FACTORED EXPRESSION** - a logic expression whose individual terms are not distributed.

**FALL TIME** - the time required for a voltage down-swing to occur. The time which elapses during the change from some specific voltage, to zero volts.

## 12. GLOSSARY (Continued)

**FALSE** - the condition of not being true; the condition of a flip-flop or logic gate being in the "0" or "off" state (flip-flop readout light off); an input that is a "0". The side of the flip-flop opposite the readout light (also all flip-flop outputs and inputs, pins D, E, F, opposite the readout light).

**"FALSE" DIRECTOR OUTPUT** - the output pin (pin D) on the "FALSE" side of the flip-flop; also known as the "FALSE" output of the flip-flop.

**"FALSE" FOLLOWER INPUT** - the input pin (pin F) on the "FALSE" side of the flip-flop which causes steering depending upon whether or not an input voltage is present.

**"FALSE" OUTPUT** - the output pin (pin D) on the "FALSE" side of the flip-flop; also known as the "FALSE" director output of the flip-flop.

**"FALSE" SIDE** - the side of the flip-flop opposite the readout light.

**FEED** - to enter an input.

**FEEDBACK** - a pulse command which is generated by one position in a register (either from a flip-flop or a gate) and is used to trigger a random position in that same register (i. e., a non-adjacent position). Usually referred to as a pulse command which is generated by the last position on the left and used to trigger the first position on the right.

**FLIP-FLOP** - an electronic computer circuit (also called a BISTABLE MULTIVIBRATOR) that is capable of displaying a binary number as either a "1" or a "0" (in this case, with a readout light that is on or off) and will change state with the proper pulse command.

**FLOW CHART** - a block pictorial representation (each block containing a brief description) of individual steps which take place in a more complicated computer operation.

**FLOW DIAGRAM** - see FLOW CHART.

**FLUSH LEFT** - the condition of two numbers of unequal digit length written, one below the other, so that first digits at the extreme left line up. The other digits to the right are then written successively one below the other until there are no more digits.

**FLUSH RIGHT** - the condition of two numbers of unequal digit length written, one below the other, so that the first digits at the extreme right line up. The other digits to the left are then written successively one below the other until there are no more digits.

**FOLLOWER INPUTS** - the input pins on the flip-flop (pins C and F) which will or will not cause a change of state to take place, depending on whether or not an input voltage is present, with the next pulse command. See STEERING INPUTS.

**FRACTION** - two numbers, written one over the other, separated by a horizontal line to show that a division can be performed; the ratio between two numbers. The top number is called the NUMERATOR and the bottom number is called the DENOMINATOR.

**FRACTION, BINARY** - a fraction whose numerator and denominator are expressed in the binary (base 2) number system.

**FRACTION, DECIMAL** - a fraction whose numerator and denominator are expressed in the decimal (base 10) number system.

**FRACTIONAL** - that portion of a number which is written to the right of a "point" (i. e., "decimal point" or "binary point") which represents that part of the number less than unity (1). This part of the number is represented by successive negative powers of the base system being used.

**FRACTIONAL, BINARY** - that portion of a binary number which is written to the right of the binary point to represent that part of the number less than unity (1). This part of the number is represented by successive negative powers of 2 (i. e.,  $1/2$ ,  $1/4$ ,  $1/8$ ,  $1/16$ ,  $1/32$ ,  $1/64$ , etc.).



12. GLOSSARY (Continued)

FRACTIONAL, DECIMAL - that portion of a decimal number which is written to the right of the decimal point to represent that part of the number less than unity (1). This part of the number is represented by successive negative powers of 10 (i. e.,  $1/10$ ,  $1/100$ ,  $1/1000$ ,  $1/10000$ , etc.).

FRACTIONAL, INFINITE - a fractional obtained by repeated division of two finite numbers. The repeated division goes on indefinitely and the digits in the fractional repeat themselves in a given sequence.

FRACTIONAL, IRRATIONAL - a fractional obtained, for instance, when extracting the root of a number which is not a perfect power, or by calculating constants such as " $\pi$ " and "e"; the digits are calculated in endless sequence, but do not demonstrate any pattern of repetition.

FULL ADDER - the complete logic circuitry which generates both a "sum" and a "carry" output.

FULL SUBTRACTER - the complete logic circuitry which generates both a "difference" and a "borrow" output.

FUNCTION, LOGIC - an expression which contains one or more logic operators indicating logic operation(s) to be performed; a specific logic operation such as "AND" or "OR".

GATE, "AND" - an electronic circuit which forms a logic gate (represented by a semi-circle) whose output is a "1" only when all of its inputs are "1". The gate output is "0" for all other combinations of inputs.

GATE, "EOR" - an "exclusive or" gate. An electronic circuit which forms a logic gate whose output is a "1" if, and only if, one of its binary inputs is a "1". The gate output is "0" for all other combinations of inputs.

GATE, INVERTER - an electronic circuit which forms a logic gate with only a single input and whose output is exactly opposite of its input.

GATE, LOGIC - an electronic circuit which performs a logic operation (i. e., "AND", "OR", etc.).

GATE, "NAND" - an electronic circuit which forms a logic gate whose output is a "0" only when all of its inputs are "1". The gate output is "1" for all other combinations of inputs. An inverted "AND" gate. Represents "NOT AND" or a negative "AND" gate logic function.

GATE, "NOR" - an electronic circuit which forms a logic gate whose output is a "1" only when all of its inputs are "0". The gate output is "0" for all other combinations of inputs. An inverted "OR" gate. Represents "NOT OR" or a negative "OR" gate logic function.

GATE, "OR" - an electronic circuit which forms a logic gate (represented by a triangle) whose output is "0" only when all of its inputs are "0". The gate output is "1" for all other combinations of inputs.

GATE, "SUM" - an electronic circuit which forms a logic gate whose output is a "1" only when the parity of all "1" inputs, regardless of how many, is odd. The gate output is "0" if the parity of all "1" inputs is even.

GATE NOMENCLATURE - names for logic gates determined by the following general format: (1) Identify the gate logic function. (2) Identify the number of identical logic gates on a printed circuit card. (3) Identify the number of inputs per gate.

GENERATOR, COMMAND - a binary counter which is used in conjunction with "AND" gates in order to generate a repetitive series of command pulses. An "n"-bit counter can generate up to  $2^n$  repetitive command pulses.

GENERATOR, PULSE - an electronic circuit (also called an ASTABLE MULTIVIBRATOR) that generates a repetitive square wave voltage output that swings from "1" (voltage present) to "0" (no voltage); the command device that makes the flip-flops work automatically.

## 12. GLOSSARY (Continued)

GROUP, TO - to use parentheses, brackets, braces, or an extension of the "overhead bar" to set off a part of a logic function from the rest of the expression, or to clarify (by isolating part(s) of a logic function) where ambiguities can exist.

HALF ADDER - the logic circuitry which generates only a "sum" output and not a "carry" output; (also called a "SUM" gate).

HALF SUBTRACTER - the logic circuitry which generates only a "difference" output; and not a "borrow" output; (also called a "SUM" gate) the half-adder and half-subtractor are identical as are the "sum" and "difference" outputs.

HALT - to stop the command pulse generator at the proper precise instant (accomplished with the use of a diode wire and a control flip-flop).

HALT COMMAND - the command signal which turns off the pulse generator (stops the pulse). This command signal triggers a control flip-flop (see "HALT") which in turn changes state and grounds out the command pulse signal through a diode wire.

HALT CONTROL - the flip-flop used with the diode wire to control the command pulse generator signal.

HANG UP - the inability of a flip-flop to be triggered from a pulse command. Examples are the use of an "AND" gate for triggering when at least one input to that "AND" gate is always "0", or the use of an "OR" gate for triggering when at least one input to that "OR" gate is always "1".

IDENTITIES, FUNDAMENTAL - logic equalities of accepted fact (i. e., need not be proven) which define basic relationships between two or more logic quantities.

IDENTITY, LOGIC - an expression of equality which denotes that for all possibilities of "1" and "0", the "TRUTH" tables are the same for the part of the expression to the left of the equality as for the part of the expression to the right of the equality.

INFINITE FRACTIONAL - a fractional obtained by repeated division of two finite numbers. The repeated division goes on indefinitely and the digits in the fractional repeat themselves in a given sequence.

INPUT, BINARY - a binary "1" or "0" which is fed to a logic gate input or flip-flop input in order to perform a logic function or flip-flop triggering operation; any pin on a flip-flop or logic gate which can receive a binary output signal.

INTRINSIC PROBLEM - the inability of a computer circuit or wiring project to work properly even though there is apparently no mistake in the circuit construction or in the wiring.

INVERTER GATE - an electronic circuit which forms a logic gate with only a single input and whose output is exactly opposite of its input.

IRRATIONAL FRACTIONAL - a fractional obtained, for instance, when extracting the root of a number which is not a perfect power, or by calculating constants such as " $\pi$ " and "e"; the digits are calculated in endless sequence, but do not demonstrate any pattern of repetition.

"L" LINE - an "L"-shaped line (reversed "L") with a long bottom which is drawn to the left of each square root remainder and is used to separate each new square root divisor from its associated remainder.

LABELS - identifying statements or numbers which are used to describe flip-flop or logic gate relative positions; also, identifying statements or numbers which describe register, command, process, control, or project functions. These statements and/or numbers may be labeled on each project (after completion) for clarification.

LAW, LOGIC - a relation which is proved or assumed to hold between other logic expressions (the relationship is expressed by a logic equality).

12. GLOSSARY (Continued)

LEAKY - the condition of a diode or transistor that does not completely block the flow of voltage and current in the direction where the flow should be blocked.

LEAP YEAR - a year which has 366 days instead of 365 days. Occurs in all non-century years divisible by 4, and in all century years divisible by 400.

LOGIC - the science of reasoning (i. e. , making use of known facts to reason out a conclusion).

LOGIC DIAGRAMS - drawings which illustrate how flip-flops and logic gates must be wired up to perform specific computer functions; the use of symbols to represent flip-flop, logic gate, and pulse generator electronic circuits without drawing out the full electrical schematic each time. Dots on the symbols are used to represent pin connections. Lines which connect these dots from one symbol to another represent all wire connections (power connections are not shown).

LOGIC GATE - an electronic circuit which performs a logic operation (i. e. , "AND", "OR", etc.).

LOGIC GATING - using logic gates to perform logic operations on flip-flop outputs.

LOGIC NOTATION - the use of symbols such as the "dot" ( $\cdot$ ), "wedge" ( $\wedge$ ), "plus" (+), "triangle" ( $\nabla$ ), "equal" (=), and the overhead "bar" ( $\bar{\quad}$ ) in conjunction with appropriate capital letters (to represent logic facts) to denote logic operation(s).

LOGIC OPERATOR - a symbol which indicates that a logic function (such as "AND" or "OR") is to be performed. Examples of logic operators are: The "wedge" ( $\wedge$ ), the "dot" ( $\cdot$ ), the "plus" (+), the "triangle" ( $\nabla$ ) and the overhead "bar" ( $\bar{\quad}$ ).

MEMORY - a wired configuration of flip-flops which stores binary information. The accumulator, which displays and stores an answer, can be considered a memory.

MILLENIUM - a time interval of 1000 years.

MINUEND - that number from which another number is to be subtracted.

MULTIPLICAND - that number which is to be multiplied by a second number.

MULTIPLICAND REGISTER - the register containing the number which is to be multiplied by a second number (in the "multiplier" computer projects).

MULTIPLICATION CONTROL - a wired configuration of flip-flops with or without logic gates which generates pulse signals to perform the multiplication process (see MULTIPLIER CONTROL).

MULTIPLIER - that number by which another number is to be multiplied.

MULTIPLIER, CUMULATIVE-ADDITION - a wired computer project which performs multiplication by successive steps of addition rather than by full logic gating.

MULTIPLIER, FULL LOGIC - a wired computer project which performs multiplication only with the use of logic gates. There are no intermediate steps of addition.

MULTIPLIER CONTROL - that register (in the cumulative-addition multiplier) which generates pulse signals for the intermediate steps of addition and stops the multiplication process when all the addition steps have been completed.

MULTIPLIER REGISTER - the register containing the number by which another number is to be multiplied (in the "multiplier" computer projects).

MULTIPLIER SENSE - logic gates (in the cumulative-addition multiplier) which determine when the last digit to the right in the multiplier register is a "1" and, if so, causes an intermediate step of addition to take place.

"NAND" GATE - an electronic circuit which forms a logic gate whose output is a "0" only when all of its inputs are "1". The gate output is "1" for all other

12. GLOSSARY (Continued)"NAND" GATE (continued)

combinations of inputs. An inverted "AND" gate. Represents "NOT AND" or a negative "AND" gate logic function.

"NAND" LOGIC OPERATION - the logic operation which is the negative of "AND" logic operation and is denoted by a "dot" and an overhead "bar" across the entire expression; the operation which answers the question: "Is at least one fact 'FALSE'?" or "Is at least one input '0'?"

NEGATION - employs the use of an overhead "bar" to denote "not A" when placed over logic fact A.

NEGATION, DOUBLE - the process of inverting twice, or inverting the negative of a logic fact A (i. e. , "not not A") to revert it to the positive (i. e. , the original fact "A").

NOISE, ELECTRONIC - abrupt power voltage fluctuations or line voltage fluctuations which cause unwanted flip-flop triggering. Noise occurs during the entering and canceling of binary numbers. When this happens, other flip-flops may change state for no observed apparent reason.

"NOR" GATE - an electronic circuit which forms a logic gate whose output is a "1" only when all of its inputs are "0". The gate output is "0" for all other combinations of inputs. An inverted "OR" gate. Represents "NOT OR" or a negative "OR" gate logic function.

"NOR" LOGIC OPERATION - the logic operation which is the negative of "OR" logic operation and is denoted by a "wedge" and an overhead "bar" across the entire expression; the operation which answers the question: "Are all the facts 'FALSE'?" or "Are all inputs '0'?"

NORMAL YEAR - a time interval of exactly 365 days.

NOTATION, LOGIC - the use of symbols such as the "dot" ( $\cdot$ ), "wedge" ( $\vee$ ), "plus" ( $+$ ), "triangle" ( $\nabla$ ), "equal" ( $=$ ), and the overhead "bar" ( $\bar{\quad}$ ) in conjunction with appropriate capital letters (to represent logic facts) to denote logic operation(s).

NUMERATOR - the top part of a fraction.

"OFF" - the condition of a flip-flop or logic gate whose output is a "0".

"OFF" STATE - the condition of a flip-flop light being off; a flip-flop in the "FALSE" or "0" state; the condition of a logic gate whose output is "0".

"ON" - the condition of a flip-flop or logic gate whose output is a "1".

"ON" STATE - the condition of a flip-flop light being on; a flip-flop in the "TRUE" or "1" state; the condition of a logic gate whose output is "1".

OPEN - no electrical connection between two specified electrical points (i. e. , pins).

OPERATOR, LOGIC - a symbol which indicates that a logic function (such as "AND" or "OR") is to be performed. Examples of logic operators are: the "wedge" ( $\vee$ ), the "dot" ( $\cdot$ ), the "plus" ( $+$ ), the "triangle" ( $\nabla$ ), and the overhead "bar" ( $\bar{\quad}$ ).

"OR" GATE - an electronic circuit which forms a logic gate (represented by a triangle) whose output is "0" only when all of its inputs are "0". The gate output is "1" for all other combinations of inputs.

"OR" LOGIC OPERATION - the logic operation denoted by the "wedge" ( $\vee$ ); the operation which answers the question: "Is at least one fact 'TRUE'?" or "Is at least one input '1'?"

OUTPUT, BINARY - any pin of a flip-flop or logic gate that generates a binary "1" or "0" voltage signal which results from one or more input signals. Any pin on a pulse generator which generates a pulse voltage signal.

OVERFLOW - the "carry" or "borrow" output generated by the last flip-flop at the extreme left in a register.

12. GLOSSARY (Continued)

PARENTHESES ( ) - symbols which denote logic grouping along with brackets and braces.

PARITY - the condition of a number being even or odd (i. e. , "even" parity or "odd" parity).

PIN - a terminology used to describe a circuit connection point. Since the printed circuit board configurations for all circuits described within are designed with inserted pin (or "terminal") connections at the proper points, the term "PIN" has been used to describe these points.

PINS, POWER - the two pins where the positive and negative power connections must be made. These power connections must be made on each circuit in order for it to operate.

POINT - a period (.) used to split a number into two parts: the left part represents that portion of the number which is greater than unity (1) and the right part represents that portion of the number which is less than unity (1). The point separates the fractional (right) from the whole number portion (left).

POINT, BINARY - a period (.) used to split a binary number into two parts: the left part represents that portion of the number which is greater than unity (1) and the right part represents that portion which is less than unity (1). The binary point separates the binary fractional (right) from the binary whole number portion (left).

POINT, DECIMAL - a period (.) used to split a decimal number into two parts: the left part represents that portion of the number which is greater than unity (1) and the right part represents that portion which is less than unity (1). The decimal point separates the decimal fractional (right) from the decimal whole number portion (left).

POLARITY - the condition of a voltage pin, battery terminal, or power supply terminal being positive ("+" ) or negative ("-"). Battery, power supply, and voltage pin polarities must be properly wired for all units in order for them to operate properly.

POST MERIDIEM - after noon (abbreviated P. M. ); denotes a time period of half a day starting from noon hour up to, but not including, zero hour (midnight) of the next day (i. e. , the time period during the afternoon and evening hours).

POWER SOURCE - a battery or a power supply which supplies voltage to all electronic units in order for them to operate.

PROBLEM, INTRINSIC - the inability of a computer circuit or wiring project to work properly even though there is apparently no mistake in the circuit construction or in the wiring.

PRODUCT - the result obtained by multiplying two or more numbers together.

PRODUCT REGISTER - the register (row of flip-flops) which indicates the answer after a multiplication is performed (in the "multiplier" computer projects).

PROJECT, BASIC - a very simple computer wiring project which usually consists of a few flip-flops, a pulse generator, but contains no logic gates.

PROJECT, COMPUTER - a wired configuration of flip-flops and logic gates which performs computer operations.

PROJECT, NON-GATED - a computer wiring project which contains no logic gates.

PULSE - a square wave voltage output that is either at "0" or at "1". The pulse signal occurs during the "down-swing" or "fall" from "1" to "0"; may be generated by a pulse generator, flip-flop, or logic gate.

PULSE GENERATOR - an electronic circuit (also called an ASTABLE MULTI-VIBRATOR) that generates a repetitive square wave voltage output that swings from "1" (voltage present) to "0" (no voltage); the command device that makes the flip-flops work automatically.

## 12. GLOSSARY (Continued)

QUAD - a time interval of 400 years. Short for QUAD CENTURY.

QUAD CENTURY - a time interval of 400 years.

QUOTIENT - the answer obtained after dividing one number by another.

QUOTIENT REGISTER - the register (row of flip-flops) which indicates the answer after the computer division process is completed (in the "divider" computer project).

RADICAL ( $\sqrt{\quad}$ ) - a mathematical notation which indicates that a root (i. e., square root) must be extracted from a number.

RADICAND - the number contained inside a root radical; the number from which a (square) root must be extracted.

RADICAND REGISTER - the register (row of flip-flops) in which is entered the number to be (square) rooted (in the "square rooter" computer project).

READOUT, COMPARATOR - the output of either a gate or a flip-flop (the computer output) in a comparator computer project which indicates that a comparison is valid if it is a "1" and invalid if it is a "0".

READOUT LAMP - the light on the flip-flop which is "on" when the "TRUE" output is a "1" and is "off" when the "TRUE" output is a "0"; the display light on the flip-flop.

REFERENCE COLUMN - a column used to translate a binary number into the decimal system. The column is headed by the appropriate power of 2 which the binary digit position represents (i. e., starting from the right and proceeding to the left, we have: ("1", "2", "4", "8", "16", etc.).

REGISTER - a row of flip-flops wired to perform a specific computer function such as to count or to shift.

REGISTER, ACCUMULATOR - the register (row of flip-flops) that displays an answer such as in the case of the "adder" and "subtractor" wiring projects.

REGISTER, ADDEND - the register (row of flip-flops) in which the number to be added is entered (in the "adder" computer project).

REGISTER, COMPLEMENTARY TRANSFORMATION - a register (row of flip-flops) which will transform, by a computer process, any number (entered into the register) into its complement and display the complement in the same register.

REGISTER, DIVIDEND - the register (row of flip-flops) in which is entered the number to be divided (in the "divider" computer project).

REGISTER, DIVISOR - the register (row of flip-flops) in which is entered the number by which a second number is being divided (in the "divider" computer project).

REGISTER, MULTIPLICAND - the register (row of flip-flops) in which is entered the number which is to be multiplied by a second number (in the "multiplier" computer projects).

REGISTER, MULTIPLIER - the register containing the number by which another number is to be multiplied (in the "multiplier" computer projects).

REGISTER, PRODUCT - the register (row of flip-flops) which indicates the answer after a multiplication is performed (in the "multiplier" computer projects).

REGISTER, QUOTIENT - the register (row of flip-flops) which indicates the answer after the computer division process is completed (in the "divider" computer project).

REGISTER, RADICAND - the register (row of flip-flops) in which is entered the number to be (square) rooted (in the "square rooter" computer project).

REGISTER, ROOT - the register (row of flip-flops) which displays the answer after the square root process is completed (in the "square rooter" computer project).

## 12. GLOSSARY (Continued)

REGISTER, SHIFT - a wired configuration of flip-flops that will shift all "1's" in a binary number either one position to the left or one position to the right with each pulse command.

REGISTER, SUBTRAHEND - the register (row of flip-flops) in which the number to be subtracted is entered (in the "subtractor" computer project).

REGISTER, TRAVELLING "1" - the register (row of flip-flops) which automatically enters a "1" at the extreme left and shifts it successively one position to the right until it reaches the last position on the right at which time it is shifted out (in the "square rooter" computer project).

REMAINDER - the quantity that remains after subtracting one number from another; the quantity that remains in excess after each division process is completed.

RESET - to change a flip-flop to the "off" state (by shorting pins A and B together or by applying current-limited voltage to the "reset" input pin B); to change all flip-flops in a register to the "0" state.

"RESET" INPUT - pin B on the flip-flop which will turn off the readout lamp when a current-limited voltage is applied at that point.

RESISTOR - a two-lead electronic device which is used to limit electronic current.

RISE TIME - the time required for a voltage up-swing to occur. The time which elapses during the change from zero volts, to some specific voltage.

ROOT - a second number which, when multiplied by itself a specified number of times, will yield back the first number.

ROOT, SQUARE - a second number which, when multiplied by itself, will yield back the first number.

ROOT CONTROL - a wired configuration of flip-flops with or without logic gates which generates pulse signals to perform the square root process (in the "square rooter" computer project).

ROOT REGISTER - the register (row of flip-flops) which displays the answer after the square root process is completed (in the "square rooter" computer project).

ROUNDING "DOWN" - (see ROUNDING OFF) leaving unchanged the first digit to the left of the "breaking point" when rounding off.

ROUNDING "DOWN", BINARY - (see ROUNDING OFF, BINARY) leaving unchanged the first binary digit to the left of the "breaking point" if the first binary digit to the right of the "breaking point" was "0".

ROUNDING "DOWN", DECIMAL - (see ROUNDING OFF, DECIMAL) leaving unchanged the first digit to the left of the "breaking point" if the first digit to the right of the "breaking point" was 4 or less.

ROUNDING OFF - the process of changing to "0" all digits in a number which are to the right of a specified position in that number, and dropping those digits if they are in the fractional portion. The first digit to the left of the "breaking point" is either increased by one unit or left unchanged depending on the first number to the right of the "breaking point".

ROUNDING OFF, BINARY - the process of changing to "0" all binary digits which are to the right of a specified "breaking point", and dropping those digits if they are in the binary fractional portion. The first digit to the left of the "breaking point" is increased by "1" if the first digit to be changed was "1", and left unchanged if the first digit to be changed was already "0".

## 12. GLOSSARY (Continued)

ROUNDING OFF, DECIMAL - the process of changing to "0" all digits which are to the right of a specified "breaking point", and dropping those digits if they are in the fractional portion. The first digit to the left of the "breaking point" is increased by 1 unit if the first digit to be changed was 5 or more, and left unchanged if the first to be changed was 4 or less.

ROUNDING "UP" - (see ROUNDING "OFF") increasing by one unit the first digit to the left of the "breaking point".

ROUNDING "UP", BINARY - (see ROUNDING OFF, BINARY) increasing by "1" the first binary digit to the left of the "breaking point" if the first binary digit to the right of the "breaking point" was "1".

ROUNDING "UP", DECIMAL - (see ROUNDING OFF, DECIMAL) increasing by 1 unit the first digit to the left of the "breaking point" if the first digit to the right of the "breaking point" was 5 or more.

"SAMPLE" COMMAND - a command pulse which causes the sample-and-hold logic to copy a number from one register into a second register, leaving the number in the first register unchanged.

SAMPLE-AND-HOLD - the process of sensing a number entered in one register and copying it into a second register without removing the basic number from the first register; a wired configuration of electronic logic gates which will perform the operation described above.

SENSE - a wired configuration of logic gate(s) which determines some condition about a register or registers (i. e. , whether a number is present in a register; whether the last digit to the right in a register is "1", etc.) and, based on this condition, will control the execution of some command.

SENSE, DIGIT - the use of an "OR" logic gate to determine whether or not a number is present in a register.

SET - to change a flip-flop to the "on" state (by shorting pins D and E together or by applying current-limited voltage to the "set" input pin E).

"SET" INPUT - pin E on the flip-flop which will turn on the readout lamp when a current-limited voltage is applied at that point.

SHIFT - to transfer all "1's" in a binary number either one position to the left or one position to the right.

SHIFT ADDER - a wired configuration of flip-flops that consists of one full adder and three shift registers. Two shift registers shift the two numbers to be added through the full adder and the answer is shifted out in the third shift register.

"SHIFT" COMMAND - a command pulse which causes the number entered in a shift register to advance one position either to the right or the left (depending on the type of shift register).

SHIFT REGISTER - a wired configuration of flip-flops that will shift all "1's" in a binary number either one position to the left or one position to the right with each pulse command.

SHIFT REGISTER, LEFT - a wired configuration of flip-flops that will shift all "1's" in a binary number one position to the left with each pulse command.

SHIFT REGISTER, RIGHT - a wired configuration of flip-flops that will shift all "1's" in a binary number one position to the right with each pulse command.

SHIFT SUBTRACTOR - a wired configuration of flip-flops and logic gates that consists of one full subtractor and three shift registers. Two shift registers shift the two numbers through the full subtractor and the answer is shifted out in the third shift register.



## 12. GLOSSARY (Continued)

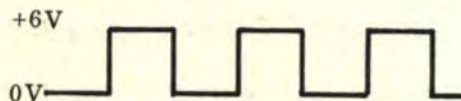
**SHORT** - a direct-wired connection between two electrical points such as touching or wiring two pins together.

**SIGNAL** - an electrical command generated by a pulse on the "down-swing" from "1" (6 volts) to "0" (0 volts) which will act as a trigger and cause one or more flip-flops to change state.

**SQUARE ROOT** - a second number which, when multiplied by itself, will yield back the first number.

**SQUARE ROOTER** - a computer project which consists of a wired configuration of flip-flops, logic gates, and pulse generators which will extract the square root from a given binary number.

**SQUARE WAVE** - an electrical fluctuation between 0 volts and some other voltage (for example, 6 volts) such that the rise time and fall time are very small compared to the time duration at 6 volts and 0 volts which should be nearly equal. Shown as follows:



**STATE** - the condition which describes a flip-flop or logic gate as being "on" ("1", "TRUE") or "off" ("0", "FALSE").

**STEERING INPUTS** - the input pins on the flip-flop (pins C and F) which will or will not cause a change of state to take place depending on whether or not an input voltage is present. The pins C and F must have opposite inputs (i. e. , one pin "1" and the other "0") to provide steering. If pin C input is a "1", the flip-flop will change state to "0" if in the "1" state, and remain "0" if already in the "0" state. If pin F input is a "1", the flip-flop will change state to "1" if in the "0" state, and will remain "1" if already in the "1" state.

**SUBROUTINE** - a specific computer process which forms part of another more complicated computer process. Example: the addition process is a subroutine in the cumulative-addition multiplier.

**SUBTRACT CONTROL** - the "halt" command which is used to control (or stop) the subtraction process in the "subtractor" configuration.

**SUBTRACTER** - a wired configuration of flip-flops and/or logic gates which will perform the arithmetic process of subtraction.

**SUBTRACTER, FULL** - the complete logic circuitry which generates both a "difference" and a "borrow" output.

**SUBTRACTER, HALF** - the logic circuitry which generates only a "difference" output and not a "borrow" output; (also called a "SUM" gate) the half-adder and half-subtractor are identical as are the "sum" and "difference" outputs.

**SUBTRACTER, LOGIC** - a wired configuration of logic gates and/or flip-flops which will perform subtraction (in binary).

**SUBTRACTER, SHIFT** - a wired configuration of flip-flops and logic gates that consists of one full subtracter and three shift registers. Two shift registers shift the two numbers through the full subtracter and the answer is shifted out in the third shift register.

**SUBTRACTER, NON-GATED BINARY** - a wired configuration of only flip-flops that will perform subtraction (i. e. , that will allow a binary number to be subtracted in one register and the answer to be displayed in a second register). Almost the same as the non-gated adder except that both upper and lower registers are "UP" counters.

12. GLOSSARY (Continued)

SUBTRACTION IDENTITY - a logic expression which defines a "difference", a "borrow", or both for two or more binary numbers.

SUBTRAHEND - that number which is to be subtracted from another number.

SUBTRAHEND REGISTER - the register (row of flip-flops) in which the number to be subtracted is entered (in the "subtractor" computer project).

SUM - the answer which results from the addition of two numbers; the binary digit which represents the addition of two other binary digits, ignoring the "carry".

"SUM" GATE - an electronic circuit which forms a logic gate whose output is a "1" only when the parity of all "1" inputs, regardless of how many, is odd. The gate output is "0" if the parity of all "1" inputs is even.

"SUM" LOGIC OPERATION - the logic operation denoted by the "plus" (+); the operation which answers the question: "Is the number of 'TRUE' facts odd?" or "Is the number of '1' inputs odd?"

TERM, LOGIC - that singular or multiple part of a logic expression which is separated by a logic operator (i. e. , grouped by a logic operator).

THEOREM, LOGIC - a logic equation which can be proved by means of a "TRUTH" table or by means of other logic equations or expressions.

TIME MACHINE - a computer project which consists of a wired configuration of flip-flops and logic gates controlled by a very accurate timing pulse which will keep time in the same manner as a clock and can also be expanded to determine days, months, years, leap years, etc.

TRANSFER - to use a command pulse to place a number (or digit) already entered in one register (or flip-flop) directly, unchanged, into a second register (or flip-flop). The

TRANSFER (continued)

original register (or flip-flop) is cleared during the process. An example is the non-gated "adder" computer project where a number is transferred from the addend register to the accumulator.

TRANSISTOR - a three-lead electronic device which is used to control the flow of electronic voltage and current (see also BASE, COLLECTOR, and EMITTER).

TRAVELLING "1" REGISTER - the register (row of flip-flops) which automatically enters a "1" at the extreme left and shifts it successively one position to the right until it reaches the last position on the right at which time it is shifted out (in the "square rooter" computer project).

TRIGGER - the input pin (pin G) on a flip-flop where a pulse command signal will cause the flip-flop to change state; to cause a flip-flop to change state.

TRIGGER INPUT - the input pin (pin G) on a flip-flop where a pulse command signal will cause the flip-flop to change state.

TROUBLESHOOT - to locate and determine the cause of problems that may occur in the wiring or operation of computer circuits.

TRUE - a proven affirmative logic fact; opposite of "FALSE"; the condition of a flip-flop or logic gate being in the "1" or "on" state (flip-flop readout light on); an input that is a "1". The side of the flip-flop with the readout light (also all flip-flop outputs and inputs, pins A, B, and C, on the same side as the readout light).

"TRUE" DIRECTOR OUTPUT - the output pin (pin A) on the "TRUE" side of the flip-flop; also known as the "TRUE" director output of the flip-flop.

12. GLOSSARY (Continued)

"TRUE" FOLLOWER INPUT - the input pin (pin C) on the "TRUE" side of the flip-flop which causes steering depending upon whether or not an input voltage is present.

"TRUE" OUTPUT - the output pin (pin A) on the "TRUE" side of the flip-flop; also known as the "TRUE" output of the flip-flop.

"TRUE" SIDE - the side of the flip-flop with the readout light.

"TRUTH" TABLE - a tabular representation of logic expressions and facts which indicates all possibilities of "TRUE" and "FALSE".

"UP" COUNTER - a binary counter that starts from zero and increases its value by "1" with each pulse signal input. In other words, it "counts upward" to any desired number.

"UP-DOWN" CONTROL - the extra logic gates wired into a binary counter so that the free-running counter will count alternately "UP" and then "DOWN", changing between "UP" and "DOWN" when an overflow occurs.

"UP-DOWN" COUNTER - a binary counter wired with extra logic gates so that the free-running counter will count alternately "UP" and then switch automatically to "DOWN" after resetting from the "UP" count. It will switch automatically back to "UP" after the "borrow" overflow is generated.

UP-SWING, VOLTAGE - the change of the output of a pulse generator, flip-flop, or logic gate, from zero volts, to a specific voltage.

VOLTAGE, CURRENT- LIMITED - voltage present when a resistor is placed in series with the power source (the value of 1,000 ohms should be used for most current limiting described in this text).

WIRE, DIODE - a wire with a diode spliced in the middle. The main use for the diode wire is to stop (or halt) the pulse from the pulse generator.

WIRES, CONNECTING - hoop-up wires of various lengths with alligator clips on each end (or with stripped ends for soldering) which are used to tie together electrical points (pin connections) as shown in the wiring diagrams.

WIRING DIAGRAM - that part of a logic diagram which shows where wiring pin connections must be made.

YEAR - a time interval of approximately 365 days, more accurately equal 365 days, 5 hours, 48 minutes, and 46 seconds.

YEAR, CENTURY - a year which ends in "00" (is divisible by 100) such as 1700, 1800, 1900, 2000, and 2100.

YEAR, LEAP - a time interval which has exactly 366 days instead of 365 days. Occurs in all non-century years divisible by 4, and in all century years divisible by 400.

YEAR, NORMAL - a time interval of exactly 365 days.

INDEX

- Accumulator, 71
- Addend register, 71
- Adder, half, 53
- Adder, logic, 75-81
- Adder, non-gated binary, 71-72
- Adder, shift, 86-88
- Addition, binary, 21, 22
- Addition identities, 10-11, 17
- "AND", 2, 7, 18, 42
- "AND" gate, 42-44
- "AND" & "OR" gate, 47-48
- "AND" & "OR" gate, negative, 48-50
- Ante meridiem, 141
- Associative laws, 6, 8
- Astable multivibrator, 39
  
- Bar, overhead, 2, 4, 8, 10
- Base, 67
- BCD counter, 74-75
- Binary, 5
- Binary addition, 21-22
- Binary coded decimal counter, 74, 75
- Binary complement, 34, 35
- Binary, conversion from decimal, 25-28
- Binary, conversion to decimal, 20, 28
- Binary count, 21
- Binary division, 23-24
- Binary "down" counter, 68-69
- Binary electronic computer circuits, 36
- Binary fractionals, 24-25
- Binary fractions, 29
- Binary multiplier, full-logic, 91-98
- Binary multiplication, 22-23
- Binary number system, 20
- Binary point, 23-25, 33
- Binary "rounding off", 34
- Binary shift register, 69-70
- Binary square root, 32-33
- Binary subtraction, 22
- Binary-to-decimal decoder, 153-154
- Binary "up" counter, 68
- Bistable multivibrator, 36
- Boards, mounting, 59
- Boolean Approach, 5
- "Borrows", 22
- Bracket, overhead, 25
- Brackets, 2, 6, 8
- Braces, 2
  
- Care of units, 66-67
- "Carry", 21
- Cautions, 58
- Century, Quad, 136
- Circuits, binary electronic computer, 36
- Circuits, electronic gate, 18
- Circuits, LIBE electronic, 36, 37, 38, 39
- Clock, time machine, 139-150
- Collector, 67
- Column, truth, 13, 15
- Commands, 62-64
- Commutative laws, 7
- Comparator, "Equal To", 115-118
- Comparator, "Greater Than", 109-112
- Comparator, "Greater Than or Equal To", 122-125
- Comparator, "Less Than", 112-115
- Comparator, "Less Than or Equal To", 126-129
- Comparator, "Unequal To", 118-122
- Comparators, 108-109
- Comparison, general, 130-134
- Complement, binary, 34-35
- Computer, electronic, circuit operation, 58
- Computer, electronic, logic expressions, 19
- Computer, electronic, logic notation, 18
- Conclusion, 2, 3, 4, 5, 36, 42
- Connections, 58-59
- Connections, power pin, 61
- Connections, wire, 60
- Converse, 10
- Conversion, binary to decimal, 20, 28
- Conversion, decimal to binary, 25-27
- Count, binary, 21
- Counter, BCD, 74-75
- Counter, binary coded decimal, 74-75
- Counter, binary, "down", 68-69
- Counter, binary, "up", 68
- Counter, "day-of-the-month", 139
- Counter, gated "up-down", 90-91
- Counter, "hour", 139
- Counter, "minute", 139
- Counter, "second", 139
- Counter, "year" BCD decade, 139
- Cumulative-addition multiplier, 99-101

INDEX (Continued)

- Debugging, 65-66
- Decimal, conversion from binary, 20, 28
- Decimal, conversion to binary, 25-27
- Decimal fractionals, 24
- Decimal point, 23
- Decoder, binary-to-decimal, 153-154
- DeMorgan's theorems, 7, 13, 14
- Denominator, 29
- Diagrams, logic, 60-61
- Dice, electronic, 150-153
- Diode wire, 59
- Direct display, 55-57
- Direct output, 56
- Director output, "false", 37
- Director output, "true", 37
- Distributed expression, 8
- Distributive laws, 8, 15
- Dividend, 23, 24
- Divider, 101-103
- Division, binary, 23-24
- Divisor, 23, 24
- "Dot", 2, 4
- Double negation, 10
- Down-swing, 37-39
- Down-swing triggering, 43, 45
  
- Electronic circuits, LIBE, 36, 37, 38, 39
- Electronic computer circuits, binary, 36
- Electronic computer circuit operation, 58
- Electronic computer logic expressions, 19
- Electronic computer logic notation, 18
- Electronic dice, 150-153
- Electronic gate circuit, 18
- Electronic gate symbols, 18
- Emitter, 67
- "EOR", 2, 3, 4, 5, 8, 10, 18, 42
- "EOR" gate, 50-52
- "Equal", 2
- "Equal To" comparator, 115-118
- "Even parity", 53
- "Exclusive OR" gate, 50-52
- Expression, distributed, 8
- Expression, factored, 8
- Expressions, logic, 19
- Extracting square root, 29-33
  
- Fact, 2, 5, 12, 36, 42
- Factored expression, 8
- Fall, 38
  
- "False", 2, 3, 4, 5, 12, 20, 36, 37, 38
- "False" director output, 37
- "False" follower input, 37
- Flip-flop, 18, 36-39
- Flip-flop, LIBE, 43
- Fractionals, binary, 24-25
- Fractionals, decimal, 24
- Fractionals, infinite, 25
- Fractionals, irrational, 25
- Fractions, binary, 29
- Fundamental identities, 7, 9, 16, 17
- Fundamental laws, 7
- Fundamental logic theorems, 7
  
- Gate, 2, 19
- Gate, "AND", 42-44
- Gate, "AND" & "OR", 47-48
- Gate, "AND" & "OR" negative, 48-50
- Gate, electronic, 18
- Gate, electronic, symbols, 18
- Gate, "EOR", 50-52
- Gate, "Exclusive OR", 50-52
- Gate input pins, 43-45
- Gate, inverter, 54-55
- Gate, logic, 42
- Gate, "NOT", 54-55
- Gate, "OR", 45-47
- Gate, parity, 53
- Gate output triggering, 55-57
- Gate, "SUM", 52-54
- Gate, "SUMMATION", 52-54
- General comparison, 130-134
- Generator, pulse, 36, 39-41
- Glossary, 155-172
- "Greater Than" comparator, 109-112
- "Greater Than or Equal To" comparator, 122-125
- Group, 2
  
- Half-adder, 53
- Horizons, unlimited, 154
  
- Identities, addition, 10-11, 17
- Identities, fundamental, 9, 16, 17
- Identities, logic, 7
- Identities, special, 10
- Inequality, 8
- Infinite fractional, 25

INDEX (Continued)

- Input, "false" follower, 37
- Input pins, 47
- Input, "reset", 37
- Inputs, 2
- Input, "set", 37
- Input, "trigger", 37-38
- Input, "true" follower, 37
- Intrinsic problems, 65-66
- Invert, 10
- Inverter, 18, 42
- Inverter gate, 54-55
- Irrational fractional, 25
- Irrational square root, 30
- "L" line, 30, 32
- Labels, 59-60
- Laws, commutative, 7
- Laws, distributive, 8, 15
- Laws, logic, 7
- Layout, physical, 58-59
- Leap year, 136
- "Less Than" comparator, 112-115
- "Less Than or Equal To" comparator, 126-129
- LIBE electronic circuits, 36, 37, 38, 39, 40, 42
- LIBE flip-flop, 43
- Logic, 2, 5
- Logic adder, 75-81
- Logic brackets, 6
- Logic diagrams, 60-61
- Logic expressions, 19
- Logic functions, 2
- Logic gates, 42
- Logic interface, "day-month-year", 139
- Logic interface, "leap-year, leap-century, century", 139
- Logic notation, 2, 18
- Logic operation, 2, 3, 4, 18
- Logic operators, 2, 12, 13
- Logic parentheses, 6
- Logic, "Sample-and-Hold", 88-90
- Logic subtracter, 81-86
- Logic symbols, 2, 18
- Logical multiplication, 8
- Logical "plus", 8
- Multiplication, logical, 8
- Multiplication, numerical, 8
- Multiplier, 22, 23
- Multiplier, binary, full-logic, 91-98
- Multiplier, cumulative-addition, 99-101
- Multivibrator, astable, 39
- Multivibrator, bistable, 36
- "NAND", 2, 4, 42-45, 48-50
- Negation, 10, 12
- Negation, double, 10
- Negative "AND" & "OR" gates, 48-50
- Nomenclature, 42
- Non-gated binary adder, 71-72
- Non-gated computer projects, 68
- Non-gated subtracter, 72-73
- "NOR", 2, 4, 42, 45, 48-50
- "NOR" output, 45
- "NOT" gate, 54-55
- Number system, binary, 20
- Numerator, 29
- Numerical "plus", 8
- Notation, electronic computer logic, 18
- Odd parity, 53
- Ohmmeter, 67
- Operation, electronic computer circuit, 58
- Operation, logic, 2, 3, 4, 18
- Operators, logic, 2, 12, 13
- "OR", 2, 3, 5, 7, 18, 42
- "OR" gate, 45-47
- "OR" output, 45
- Output, direct, 56
- Output, "false" director, 37
- Output pins, 47
- Output, pulse, 39
- Output, "true" director, 37
- Outputs, 2, 38
- Overhead bar, 2, 4, 8, 10
- Overhead bracket, 25
- Parentheses, 2, 6, 8, 10
- Parity, 3
- Parity, even, 53
- Parity gate, 53
- Parity, odd, 53
- Physical layout, 58-59
- Mounting, 58-59
- Mounting boards, 59
- Multiplicand, 22, 23
- Multiplication, binary, 22-23

INDEX (Continued)

- Pins, gate input, 43, 45  
 Pins, input, 47  
 Pins, output, 47  
 "Plus", 2  
 "Plus", logical, 8  
 "Plus", numerical, 8  
 Point, binary, 23, 24, 25, 33  
 Point, decimal, 23  
 Positive, 10  
 Post meridiem, 141  
 Power pin connections, 61  
 Power sources, 61-62  
 Problems, intrinsic, 65-66  
 Product, 23  
 Projects, advanced computer, 74  
 Projects, non-gated computer, 68  
 Pulse generator, 36, 39-41  
 Pulse output, 39
- Quad century, 136  
 Quotient, 23, 24
- Radical, 29  
 Radicand, 104  
 Reference column, 26, 28  
 Register, addend, 71  
 Register, binary shift, 69-70  
 Register, complementary transformation, 70-71  
 Register, "month" shift, 139  
 Register, "Travelling 1", 105  
 Remainder, 23, 24, 30  
 Remainder, binary, 32, 33  
 Repair of units, 66-67  
 "Reset", 64, 65  
 "Reset" input, 37  
 Rise, 38  
 Root, square, 29-32  
 Root, square, binary, 32-33  
 Rooter, square, 104-108  
 Rounding "down", 34  
 Rounding "off", 34  
 Rounding "up", 34
- "Sample-and-Hold" logic, 88-90  
 Science of reasoning, 2  
 "Senses", 62-64  
 "Set", 64-65  
 "Set" input, 37  
 Shift adder, 86-88  
 Sources, power, 61-62  
 Special identities, 10  
 Square root, 29-31  
 Square root, binary, 32-33  
 Square rooter, 104-108  
 Square wave, 37, 39  
 Subtractor, logic, 81-86  
 Subtractor, non-gated binary, 72-73  
 Subtraction, binary, 22  
 "SUM", 2, 3, 5, 10, 18, 42  
 "SUM" gate, 52-54  
 "SUMMATION" gate, 52-54  
 Symbols, 2  
 Symbols, electronic gate, 18  
 Symbols, logic, 18  
 System, binary number, 20
- Tables, truth, 12, 13, 14, 15, 16, 17, 21  
 Theorems, DeMorgan's, 7, 13, 14  
 Theorems, logic, 7  
 Time machine, 134-138  
 Time machine clock, 139-150  
 Transistor, 67  
 Triangle, 2, 3  
 Triggering, down-swing, 43, 45  
 Triggering, gate output, 55-57  
 Trigger input, 37-38  
 "True", 2, 3, 4, 5, 20, 36, 37, 38  
 "True" director output, 37  
 "True" follower input, 37  
 Truth column, 13, 15  
 Truth tables, 12, 13, 14, 15, 16, 17, 21
- "Unequal To" comparator, 118-122  
 Units, care and repair of, 66-67  
 Unlimited horizons, 154  
 "Up-Down" control, 91  
 "Up-Down" counter, gated, 90-91  
 Up-swing, 37, 38
- Year, 136
- Wave, square, 37, 39  
 Wedge, 2, 4

INDEX (Continued)

Wire connections, 60

Wire, diode, 59

Wires, 59

Wiring, 60-61



# A NEW APPROACH TO DIGITAL COMPUTER LOGIC

FANTASTIC DO-IT-YOURSELF COMPUTER PROJECTS  
USING ONLY 4 SIMPLE ELECTRONIC CIRCUITS !!!!!

- BINARY COUNTERS
- ELECTRONIC DICE
- SHIFT REGISTERS
- COMPARATORS
- BCD COUNTERS
- BCD DECODER
- ADDERS & SUBTRACTORS
- SAMPLE & HOLD LOGIC
- MULTIPLIER
- DIVIDER
- SQUARE ROOTER
- TIME MACHINE

# LIBE