

Price: \$2.50

# **INTRODUCTION TO DATA PROCESSING**

## **Maintenance Training**

August 1966

900909A



SCIENTIFIC DATA SYSTEMS • 701 South Aviation Boulevard • El Segundo, Calif., 90245 • 213/772-4511

# CONTENTS

Section No.	Page No.	Section No.	Page No.
1	1		
<u>NUMBER SYSTEMS</u>			
INTRODUCTION			
Counting	1	Phantom OR Gate	29
General Expression	2	The AND/OR Gate	30
Octal Numbering System	2	Logic Amplifiers	30
Binary Number System	3	BAND - Buffered AND	32
Summary	3	NAND - Negative AND or Not AND	32
		Cable Driver	32
		Receiver-Inverter	33
		Receiver-Inverter-Buffer	34
		Flip-Flop	34
		NAND Flip-Flop	36
		"Super" NAND Flip-Flop	38
		DC Flip-Flop	39
		Repeater Flip-Flop	40
2	5		
<u>CONVERSION BETWEEN NUMBERING SYSTEMS</u>			
INTRODUCTION			
CONVERTING BINARY OR OCTAL TO DECIMAL			
CONVERTING FRACTIONAL NUMBERS			
Binary/Octal to Decimal	6	6	43
Decimal to Binary/Octal	6	<u>HOW A COMPUTER WORKS</u>	
Conversion of Fractions	7	INTRODUCTION	
Comparisons	8	ARITHMETIC-LOGICAL UNIT	
		LOCATIONS AND ADDRESSES	
		PICTORIAL REPRESENTATION OF MEMORY	
		LOCATION AND ADDRESS TERMINOLOGY	
		INSTRUCTIONS AND PROGRAMS	
		Format of an Instruction	44
		Location of Instructions	44
		Interpretation of the Address Part	44
		HOW A COMPUTER ADDS	
		Addition Instructions	45
		Sequence of Performing Instructions	45
		Detail of the Addition Operation	45
		Repeated Operations	46
		WHAT HAPPENS IN THE EXECUTION OF AN INSTRUCTION	
		REPEATED OPERATIONS	
		Loops	48
		MEANINGS OF THE ADDRESS PORTION OF INSTRUCTIONS	
		Address of Data to be Taken from Memory	51
		Address of Data to be Put into Memory	51
		Address of Instructions	51
		Unconditional Transfer Addresses	51
		Conditional Transfer Addresses	51
		Addresses which are Absolute Numbers	51
		Address as Identification of Input/ Output Units	52
		Address as Identification of Indicator Units	52
3	11		
<u>USING THE NUMBERING SYSTEM</u>			
INTRODUCTION			
Addition	11		
Subtraction	11		
Multiplication	12		
Division	13		
Complements and Complement Arithmetic	13		
4	17		
<u>LOGICAL ALGEBRA</u>			
INTRODUCTION			
SYMBOLY AND RULES OF LOGICAL STATES			
The AND Function	18		
OR	18		
NOT	19		
NOR	19		
NAND	20		
VEITCH DIAGRAMS			
Mechanization of Logic	25		
5	27		
<u>LOGIC ELEMENTS</u>			
INTRODUCTION			
STANDARD LOGIC LEVELS			
The AND Function	27		
Expander AND Gate	27		
The OR Function	28		
Gated Input OR Gate	29		

## CONTENTS (Cont'd)

Section No.	Page No.	Figure No.	Page No.
7	<u>STORAGE SECTIONS</u>		
	MAGNETIC CORE MEMORY		
	Basic Element of a Core Memory		
	Magnetic Field of a Core		
	Using Two Wires to Pass Current		
	Inhibiting Current		
	The Read Operation		
	Core Addressing		
	Reading from Core		
	Core Arrays		
	<u>APPENDIX A. CONVERSION TABLES</u>		
	Octal-Decimal Integer Conversion Table		
	Octal-Decimal Fraction Conversion Table		
	Table of Powers of Two		
	<u>TABLES</u>		
	Table No.	Page No.	
	3-1 Binary Addition, A + B	11	
	3-2 Octal Addition	11	
	3-3 Binary Subtraction, A - B	12	
	3-4 Octal Subtraction	12	
	3-5 Binary Multiplication	12	
	3-6 Octal Multiplication	12	
	<u>ILLUSTRATIONS</u>		
	Figure No.	Page No.	
	4-1 AND Truth Table	18	
	4-2 AND Switching Circuit	18	
	4-3 AND Symbol	18	
	4-4 OR Truth Table	18	
	4-5 OR Switching Circuit	19	
	4-6 OR Logic Symbol	19	
	4-7 NOT Truth Table	19	
	4-8 NOT Switching Circuit	19	
	4-9 NOT Logic Symbol	19	
	4-10 NOR Operation	20	
	4-11 NOR Switching Circuit	20	
	4-12 NOR Logic Symbol	20	
	4-13 NAND Operation	20	
	4-14 NAND Switching Circuit		20
	4-15 NAND Logic Symbol		21
	4-16 Logic Symbols		21
	4-17 Veitch Diagrams		23
	4-18 Veitch Diagram Rules		25
	4-19 Mechanization of Logic		25
	4-20 Simplified Mechanization of Logic		26
	5-1 Cable Driver		33
	5-2 Symbolic Receiver "Pick-Off"		33
	5-3 Receiver-Inverter		34
	5-4 Receiver-Inverter-Buffer		34
	5-5 Flip-Flop		35
	5-6 Central Latch and DC Set Input		35
	5-7 Setting/Resetting Circuitry		35
	5-8 Setting/Resetting and Central Latch		36
	5-9 NAND Flip-Flop		37
	5-10 Triggering Circuitry		37
	5-11 Super NAND Flip-Flop		38
	5-12 Central Latch and Output Buffer		39
	5-13 DC Flip-Flop		39
	5-14 Repeater Flip-Flop		41
	6-1 Two Numbers in Memory		43
	6-2 Memory Before Addition Operation		45
	6-3 Memory Before Executing First Instruction		45
	6-4 Memory After First Instruction		46
	6-5 Memory After Second Instruction		46
	6-6 Memory After Last Instruction		46
	6-7 Addition Program		47
	6-8 How A Computer Executes An Instruction		48
	6-9 Program of Addition, Storage and Modification of Address		49
	6-10 COMPARE and TRANSFER IF EQUAL Instructions		50
	7-1 Core		53
	7-2 Magnetized Core		53
	7-3 Magnetization Curve		53
	7-4 Coil with One Turn		54
	7-5 Magnetization Curve		54
	7-6 Two Wire Coil		55
	7-7 Inhibit Current Scheme		55
	7-8 Conditions of Inhibit and No-Inhibit Current		55
	7-9 Core Lacing and Various Currents		56
	7-10 Sixteen-Bit Core Storage Arrangement		56
	7-11 Effects of One Core Half-Current-Flux Change		57
	7-12 8 x 8 Core Plane		58
	7-13 X and Y Drive Lines for 4-Bit Words in Core Memory		59

# SECTION 1. NUMBER SYSTEMS

## INTRODUCTION

A mandatory requirement to understanding of modern digital systems is an understanding of various numbering systems. After obtaining an understanding of the numbering systems, facility in their use comes with practice.

Nearly all of us have been trained to use the decimal number system. We are so familiar with the system and manipulation of decimal numbers that we have never bothered to analyze the system to determine the exact meaning of each integer and decimal fraction. The rules of other number systems are the same as those of the decimal system. Therefore, as a prelude to study of other systems, an analysis of the decimal system will be presented because of its familiarity.

The decimal system uses 10 different symbols, each representing a discrete value, to display all numerical quantities. These are: 0, 1, 2, 3, 4, 5, 6, 7, 8 and 9. These symbols are identified by a variety of names such as admissible marks, basic characters, basic figures, admissible figures. We will use the term basic symbols, or symbols.

In the decimal system, the name for which was derived from the latin "decem" meaning ten, it is possible to describe with any one basic symbol any quantity of things in the range of 0 through 9. If it is desired to describe more than 9 things, more than one basic symbol is required: e.g., 10, or 27, or 9376. From this, two facts stand out:

a: In a numbering system using R number of basic symbols it is possible to represent only R-1 things using only one basic symbol.

b: To describe more than R-1 things more than one basic symbol must be used in some ordered arrangement to describe the quantity of things.

Therefore we can now define some terms and make some basic rules:

a. RADIX or BASE of a system is the number of basic symbols which comprise the system.

## RULE 1

In any numbering system with a Radix R, a maximum of R-1 things may be described using only one basic symbol.

### Counting

Using the basic symbols of a numbering system to describe quantities or values of things is called counting. In counting there are rules which are followed. We should analyze how counting is actually done and then formulate the rule.

Assume counting in sequence from 0 to some 3-digit number.

Starting from 0, add 1 to the least significant digit (LSD) until every basic symbol has been used.

To count beyond 9 add 1 to the next most significant digit or second digit position and start from 0 through the basic symbols again in the LSD position.

Continue this process until all basic symbols have been used in the second digit position. When it now becomes necessary to add 1 to the second digit position, but all symbols have been used in the second digit position, it will be necessary to use a third digit position.

Place a 1 in the third digit position and 0's in the second digit position and LSD position. Continue the process by repeating steps outlined above.

This same process can go on for additional digit positions but it would merely be repetition of the above steps. The major facts which can be obtained from this analysis of counting in the decimal system are:

a. A basic symbol in the LSD position has a value of the basic symbol.

b. A basic symbol in the second, third, or higher digit position has a value which depends on its position.

Example

$$\begin{aligned}
 3 &= 3 \times 1 &= 3 \times 10^0 \\
 30 &= 3 \times 10 &= 3 \times 10^1 \\
 300 &= 3 \times 100 &= 3 \times 10^2 \\
 3000 &= 3 \times 1000 &= 3 \times 10^3
 \end{aligned}$$

c. The basic symbol is called a coefficient and the value it represents is determined by its position. Position values are determined by the radix raised to some exponent value.

Example

$$\begin{aligned}
 37501 &= 1 \times 10^0 = 1 \\
 &= 0 \times 10^1 = 00 \\
 &5 \times 10^2 = 500 \\
 &7 \times 10^3 = 7000 \\
 &3 \times 10^4 = 30000 \\
 \hline
 &37501
 \end{aligned}$$

The number 37501 in the above example is shorthand notation of the sum of the coefficients times the radix raised to some exponent level.

Thus far we have spoken only of integer numbers, and decimal fractions have not been mentioned. It has been assumed that these were whole numbers and a decimal point or radix point to the right of the LSD. To express fractional numbers basic symbols are used to the right of the radix point in digit positions which correspond to the negative coefficient value of the radix.

Continue the system by using increased negative exponents on the radix as we move to the right farther away from the radix point.

Example

					↓				
					Radix Point				
Coefficient Position	4	3	2	1	.	-1	-2	-3	-4
Position Exponent	$R^3$	$R^2$	$R^1$	$R^0$		$R^{-1}$	$R^{-2}$	$R^{-3}$	$R^{-4}$

We have explained what digits to the left of the radix point mean, but what do the digits to the right of the radix point mean?

$$R^{-1} = \frac{1}{R}, R^{-2} = \frac{1}{R^2}, R^{-N} = \frac{1}{R^N}$$

Therefore the value of digits to the right of the radix point also have values which depend on position.

Examples

$$\begin{aligned}
 .3 &= 3 \times 10^{-1} = \frac{3}{10} \\
 .03 &= 3 \times 10^{-2} = \frac{3}{100} \\
 .003 &= 3 \times 10^{-3} = \frac{3}{1000} \\
 \\ 
 37.25 &= 3 \times 10^1 = 30 \\
 &7 \times 10^0 = 7 \\
 &2 \times 10^{-1} = .2 \\
 &5 \times 10^{-2} = .05 \\
 \hline
 &37.25
 \end{aligned}$$

RULE 2

The position a basic symbol occupies determines the value it represents. The value of a digit position is always determined by a value of the radix raised to some exponent power.

General Expression

To express a value in any numbering system the following general expression applies:

$$\begin{aligned}
 N &= A_m R^m + A_{m-1} R^{m-1} + \dots + A_2 R^2 + A_1 R^1 + A_0 R^0 \\
 &+ A_{-1} R^{-1} + A_{-2} R^{-2} + \dots + A_{-(n-1)} R^{-(n-1)} + A_{-n} R^{-n}
 \end{aligned}$$

Note that in the general expression there is no radix point indicated. The radix point always appears between  $A_0 R^0$  and  $A_{-1} R^{-1}$ .

Octal Numbering System

The previous discussion, although it referred to the decimal system, has direct application to other numbering systems. For instance, assume a numbering system which has only 8 basic symbols: 0, 1, 2, 3, 4, 5, 6, 7. From this we can determine:

$$\begin{aligned}
 \text{Radix} &= 8 \\
 \text{Highest single digit value} &= R-1 = 8-1 = 7.
 \end{aligned}$$

This system is called the octal system since there are 8 discrete values permissible using only one basic symbol.

To count in this system apply the same rules as previously used in the decimal system.

Example

0	25	75
1	26	76
2	27	77
3	30	100
4	31	101
5	:	:
6	:	:
7	35	105
10	36	106
11	37	107
12	40	110

To express a number, say 3765.125 in octal it is written the same as in decimal. The radix point is used as before. But the number no longer represents the same number of "things" as it would if it were a decimal number. Looking at 3765.125 it is impossible to tell in which number system it was written. Therefore to be clear about just what quantity is intended, the number system must be specified. This is done by subscripting the last digit of the number with the radix.

Example

3765.125 <sub>10</sub>	26347.213 <sub>8</sub>
101211.212 <sub>3</sub>	10110.11 <sub>2</sub>

The octal system will be the subject of further discussion later. Before going into this we should become familiar with another number system that is used in digital computer systems: the BINARY number system.

Binary Number System

The binary number system has 2 basic symbols: 0, 1 and thus the radix is 2. To count in this system the same rules apply as were previously used.

Example

Decimal	Octal	Binary
0	0	0
1	1	1
2	2	10
3	3	11
4	4	100
5	5	101
6	6	110
7	7	111
8	10	1000
9	11	1001
10	12	1010
11	13	1011
12	14	1100

Decimal	Octal	Binary
13	15	1101
14	16	1110
15	17	1111
16	20	10000

Summary

We have covered three number systems and learned how to count in each with the major reference being the decimal system. Now it is possible to make comparisons and to work several examples in each system.

$$\begin{aligned}
 10_{10} &= 12_8 = 1010_2 \\
 37_{10} &= 45_8 = 100101_2 \\
 25_{10} &= 31_8 = 11001_2 \\
 10.125_{10} &= 12.1_8 = 1010.001_2 \\
 25.375_{10} &= 31.3_8 = 11001.011_2 \\
 37.5_{10} &= 45.4_8 = 100101.1_2
 \end{aligned}$$

If you have trouble understanding these comparisons, perhaps it would be worthwhile for you to use positional values on these and other numbers of your choosing to go from numbers in other than the decimal system to determine their decimal value.

Remember that the general expression for a number in any system is:

$$N = A_m R^m + A_{m-1} R^{m-1} + \dots + \underbrace{A_0 R^0}_{\text{Radix Point}} + A_{-1} R^{-1} + A_{-2} R^{-2} + \dots + A_{-n} R^{-n}$$

Examples

(1)  $721.32_8 =$

$$\begin{array}{cccccc}
 (7 \times 8^2) & + & (2 \times 8^1) & + & (1 \times 8^0) & + & (3 \times 8^{-1}) & + & (2 \times 8^{-2}) \\
 \uparrow & & \uparrow & & \uparrow & & \uparrow & & \uparrow \\
 A_2 & & A_1 & & A_0 & & A_{-1} & & A_{-2} \\
 R^2 & & R^1 & & R^0 & & R^{-1} & & R^{-2}
 \end{array}$$

$$\begin{aligned}
 &(7 \times 64) + (2 \times 8) + (1 \times 1) + (3 \times \frac{1}{8}) + (2 \times \frac{1}{64}) \\
 &448 + 16 + 1 + (3 \times .125) + (2 \times .015625) \\
 &448 + 16 + 1 + .375 + .03125 = 448.0000 \\
 &16.0000 \\
 &1.0000 \\
 &.3750 \\
 &+.03125 \\
 \hline
 &465.40625
 \end{aligned}$$

$$721.32_8 = 465.40625_{10}$$

$$(2) 101.01_2 =$$

$$1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 0 \times 2^{-1} + 1 \times 2^{-2}$$

$$1 \times 4 + 0 \times 2 + 1 \times 1 + \frac{0}{2} + \frac{1}{4}$$

$$4 + 0 + 1 + 0 + .25 = 4.00$$

$$1.00$$

$$\underline{.25}$$

$$5.25$$

$$101.01_2 = 5.25_{10}$$

## SECTION 2. CONVERSION BETWEEN NUMBERING SYSTEMS

### INTRODUCTION

In learning the numbering systems and counting in the various systems it was implied that there was a method of going from one system to another, although it was not stated. It is essential that digital systems maintenance personnel know how to get from one system to another with facility. In the following discussion only three number systems will be used: decimal, octal and binary. The methods discussed will, however, be usable in any system.

Digital equipment does its work by using binary number manipulations. The reason for this is the ease with which a binary system can be implemented electrically. For instance a light being on or off, a transistor conducting or not conducting, relay contacts being opened or closed; these are all usable as binary system indicators since there are only 2 states represented in each case. This matches the two basic symbols 0 and 1. Either state of a two-state element may be called either 0 or 1 in any given system. For instance an "on" light may be called 1 and thus the "off" light would be called 0. However, nothing would be wrong with calling the "off" light 1 and the "on" light 0. Once the stated condition is named, however, it must be used from that point onward.

### CONVERTING BINARY OR OCTAL TO DECIMAL

To convert a binary or octal number to a decimal number is a reasonable and understandable process. Remembering the positional system, and remembering that the general expression for a number is  $N = A_m R^m + A_{m-1} R^{m-1} + \dots + A_1 R^1 + A_0 R^0 + A_{-1} R^{-1} + A_{-2} R^{-2} + \dots + A_n R^{-n}$ , it would be easy to convert from any system other than decimal to the decimal system. All one needs to do is multiply the coefficient by the radix raised to the appropriate power and add the products to arrive at the decimal value of the number. Remember that decimal arithmetic is used in conversion manipulations.

#### Examples

$$\begin{aligned}
 (1) \quad 321.5_8 &= (3 \times 8^2) + (2 \times 8^1) + (1 \times 8^0) + (5 \times 8^{-1}) \\
 &= (3 \times 64) + (2 \times 8) + (1 \times 1) + (5 \times .125) = 321.5_8 \\
 &= 209.625_{10}
 \end{aligned}$$

$$(2) \quad 11101.1_2$$

$$\begin{aligned}
 &1 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1} \\
 &1 \times 16 + 1 \times 8 + 1 \times 4 + 0 \times 2 + 1 \times 1 + 1 \times \frac{1}{2} =
 \end{aligned}$$

$$16 + 8 + 4 + 0 + 1 + .5 = 11101.1_2 = 29.5_{10}$$

The above examples work well when there is available to you a table of values of 8 and 2 raised to positive and negative exponents (see Appendix A). If you do not use the tables, you can convert by continued multiplication for integer conversion as follows:

#### Examples (Octal to Decimal)

$  \begin{array}{r}  (1) \quad 321_8 \\  \times 8 \\  \hline  24 \\  +2 \\  \hline  26 \\  \times 8 \\  \hline  208 \\  +1 \\  \hline  209 \\  \hline  \end{array}  $	$  \begin{array}{r}  (2) \quad 705_8 \\  \times 8 \\  \hline  56 \\  +0 \\  \hline  56 \\  \times 8 \\  \hline  448 \\  +5 \\  \hline  453 \\  \hline  \end{array}  $	$  \begin{array}{r}  (3) \quad 256_8 \\  \times 8 \\  \hline  16 \\  +5 \\  \hline  21 \\  \times 8 \\  \hline  168 \\  +6 \\  \hline  174 \\  \hline  \end{array}  $	$  \begin{array}{r}  (4) \quad 100_8 \\  \times 8 \\  \hline  8 \\  +0 \\  \hline  8 \\  \times 8 \\  \hline  64 \\  +0 \\  \hline  64 \\  \hline  \end{array}  $
$321_8 = 209_{10}$	$705_8 = 453_{10}$	$256_8 = 174_{10}$	$100_8 = 64_{10}$

#### Examples (Binary to Decimal)

$  \begin{array}{r}  (1) \quad 11010_2 \\  \times 2 \\  \hline  2 \\  +1 \\  \hline  3 \\  \times 2 \\  \hline  6 \\  +0 \\  \hline  6 \\  \times 2 \\  \hline  12 \\  +1 \\  \hline  13 \\  \times 2 \\  \hline  26 \\  +0 \\  \hline  26 \\  \hline  \end{array}  $	$  \begin{array}{r}  (2) \quad 101101_2 \\  \times 2 \\  \hline  2 \\  +0 \\  \hline  2 \\  \times 2 \\  \hline  4 \\  +1 \\  \hline  5 \\  \times 2 \\  \hline  10 \\  +1 \\  \hline  11 \\  \times 2 \\  \hline  22 \\  +0 \\  \hline  22 \\  \times 2 \\  \hline  44 \\  +1 \\  \hline  45 \\  \hline  \end{array}  $
$11010_2 = 26_{10}$	$101101_2 = 45_{10}$



## CONVERTING FRACTIONAL NUMBERS

To continue this process for fractional numbers would not work since the fractional portion is determined with negative exponential values of the radix ( $R^{-n}$ ). There is no easy way to convert from the octal or binary system to the decimal system of numbers for fractions. The positional system appears to be the best method, and at least the easiest to remember. It is not the objective of this book to describe every number conversion system, but rather to explain one or two methods. Additional methods are available in standard texts.

### Binary/Octal to Decimal

For fractional numbers, the method recommended is positional notation expansion as demonstrated in the following examples:

#### Examples

(1)  $.432_8$

$$4 \times 8^{-1} + 3 \times 8^{-2} + 2 \times 8^{-3}$$

$$\frac{4}{8} + \frac{3}{64} + \frac{2}{512} =$$

$$\frac{256}{512} + \frac{24}{512} + \frac{2}{512} = \frac{282}{512} = .5507_{10}$$

$$.432_8 = .5507_{10}$$

$$512 \overline{) .5507}$$

$$\begin{array}{r} 282.0000 \\ -2560 \\ \hline 2600 \\ -2560 \\ \hline 4000 \end{array}$$

(2)  $.253_8$

$$2 \times 8^{-1} + 5 \times 8^{-2} + 3 \times 8^{-3}$$

$$\frac{2}{8} + \frac{5}{64} + \frac{3}{512} =$$

$$\frac{128}{512} + \frac{40}{512} + \frac{3}{512} = \frac{171}{512} = .334_{10}$$

$$.253_8 = .334_{10}$$

$$512 \overline{) .33398}$$

$$\begin{array}{r} 171.000 \\ -1536 \\ \hline 1740 \\ -1536 \\ \hline 2040 \\ -1536 \\ \hline 5040 \\ -4598 \\ \hline 4420 \end{array}$$

(3)  $.1011_2$

$$1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3} + 1 \times 2^{-4} =$$

$$\frac{1}{2} + \frac{0}{4} + \frac{1}{8} + \frac{1}{16} =$$

$$\frac{8}{16} + \frac{2}{16} + \frac{1}{16} = \frac{11}{16} = .6875$$

$$.1011_2 = .6875_{10}$$

$$16 \overline{) 11.000}$$

$$\begin{array}{r} 6875 \\ -96 \\ \hline 140 \\ -128 \\ \hline 120 \\ -112 \\ \hline 80 \end{array}$$

(4)  $.0101_2$

$$0 \times 2^{-1} + 1 \times 2^{-2} + 0 \times 2^{-3} + 1 \times 2^{-4}$$

$$\frac{0}{2} + \frac{1}{4} + \frac{0}{8} + \frac{1}{16} =$$

$$\frac{1}{4} + \frac{1}{16} = \frac{4}{16} + \frac{1}{16} = \frac{5}{16} = .3125_{10}$$

$$.0101_2 = .3125_{10}$$

$$16 \overline{) .3125}$$

$$\begin{array}{r} 5.00 \\ -48 \\ \hline 20 \\ -16 \\ \hline 40 \\ -32 \\ \hline 80 \end{array}$$

### Decimal to Binary/Octal

Thus far we have been stating a number in the binary or octal systems and converting it back to its decimal equivalent. The next step is to go from the decimal system to the binary or octal system. For integer numbers, the following general rules apply:

- Divide the original decimal by the radix. Use decimal arithmetic.
- The remainder will be the least significant digit of the equivalent number in the new radix number system.
- Divide the quotient by the radix. The remainder will be the next to least significant digit.
- Repeat the above steps until a quotient of 0 is obtained. The remainder in each case will be the next most significant digit in the equivalent number of the new radix system.

#### Examples (Decimal to Octal)

(1)

$$8 \overline{) 2732_{10}}$$

$$\begin{array}{r} 341 \\ -24 \\ \hline 33 \\ -32 \\ \hline 12 \\ -8 \\ \hline 4 \rightarrow 4 \end{array}$$

$$8 \overline{) 42}$$

$$\begin{array}{r} 5 \\ -32 \\ \hline 21 \\ -16 \\ \hline 5 \rightarrow 5 \end{array}$$

$$8 \overline{) 341}$$

$$\begin{array}{r} 42 \\ -32 \\ \hline 21 \\ -16 \\ \hline 5 \rightarrow 5 \end{array}$$

$$8 \overline{) 42}$$

$$\begin{array}{r} 5 \\ -40 \\ \hline 2 \rightarrow 2 \end{array}$$

$$8 \overline{) 0}$$

$$\begin{array}{r} 0 \\ -0 \\ \hline 0 \rightarrow 0 \end{array}$$

$$2732_{10} = 5254_8$$

(2)

$$\begin{array}{r}
 8 \overline{) 948} \\
 \underline{72} \\
 38 \\
 \underline{32} \\
 64 \\
 \underline{64} \\
 0 \rightarrow 0
 \end{array}$$

$$\begin{array}{r}
 8 \overline{) 118} \\
 \underline{8} \\
 14 \\
 \underline{8} \\
 68 \\
 \underline{64} \\
 4 \rightarrow 4
 \end{array}$$

$$\begin{array}{r}
 8 \overline{) 118} \\
 \underline{8} \\
 38 \\
 \underline{32} \\
 6 \rightarrow 6
 \end{array}$$

$$\begin{array}{r}
 8 \overline{) 14} \\
 \underline{8} \\
 6 \rightarrow 6
 \end{array}$$

$$8 \overline{) 1} \rightarrow 1$$

$$7584_{10} = 16,640_8$$

(3)

$$\begin{array}{r}
 8 \overline{) 10} \\
 \underline{8} \\
 2 \rightarrow 0
 \end{array}$$

$$\begin{array}{r}
 8 \overline{) 10} \\
 \underline{8} \\
 2 \rightarrow 2
 \end{array}$$

$$8 \overline{) 1} \rightarrow 1$$

$$80_{10} = 120_8$$

Examples (Decimal to Binary)

(1)  $2 \overline{) 156}$

$$\begin{array}{r}
 2 \overline{) 156} \text{ r } 0 \rightarrow 0 \\
 2 \overline{) 139} \text{ r } 0 \rightarrow 0 \\
 2 \overline{) 19} \text{ r } 1 \rightarrow 1 \\
 2 \overline{) 9} \text{ r } 1 \rightarrow 1 \\
 2 \overline{) 4} \text{ r } 1 \rightarrow 1 \\
 2 \overline{) 2} \text{ r } 0 \rightarrow 0 \\
 2 \overline{) 1} \text{ r } 0 \rightarrow 0 \\
 0 \text{ r } 1 \rightarrow 1
 \end{array}$$

$$156_{10} = 10011100_2$$

(2)  $2 \overline{) 128}$

$$\begin{array}{r}
 2 \overline{) 128} \text{ r } 0 \rightarrow 0 \\
 2 \overline{) 64} \text{ r } 0 \rightarrow 0 \\
 2 \overline{) 32} \text{ r } 0 \rightarrow 0 \\
 2 \overline{) 16} \text{ r } 0 \rightarrow 0 \\
 2 \overline{) 8} \text{ r } 0 \rightarrow 0 \\
 2 \overline{) 4} \text{ r } 0 \rightarrow 0 \\
 2 \overline{) 2} \text{ r } 0 \rightarrow 0 \\
 2 \overline{) 1} \text{ r } 0 \rightarrow 0 \\
 0 \text{ r } 1 \rightarrow 1
 \end{array}$$

$$128_{10} = 10000000_2$$

(3)  $2 \overline{) 39}$

$$\begin{array}{r}
 2 \overline{) 39} \text{ r } 1 \rightarrow 1 \\
 2 \overline{) 9} \text{ r } 1 \rightarrow 1 \\
 2 \overline{) 4} \text{ r } 1 \rightarrow 1 \\
 2 \overline{) 2} \text{ r } 0 \rightarrow 0 \\
 2 \overline{) 1} \text{ r } 0 \rightarrow 0 \\
 0 \text{ r } 1 \rightarrow 1
 \end{array}$$

$$39_{10} = 100111_2$$

(4)  $2 \overline{) 80}$

$$\begin{array}{r}
 2 \overline{) 80} \text{ r } 0 \rightarrow 0 \\
 2 \overline{) 40} \text{ r } 0 \rightarrow 0 \\
 2 \overline{) 20} \text{ r } 0 \rightarrow 0 \\
 2 \overline{) 10} \text{ r } 0 \rightarrow 0 \\
 2 \overline{) 5} \text{ r } 0 \rightarrow 0 \\
 2 \overline{) 2} \text{ r } 1 \rightarrow 1 \\
 2 \overline{) 1} \text{ r } 0 \rightarrow 0 \\
 0 \text{ r } 1 \rightarrow 1
 \end{array}$$

$$80_{10} = 1010000_2$$

From the rules and the examples the conversion of decimal to binary or octal systems should be clear. If you have any trouble following the examples with the rules given, work some problems on your own and convert and re-convert until you are familiar with the system. Remember, this is for integer values only. You have not yet been exposed to fractions and their conversion.

Conversion of Fractions

To convert decimal fractions to octal or binary fractions the following general rules apply:

- Multiply the decimal fraction by the radix of the number system to which you wish to convert.
- Record the integer portion, which will be the MSD of the converted fraction.
- Multiply the fractional portion of the result by the radix.
- The integer portion of the result will be the next most significant digit of the converted fraction.
- Continue the above steps, obtaining one less significant digit with each operation until the fractional portion is all zeros or you have obtained the desired number of places.

Examples (Decimal to Octal Fractions)

(1)  $\frac{.525_{10}}{8} \rightarrow 4$

(4)  $\frac{.200}{8} \rightarrow 1$

(1)  $\frac{.600}{8} \rightarrow 4$

(4)  $\frac{.800}{8} \rightarrow 6$

(6)  $\frac{.400}{8} \rightarrow 3$

(3)  $\frac{.200}{8} \rightarrow 3$

$$.525_{10} = .41463_8 +$$

(2)  $\frac{.375_{10}}{8} \rightarrow 3$

(3)  $\frac{.000}{8} \rightarrow 3$

$$.375_{10} = .3_8$$

(3)  $.2163_{10}$

(1).	$\frac{7304}{8}$	→ 1
(5).	$\frac{8432}{8}$	→ 5
(6).	$\frac{7456}{8}$	→ 6
(5).	$\frac{9648}{8}$	→ 5
(7).	$\frac{7184}{8}$	→ 7

$.2163_{10} = .15657_8^+$

(4)  $.975_{10}$

(7).	$\frac{800}{8}$	→ 7
(6).	$\frac{400}{8}$	→ 6
(3).	$\frac{200}{8}$	→ 3
(1).	$\frac{600}{8}$	→ 1
(4).	$\frac{800}{8}$	→ 4

$.975_{10} = .76314_8^+$

$.875_{10}$

(7).	$\frac{000}{8}$	→ 7
------	-----------------	-----

$.875_{10} = .7_8 = .111_2$   
and  $111_2 = 7_{10} = 7_8$

$.875_{10}$

(1).	$\frac{750}{2}$	→ 1
(1).	$\frac{500}{2}$	→ 1
(1).	$\frac{000}{2}$	→ 1

Examples (Decimal to Binary Fractions)

(1)  $.525_{10}$

(1).	$\frac{050}{2}$	→ 1
(0).	$\frac{100}{2}$	→ 0
(0).	$\frac{200}{2}$	→ 0
(0).	$\frac{400}{2}$	→ 0
(0).	$\frac{800}{2}$	→ 0
(1).	$\frac{600}{2}$	→ 1
(1).	$\frac{200}{2}$	→ 1

$.525_{10} = .100011_2^+$

(2)  $.333_{10}$

(0).	$\frac{666}{2}$	→ 0
(1).	$\frac{332}{2}$	→ 1
(0).	$\frac{664}{2}$	→ 0
(1).	$\frac{328}{2}$	→ 1
(0).	$\frac{656}{2}$	→ 0
(1).	$\frac{312}{2}$	→ 1
(0).	$\frac{624}{2}$	→ 0
(1).	$\frac{248}{2}$	→ 1

$.333_{10} = .01010101_2^+$

$.325_{10}$

(2).	$\frac{600}{8}$	→ 2
(4).	$\frac{800}{8}$	→ 4
(6).	$\frac{400}{8}$	→ 6
(3).	$\frac{200}{8}$	→ 3

$.325_{10} = .2463_8^+$

$.325_{10}$

(0).	$\frac{650}{2}$	→ 0
(1).	$\frac{300}{2}$	→ 1
(0).	$\frac{600}{2}$	→ 0
(1).	$\frac{200}{2}$	→ 1
(0).	$\frac{400}{2}$	→ 0
(0).	$\frac{800}{2}$	→ 0
(1).	$\frac{600}{2}$	→ 1
(1).	$\frac{200}{2}$	→ 1
(0).	$\frac{400}{2}$	→ 0
(0).	$\frac{800}{2}$	→ 0
(1).	$\frac{600}{2}$	→ 1
(1).	$\frac{200}{2}$	→ 1

$.325_{10} = 010100110011_2^+$

(3)  $.404_{10}$

(0).	$\frac{808}{2}$	→ 0
(1).	$\frac{616}{2}$	→ 1
(1).	$\frac{232}{2}$	→ 1
(0).	$\frac{524}{2}$	→ 0
(1).	$\frac{048}{2}$	→ 1

$.404_{10} = .01101_2^+$

(4)  $.975_{10}$

(1).	$\frac{950}{2}$	→ 1
(1).	$\frac{900}{2}$	→ 1
(1).	$\frac{800}{2}$	→ 1
(1).	$\frac{600}{2}$	→ 1
(1).	$\frac{200}{2}$	→ 1
(0).	$\frac{400}{2}$	→ 0

$.975_{10} = .111110_2^+$

Dividing the binary equivalent into groups of three starting at the binary point, we find

.010	100	110	011
.2	4	6	3

It is apparent that the binary equivalent can be easily converted to octal equivalent by dividing the binary number into groups of 3 binary digits (bits) commencing at the radix point. Then it is possible to convert to octal equivalent by inspection of each group of 3 bits using the knowledge that:

- |         |         |
|---------|---------|
| 0 = 000 | 4 = 100 |
| 1 = 001 | 5 = 101 |
| 2 = 010 | 6 = 110 |
| 3 = 011 | 7 = 111 |

Comparisons

Of interest now would be comparisons of decimal system numbers converted to both octal and binary equivalents.

Examples

010/101/010/100/001/000. 010/010/101/000/001  
2 5 2 4 1 0 , 2 2 5 0 1

001, 101, 011, 111, 110, 110, 111, 011, 110. 111, 111  
1 5 3 7 6 6 7 3 6 . 7 7

This completes the discussion of conversion from one number system to another. It will be essential that you have a thorough understanding of binary and octal systems and are able to convert decimal, octal and binary with ease and confidence.



# SECTION 3. USING THE NUMBERING SYSTEM

## INTRODUCTION

The systems of numbers and counting, and the methods of converting a number from one system to its equivalent in another system were presented in Sections I and II. The next step is to manipulate the numbers in order to make them do useful work. The basic elements of arithmetic: addition, subtraction, multiplication and division are the operations which will be covered.

### Addition

Addition in binary or octal number systems is performed in the exact manner that is employed in the decimal system. The generation of a "carry" is in effect when the sum of the digits exceeds (overflows) the basic symbols. To make the procedure a little more clear, "truth" tables for both binary and octal addition are given below:

TABLE 3-1. BINARY ADDITION, A + B

A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

TABLE 3-2. OCTAL ADDITION

		AUGEND							
		0	1	2	3	4	5	6	7
ADDEND	1		2 <sup>0</sup>	3 <sup>0</sup>	4 <sup>0</sup>	5 <sup>0</sup>	6 <sup>0</sup>	7 <sup>0</sup>	0 <sup>1</sup>
	2			4 <sup>0</sup>	5	6 <sup>0</sup>	7 <sup>0</sup>	0 <sup>1</sup>	1 <sup>1</sup>
	3				6 <sup>0</sup>	7 <sup>0</sup>	0 <sup>1</sup>	1 <sup>1</sup>	2 <sup>1</sup>
	4					0 <sup>1</sup>	1 <sup>1</sup>	2 <sup>1</sup>	3 <sup>1</sup>
	5						2 <sup>1</sup>	3 <sup>1</sup>	4 <sup>1</sup>
	6							4 <sup>1</sup>	5 <sup>1</sup>
	7								6 <sup>1</sup>

Legend:  $\begin{matrix} x & y \\ \hline & \end{matrix}$      $y$  = Carry    Augend  
 $\begin{matrix} x & \\ \hline & \end{matrix}$  = Sum     $\begin{matrix} + & \text{Augend} \\ \hline & \text{Sum} \end{matrix}$

Therefore, to add in either system we can use the tables above in order to aid in understanding.

### Examples

Decimal	Octal Equiv	Binary Equiv
$\begin{array}{r} 29 \\ +20 \\ \hline 49_{10} \end{array}$	$\begin{array}{r} 35 \\ +24 \\ \hline 61_8 \end{array}$	$\begin{array}{r} 011101 \\ +010100 \\ \hline 110001_2 \end{array}$
$49_{10} = 61_8 = 110001_2$		
$\begin{array}{r} 67 \\ +15 \\ \hline 82_{10} \end{array}$	$\begin{array}{r} 103 \\ 17 \\ \hline 122_8 \end{array}$	$\begin{array}{r} 001000011 \\ 001111 \\ \hline 001010010_2 \end{array}$
$\begin{array}{r} 201 \\ 49 \\ 169 \\ 17 \\ \hline 436_{10} \end{array}$	$\begin{array}{r} 311 \\ 61 \\ 251 \\ 21 \\ \hline 664_8 \end{array}$	$\begin{array}{r} 11001001 \\ 00110001 \\ 10101001 \\ 00010001 \\ \hline 110110100_2 \end{array}$
$\begin{array}{r} 320 \\ 251 \\ 427 \\ \hline 998_{10} \end{array}$	$\begin{array}{r} 500 \\ 373 \\ 653 \\ \hline 1746_8 \end{array}$	$\begin{array}{r} 101000000 \\ 011111011 \\ 110101011 \\ \hline 1111100110_2 \end{array}$
$\begin{array}{r} 27 \\ 32 \\ 10 \\ \hline 69_{10} \end{array}$	$\begin{array}{r} 33 \\ 40 \\ 12 \\ \hline 105_8 \end{array}$	$\begin{array}{r} 011011 \\ 100000 \\ 001010 \\ \hline 1000101_2 \end{array}$

As can be seen, "carries" in any system are treated the same. Addition of more than two rows of numbers is not necessary because in computer work there is no requirement to add or manipulate more than two numbers at a time.

### Subtraction

Truth tables for octal and binary number systems are given below for reference in working the examples. Little time is spent dwelling on points because the rules are the same regardless of the system.

TABLE 3-3. BINARY SUBTRACTION, A - B

A	B	Difference	Carry
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

TABLE 3-4. OCTAL SUBTRACTION

		MINUEND							
		0	1	2	3	4	5	6	7
SUBTRAHEND	0	0 <sup>0</sup>	1 <sup>0</sup>	2 <sup>0</sup>	3 <sup>0</sup>	4 <sup>0</sup>	5 <sup>0</sup>	6 <sup>0</sup>	7 <sup>0</sup>
	1	7 <sup>1</sup>	0 <sup>0</sup>	1 <sup>0</sup>	2 <sup>0</sup>	3 <sup>0</sup>	4 <sup>0</sup>	5 <sup>0</sup>	6 <sup>0</sup>
	2	6 <sup>1</sup>	7 <sup>1</sup>	0 <sup>0</sup>	1 <sup>0</sup>	2 <sup>0</sup>	3 <sup>0</sup>	4 <sup>0</sup>	5 <sup>0</sup>
	3	5 <sup>1</sup>	6 <sup>1</sup>	7 <sup>1</sup>	0 <sup>0</sup>	1 <sup>0</sup>	2 <sup>0</sup>	3 <sup>0</sup>	4 <sup>0</sup>
	4	4 <sup>1</sup>	5 <sup>1</sup>	6 <sup>1</sup>	7 <sup>1</sup>	0 <sup>0</sup>	1 <sup>0</sup>	2 <sup>0</sup>	3 <sup>0</sup>
	5	3 <sup>1</sup>	4 <sup>1</sup>	5 <sup>1</sup>	6 <sup>1</sup>	7 <sup>1</sup>	0 <sup>0</sup>	1 <sup>0</sup>	2 <sup>0</sup>
	6	2 <sup>1</sup>	3 <sup>1</sup>	4 <sup>1</sup>	5 <sup>1</sup>	6 <sup>1</sup>	7 <sup>1</sup>	0 <sup>0</sup>	1 <sup>0</sup>
	7	1 <sup>1</sup>	2 <sup>1</sup>	3 <sup>1</sup>	4 <sup>1</sup>	5 <sup>1</sup>	6 <sup>1</sup>	7 <sup>1</sup>	0 <sup>0</sup>

Legend:  $x^y$      $y$  = Carry  
                    $x$  = Difference     $\begin{array}{r} \text{Minuend} \\ -\text{Subtrahend} \\ \hline \text{Difference} \end{array}$

Examples

Decimal	Octal Equivalent	Binary Equivalent
$\begin{array}{r} 23 \\ -15 \\ \hline 8_{10} \end{array}$	$\begin{array}{r} 27 \\ -17 \\ \hline 10_8 \end{array}$	$\begin{array}{r} 010111 \\ -001111 \\ \hline 001000_2 \end{array}$
$\begin{array}{r} 64 \\ -25 \\ \hline 39_{10} \end{array}$	$\begin{array}{r} 100 \\ -31 \\ \hline 47_8 \end{array}$	$\begin{array}{r} 00100000 \\ -011001 \\ \hline 100111_2 \end{array}$
$\begin{array}{r} 219 \\ -146 \\ \hline 73_{10} \end{array}$	$\begin{array}{r} 333 \\ -222 \\ \hline 111_8 \end{array}$	$\begin{array}{r} 011011011 \\ -010010010 \\ \hline 001001001_2 \end{array}$
$\begin{array}{r} 346 \\ -165 \\ \hline 181_{10} \end{array}$	$\begin{array}{r} 532 \\ -245 \\ \hline 265_8 \end{array}$	$\begin{array}{r} 101011010 \\ -010100101 \\ \hline 010110101_2 \end{array}$

Multiplication

Binary and octal multiplication are identical to decimal multiplication. The major difference is that the tables are different and must be memorized if one is to become rapid and expert in their use. The tables for each system are given as follows:

TABLE 3-5. BINARY MULTIPLICATION

		MULTIPLIER	
		0	1
MULTIPLICAND	0	0	0
	1	0	1

Examples

$A = 0$      $B = 1$      $A \times B = 0$   
 $A = 1$      $B = 1$      $A \times B = 1$   
 $A = 0$      $B = 0$      $A \times B = 0$   
 $A = 1$      $B = 0$      $A \times B = 0$

There are no other combinations and remembering the table is relatively easy.

TABLE 3-6. OCTAL MULTIPLICATION

		MULTIPLIER							
		0	1	2	3	4	5	6	7
MULTIPLICAND	0	0	0	0	0	0	0	0	0
	1	0	1	2	3	4	5	6	7
	2	0	2	4	6	10	12	14	16
	3	0	3	6	11	14	17	22	25
	4	0	4	10	14	20	24	30	34
	5	0	5	12	17	24	31	36	43
	6	0	6	14	22	30	36	44	52
	7	0	7	16	25	34	43	52	61

Examples

$A = 5$      $B = 7$      $A \times B = 43_8$   
 $A = 2$      $B = 4$      $A \times B = 10_8$   
 $A = 6$      $B = 3$      $A \times B = 22_8$

Examples: using the multiplication tables.

Decimal	Octal	Binary
$\begin{array}{r} 7 \\ 5 \\ \hline 35_{10} \end{array}$	$\begin{array}{r} 7 \\ 5 \\ \hline 43_8 \end{array}$	$\begin{array}{r} 111 \\ 101 \\ \hline 111 \\ 000 \\ \hline 111 \\ 100011_2 \end{array}$
$\begin{array}{r} 8 \\ 4 \\ \hline 32_{10} \end{array}$	$\begin{array}{r} 10 \\ 4 \\ \hline 40_8 \end{array}$	$\begin{array}{r} 1000 \\ 100 \\ \hline 10000_2 \end{array}$

<u>Decimal</u>	<u>Octal</u>	<u>Binary</u>
27	33	011011
8	10	1000
<u>216</u> <sub>10</sub>	<u>330</u> <sub>8</sub>	<u>011011000</u> <sub>2</sub>
51	63	110011
2	2	10
<u>102</u> <sub>10</sub>	<u>146</u> <sub>8</sub>	<u>001100110</u> <sub>2</sub>
100	144	1100100
63	77	111111
<u>300</u>	<u>1274</u>	<u>1100100</u>
600	1274	1100100
<u>6300</u> <sub>10</sub>	<u>14234</u> <sub>8</sub>	1100100
		1100100
		1100100
		<u>1100010011100</u> <sub>2</sub>

<u>Decimal</u>	<u>Octal</u>	<u>Binary</u>
5.9	5.71	101.111001
5 <u>29.5</u>	5 <u>35.4</u>	101 <u>11101.100</u>
-25	-31	-101
<u>45</u>	<u>44</u>	<u>1001</u>
-45	-43	-101
<u>0</u>	<u>10</u>	<u>1001</u>
	-5	-101
	<u>3 R</u>	<u>1000</u>
		-101
		<u>110</u>
		-101
		<u>1000</u>
		-101
		<u>11 R</u>
2.5	2.4	10.1
10 <u>25.0</u>	12 <u>31.0</u>	1010 <u>11001.000</u>
-20	-24	-1010
<u>50</u>	<u>50</u>	<u>1010</u>
-50	-50	-1010
<u>0</u>	<u>0</u>	<u>0</u>
3	3	11.0
13 <u>39</u>	15 <u>47</u>	1101 <u>100111.00</u>
-39	-47	-1101
<u>0</u>	<u>0</u>	<u>1101</u>
		-1101
		<u>0</u>

The reader should go through the above examples to develop an understanding of the processes.

### Division

Octal and binary division are identical to decimal division. Remember, when analyzing the examples below, that you must use the multiplication table of the system in which you are doing the division.

<u>Decimal</u>	<u>Octal</u>	<u>Binary</u>
6.8	6.63	110.110011
4 <u>27.2</u>	4 <u>33.15</u>	100 <u>011011.001101</u>
-24	-30	-100
<u>32</u>	<u>31</u>	<u>101</u>
-32	-30	-100
<u>00</u>	<u>15</u>	<u>110</u>
	-14	-100
	<u>1 R</u>	<u>100</u>
		-100
		<u>110</u>
		-100
		<u>101</u>
		-100
		<u>1 R</u>

11.0	13.0	1011.0
11 <u>121.0</u>	13 <u>171.00</u>	1011 <u>1111001.00</u>
-11	-13	-1011
<u>11</u>	<u>41</u>	<u>10000</u>
-11	-41	-1011
<u>0</u>	<u>0</u>	<u>1011</u>
		-1011
		<u>0</u>

The binary division example is stopped with a remainder but could have been just as easily carried on.

### Complements and Complement Arithmetic

SDS computer systems hold negative numbers in memory in two's-complement form. Single precision numbers have the most significant digit position of a computer word as the sign bit and the remainder of the word represents the magnitude of the number. This convention allows the sign of a number to be used as an integral part of the number in all arithmetic operations and obviates the need for keeping track of a detached sign with computer logic. A "0" bit denotes a positive sign and a "1" denotes a negative sign. In this system, the negative of a number is its two's complement.



An algorithm for finding the two's complement of a binary number is:

To find the two's complement of a binary number B that has N significant bits, including the sign bit, subtract it from the number  $2^{N+1}$  expressed in binary form. (The number  $2^{N+1}$  is a "one" followed by N zeroes.)

A more simple method of deriving the two's complement of a binary number is to invert all 0's and 1's and add 1 to the least significant digit. (The inversion of 1's and 0's forms what is called a one's complement.) This is the same as subtracting a binary number from an equal number of 1 digits, and adding 1 to the least significant digit.

Examples: Find the two's complement of the given numbers.

Method 1

$$\begin{array}{r} -25_{10} = -(011001_2) \quad 1000000 \\ \quad \quad \quad \quad \quad (-) 011001 \\ \hline \quad \quad \quad \quad \quad 1001111_2 \end{array}$$

$$\begin{array}{r} -9_{10} = -(001001_2) \quad 1000000 \\ \quad \quad \quad \quad \quad (-) 001001 \\ \hline \quad \quad \quad \quad \quad 1101111_2 \end{array}$$

$$\begin{array}{r} -2_{10} = -(000010_2) \quad 1000000 \\ \quad \quad \quad \quad \quad - 000010 \\ \hline \quad \quad \quad \quad \quad 1111110_2 \end{array}$$

Method 2

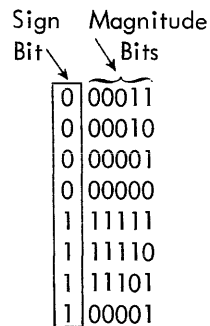
$$\begin{array}{r} -25_{10} = -(011001_2) \quad 100110 \text{ (one's complement)} \\ \quad \quad \quad \quad \quad + 1 \\ \hline \quad \quad \quad \quad \quad 100111_2 \text{ (two's complement)} \end{array}$$

$$\begin{array}{r} -9_{10} = -(001001_2) \quad 110110 \text{ (one's complement)} \\ \quad \quad \quad \quad \quad + 1 \\ \hline \quad \quad \quad \quad \quad 110111_2 \text{ (two's complement)} \end{array}$$

$$\begin{array}{r} -2_{10} = -(000010_2) \quad 111101 \text{ (one's complement)} \\ \quad \quad \quad \quad \quad + 1 \\ \hline \quad \quad \quad \quad \quad 111110_2 \text{ (two's complement)} \end{array}$$

In SDS systems, the sign bit is the first bit position to the left of the most significant magnitude bit. Thus, if an SDS computer word was only 6 bits long, instead of 24, some common decimal values would be represented in binary format as follows, considering that only negative numbers are complemented and the binary point is to the right of the least significant digit.

Decimal Number	Octal Equivalent	Complement Plus 1	Binary Equivalent
3	03	--	0 00011
2	02	--	0 00010
1	01	--	0 00001
0	00	--	0 00000
-1	(-)01	77	1 11111
-2	(-)02	76	1 11110
-3	(-)03	75	1 11101
-31	(-)37	41	1 00001



When using two's complement notation, an N-bit integer can be expressed as:

$$A_{n-1}[-(2^{n-1})] + A_{n-2}(2^{n-2}) + \dots + A_1(2^1) + A_0(2^0).$$

Thus,

$$\begin{aligned} +5 = 0101 &= 0[-(2^3)] + 1(2^2) + 0(2^1) + 1(2^0) \\ &= 0 + 4 + 0 + 1 = +5 \end{aligned}$$

and, taking the two's complement of 0101 to obtain the binary equivalent of -5,

$$\begin{aligned} -5 = 1011 &= 1[-(2^3)] + 0(2^2) + 1(2^1) + 1(2^0) \\ &= -8 + 0 + 2 + 1 \\ &= -8 + 3 = -5. \end{aligned}$$

The following examples show how two's complement numbers automatically yield the correct result when used arithmetically in a computer.

Decimal Number	Binary Equivalent
+20	010100
-03	111101
+17	(1) 010001 = $21_8 = 17_{10}$

Lost Carry →

Note that the carry out of the most significant (sign bit) position is "lost". Nevertheless, the value remaining is the correct answer because the positive "carry" into the negative sign position "cancels" the negative sign bit.

Decimal Number	Binary Equivalent
-32	100000
+24	011000
-8	(1) 111000 = $-10_8 = -8_{10}$

No Carry →

To find the decimal equivalent of a binary two's complement number:

- A. Subtract one. Form the one's complement
- C. Find the decimal equivalent. The negative of this result is the decimal equivalent.

As the above examples indicate, the sign bit is an integral part of the number to which it is attached and its value, plus or minus, is automatically taken care of during the use of the two's complement arithmetic. This property is used when numbers of different length are added. For example, assume that these two signed, two's complemented, negative numbers of 6-bit and 3-bit length are added:

<u>Decimal</u>		<u>Binary</u>
-21		101011
-03		101
<u>-24</u>	≠	<u>110000</u> = -16 <sub>10</sub>

Note that the third least significant bit of the first number is added to the sign bit of the second number causing an erroneous result. This error is corrected by filling in the empty bit positions with the value of the sign bit of the shorter number.

<u>Decimal</u>		<u>Binary</u>
-21		101011
-03		111101
<u>-24</u>	=	(1) <u>101000</u> = -24 <sub>10</sub>

This property suggests:

- 1) Filling the empty bit positions with the sign value of a positive number, that is, zero, has no changing effect on the result, and
- 2) If the two's complement is taken by the method suggested where N is the largest number's length, the sign value is automatically appended to the

smaller number. For example, if the complement of 03 is taken using N = 6,

$$\begin{array}{r} 100000 \\ 011 \\ \hline 111101 \end{array}$$

the sign is properly appended to the number.

This procedure is called "extending" the sign of a number.

Note that if the sign bit of -5 (1011) is extended four places to 1111011, this can be interpreted as

$$\begin{aligned} & 1(-2^7) + 1(2^6) + 1(2^5) + 1(2^4) + 1(2^3) + 0(2^2) \\ & \quad + 1(2^1) + 1(2^0) \\ = & -128 + 64 + 32 + 16 + 8 + 0 + 2 + 1 \\ = & -128 + 123 = -5. \end{aligned}$$

Thus the actual value of a binary number is unchanged by the sign-bit extension.

It was previously noted that when performing addition or subtraction in the computer, carries out of the sign bit do not always signify a true overflow condition or cause the overflow indicator to set. For example, in an addition it is impossible to produce a true overflow if the signs of the operands are unlike. A true overflow occurs when the result of an addition or subtraction results in a number too large to be expressed by the machine number range. The computer sets the overflow indicator during an addition only when the signs of the two operands are the same, but the sign of the result is opposite. In a subtraction, accomplished in the computer by forming the two's complement of the subtrahend and then adding to the minuend, the test for overflow is similar to that for addition. That is, overflow occurs when both numbers have the same sign after the subtrahend has been complemented, but the sign of the result is opposite.



## SECTION 4. LOGICAL ALGEBRA

### INTRODUCTION

Ordinary algebra is the symbolic expression for relationship of mathematical variables. This algebra we learned in high school.

There is another variety of algebra known as logical or Boolean algebra. Logical algebra differs from ordinary algebra in two respects.

- a. The Symbols do not represent numerical values.
- b. Arithmetic operations are not performed.

Boolean algebra is an aid for analyzing logical thought. In fact, George Boole, the developer of this algebra, published the algebra in a book entitled "The Laws of Thought". Boolean algebra is ideal for describing the action of switching circuits which is important in digital computer design since a switch may represent the characteristic ON and OFF states of computer intelligence. The operation of a digital computer can be described by logic equations using Boolean symbology.

### SYMBOLY AND RULES OF LOGICAL STATES

Letter symbols are used to represent dependent or independent variables which are always two valued: either zero or one; true or false; voltage or no voltage, etc.

Every logical quantity must exist in one or the other of the two possible states. No other values are allowed. In addition a logical quantity is single valued; that is, no quantity may be simultaneously both true and false.

Any quantity that is true is equal to any other quantity that is true and any quantity that is false is equal to any other quantity that is false.

Every logical quantity has an opposite. If the quantity is true the opposite, or reverse of that quantity is false, and vice versa.

Letters of the alphabet are generally used to represent the variables of Boolean algebra. These letter symbols are known as terms.

Logic operations that can be performed in algebra are:

- a. AND = X =  $\cdot$  =

- b. OR = +

- c. Inverse or negative = symbol over barred, e. g.  $\bar{A}$  or  $\bar{B}$ .

The X,  $\bullet$ , + of logic algebra in no way resembles the function of these same symbols when used in ordinary algebra.

a. The AND operation - a combination of variables has a true state output when all the variables are true, and for all other conditions the output state is false. The AND operation is represented by a dot or absence of any symbol between variables.

b. The OR operation - a combination of variables has a true state output when any one or more of the variables is true and a false state output when all variables are false. The OR operation is represented by a + between variables.

c. An expression is a combination of terms and operators.

A logical equation is a complete statement of equality of the two expressions separated by the = sign.

To express the complement of a logic term, expression, or equation, place a bar over it.

Examples:

	<u>Word description</u>
(1) $AB + \bar{C}$	A <u>and</u> B or not C
(2) $A + B = C$	A <u>or</u> B equal C
(3) $\overline{A + B} = \bar{C}$	Not (A or B) equals the complement of C
(4) $\overline{A + B} = \bar{A} \bar{B}$	Not (A or B) equal the complement of A and the complement of B
(5) $\overline{ABC} = \bar{A} + \bar{B} + \bar{C}$	Not (A and B and C) equals Not A or Not B or Not C

The operation described by the overbarred symbol or term is called the NOT operation. It denotes the inverse or complement of the term or symbol that is overbarred.

There are two additional operations which are combinations of the fundamental three (AND, OR, NOT). These additional operations are NOR and NAND.

NOR and NAND can be considered to be:

NOR =  $\overline{OR}$  i. e., the output of the OR is complemented.

NAND =  $\overline{AND}$  i. e., the output of the AND is complemented.

To go into these logic operations in more detail, it is necessary to define further some of the principles which must be considered. In dealing with logic the binary 1 is taken to mean the true state and the binary 0 is taken to mean the false state. In computers, voltage levels are used to represent true or false, for instance false = 0 volts, true = +8 volts might well be a set of conditions within a computer.

The AND Function

Expressed as a logical equation, the AND function might be used as follows:

$f = A \cdot B \cdot C = ABC = (A) (B) (C)$  where f denotes a logical function - in this case the AND.

A, B, and C represent true logical quantities.

The equation says that f will be true only when A and B and C are true all at the same time. One method of showing all possible conditions of all variables A, B, C, and the function f is the truth table. Figure 4-1 is a truth table of the above equation which uses 1 and 0 to replace true and false respectively.

$f = A \cdot B \cdot C$

A	B	C	f
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

Figure 4-1. AND Truth Table

The AND function might further be illustrated by a series of switches as shown below. If the light when lighted represents the true state of the function, the only way the true state at the light can be obtained

is for all switches to be in the true (closed) state.

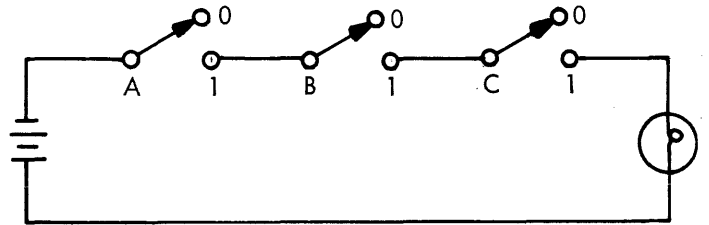


Figure 4-2. AND Switching Circuit

The logic symbol of an AND operation is:

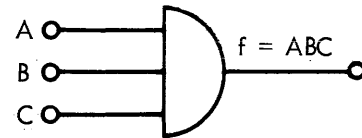


Figure 4-3. AND Symbol

The symbol of Figure 4-3 is known as the AND gate.

OR

Expressed in a logical equation the OR function might be used as follows:

$f = A + B + C$  where f denotes a logical function, in this case the OR.

A, B, and C represent true logical quantities.

The equation says that the function f will be true if A OR B OR C are true. Figure 4-4 is the truth table of the above function which uses 1 and 0 for true and false respectively.

$f = A + B + C$

A	B	C	f
0	0	0	0
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	1

Figure 4-4. OR Truth Table

The OR function might further be illustrated by a series of switches arranged in parallel as shown below. The light when lighted represents the TRUE state of the function. The only way the FALSE state at the light

can be obtained is for all switches to be FALSE (open). The TRUE state will be realized when any one of the switches is TRUE (closed).

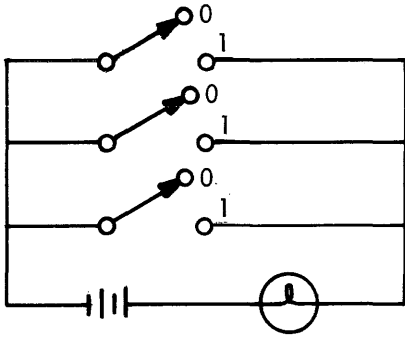


Figure 4-5. OR Switching Circuit

The logic symbol for an OR operation is:

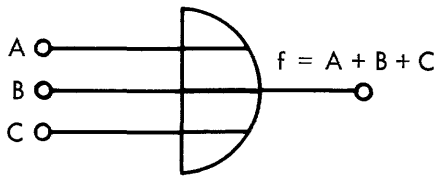


Figure 4-6. OR Logic Symbol

The symbol of Figure 4-6 is known as the OR gate.

### NOT

The NOT operation is a negating, inverting, or complementing operation. The implication is that you get from the NOT operation the opposite of what you put into the input of the operation.

The NOT operation is indicated by a bar drawn over the logical quantity. For instance:

$$f = \bar{A} \quad \text{Where } f \text{ denotes the logical function} \\ \text{NOT } A \text{ denotes a true logical quantity.}$$

When reading this equation, one would read, or say, "f equals NOT A" or "f equals A overbar" or "f = A-bar".

If A is true then  $\bar{A}$  must be false and vice versa. A truth table for the NOT function is given below in Figure 4-7.

$$f = \bar{A}$$

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

Figure 4-7. NOT Truth Table

A circuit which may be thought of as representative of the NOT is shown in Figure 4-8 below. The light f is TRUE when it is lit. In this case, the only way the TRUE state of the light f can be obtained is for the switch A to be open (false). The switch can be labeled any way we choose, thus the two conditions given in the truth table, Figure 4-7.

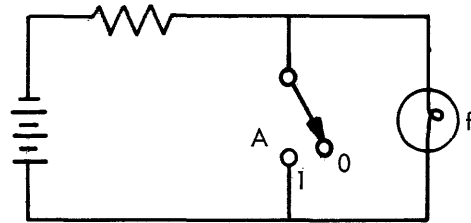


Figure 4-8. NOT Switching Circuit

The logic symbol for a NOT operation as used by SDS is:

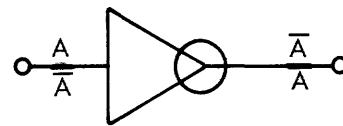


Figure 4-9. NOT Logic Symbol

The symbol of Figure 4-9 is known as an Inverter.

### NOR

The NOR operation is a combination of OR and NOT, two of the three fundamental logical operations. The word NOR is a contraction of the statement "NOT OR".

Expressed in a logical equation the NOR operation might be used as follows:

$$f = \overline{A + B + C} \quad \text{Where } f \text{ denotes the logical NOR} \\ \text{operation } A, B, C \text{ are TRUE} \\ \text{logical quantities.}$$

This equation says:

- f is false if any one or more of the logical quantities is true.
- f is true only when all of the logical quantities are false.

The truth table of the NOR operation is:

$$f = \overline{A + B + C}$$

A	B	C	f
0	0	0	1
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	0

Figure 4-10. NOR Operation

The NOR operation might be further illustrated by a switching circuit as shown in Figure 4-11. The light will represent TRUE when lighted, and from inspection of Figure 4-11, it will be readily seen that the light f can be TRUE only when all the logical quantities (switches) are FALSE (open). Any one or more logical quantity being TRUE will cause the light f to extinguish and thus be FALSE.

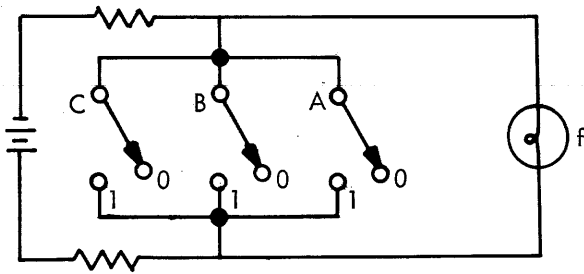


Figure 4-11. NOR Switching Circuit

The logic symbol for the NOR operation is a combination of OR and NOT in SDS logic symbology. As shown in Figure 4-12.

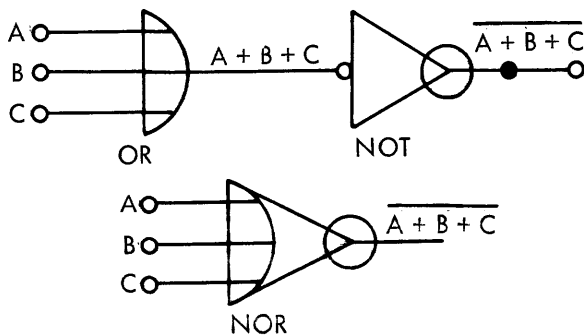


Figure 4-12. NOR Logic Symbol

The symbol of Figure 4-12 is known as the NOR gate.

## NAND

The NAND operation is a combination of AND and NOT, two of the three fundamental logical operations. The word NAND is a contraction of the statement NOT AND.

Expressed in a logical equation the NAND operation might be used as follows:

$$f = \overline{A B C}$$

Where f denotes the logical operation NAND. A, B, C are true logical quantities.

This equation says:

- f is true when any one or more of the logical quantities is false.
- f is false only when all the logical quantities are true.

The truth table of the NAND operation is Figure 4-13.

$$f = \overline{A B C}$$

A	B	C	f
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	0

Figure 4-13. NAND Operation

The logical NAND operation might be further illustrated by a switching circuit as shown in Figure 4-14. The light will represent true when lighted. From inspecting Figure 4-14, it will be readily seen that the light f will be false (unlit) only when all switches are true (closed).

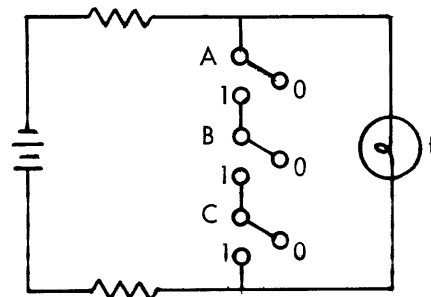


Figure 4-14. NAND Switching Circuit

The logic symbol for the NAND operation is a combination of the AND and NOT in SDS logic symbology as shown in Figure 4-15.

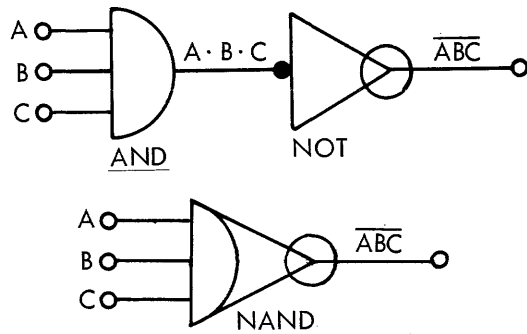


Figure 4-15. NAND Logic Symbol

The symbol of Figure 4-15 is known as the NAND gate.

Summary of the 5 operations and their symbols.

LOGICAL QUANTITIES

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

LOGICAL FUNCTION

(OR)	*NOT	f(NOR)	f(AND)	**f(NOT)	f(NAND)
0	1	1	0	1	1
1	0	0	0	1	1
1	0	0	0	1	1
1	0	0	0	1	1
1	0	0	0	1	1
1	0	0	0	1	1
1	0	0	1	0	0

\* using OR as input to NOT function.  
 \*\* using AND as input to NOT function.  
 All other functions use A, B, C, as inputs.

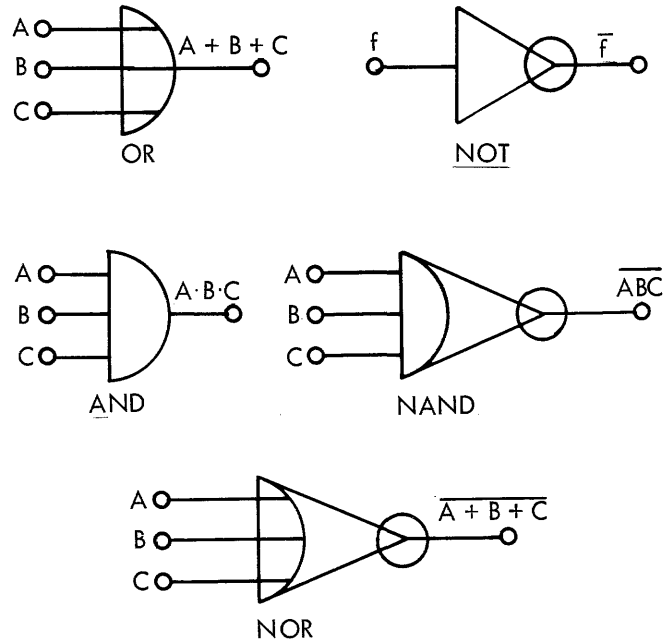


Figure 4-16.

In the preceding work discussed in this section, we have gone over the operation of AND, OR, NOT, NOR, and NAND. These operations have been discussed as a method of leading into the algebraic properties of logic and logical algebra. From what has been said previously it is now possible to discuss other properties of logical algebra. These other properties will be stated in logical algebraic form, and where necessary, additional comments will be made. From what has gone before, many of the statements will be self explanatory.

Identities are statements of absolute equality between the left member and the right member separated by an equality sign. In the case listed below the equations are correct, regardless of the values of the variables. The reader should verify the validity of the following statements for himself by substituting the "true" (1) and "false" (0) for the variables in all possible situations.

IDENTITIES

- $A + \bar{A} = 1$
- $A\bar{A} = 0$
- $A \cdot A = A$
- $A + \bar{A}B = A + B$
- $A + AB = A$
- $A + A = A$
- $0 + A = A$
- $1A = A$
- $1 + A = 1$
- $0 \cdot A = 0$
- $(A + B)(A + C) = A + BC$
- $\bar{\bar{A}} = A$
- $\overline{(A+B)} = \bar{A} \times \bar{B}$  or  $\bar{A}\bar{B}$
- $\overline{\bar{A}\bar{B}} = \bar{A} + \bar{B}$
- $A(B+C) = AB + AC$



Examples of proofs for the above identities.

$$(1) \quad A + \bar{A} = 1 \quad \text{If } A = 1 \text{ then } \bar{A} = 0$$

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

$$\text{If } A = 0 \text{ then } \bar{A} = 1$$

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

The statement is true for all possible values of A, and thus the identity has been proven.

(4)	A	$\bar{A}$	B	$\bar{A}\bar{B}$	$A + \bar{A}B$	=	A+B
	1	0	0	0	1	=	1
	0	1	0	0	0	=	0
	1	0	1	0	1	=	1
	0	1	1	1	1	=	1

This table proves that  $A + \bar{A}B = A + B$

$$(11) \quad (A + B)(A + C) = A + BC$$

A	B	C	(A+B)	(A+C)	BC	(A+B)(A+C)	=	A + BC
0	0	0	0	0	0	0 x 0	=	0 + 0
0	0	1	0	1	0	0 x 1	=	0 + 0
0	1	0	1	0	0	0 x 0	=	0 + 0
0	1	1	1	1	1	1 x 1	=	0 + 1
1	0	0	1	1	0	1 x 1	=	1 + 0
1	0	1	1	1	0	1 x 1	=	1 + 0
1	1	0	1	1	0	1 x 1	=	1 + 0
1	1	1	1	1	1	1 x 1	=	1 + 1

Therefore the identity has been proven since the identity holds true for all possible combinations of variables.

The reader should work out this same sort of proof for any of the identities he feels there may be some question about. It is mandatory that the reader be able to justify every identity and in addition be able to use these identities from memory since these are used in many cases to simplify logical equations.

The following identities may seem trivial but they prove some very important features of the operations stated. These features will be covered with each of the identities which are called theorems.

$$1. \quad A + B = B + A$$

$$AB = BA$$

These two statements say that the order of arrangement of the logical quantities does not affect the truth of the

statement. In other words, the AND and OR operations are commutative.

$$2. \quad A + (B + C) = (A + B) + C = A + B + C$$

$$A(BC) = (AB)C = ABC.$$

These statements say that the order of performing the logical operations in a given statement does not affect the end result, which in each case, for each line above, will be the same. In other words the AND and OR operations are associative.

The next step is to apply the information given previously in this section. In application the aim will be to simplify logic statements which appear at first glance to be complicated.

Given the following statement, simplify it as much as possible using the identities given previously.

$$f = A B C + A B \bar{D} + A \bar{C} + \bar{A} \bar{B} \bar{C} \bar{D} + \bar{A} C$$

rearrange the terms and group:

$$f = A B C + A \bar{C} + \bar{A} \bar{B} \bar{C} \bar{D} + \bar{A} C + A B \bar{D}$$

$$= A(B C + \bar{C}) + \bar{A}(\bar{B} \bar{C} \bar{D} + C) + A B \bar{D}$$

Apply the simplifying identity  $A + \bar{A} B = A + B$  to the first two terms as follows:

$$B C + \bar{C} = B + \bar{C}$$

$$\bar{B} \bar{C} \bar{D} + C = \bar{B} \bar{D} + C$$

Substitute the result in the statement to get

$$f = A(B + \bar{C}) + \bar{A}(\bar{B} \bar{D} + C) + A B \bar{D}$$

Expand to get

$$f = A B + A \bar{C} + \bar{A} \bar{B} \bar{D} + \bar{A} C + A B \bar{D}$$

Apply the simplifying identity  $A + A B = A$  to the terms  $A B + A B \bar{D}$

$$A B + A B \bar{D} = A B$$

Substitute the result to get

$$f = A B + A \bar{C} + \bar{A} \bar{B} \bar{D} + \bar{A} C$$

No further simplification appears possible. Therefore the following identity:

$$A B C + A B \bar{D} + A \bar{C} + \bar{A} \bar{B} \bar{C} \bar{D} + \bar{A} C = A B + A \bar{C} + \bar{A} C + \bar{A} \bar{B} \bar{D}$$

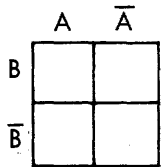
By this simplification it was possible to reduce the number of logical quantities from 14 to 9, which represents

considerable reduction in logical statement complexity and in hardware when it is implemented.

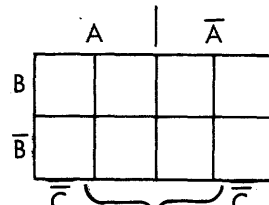
## VEITCH DIAGRAMS

There is no assurance in the reduction or simplification system used that the statement has been reduced to its absolutely simplest form. At least this is the case for a beginner or someone not completely familiar with reduction possibilities. There should be a more certain, less laborious method to use in simplifying logic statements. A simplification system, for a reasonable number of variables is the Veitch diagram.

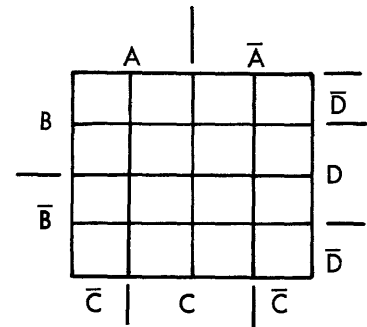
Veitch diagrams are a quick and easy way to find the simplest logical equation needed to express a given function. Veitch diagrams may be constructed for any number of variables, but they become difficult to use when more than 4 variables are involved. Veitch diagrams for use with two through six variables are illustrated in Figure 4-17.



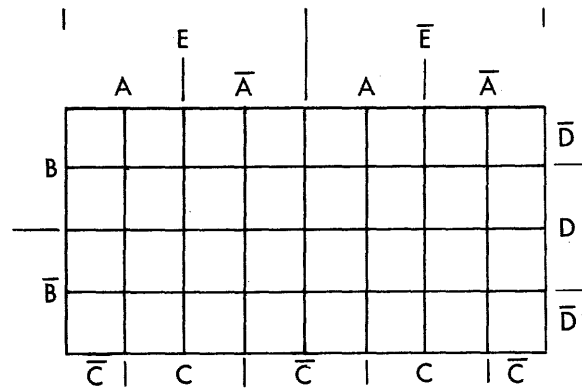
2 VARIABLES



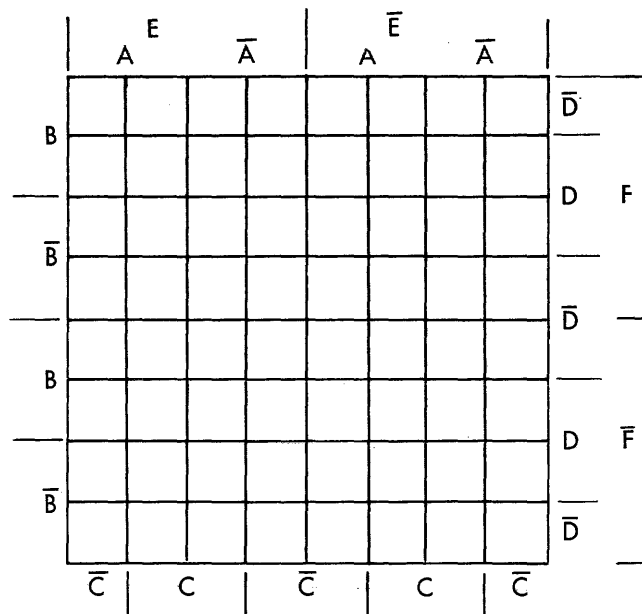
3 VARIABLES



4 VARIABLES



5 VARIABLES



6 VARIABLES

Figure 4-17. Veitch Diagrams

Each variable has 2 possible states. Therefore, the number of squares in the diagram, to represent all states must be equal to  $2^n$ , where n is the number of variables to be indicated by the diagram. Thus, for 4 variables there must be  $2^4$  squares, or 16 squares. For an 8 variable diagram, the number of squares would be 256, a rather large diagram. For variables in excess of 5 or 6, it is recommended that the student refer to Boolean algebra texts for other methods of simplification.

Each square of a Veitch diagram refers to a unique combination of variables. This is illustrated below for a 4 variable Veitch diagram.

		A		$\bar{A}$				
B		1	2	3	4	$\bar{D}$	1	$A B \bar{C} \bar{D}$
		5	6	7	8	D	2	$A B C \bar{D}$
$\bar{B}$		9	10	11	12	$\bar{D}$	7	$\bar{A} B C D$
		13	14	15	16	D	16	$\bar{A} \bar{B} \bar{C} \bar{D}$
		$\bar{C}$	C	$\bar{C}$	C			

To illustrate the use of the Veitch diagram the same equation as was previously used to illustrate the simplification identities will be diagrammed.

$$f = A B C + A B \bar{D} + A \bar{C} + \bar{A} \bar{B} \bar{C} \bar{D} + \bar{A} C$$

4 variables: A, B, C, D

		A		$\bar{A}$		
B		1	2	3	4	$\bar{D}$
		5	6	7	8	D
$\bar{B}$		9	10	11	12	$\bar{D}$
		13	14	15	16	D
		$\bar{C}$	C	$\bar{C}$	C	

The numbers in each square are for clarity in explanation only and are not a part of a standard Veitch diagram.

Plot the terms of the logical function term by term. The diagram, when this operation is completed, would appear as shown below. A "1" placed in each square is representative of a term. Remember that a term entered in a Veitch diagram may cause more than one square to have a 1 entered in it. Be certain you have placed a 1 in every square which represents the term.

		A		$\bar{A}$			
B		1	1	1		$\bar{D}$	$ABC = 2 \& 6$
		1	1	1		D	$AB\bar{D} = 1 \& 2$
$\bar{B}$				1		$\bar{D}$	$A\bar{C} = 1, 5, 9 \& 13$
				1	1	D	$\bar{A}\bar{B}\bar{C}\bar{D} = 16$
		$\bar{C}$	C	$\bar{C}$	C		$\bar{A}C = 3, 7, 11, 15$

There are a few rules which are used with the Veitch diagram to insure that the resultant simplified equation is in fact correct:

1. If 1's are located in adjacent squares or at opposite ends of any row or column, one of the variables may be dropped.

Example: adjacent squares 15 & 16

$$\begin{aligned} (15) \quad & \bar{A} \bar{B} C \bar{D} \\ (16) \quad & \bar{A} \bar{B} \bar{C} \bar{D} \\ & \bar{A} \bar{B} C \bar{D} + \bar{A} \bar{B} \bar{C} \bar{D} \\ & \bar{A} \bar{B} \bar{D} (C + \bar{C}) = \bar{A} \bar{B} \bar{D} \end{aligned}$$

Opposite ends of a row or column 13 & 16

$$\begin{aligned} (13) \quad & \bar{A} \bar{B} \bar{C} \bar{D} \\ (16) \quad & \bar{A} \bar{B} C \bar{D} \\ & \bar{A} \bar{B} \bar{C} \bar{D} + \bar{A} \bar{B} C \bar{D} \\ & \bar{B} \bar{C} \bar{D} (A + \bar{A}) = \bar{B} \bar{C} \bar{D} \end{aligned}$$

2. If any row or column of squares, any block of 4 squares or the 4 end squares of any adjacent rows or the four corner squares are filled with 1's, two of the variables may be dropped.

Example: Consider squares 1, 5, 9, 13. These could be represented by  $A \bar{C}$ , since B &  $\bar{B}$ , D &  $\bar{D}$  have been used. This can be better illustrated by simplifying just the 4 terms represented by these squares.

$$\begin{aligned} (1) \quad & A B \bar{C} \bar{D} \\ (5) \quad & A B C \bar{D} \\ (9) \quad & \bar{A} \bar{B} \bar{C} \bar{D} \\ (13) \quad & \bar{A} \bar{B} C \bar{D} \\ & A B \bar{C} \bar{D} + A B C \bar{D} + \bar{A} \bar{B} \bar{C} \bar{D} + \bar{A} \bar{B} C \bar{D} \\ & A B \bar{C} (\bar{D} + D) + \bar{A} \bar{B} \bar{C} (D + \bar{D}) \\ & A B \bar{C} + \bar{A} \bar{B} \bar{C} \\ & A \bar{C} (B + \bar{B}) = A \bar{C}. \end{aligned}$$

Thus squares 1, 5, 9, 13 can be represented by the term  $A \bar{C}$ .

3. If any 2 adjacent rows or columns, or the top and bottom rows, or the right and left hand columns are completely filled with 1's, three of the variables may be dropped.

Example: For this example, assume that the two left columns are filled with 1's. Thus we would be considering squares 1, 2, 5, 6, 9, 10, 13, 14.

$$\begin{aligned}
 & (1) \quad (2) \quad (5) \quad (6) \quad (9) \quad (10) \quad (13) \quad (14) \\
 & A\bar{B}\bar{C}\bar{D} + A\bar{B}C\bar{D} + A\bar{B}\bar{C}D + A\bar{B}CD + A\bar{B}\bar{C}\bar{D} + A\bar{B}C\bar{D} + A\bar{B}\bar{C}D + A\bar{B}CD \\
 & A\bar{B}\bar{D}(\bar{C}+C) + A\bar{B}D(\bar{C}+C) + A\bar{B}\bar{D}(\bar{C}+C) + A\bar{B}D(\bar{C}+C) \\
 & A\bar{B}\bar{D} + A\bar{B}D + A\bar{B}\bar{D} + A\bar{B}D \\
 & A\bar{B}(\bar{D} + D) + A\bar{B}(\bar{D} + D) \\
 & A\bar{B} + A\bar{B} = A(\bar{B} + B) = A
 \end{aligned}$$

This same example can be worked out for the other conditions which permit dropping three variables. This is left as a reader's exercise.

4. To reduce the original equation to its simplest form, sufficient simplification must be made until all 1's have been included in the final equation. Ones may be used more than once and largest possible combinations should be used.

The rules above are illustrated in Figure 4-18.

Now we can look at the Veitch diagram and come up with the simplified resultant equation.

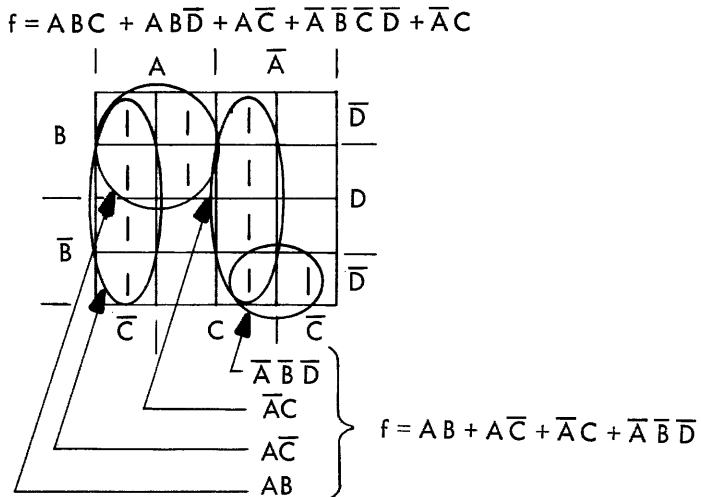


Figure 4-18. Veitch Diagram Rules

The ovals and circles are used to insure groupings and also to insure that all 1's are used at least once in the simplified statement. The circle cannot be used for the four corners or ends of columns or rows as detailed in the rules. However, X's or 0's may be used to indicate usage.

From the rules, illustrations, and discussions it should have been deduced by the reader that to drop a variable both the variable and its inverse must appear in two terms with the other variables of the terms being identical. The only simplification identity that has been applied is  $A + \bar{A} = 1$ . This is best illustrated by looking at squares 15 & 16. It is obvious that both C and  $\bar{C}$  can be dropped. Further, looking at column 1, squares 1, 5, 9, 13, it is again obvious that B and  $\bar{B}$ , and D and  $\bar{D}$  appear along with  $\bar{A}\bar{C}$  and thus B,  $\bar{B}$ , D, and  $\bar{D}$  can be dropped.

Learning to recognize these essentials, and which variables can be dropped, is the secret to success with use of the Veitch diagram in logic simplification.

### Mechanization of Logic

Logic mechanization may take many forms. At this point it is not worth while to discuss what forms they may take, but rather to look at the logic symbols of mechanization. For instance we could look at the logic described by the equation,

$$f = ABC + A\bar{B}\bar{D} + A\bar{C} + \bar{A}\bar{B}\bar{C}\bar{D} + \bar{A}\bar{C}$$

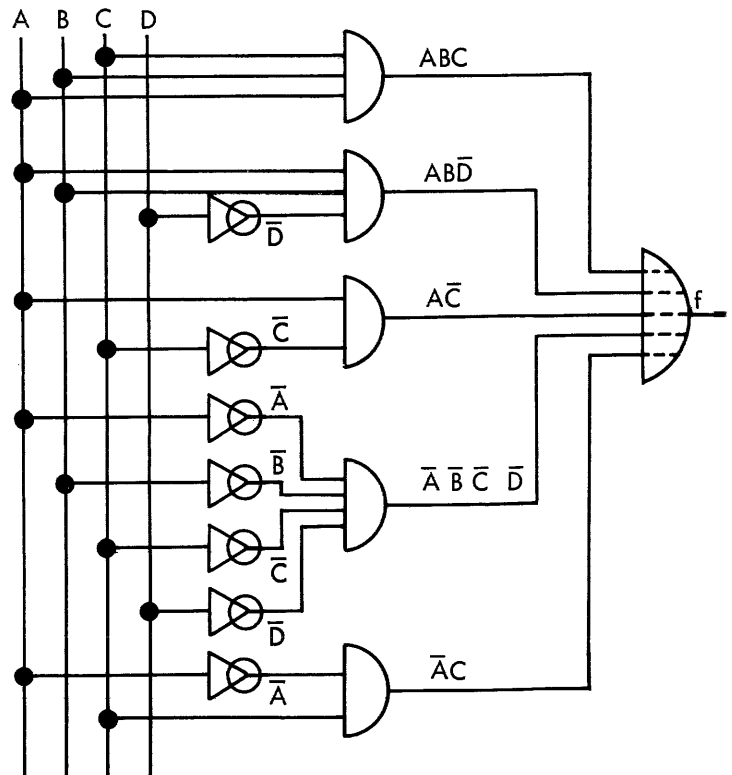


Figure 4-19. Mechanization of Logic

This is the mechanized version of the same equation with which we started earlier in this section.

To see the reduction in mechanization complexity, the simplified and reduced version of the equation is shown below.

$$f = AB + A\bar{C} + \bar{A}C + \bar{A}\bar{B}\bar{D}$$

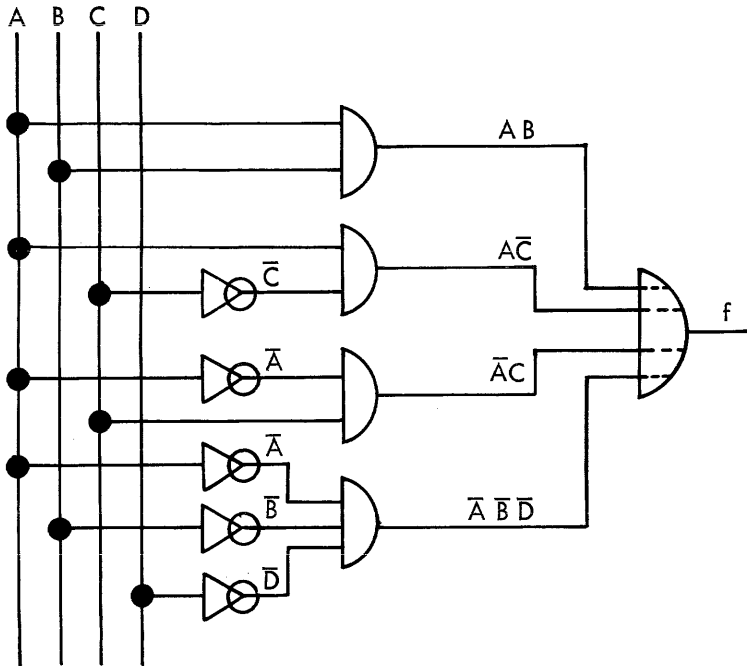


Figure 4-20. Simplified Mechanization of Logic

It is not worthwhile to go further with mechanization of logic since this will be the subject of your study in the SDS School for which you are registered.

## SECTION 5 LOGIC ELEMENTS

### INTRODUCTION

Numbering system manipulation, logical algebra operations, symbols, and terms have been the subject of previous sections. The actual electronics of implementation used in hardware has not been presented. In this section we will be concerned with the general theory of various standard electronic circuits used in SDS computers.

Prior to starting this study, it is necessary that we understand the objective being sought. The general understanding of how a particular circuit works rather than the specific considerations of resistor, transistor, capacitor or voltages chosen by the design engineer is the main objective. It is true that the understanding of the "why" would be beneficial if we were to be prepared to design equipment. This is not the case and we will concern ourselves only with "how".

### STANDARD LOGIC LEVELS

The SDS logic circuits are based on the use of diode gating and representation of logic signals as DC voltage levels. The standard logic levels will be as shown below. Bear in mind that the value of true could be +4 or +6, but for this discussion +8 has been chosen.

$$\begin{aligned} 0 &= \text{False} = 0\text{V} \\ 1 &= \text{True} = +8\text{V} \end{aligned}$$

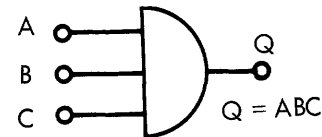
Functions of any conceivable complexity can be carried out by the proper combination of signals according to simple rules. The gating configuration used to implement the desired functions can be AND, OR, NAND, NOR or combinations of all of these. The results of gating operations are usually presented to flip-flops that perform the function of remembering the output condition of gates even though the outputs may have been transitory. The flip-flop circuits used are essentially all the same.

### The AND Function

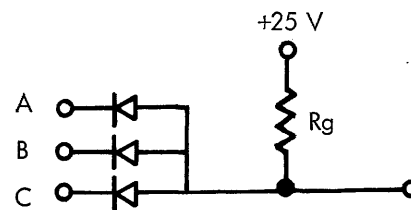
The AND functions are those that have two or more terms, all of which must be true in order for the logical expression to be true. That is, if  $Q = A \cdot B \cdot C$ , then Q will be true only when A and B and C are true. SDS AND gates are formed by using diodes with a common anode connection such that any input held at ground

will hold the output at ground. Only when each input is at true, a nominal +8V, will the output rise to +8V or true.

Standard AND gate



Symbol



Circuit

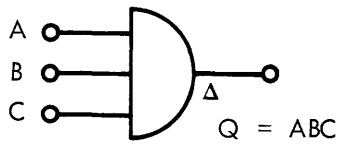
If the A input is false (0 V) the top diode will conduct heavily and the anode will be at approximately 0 V due to the  $1R$  drop across  $R_g$ . If then inputs B and/or C are true (+8 V) they will be reverse biased and cannot in that condition affect the 0 V established at the output by the top diode. If all diodes have 0 V inputs then the output is 0 V and the current through  $R_g$  is the result of all diodes conducting simultaneously.

If all inputs are true (+8 V) the diodes still conduct more or less equally but the drop across  $R_g$  is only 17 V and the anodes and hence the output is locked at about +8 V, or true. The voltage at the anodes will differ from 8 V by the voltage drop across a diode, which from a practical point of view is insignificant.

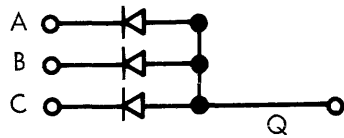
The AND gate shown above is only a 3-input AND. The standard AND gate can be expanded to include up to 30 input terms by adding more diodes. These diodes are provided by the expander AND gates.

### Expander AND Gate.

Expander AND gates are designated symbolically by a  $\Delta$  at the output terminal. Physically they are the same as the AND gate except that they do not include gate load resistors.

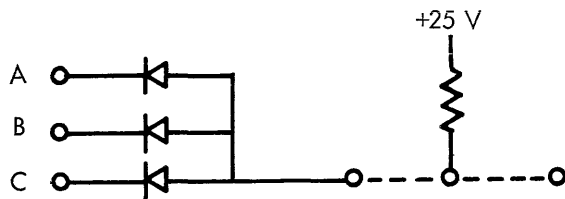


Symbol

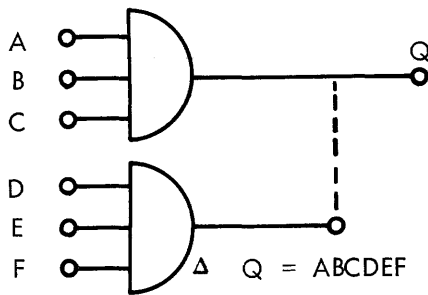


Circuit

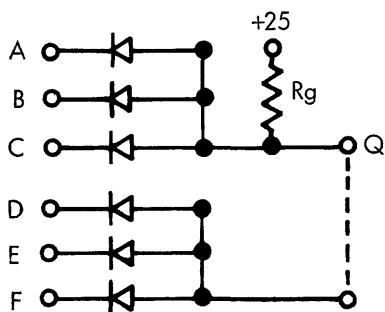
The operation of the expander AND gate is identical with that described for the standard AND, if one added an external load resistor with +25 V as shown below:



In actual practice the expander AND gate is combined with the standard AND gate as shown below:



Symbol

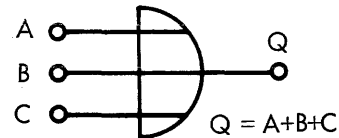


Circuit

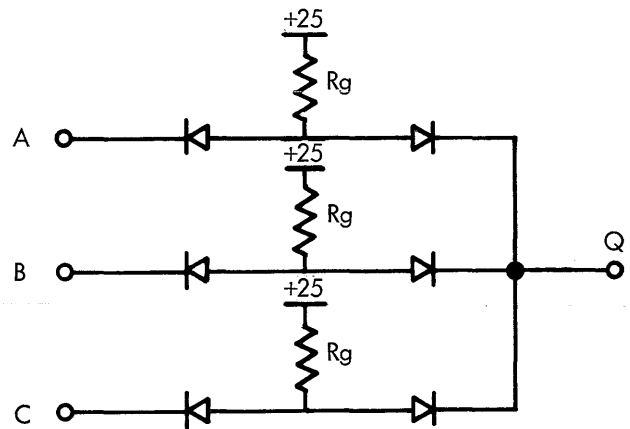
### The OR Function

OR functions are those that have two or more terms any of which being true will cause the logical expression to be true. That is, if  $Q = A + B + C$ , Q will be true if A or B or C is true. SDS OR gates are diode circuits, arranged so that any input held at a nominal +8 V (true) will cause the output to be true (+8 V).

The standard OR gate:



Symbol



Circuit

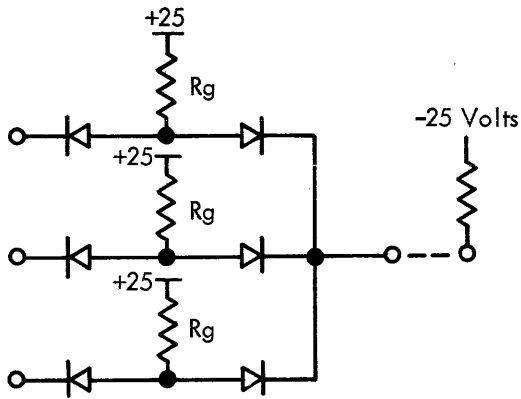
Inspection of the standard OR gate circuit reveals that the input diode and associated resistor  $R_g$  are identical to the configuration of the AND gate. The difference between a standard AND gate and a standard OR gate is the inclusion of the output diodes and individual diode load resistors on the OR gate.

In the standard OR gate the anode of any input diode will follow the input. The voltage at the bottom of any load resistor will be determined by the input to its associated diode. Thus an output diode anode has on it the voltage appearing at the input terminal associated with it.

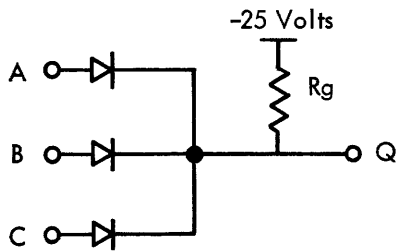
To make the standard OR diodes work at all there must be a load resistor and source of voltage attached to their common-connected cathodes. Though it is not shown in the diagram of the circuit, it must be there to have the circuit work at all. The circuit configuration would thus be:

## Gated Input OR Gate

Inspection of the diode previously shown reveals that the input diode and associated resistor of the standard OR gate are identical to the circuit for an AND gate. The basic difference between a standard AND gate and a standard OR gate is the inclusion of the output diode on the OR gate. It follows that the outputs of standard AND gates can be ORed by adding the required diodes in the OR-facing direction. Diodes connected in this manner are referred to as gated input OR gates. A connection for such an arrangement could be:

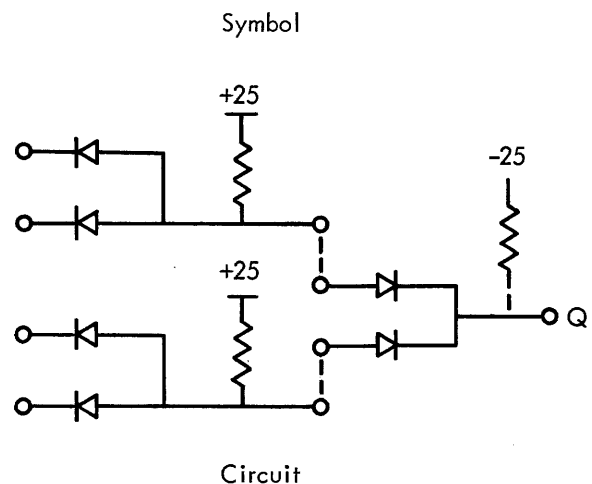
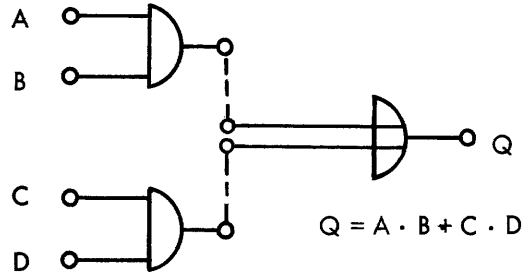


The AND oriented input diodes can be ignored for this circuit explanation since this has been explained in the discussing of the standard AND. Thus the OR circuit can be explained using only the following circuit.



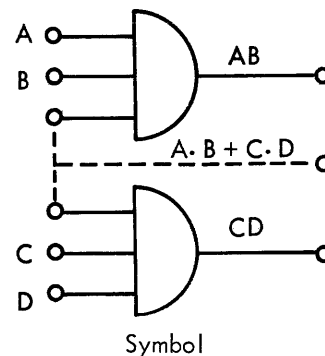
If an OR facing diode has +8 V at the input or anode terminal, the output at its cathode will be +8 V regardless that the other diodes have 0 V at their anode. With 0 V at the anode and +8 V at a cathode, a diode is reverse biased and effectively out of the circuit. Thus with any one or more inputs true (+8 V) the output will be true.

In this type of circuit it is essential that all unused input terminals be grounded. If this precaution is not taken, the output of the standard OR will be true at all times regardless of the inputs at the used terminals. To explain the reason for the necessity of grounding unused input terminals, assume that input C is unused, ungrounded, and terminals A and B are used and have false inputs. Under these conditions we would normally expect a false output at Q. However, with terminal C open, the anode of the C output diode would attempt to go to +25 V. The result would be conduction through the C output diode since it is now forward-biased. As a result, the output terminal would go positive and remain there. A positive output at Q represents a true condition which is not what we would expect with false at both used input terminals. Thus under any circumstance the output would be true, if an unused terminal of a standard OR were left ungrounded.

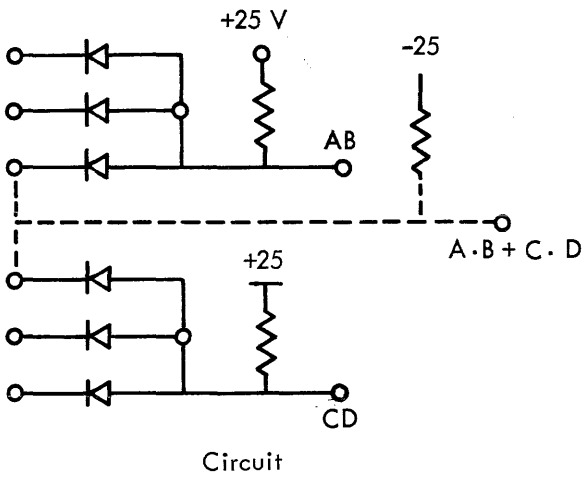


## Phantom OR Gate

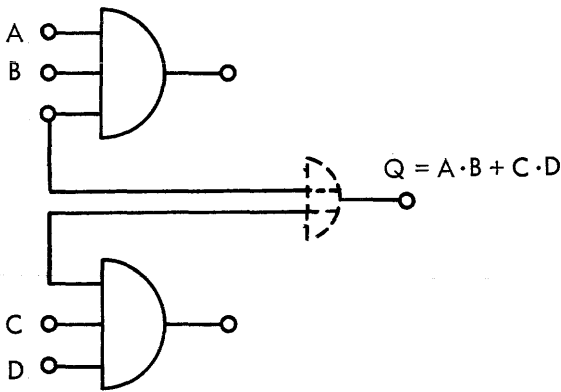
A more common and versatile way to obtain diodes in the OR-facing direction when ORing the outputs of AND gates is to use an extra input diode of the AND gate as the output diode.







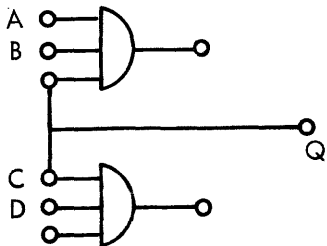
Inspection will show that this diode configuration is the same as that shown for the gated input OR gate. To clarify the logical operation on logic diagrams, the connection of the diodes is sometimes drawn as follows:



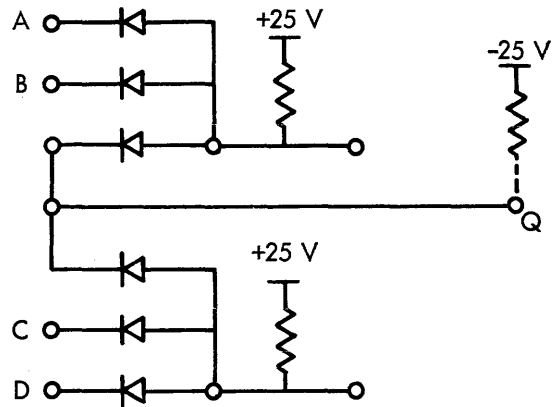
The OR gate represented by the dashed lines is referred to as a phantom OR gate and readily clarifies the fact that the output of the AND gates has been ORed together.

The AND/OR Gate

The circuit configuration described above with the OR portion being called a phantom OR is generally referred to at SDS as the AND/OR gate. The AND/OR gate is used as shown below:



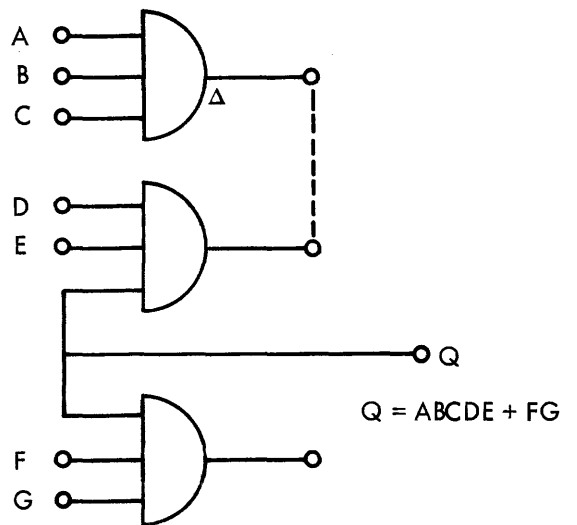
Symbol



Circuit

NOTE: It is felt that sufficient operation and description of diode circuitry has been presented to this point. If the reader feels the necessity to apply voltages and analyze the individual circuits, it is left to him as an exercise to do so. Understanding of these circuits is essential to understanding the computer when it is studied in school.

It is possible to increase the inputs to the AND/OR gate by use of an expander AND as shown in the following symbols. A drawing of the circuit is not shown. The reader can draw the circuit, if he feels the necessity. Enough information on circuits has been given thus far to make the drawing a simple task.

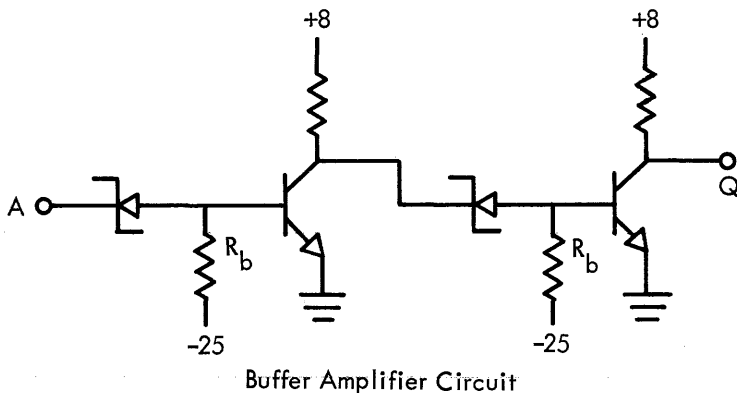
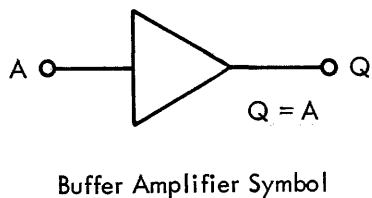
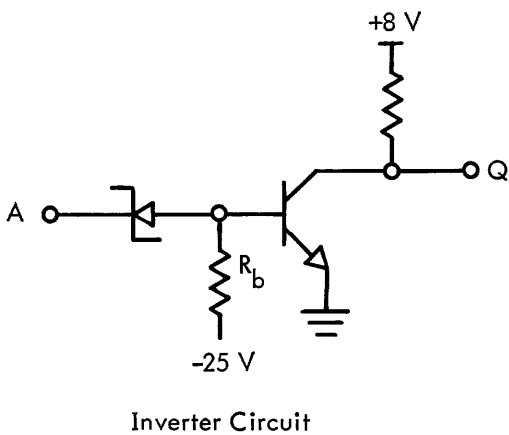
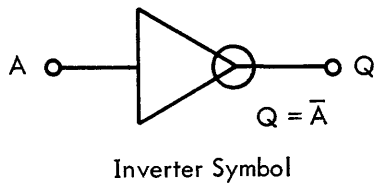


Symbol for Expanded AND/OR

## Logic Amplifiers

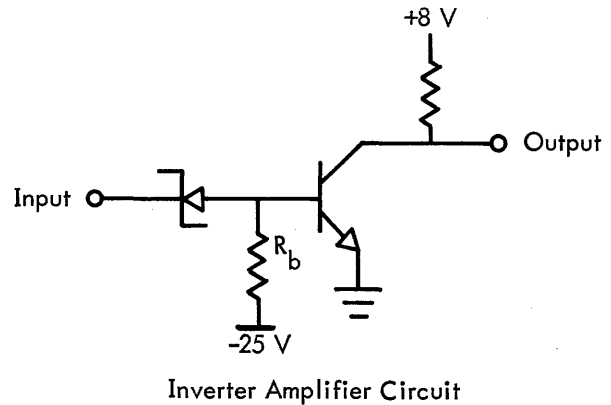
Logic amplifiers are used to amplify the output of diode gate structures, to generate the logical complement or inverse of an input signal, and to provide additional driving capability for heavily loaded logic terms.

SDS Logic amplifiers consist of an input diode with a controlled reverse voltage breakdown point (Zener diode) followed by either a single-grounded emitter transistor in the case of an inverter or two transistors in the case of a buffer amplifier.



NOTE: In practice the base resistor and coupling Zener diode to the second transistor may be eliminated.

To explain the operation of these circuits, a discussion of the inverter circuit is undertaken. The buffer amplifier will be explained also since the two circuits are identical in theory of operation. The only difference being that an input signal to an inverter is inverted once and in a buffer amplifier it is inverted twice.



When the input is held at 0 V (false) the input diode and base resistor provide a negative voltage bias to prevent the transistor from conducting. The Zener diode will have broken down and be conducting in a reverse direction under this condition and be holding the base at a nominal -4 V. With the base-emitter cut off, the transistor is in a cut-off state and collector voltage will rise to +8 V (true).

When the input rises to +8 V, the Zener diode will conduct more heavily in the reverse direction, but still maintain the nominal 4V drop across it. Therefore, +4V will be attempting to appear at the base of the transistor. The transistor base-emitter junction will thus be forward-biased and will, due to base-emitter current, limit the voltage at the base to some fraction of a volt near ground. The transistor being in full conduction will drop the collector voltage to near ground potential ( $\approx +.6V$ ) and the output will be false (approximately 0 V).

Therefore, with false (0 V) in we have gotten true (+8 V) out and with true in we have gotten false out, and we have an inverter amplifier.

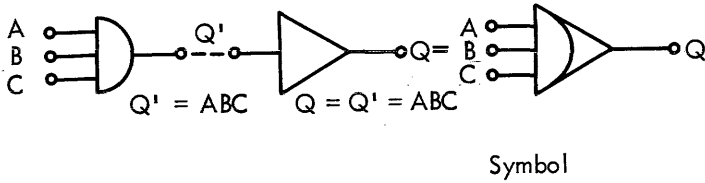
This same explanation applies to the second transistor circuit in series in the buffer amplifier circuit. An input signal goes through two inversions, one each as it passes through each transistor circuit, and as a result comes out unchanged in state but bolstered in capability, and we have a buffering (non-inverting) amplifier.

In SDS equipment there are combinations of all the circuits discussed previously to generate specific functions for specific needs in mechanization of logic. The symbols and functions of these combination circuits will

be given and discussed briefly. However, the circuits for them will not be analyzed since such discussion would be repetition of analyses which have been given previously.

BAND - Buffered AND

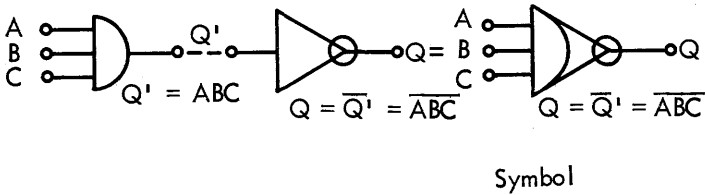
This is a combination of the non-inverting buffer amplifier and AND circuitry.



Logically the output of this combination of circuitry is the same as the output of a standard AND. The buffer amplifier is used to bolster the output to make the output capable of driving a heavier load than the direct AND output.

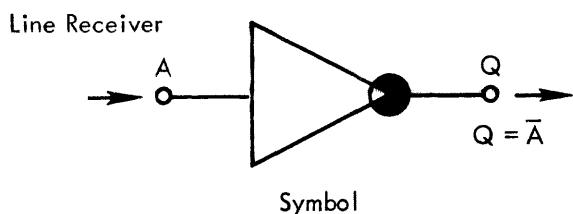
NAND - Negative AND or Not AND

This is a combination of the inverting amplifier and the standard AND circuitry.



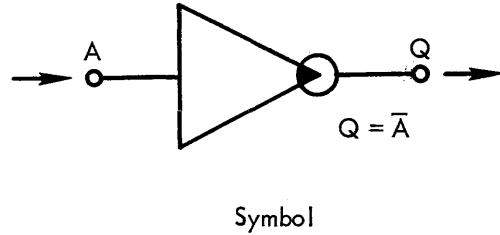
Logically the output of this combination of circuitry is the complement, or the inverse of the output of the standard AND. The inverting is used to accomplish the inversion as well as bolster the output signal drive capabilities.

The notable thing here is the significance of the small circle used with the logical amplifier symbol. In every case where a circle is used at the output of a logical amplifier symbol the indication is that the input signal has been inverted. This is true whether the circle is unfilled (as shown above) or partially filled as shown below:



The line receiver is used in receiving signals from some remote unit, inverting the signal for use, and changing the amplitude level where necessary. Logically, it serves the same function as an inverting amplifier, even though in actual use it does change signal amplitudes to the desired level where necessary.

Line Driver



The line driver is used in preparing a signal for transmission from one unit to another unit, to match the transmission line impedance, invert the signal and change the amplitude of the signal where necessary. Logically, it serves the same function as an inverting amplifier even though in actual use it does change levels of signals and does match the impedance of the transmission line.

Cable Driver

When it is desirable to distribute signals by means of 33-ohm coaxial cable, a cable driver is used rather than an ordinary inverter or buffer amplifier. Figure 5-1 shows a cable driver which may be used to drive one or two terminated 33-ohm cables. The output is from two unterminated collectors in parallel and hence must be terminated by 33 ohms to +4V at the receiving end of each cable connected to the driver. Further details on usage will be discussed when receiver-inverters and receiver-inverter-buffers are considered.

Input to Q1 is made by means of a conventional diode gate of the sort that was discussed in the NAND gate. If Q1 is cut off, Q2 is saturated and Q1's collector is held at Q2's Vbe sat = +0.8V. If Q1 is saturated, its collector is at Vbe sat = +0.4V. Since Q1's collector shifts only 0.4V from saturation to cutoff, there is no need for shunting D1-D3 with the familiar resistor-capacitor combination to compensate for Miller effect capacitance.

The driving output to the cable or cables is taken from the paralleled collectors of Q3 and Q4. Such a parallel combination has not only twice the power handling capacity of a single transistor but also only half the saturation resistance, permitting the "false" output to approach ground despite the large current handled by the parallel combination.

To take advantage of a parallel combination, the bases must be driven equally. If the bases were merely paralleled like the collectors and emitters, the transistor with the lower  $V_{be}$  would receive most of the available drive current and the transistors would not do equal amounts of work. So as not to encounter such a problem, the circuit of Figure 5-1 mutually isolates the drives to Q3 and Q4 whenever these transistors are saturated. If Q2 is saturated, collector current flows down through R1 and D4 and through R2 and D5. Thus points P1 and P2 are both at a potential of  $0.4 + 0.7V = +1.1V$ . Since VR1 and VR2 both have a Zener voltage of 3.3V, Q3 and Q4 are reverse-biased  $1.1 - 3.3 = -2.2V$ . If Q2 is cut off, however, points P1 and P2 are mutually isolated

by diodes D4 and D5 connected back to back in series, and Q3 and Q4 are saturated by current from separate 600-ohm drive resistors R1 and R2 respectively.

Since Q3 and Q4 are handling rather heavy base currents in order to switch heavy collector currents, there is the possibility of the circuit being slowed down by a stored charge in the base. So that stored charge will not be a problem, the capacitors shunting VR1 and VR2 are made considerably larger than would be necessary only to compensate for Miller-effect capacitance. Thus the bases of transistors Q3 and Q4 are over driven while the transistors are switching from one state to the other to "speed up" the transition.

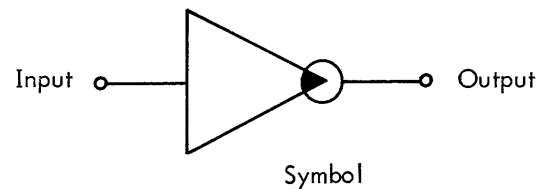
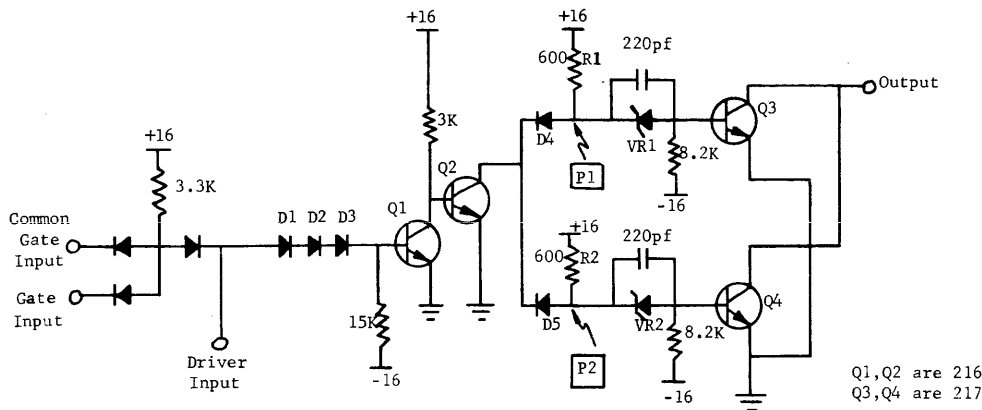


Figure 5-1. Cable Driver

### Receiver-Inverter

Signals are "picked off" 33-ohm coaxial cables by receivers as shown symbolically in Figure 5-2. Coaxial cables must be terminated in their characteristic impedance of 33 ohms regardless of receiver placement so that reflections do not occur in the line. The loading effect of the receivers is sufficiently small that the impedance discontinuities in the cable created by their presence do not cause significant reflections. Since, in addition, receivers draw very little current, a very large number of receivers may be driven from 33 ohm coaxial cable by one cable driver.

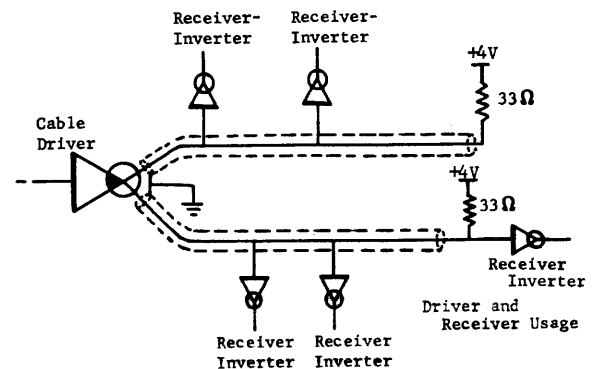


Figure 5-2. Symbolic Receiver "Pick-Off"

Figure 5-3 shows a receiver-inverter. As the name implies, the output is the inverse of the logical signal present on the coaxial line to which it is connected. There is also a receiver-inverter-buffer which has both inverted and non-inverted outputs. This latter circuit shall be discussed later.

Suppose that the input to the receiver-inverter is raised to +4 V. This saturates Q1 and places its emitter at  $4 - 0.4 = +3.6$  V. Point P1 is then placed at +0.96 V. This, however, is more positive than +0.80 V,  $V_{be}$  sat of Q2. Thus Q2 is saturated by the current flowing into its base from the 2.7K resistor and D2 is reverse biased (as is, of course, D3).

If the input to Q1 is connected to ground, Q2 should be cut off. If Q2 is cut off, P1 may drop to -1.4 V, pulled down by the 1.2K resistor to -16 V and limited to -1.4 V by the forward drops across D2 and D3. Q1 may be on the verge of conduction, but the base of Q2 has dropped sufficiently negative to cut it off completely.

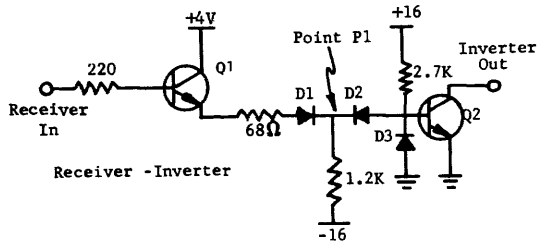


Figure 5-3. Receiver-Inverter

#### Receiver-Inverter-Buffer

Figure 5-4 shows a circuit similar to the receiver-inverter but which also includes a non-inverted output ("buffer out"). If the input is raised to +4 V, Q1 is saturated and point P1 is raised sufficiently positive to reverse bias diodes D2 and D3 and transistors Q2 and Q3 are saturated by their individual 5.6K resistors to +16 V. Q4's base is lowered to +0.4 V by Q3 saturating and is therefore cut off.

If the input is connected to ground, point P1 is pulled down by the 1.2K resistor to -16 V. P1 drops to -1.4 V, limited by the forward drops across D2 and D4. With P1 at this voltage Q1 is only on the verge of conduction, and both Q2 and Q3 are cut off with a reverse base bias of -0.7 V in both cases. Q4 is then saturated by the current flowing down through Q3's collector load resistor.

Since both the receiver-inverter and the receiver-inverter-buffer have the same input circuit, the usage rules for both are the same. It should be remembered, however, that the extra transistor in the "buffer output" causes the non-inverted output to be delayed somewhat with respect to the inverted output.

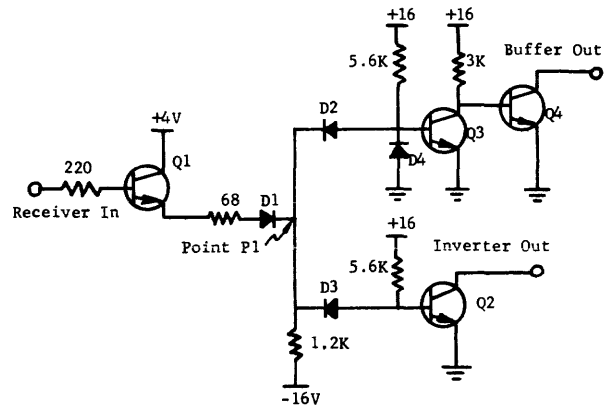


Figure 5-4. Receiver-Inverter-Buffer

#### Flip-Flop

A flip-flop is illustrated in Figure 5-5. In actuality, there are several groups of "Set", "Reset", "Enable" inputs and two "Gate Outputs", all groups sharing the same "Aux. Set" and "Aux. Reset" inputs. One group is shown for clarity.

Figure 5-6 represents only the central latch and the DC set input. Suppose the flip-flop is in the "Reset" state, this implies Q1 is saturated. Thus Q1's collector is at 0.4 V, and Q3 must also be saturated. Q3, however, is drawing practically no collector current but has considerable base current and hence adds only 50 millivolts to  $V_{cd}$  of Q1, so that the voltage at Q3's collector is 0.45 V. However, the base threshold to turn on Q2 is 0.6 V (0.8 is required for saturation), and hence Q2 is turned off. Since Q2 is cut-off presents a very high impedance to the emitter of Q4, Q4's base-collector diode becomes forward-biased and Q4's base current flows out of its collector into the base of Q1, keeping it saturated.

The cross-coupling transistors Q3 and Q4 operate with essentially constant base current (except when "DC Set" is used) which makes them capable of extremely fast switching. Consequently, this is one of the fastest forms of cross-coupling that can be used in a flip-flop.

Suppose that while the flip-flop is in the reset state as described above, the DC set input is momentarily grounded or connected through a resistance to a negative level (the diode D2 on the DC set input prevents excursions more negative than -0.7 V). This causes the diode D1 to become forward biased and the current that was flowing into the base of Q4 and from Q4 to Q1 now flows to ground instead. This cuts off Q4, which cuts off Q1, and hence Q3's base current flows to the base of Q2, saturating Q2 and thus placing the flip-flop in the "Set state".

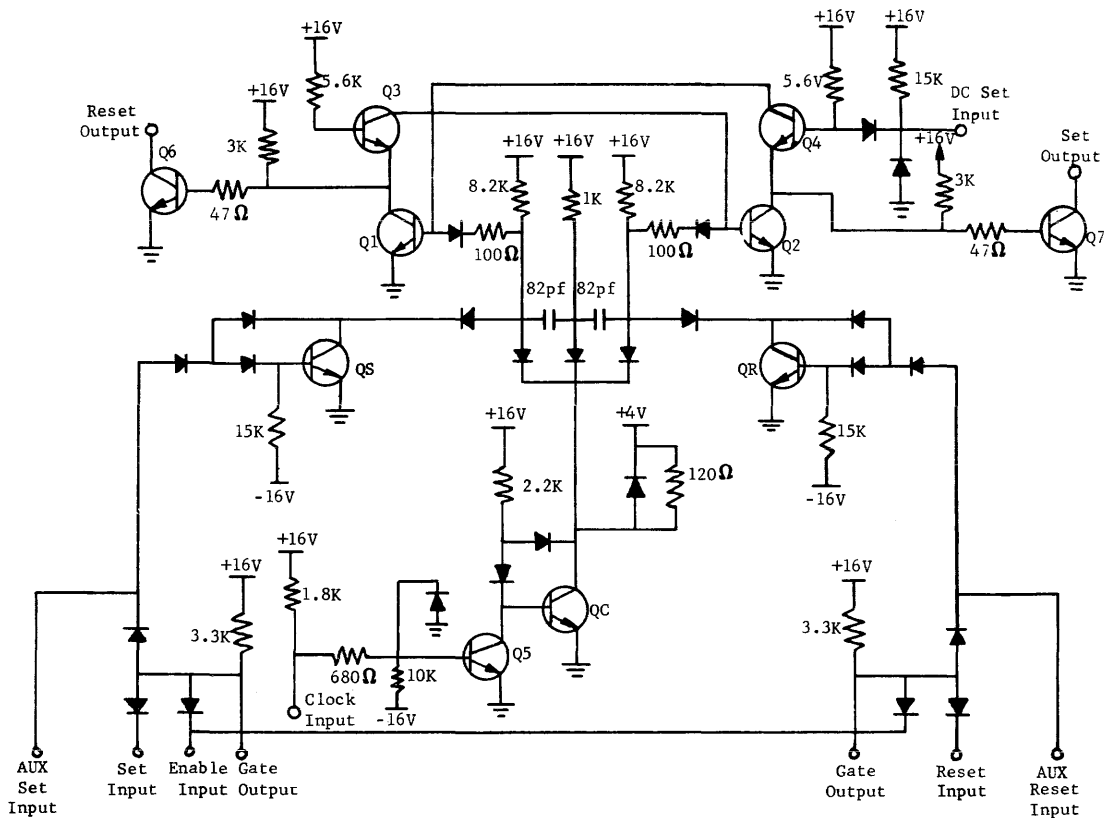


Figure 5-5. Flip-Flop

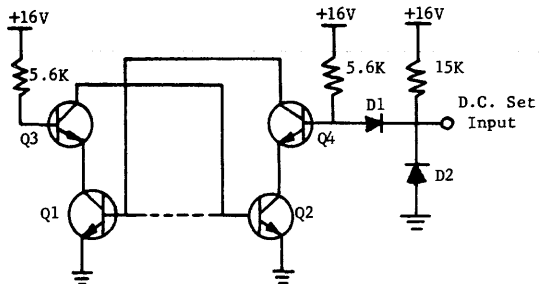


Figure 5-6. Central Latch and DC Set Input

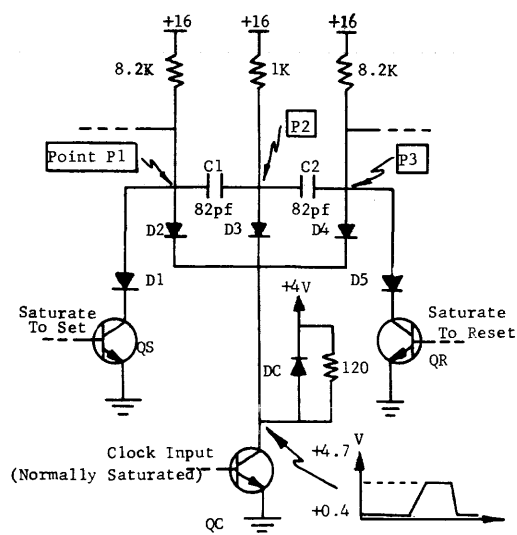


Figure 5-7. Setting/Resetting Circuitry

Figure 5-7 shows just the circuitry which is used in setting and resetting the flip-flop. Suppose that the flip-flop is in the "Reset" state (Q1 saturated) and we wish to set it by saturating QS. In the absence of a clock pulse, this would have no effect because QC is already saturated. However, suppose QC is cut off and QS is saturated. Then the collector of QC rises to +4V plus the drop across DC ( $4.0\text{V} + 0.7\text{V}$ ) = 4.7V and point P2 rises to 4.7V plus the drop across D3 ( $4.7\text{V} + 0.7\text{V}$ ) = 5.4V. If QS is saturated, however, point P1 is held down to  $V_{ce\text{ sat}}$  of QS plus the drop across D1 ( $0.4 + 0.7 = +1.1\text{V}$ ). This leaves C1 charged, with a voltage across it of  $(5.4\text{V} - 1.1\text{V}) = 4.3\text{V}$  with the right-hand end (P2) positive.

When the clock pulse falls (which occurs very quickly), P2 falls down rapidly to +1.1V. Since the voltage across C1 can't change instantaneously, C1 must still have 4.3V across it after P2 has dropped to 1.1V, so P1 drops to  $(1.1 - 4.3) = 3.2\text{V}$ . At this point, refer to Figure 5-8 which shows the circuitry of Figure 5-7 together with the central latch. Now that P1 has gone negative, D6 is, for the first time, forward biased while both D1 and D2 are reverse biased.

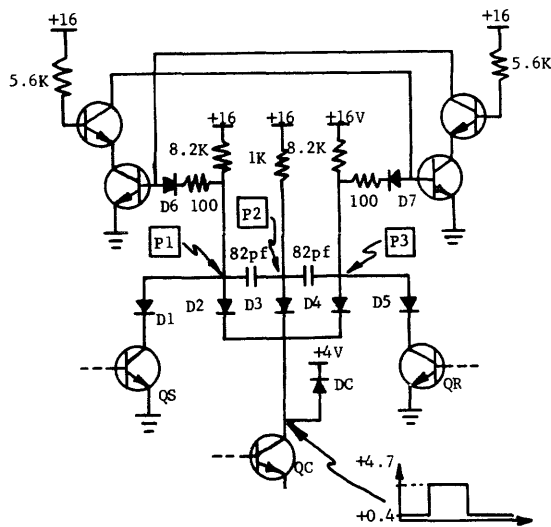


Figure 5-8. Setting/Resetting and Central Latch

The only way C1 can discharge then, is to draw current out of the base of Q1, cutting it off. The 100-ohm resistor in series with D6 forces the discharge to take place slowly enough to insure that Q1 stays cut off until the latch is fully regenerated.

QR, on the other hand, remained cut off during this procedure; clearly, one should not try to set and reset a flip-flop at the same time. Since QR was cut off, P3 rose and fell with P2 and no charge was built up on C2.

We can refer back to Figure 5-5 for most of the remainder of our discussion of this flip-flop. Q6 and Q7 are buffer amplifiers. If connected through load resistors to +4V, they will provide logical voltage outputs which will be "true" when the flip-flop is in the respective states indicated by the two outputs. If, for example, Q1 is cut off, we want Q6 saturated. However, when Q1 is cut off, current from Q3 is going to the base of Q2 to keep it saturated. Thus we need the 3K resistors to +16 V to provide drive to Q6 and Q7. The 47-ohm resistors going to the bases of Q6 and Q7 prevent these transistors from diverting drive from Q3 or Q4 which should be going to the bases of Q2 and Q1 respectively.

QS and QR are straightforward inverting amplifiers with the exception of the antisaturation circuit. This circuit is necessary because there are normally several sets of input gates and thus the drives to the bases of QS and QR vary with the number of input gates given "true" inputs at a given time. Some flip-flops have inputs arranged so that the drive to turn on the input amplifiers QS and QR is constant (see, for example, the NAND-flop) and hence antisaturation circuits on these transistors are not necessary.

The clock amplifier, composed of QC and Q5, is a comparatively straightforward non-inverting amplifier with an input impedance of 600 ohms (many such clock inputs in parallel have a combined impedance of 33 ohms which is the impedance of the lines on which the clock is distributed).

The upper limit of QC's collector swing is clamped, as has been observed. In addition, QC is also equipped with an antisaturation circuit so that the saturation potentials of QR, QS, and QC are equalized.

### NAND-Flip-Flop

Figure 5-9 shows a variation on the flip-flop called a NAND flip-flop. The central latch and buffer amplifiers are the same as in the flip-flop previously discussed. The triggering circuitry, however, is somewhat different.

In Figure 5-10 is shown just the setting and resetting circuitry. Normally, QC is saturated, P2 is at +0.4 V and P1 and P3 are at +1.1 V. If either QR1 or QS1 is saturated, neither P1 nor P3 will be affected as long as QC is saturated (and has the same  $V_{ce\ sat}$  as QS1 and QR1) because, in the case of P1,  $V_{ce\ sat}$  of QC plus the drop across D1 equals  $V_{ce\ sat}$  of QR1 plus the drop across D3, and similarly for P3. Suppose we cut off QR1, leaving QS1 saturated. When the clock pulse rises to +4 V, P2 rises to 4.7 V and P1 rises to +5.4 V (4.7 plus the additional drop across D1). But if QS1 is saturated, P3 is at 0.4 V + 0.7 V = 1.1 V and D2 is back-biased. Thus P3 is held at the same voltage it was before the clock pulse. When the clock pulse falls, P3 is unchanged, but P1 goes from +5.4 V to 1.1 V, a fall of 4.3 V. Since the voltage across C1 can't change instantaneously, the upper terminal of C1, which was at +0.7 V (drop across D5), falls the same amount as P1 to become 0.7 V - 4.3 V = 3.6 V. This, as in the case of the previously-discussed flip-flop, forward biases D7 and cuts off the leftmost of the two latch transistors shown in Figure 5-10 by pulling current out of its base.

Since the drive applied to the bases of QS1 and QR1 when it is desired to saturate them does not vary as it does in the previously-discussed flip-flop, no anti-saturation circuits are needed on these transistors and hence none is needed on the clock amplifier as long as the transistors have close to the same  $V_{ce\ sat}$ .

Inputs to the NAND-flop are made through, as the name implies, a system of NAND gates. The gates are similar to the NAND gate discussed previously, except that the two gates which include QR1 and QS1 do not include the 47 pf capacitor and associated resistor to compensate for Miller-effect capacitance. Miller-effect capacitance only occurs when the collector moves in response to a signal at the base. In this case, the logic usage rules dictate that the input signals be present in time for QR1 and QS1 to assume their final states before the clock

pulse rises. Thus no Miller-effect occurs because the collectors of QR1 and QS1 are held fixed by the clock amplifier whenever their base signals are changing.

To set the flip-flop, we want to saturate the leftmost latch transistor and cut off the right one. Thus we want QS1 cut off and QR1 saturated. To saturate QS1, the "Low True Reset" input must be high (+4 V) and either the "Reset in a" or the "Reset in b" or the "Gate Common" must be false (low). To cut off QS1 either the "Low True Set in a" or the "Low True Set in b" must be false (low) or the "Set in a" and the "Set in b" and "Gate Common" must be true (high). Thus conditions for placing the NAND-flop in a given state are: one or more of the "Low True" inputs for the desired state must

be low, or all of the "Set/Reset" inputs for the desired state and the "Gate Common" must be true; in addition, all of the "Low True" inputs for the opposite state must be High (normal true = 4 V) and at least one of the "Set/Reset" inputs for the opposite state or the gate common must be false. If these conditions are met for at least 10 nsec before the start of the clock pulse and maintained during the clock pulse, the NAND-flop will enter the desired state when the clock pulse falls.

The clock amplifier is a non-inverting amplifier similar to that used in the previously-discussed flip-flop, except that QC has no antisaturation circuit. This is because QS1 and QR1 have none, and it is desirable for all three transistors to have the same  $V_{ce\ sat}$ .

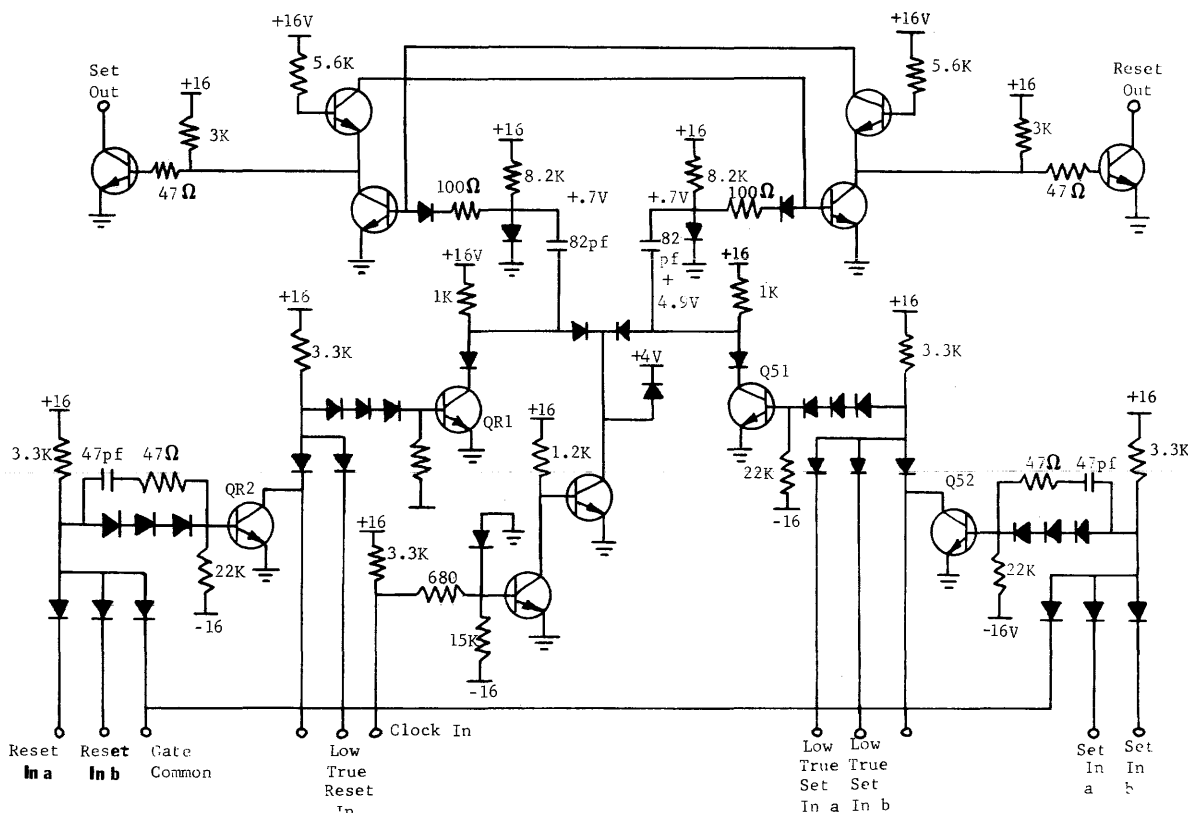


Figure 5-9. NAND Flip-Flop

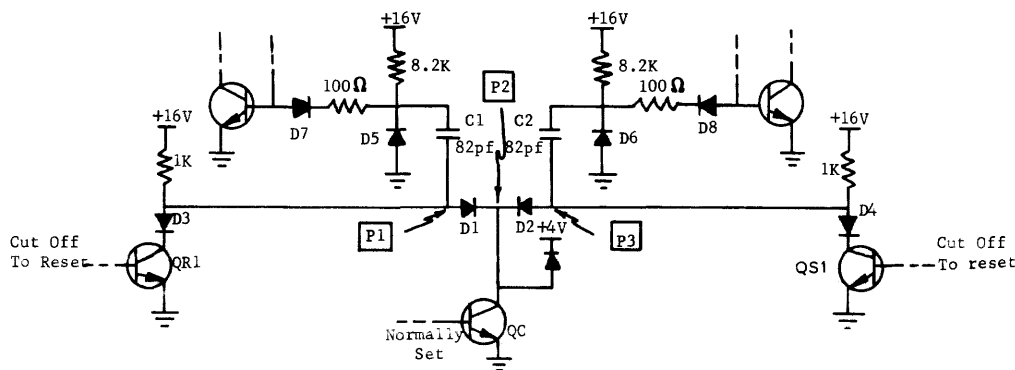


Figure 5-10. Triggering Circuitry



## "Super" NAND Flip-Flop

The circuit of Figure 5-11 is a "super NAND flip-flop" with more flexible input gating than the "NAND flip-flop" previously discussed. The method of setting and resetting the central latch is the same as that used in the "NAND flip-flop".

The central latch, however, contains the addition of D1-D4, as shown in Figure 5-12. These diodes limit the upper positive excursions of the collectors of latch transistors Q1 and Q2, which is necessary because of the way in which Q5 is used. Suppose Q2 is saturated. Then its  $V_{ce}$  is +0.4 V and Q5 is also saturated. But because Q5 has considerable base current but very small collector current, its  $V_{ce}$  is only about +0.05 V.

This places +0.45 V at the base of Q6 which is insufficient to turn it on, since the base threshold is +0.6 V. If we cut off Q2, however, its collector is pulled up to +2.1 V by the 3K resistor R. This cuts off Q5 in the normal mode but forward biases its base-collector diode, feeding Q5's base current to the base of Q6 and saturating the latter transistor. When Q5 is operating in this latter mode, its base is at +1.6 V. To avoid breaking down its base-emitter junction, D1, D2 and D4 are provided to limit Q2's collector (and hence Q5's emitter) to +2.1 V.

The rest of the circuitry in the "super NAND flip-flop" has been discussed previously; the changes in the central latch and output buffers were made only to permit the latch transistors to drive extra outputs without overload.

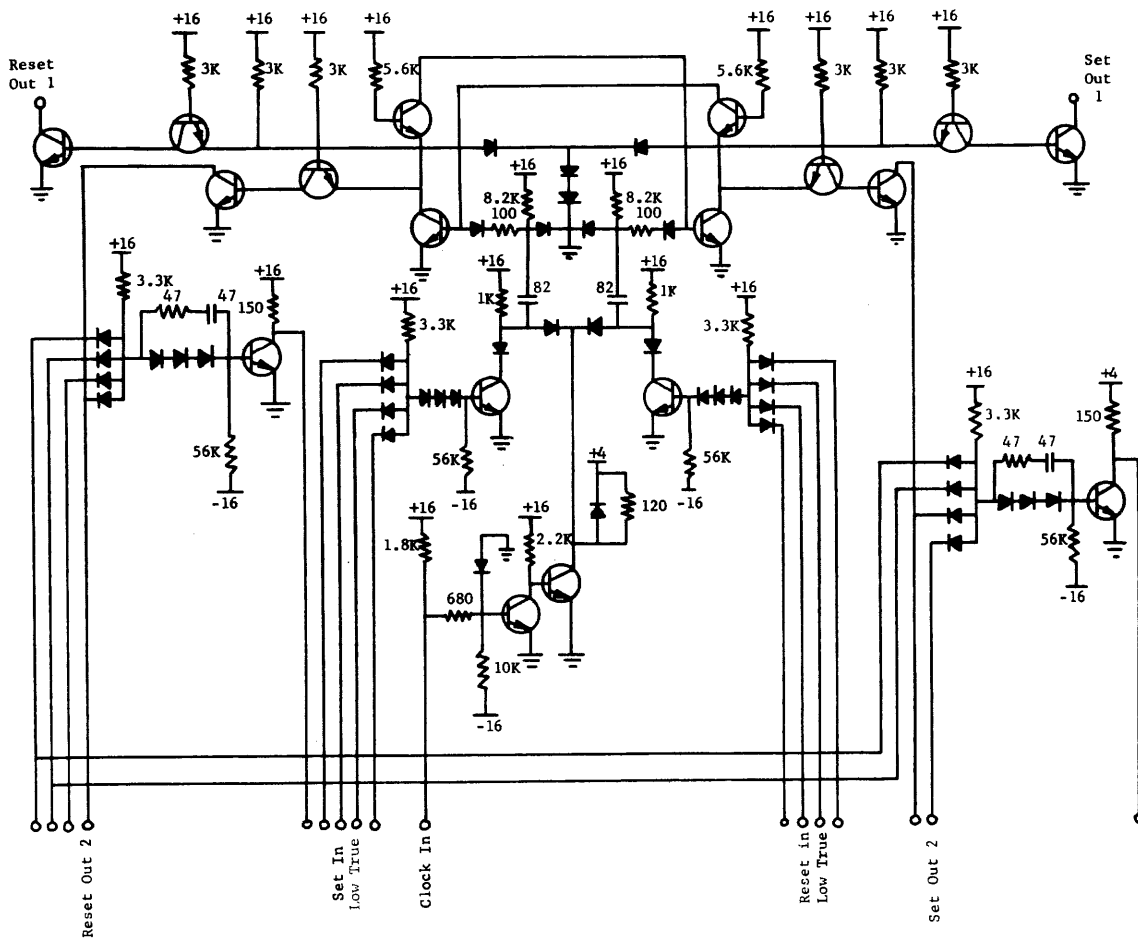


Figure 5-11. Super NAND Flip-Flop

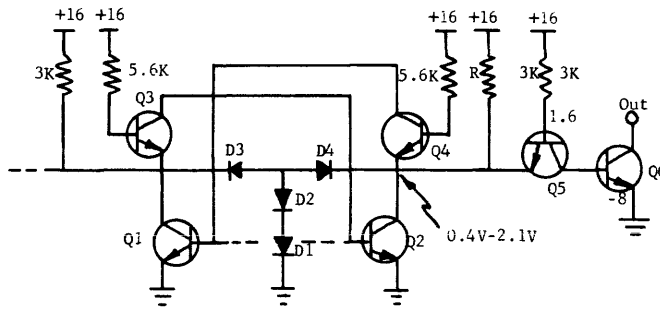


Figure 5-12. Central Latch and Output Buffer

### DC Flip-Flop

Unlike other B-Series flip-flops, the "DC flip-flop" in Figure 5-13 is not clocked. It also uses diode cross-coupling in the latch.

The operation of the diode cross-coupled latch is as follows. Suppose Q1 is saturated. Its collector is then at +0.4V and P2 is at  $0.4 + 0.7 = +1.1V$ . P4 is at  $+1.1 - 0.7 = +0.4V$  and the base of Q2 is at  $+0.4 - 2 \times 0.7 = -1.0V$ . Thus Q2 is cut off, and point P1 is permitted to rise to Q1's  $V_{be\ sat} = 0.8V + 3 \times 0.7 = +2.9V$ , and Q1 is thus kept saturated. The collector of Q2 is actually permitted to rise to  $V_{be\ sat}$  of Q4 plus the drop across the Zener diode VR2 ( $0.8V + 3.3V = 4.1V$ ). Thus the diode from Q2's collector (+4.1V) to P1 (+2.9V) is back-biased and the drive which keeps Q1 saturated comes from the 8.2K resistor from P1 to +16V.

So far, we have assumed that all of the "set" and "reset" inputs are grounded, permitting points P3 and P4 to

assume any positive voltage. If, however, the two inputs of any pair of set inputs (the two inputs of pair 1 are ANDed, as are the inputs of pairs 2 and 3) are both raised to +4V, point P3 and hence the base of Q1 will receive drive from the 3K resistor associated with the activated pair of inputs. Thus Q1 will become saturated even if it was previously cut off. The same reasoning may be applied to resetting the flip-flop. This flip-flop will begin regeneration whenever the logical arrangement of the inputs calls for a change of state.

Zener diodes are used to couple the output buffers to the central latch. Since the cross-coupling diode to the collector of a latch transistor becomes back-biased when the transistor is cut off, the output buffer determines the maximum collector excursion of the latch transistor. Thus the collector of the latch transistor can rise to +4.1V ( $= 0.8V V_{be}$  for the buffer transistor +3.3V for the Zener diode), saturating the output buffer.

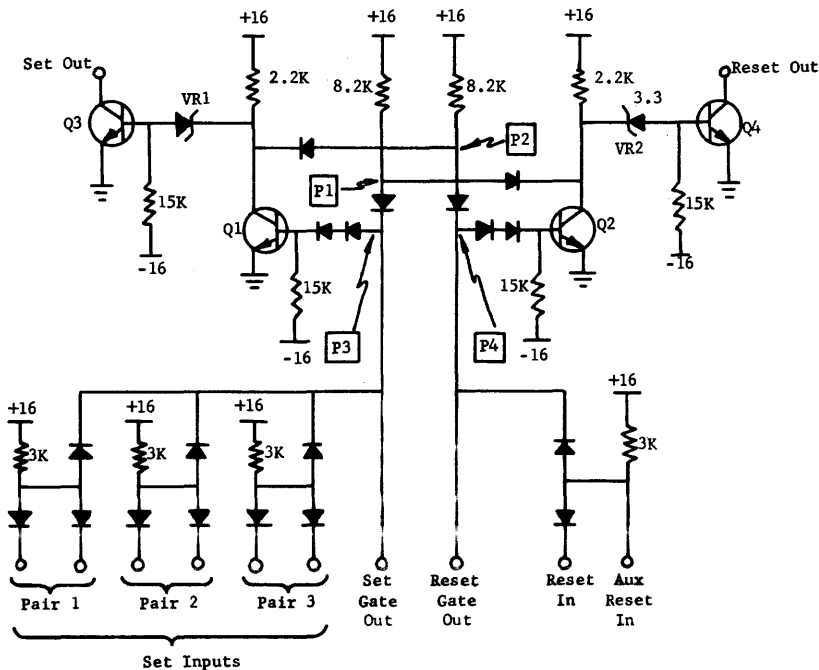


Figure 5-13. DC Flip-Flop

## Repeater Flip-Flop

The function of the repeater flip-flop is to Set and Reset upon the receipt of either +8 VDC or 0 VDC respectively, at the Aux Set In point, so long as there is +8 VDC at the Common Hold In point, which is called an "enable" voltage. If there is 0 VDC at the Common Hold In point, then the repeater will not change state. The Common Clock In furnishes the circuit with a timing clock for synchronous operation with other flip-flops. The clock's negative-going transition causes the repeater to change state according to its input logic. In the DC Set In mode, a negative transition at the DC Set In point will cause the repeater to assume the Set state, regardless of the condition of the Common Hold In point and the Common Clock In point. Thus, the DC Set In point provides a means of setting the repeater asynchronously. The repeater is also provided with a Register In entry, which allows the repeater to be wired to the output of an external register, normally a dynamic register. Under this modus operandi, the repeater will copy the contents of the output stage of the external register, regardless of the condition of the Common Hold In point. It is, however, dependent upon the receipt of the clock at the Common Clock In point, and as such, operates in the synchronous mode.

### Description

Assume for the moment that the repeater is in the Reset state. In this state, the transistors will assume the following conditions:

Q1, Q2, and Q5 Off      Q3 and Q4 On

Q1 is the trigger stage for the reset input; Q5 is the trigger stage for the set input.

### Set Operation

Assuming that the Common Hold In point, pin 29, is "enabled" by +8 VDC, diodes CR7 and CR12 will be reverse-biased. With 0VDC at the Aux Set In point, pin 30, the junction of R18 and VR2 will be at  $\approx -3$ VDC, keeping the base of Q5 negative in respect to its emitter (reverse-biased), so Q5 will be off. When the Aux Set In point, however, rises to +8 VDC, the bias point at the junction of R18 and VR2 will rise positive enough to forward bias the base of Q5, and turn Q5 on. With Q5 on, its collector falls towards 0 VDC (ground), and current in the collector circuit causes pin 1 of T2 to go negative in respect to pin 2. By transformer action, pin 4 of T2 goes positive in respect to pin 3, which in turn reverse-biases CR14 and CR15, thus causing no further change in the repeater. With Q5 on, it is said to be "primed". With its collector at 0 VDC, CR 6 in the collector circuit is forward-biased, causing the potential at the cathode of VR1 in the base circuit of Q1 to be 0 VDC, keeping the base of Q1 reverse-biased and thus keeping Q1 off.

The system clock appears at the **Common Clock In points**, pins 43 and 42. The secondary of the input transformer has one leg tied to ground, and the other leg tied to the cathodes of CR23 and CR24. When the transformer-coupled clock goes positive at the secondary, the diodes CR23 and CR24 are reverse-biased, and therefore, the positive swing of the clock has no effect upon the circuit. When the clock swings in the negative direction, however, the diodes CR23 and CR24 will be forward-biased, and will couple the negative-going transition to the cathode of CR22 and to the base of Q5. Since Q1 is in the off condition, the clock will have little effect upon the base bias of Q1. Q5, however, is in the on condition, and the negative swing of the clock will pull down the potential of the base of Q5 negative enough to cause Q5 to turn off. When Q5 turns off, its collector rises towards +8 VDC, causing the current in T2 to change direction momentarily, reversing the field, and, by transformer action, will cause pin 4 of T2 to swing in the negative direction. This negative transition will forward-bias CR14 and CR15, causing the bases of Q4 and Q3 to go negative in respect to their emitters, turning both transistors off. When Q4 turns off, its collector rises toward +8 VDC, which reverse-biases CR11, allowing the base of Q2 to go positive through the voltage divider network R1, R11, and R15, turning Q2 on. After the clock has left, Q5 will turn back on if the Aux Set In point is again at +8 VDC. With Q3 and Q4 off, their collectors, and the Set outputs, will be at +8 VDC. With Q2 on, its collector, and the Reset outputs, will be at 0 VDC. The repeater is now in the Set state.

### Reset Operation

Let us assume for the moment that the repeater is in the Set state with the Common Hold In point at +8 VDC ("enabled"), and the Aux Set In point at 0 VDC (no set input). In this condition, the base of Q5 will be at a negative potential in respect to the grounded emitter, and Q5 therefore will not be "primed". Q5, being off, puts its collector potential at +8 VDC, which reverse-biases CR6. CR7 is also reverse-biased by the "enable". Therefore a voltage divider network of R4, VR1, and R14 will cause the base of Q1 to become positive in respect to its emitter, turning Q1 on. Q1 is now "primed". When Q1 turns on, its collector goes towards 0 VDC, causing current in its collector circuit to put pin 1 of T1 negative in respect to pin 2. Through transformer action, pin 4 of T1 becomes more positive than pin 3, and CR13 is thus reverse-biased, causing no further change in the circuit. When the clock appears at the Common Clock In point, it forward-biases CR23, CR22, and CR19. This causes the potential at the cathode of Zener diode, VR1 to drop, making the base of Q1 negative in respect to its emitter, turning Q1 off. When Q1 turns off, its collector rises towards +8 VDC, causing the field of T1 to change in the same manner as T2 did. Pin 4 of T1 will then go negative in respect to pin 3, and CR13 will be forward-biased, which, in turn,

causes Q2 to turn off. When Q2 turns off, its collector rises towards +8 VDC which reverse-biases CR8, allowing the base potential of Q4 to rise, turning Q4 on. At the same time, CR4 in the collector circuit of Q2 is reverse-biased, allowing the base of Q3 to rise, turning Q3 on. With the collector of Q4 near 0 VDC, CR11 is forward-biased, keeping the base of Q2 reverse-biased. The repeater is now in the Reset state.

### DC Set Input Operation

The DC Set In point, pin 31, if used, will be normally at +8 VDC, thus keeping diodes CR9 and CR10 reverse-biased. This allows the repeater to operate in the normal manner. When, however, the DC Set In point goes negative, it forward-biases CR9 and CR10. CR9 will couple the negative transition to the base of Q4 through C6 and R13, causing Q4 to turn off. CR10 will couple the negative transition to the base of Q3 via R11 and C4, turning Q3 off. With Q4 off, its collector will go towards 0 VDC, forward-biasing CR11

and causing the base of Q2 to go negative in respect to its emitter, turning Q2 off. The repeater is now in the Set state. Notice that the Common Hold In point and the Common Clock In point had no effect upon the DC Set In operation.

### Register Input Operation

If the Register In operation option is chosen, the Register In point, pin 23, will be wired to an external register. The condition of the output stage of the external register will determine whether the repeater will Set or Reset. Pin 23 is connected directly to the base of Q5. Therefore, if the external stage is at +8 VDC, it will "prime" Q5 (turn it on). When the clock appears, Q5 will turn off, causing the same chain of events as the normal Aux Set In operation did. The same is true for the Reset condition. Notice, however, that in the Register In operation, the circuit did not rely upon the condition of the Common Hold In.

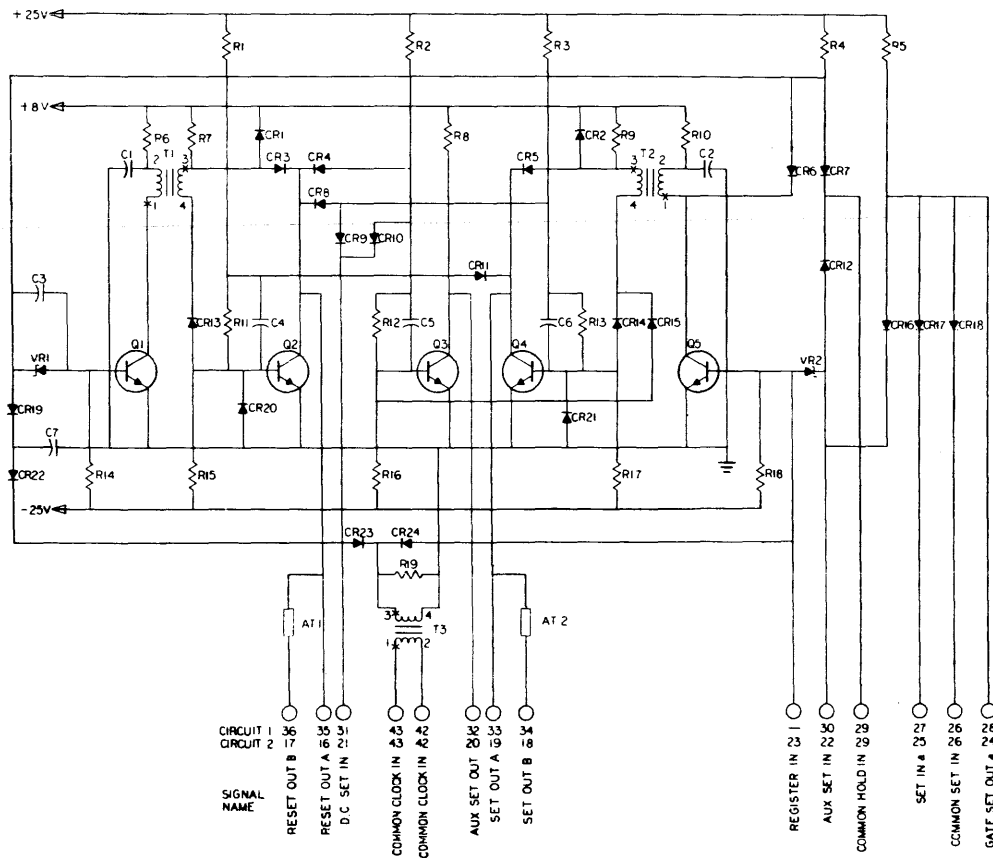


Figure 5-14. Repeater Flip-Flop



# SECTION 6. HOW A COMPUTER WORKS

## INTRODUCTION

A digital computer is basically a high speed device designed to manipulate numbers. The computer has built into it all the capabilities desired by its producer. It is designed to have various means of placing data within the machine and giving information via a variety of output equipment.

The configuration of equipment using the information input and output equipment with a digital computer is called a data processing system. This system is a useless conglomeration of hardware unless a man tells "the system" what it is to do. To "tell" the system what to do, the man writes a program. This program is then put into the computer and when the system is started correctly this system will "operate under program control". The system in running will do just those operations the programmer has specified, in the sequence he has specified, using the data he has specified, and giving answers he has specified. If for any reason, the programmer has made a mistake, the system will make that same mistake. Therefore, it is essential that the programmer know exactly what is to be done, and that he makes no error in telling the system, by the program, exactly what is to be accomplished.

It is, therefore, important that the reader, as a maintenance man for data processing systems, know the internal functionings of the computer and how it handles the program written for it. This discussion will limit itself to what happens inside a computer rather than how it happens. How it happens will be a matter for later learning.

Continual reference will be made to information as decimal digits and letters of the alphabet. It should be remembered that inside the machine these are stored and manipulated as binary codes; however, the convention adopted here causes no loss of generality.

## ARITHMETIC-LOGICAL UNIT

The arithmetic-logical unit (ALU) and the control unit usually are located physically in the same cabinet. Frequently the ALU is broadened to include the control unit also, but this seldom causes any confusion. The cabinet is large. This is understandable because it contains the fast-access storage device, and the registers for holding the data being operated upon and for controlling operations. It also houses all the associated electronic hardware necessary to accomplish

arithmetic and logical operations and to control the transfer of information among the various units--input, output, and storage. This latter is called "switching", which is rather an obvious name for a function primarily concerned with setting up the electronic communication lines to transfer information between any two prescribed points. Everything discussed in this section occurs in the ALU and the control units.

## LOCATIONS AND ADDRESSES

Because different equipments vary in specifications of their memories this section, for purposes of simplicity, assumes specifications which are completely representative and will serve to provide an understanding of what happens inside the machine. This fictitious computer has a magnetic storage, with information stored in the form of "words" consisting of five characters each. There is a capacity of 2,048 words. The words all have a specific location in magnetic storage, and every location has an address, ranging from 0000 to 2047. Convention, based on requirements of the earlier computers, dictates that the first location have the address 0000, not 0001, and this practice is carried over into present day computer terminology. Thus the address of the last location is always one less than the total number of words that can be stored.

## PICTORIAL REPRESENTATION OF MEMORY

The 2048 words of memory can be represented pictorially as many lots on a long street. Each lot contains five characters, and each has an address of its own; the first lot is the street (memory) address 0000, the second is 0001 and so on up to the last, which is 2047. Now assume that two of the lots have information or data in them; the lot with the address 1000 has the number 25640 and the next lot address 1001, has the number 14628. The rest of the lots are empty; that is, there is no information in them. A special character is assigned as a "blank", or contains all 0's (00000). After placing the two numbers in the memory locations with addresses 1000 and 1001, memory can be represented as:

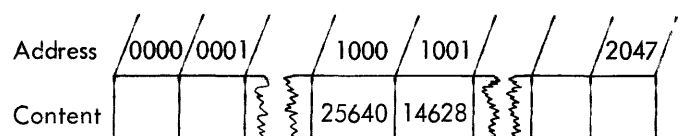


Figure 6-1. Two Numbers in Memory

It should be remembered that the numbers in memory are stored as binary code characters, not as decimal characters, but no generality is lost in picturing them and talking about them as ordinary decimal numbers.

For purposes of illustration in this section, it will be assumed that any information required is available in memory, without considering how it got there. In the next section, memory will be explained.

### LOCATION AND ADDRESS TERMINOLOGY

It is a bit cumbersome to continue referring to the location in memory as "the location whose address is 1000", or some other address. Consequently, the convention will be adopted that address 1000 and location 1000 will mean the same thing i.e., the location in memory with address 1000. Therefore, in the example being used, the number 25640 is located or stored at address 1000 and 14628 at address 1001.

### INSTRUCTIONS AND PROGRAMS

The operation to be performed by the computer is to add the two numbers stored at addresses 1000 and 1001 and put the sum into address 1002. So far, the machine has the two numbers, but no way of knowing what it is supposed to do. Obviously, some "instructions" are required. It would be well to find out just what "instructions" are and what "instructions" look like. An order telling the computer what operation to perform is called an instruction. A series of instructions to be followed by the machine in performing a sequence of operations is called the program. An instruction, for example, may tell the machine to ADD. This tells it what it is supposed to do, but it does not tell it what number it is to add. The number to be added must, of course, be in memory and the instruction must include information telling the equipment just what number is to be added. Because every location in memory has an address which the machine can find immediately, inclusion of the address as part of the instruction will give it the necessary information to find the number to be added. An instruction, then, must consist of two parts: an operation part, telling the computer what to do, and an address part, telling it where to get the information to be operated on.

With four digits required to identify every one of the 2048 locations in memory, a complete instruction can be put into one 5-character word provided its operation part consists of only one character since one character has a possibility of being any one of 64 combinations of binary digits. That many different operations could be provided and still limit the length of the instruction to one word of 5 characters. In practice, this is not ample; some computers have almost that many different operations, but others operate with only 16 or 32.

It is important to note that the instruction does not include the number to be added; it only tells the machine where to go in memory to find the number.

### Format of an Instruction

The instruction, then, consists of a one-character operation part and a four-character address part. The computer interprets the operation part and sets up the circuits necessary to accomplish that particular operation. The address part tells it where to get the information. The operation of adding to be performed in the example obviously requires an ADD instruction. The character "A" will be assigned for this purpose. The equipment will recognize this character "A" as an order to add. Therefore, if a number located at address 1500 is to be added, the instruction in words would read ADD 1500, but in machine language it would be abbreviated to A1500, exactly five characters for one word.

### Location of Instructions

It was noted that the instructions as well as data are stored in memory. Consequently, before the addition of the example can be accomplished, it will be necessary to have in memory the necessary instructions for performing this operation. Because memory contains some instructions and information to be operated on, it is well to keep them separate by assigning blocks of addresses in memory for the exclusive purpose of storing instructions and other blocks for storing data. In this example, instructions will be stored in memory at the beginning of address 0000.

### Interpretation of the Address Part

The instruction "A1500" was assigned as that which would tell the computer to add the number located at address 1500 to something. It is important to note that this instruction does not mean to add the number 1500 to another number. The distinction between address in memory, which is simply a location and the content of that location must be borne in mind at all times. The reason for the necessity of this distinction will become apparent in later paragraphs.

### HOW A COMPUTER ADDS

Returning to the addition of the two numbers, 25640 at address 1000 and 14628 at address 1001, it is apparent that instructions are necessary and must be placed in memory to perform this operation. Operations are not performed in memory, but in a special type storage device called a register. The register in this equipment has a capacity of ten characters or two words, and following standard practices it will be referred to as an accumulator. The methods of addition will be to place the number located at address 1000 into the accumulator,

and add to it the other number, putting the sum of the two into the accumulator. This sum will then be stored at address 1002.

### Addition Instructions

It is now necessary to determine what instructions are required to accomplish this addition. The first one must place the number 25640, from address 1000 in the accumulator. Is the ADD instruction sufficient for this purpose? Not by the method of operation decided above. There is a chance that the data from a previous operation may be in the accumulator and the ADD instruction would actually add 25640 to whatever was present before. The accumulator must first be set to all zeros, and because addition is a frequent operation, it will be convenient to have an instruction which will first clear the accumulator (i.e., set it so it contains all zeros), and then place in it the number 25640. This instruction will be called RESET ADD and will be denoted by the letter code "R". Therefore, the first instruction is RESET ADD 1000 in code R1000, which means: set the accumulator to 0 and then add into it the number located at address 1000.

Once the number 25640 is in the accumulator, the ADD instruction will accomplish the following: take the number found at the location specified by the address, add it to the number in the accumulator, and place the sum of the two in the accumulator, removing the

original number there. Therefore, ADD 1001, in code A1001, will take the number 14628 found in address 1001 and add it to 25640 in the accumulator, and place the sum 40268 in the accumulator, wiping out the 25640 that was there.

It is now necessary to store the number 40268 in location 1002. Another operation is involved, which will be called STORE the character, code "S". The instruction STORE 1002, in code S1002, will instruct the machine to take the number in the accumulator and store it in the location with address 1002 wiping out any information in that location. The three instructions, then, will accomplish the addition desired. It is only necessary to place the instruction in memory and determine the method of sequence of operation in the equipment.

### Sequence of Performing Instructions

It appears that a simple way of having a computer perform a sequence of operations is to have it start with the instruction at the first address, do whatever the instruction says, take the instruction of the next address, and so on. This turns out to be very convenient. The computer of this example is designed to perform operations in just that manner; therefore, the three instructions necessary to accomplish the addition are placed in memory beginning with the location at address 0000. Before beginning the addition operation, memory looks like this:

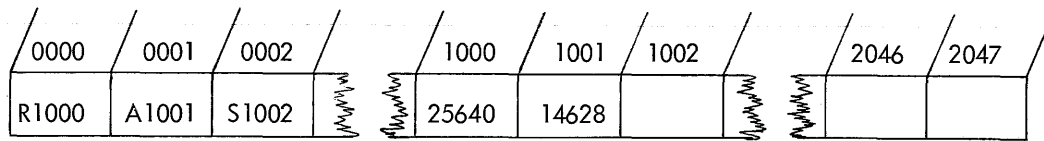


Figure 6-2. Memory Before Addition Operation

### Detail of the Addition Operation

It will be helpful in obtaining an understanding of machine operation to repeat the addition example in detail, depicting exactly what happened in carrying out each instruction. It is assumed that before beginning the sequence of instructions, the accumulator contains the number 182359. It makes no difference what number, if any, is actually there, because the

instruction will make sure that it is cleared before the addition begins. Similarly, though memory is assumed to be empty except for the three instructions, and the two numbers to be added, it makes no difference if there are other data locations not used.

Before executing the first instruction, memory and the accumulator look like this. The arrow indicates that the first instruction is the next to be carried out:

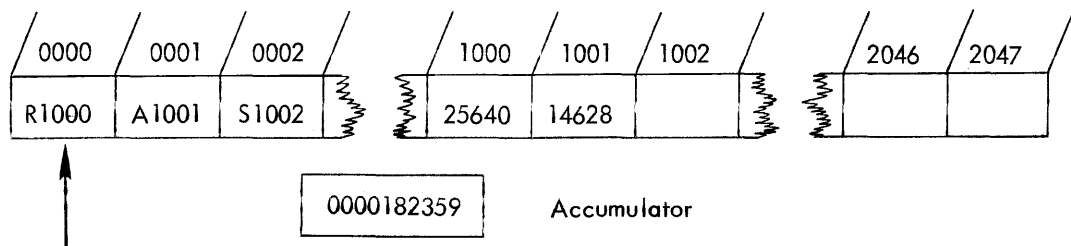


Figure 6-3. Memory Before Executing First Instruction



After executing the first instruction RESET ADD 1000, the memory looks like Figure 6-4. (The arrow indicates

that the second instruction is the next to be performed.)

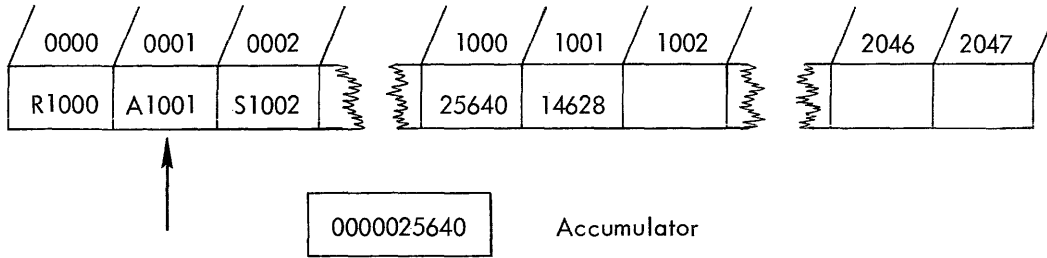


Figure 6-4. Memory After First Instruction

Notice that the RESET ADD 1000 instruction erased the number 182359 that was in the accumulator. Also notice that memory address 1000 still contains the number 25640. Although transferred to the accumulator, it still remains available in memory for further use if

required. After execution of the second instruction ADD 1001, memory and the accumulator look like Figure 6-5. (The arrow indicates that the third instruction is the next to be performed.)

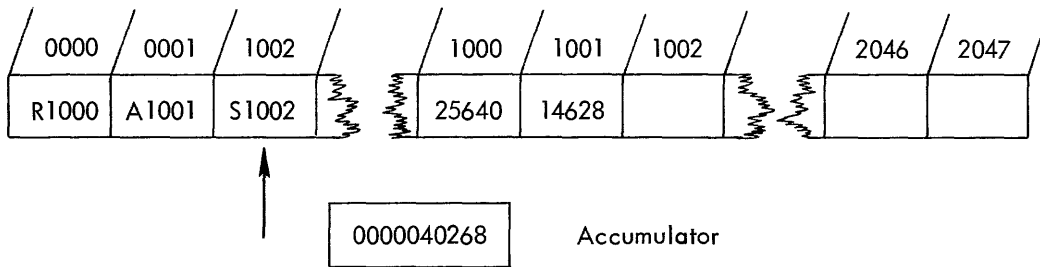


Figure 6-5. Memory After Second Instruction

Note that the accumulator now contains the sum of the two numbers which has replaced the 25640 that was there.

Finally, after execution of the last instruction STORE 1002, memory and accumulator look like Figure 6-6.

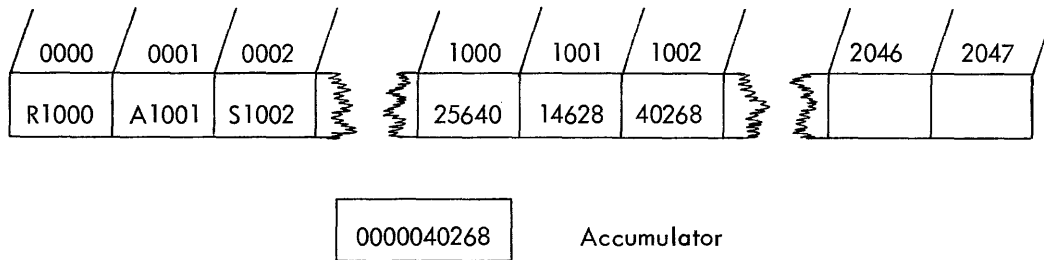


Figure 6-6. Memory After Last Instruction

Notice that the STORE instruction placed the contents of the accumulator in memory location 1002, but left the number in the accumulator where it is available for other operations.

### Repeated Operations

In practice it is seldom that only two members are added together for a single sum. An adding machine or pencil and paper would be faster and more economical than using a computer. Suppose that instead of adding a

single pair of numbers, the problem called for adding 500 pairs and storing 500 different sums. The first 500 numbers are stored in memory, beginning at location 0500 (and of course ending at 0999): The next 500 are stored in location 1000 through 1499. Answers are to be put in locations 1500 through 1999. The problem calls for adding the numbers in location 0500 to the number in location 1000 and storing the sum in 1500, then the number in 0501 to the one in 1001 and storing the sum in 1501, etc. Obviously the way to do this is to use the three instructions already developed. See

Figure 6-7.

RESET ADD	0500	R0500
ADD	1000	A1000
STORE	1500	S1500
RESET ADD	0501	R0501
ADD	1001	A1001
STORE	1501	S1501
==	=	==
RESET ADD	0999	R0999
ADD	1499	A1499
STORE	1999	S1999

Figure 6-7. Addition Program

Not only would the person writing instructions soon become finger weary, but 1500 separate instructions are required. Since the original data and the answers require 1500 words of storage in memory, a total of 3000 words is beyond the 2048 words of capacity available. Therefore, to satisfy the joint requirements of cutting down the number of instructions for the operation and for minimizing the amount of storage space needed to carry the program in memory, a more practical solution is needed.

A brief study of the instructions revealed that a regular cycle of RESET ADD, ADD, and STORE is formed. The only difference is in the addresses. The addresses, it will be noted, increase in a regular pattern. The address of each instruction being one more than the address of the last similar instruction; RESET ADD 1501 follows RESET ADD 0500, RESET ADD 0502 follows RESET ADD 0501. If it would be possible to go through the first three instructions then add 1 to each of the three addresses and repeat the program; add 1 again to each of the three instructions and repeat the program; etc., it appears that a large number of instructions (1500 in this case) could be reduced to a very few. Can this be done? Instructions are not data but is it possible to operate on them in the same manner as on data? Before this question is answered it will be instructive to consider in detail exactly what happens in the machine during the execution of a single instruction.

#### WHAT HAPPENS IN THE EXECUTION OF AN INSTRUCTION

For the sake of illustration, memory contains three instructions used in the addition example:

Memory Location	0000	R1000
	0001	A1001
	0002	S1002

Obviously, the first order of business for the machine is to find the instruction in location 0000; it can do nothing until it has the instruction available. More generally, after completing the operation called for by the instruction the machine must find the next one.

It needs some means of keeping track of where it is in performing a long series of operations. Because memory is only a place to store instructions and data and does not enter into any computations it appears that something else must be provided to enable the equipment to know what instruction it is to perform next. Likewise, it appears that the equipment needs some place to put an instruction temporarily while it examines the operation part to determine what is to be done and the address part to determine where the data is to come from or where it is to be stored. The function of keeping track of what instruction is to be executed is done by a special register called the Address Counter. The temporary holding place for the instruction is the Instruction Register. Both are part of the control unit.

Suppose now that the Address Counter is initially set at 0000 and the addition operation commences. The first thing the equipment does is to inspect the number located in the Address Register -- 0000. It goes to this address and transfers the contents -- the first instruction -- to the Instruction Register. At the same time, a "1" is added to the contents of the Address Counter so it now contains 0001. The transfer of the instruction to the Instruction Register does not remove it from memory; it still remains available for use later just as other data.

The Instruction Register may be considered as consisting of two parts. Into one part the operation code is placed. Into the other, the address. The equipment inspects the operation code and interprets it as an order to accomplish a specific function (in this case to RESET ADD) and sets up the electronic circuits necessary to do this operation. When the circuits are properly established it looks at the address part of the Instruction Register and determines what location is involved and performs the RESET ADD operation on the number in that location. The operation completed, the equipment turns to the Address Counter where it now finds 0001. The operational cycle is repeated in the same manner as the first instruction and when the next instruction is to be performed, the Address Register reads 0002. The complete carrying out of one instruction may then be considered as involving the following separate steps which are also shown in Figure 6-8.

- a. Determine location of instruction
- b. Obtain instruction; reset address counter
- c. Interpret instruction and set up necessary circuits
- d. Execute the instruction

The first three steps (a, b, c) are often combined into the interpretation part of the cycle. The last step is called the execution part. Regardless of the nomenclature and type of computer, the instruction cycle can be considered as consisting of the four steps described. A complete understanding of what occurs in carrying out an instruction will make clear the general operation within a digital computer.

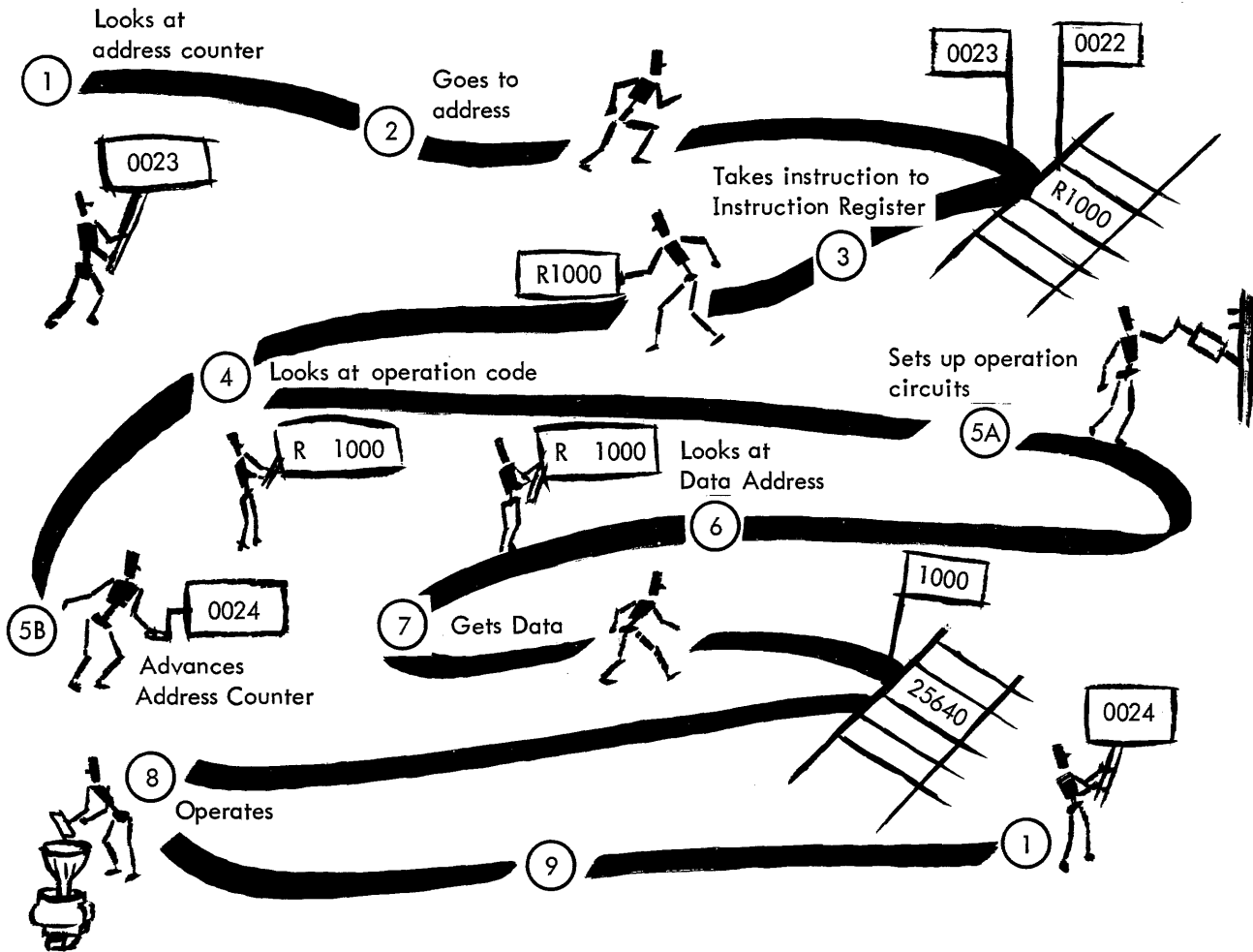


Figure 6-8. How A Computer Executes An Instruction

REPEATED OPERATIONS

With an understanding of what happens in the instruction cycle, the method of changing the addresses of instructions to permit condensing the length of the program required to perform repeated sequences of operation can be understood. Instructions that will be remembered are stored in memory in exactly the same manner as data. In the examples so far, all data have been straight numbers and the three operation codes have been letters, but it is evident that data can include alphabetical information and likewise that there is nothing to prevent some operation code from being numbers. If the entire contents of memory could be spread out for inspection, instructions and data could not be differentiated from one another. The machine treats each impartially; how it considers each depends entirely on what part of the instruction cycle it is in. If by some chance data is stored in the location of an instruction, as soon as the address counter reaches the address of that location, the equipment will transfer the content -- the data -- to the instruction register, treat the first digit as a operation code and the last four as address and proceed to

execute that instruction. Similarly, if the address part of an instruction is the location of another instruction (or the same one, for that matter), the equipment will take the contents of the specified location -- the instruction -- and perform the designated operation on it. The first condition of course is to be avoided and presents a programming error, the second is of considerable use in programming repeated sequences of operations.

Loops

The method of programming a repeated sequence of instruction can now be developed. The data consists of 500 numbers in locations 0500- 0999. 500 more numbers in 1000-1499, and the pairs of numbers in locations 500 apart are to be added with the resulting sum stored in sequential locations 1500-1999. From previous work it is known that the following instructions will add the first pair and store the resulting sum:

RESET ADD	0500	R0500
ADD	1000	A1000
STORE	1500	S1500

The next step is to take the first instruction and increase the address by "1". This can be accomplished by placing the first instruction in the accumulator, adding "1" to it, and storing the result back in the location of the first instruction. The same thing is to be done with the second and third instructions. With the three instructions used in the initial addition example, the location of storage was not incorporated with the program. Several more instructions are now to be added. It will be convenient if the listing shows where the instructions are to be placed in memory. The first three instructions are:

```

0000  RESET ADD  0500  R0500
0001  ADD       1000  A1000
0002  STORE    1500  S1500

```

Because the address counter increases by one with each instruction, the next goes in memory location 0003. This instruction is to place RESET ADD 0500 (in code R0500) into the accumulator. After the STORE instruction is executed the sum of the first addition is still there. However, RESET ADD will clear the accumulator and place into it the contents of the address placed into it. The address of R0500 is 0000. Therefore, the next instruction is:

```

0003  RESET ADD  0000  R0000

```

A "1" is to be added to this, but so far this digit is not available. Therefore, a "1" will be assumed to be in memory in some unused location. There is room starting with 2001. So in 2001 the quantity 00001 is stored. The next instruction is:

```

0004  ADD       2001  A2001

```

After completing this operation the accumulator contents are R0501. This is exactly what is desired and it is stored back in location 0000:

```

0005  STORE    0000  S0000

```

A similar set of instructions will change ADD 1000 to ADD 1001 and another set of three STORE 1500 to STORE 1501.

It is probably obvious by now that the notations as to the contents of the accumulator will be quite helpful. Therefore, in writing the complete sequence of instructions, so far completed, this will be included and a systematic format for writing instructions adopted. Two other new conventions will also be introduced. Usually in writing instructions the exact figure or contents of the address locations are not known. All that is specified is that the contents of a certain address will be operated on in accordance with the instruction. A short form for "contents of address ---" will be helpful. This will be denoted by (m), where "m" is the address. The second convention concerns a marking for addresses

which will change during the course of a problem and are keyed by enclosing the starting address in squared brackets. In this example the first RESET ADD address 0500 (because it is going to change during the course of the operation) the instruction will be written RESET ADD [0500]. The brackets are an aid to the programmer in being certain that all variable addresses are modified during the course of the sequence of operations. Consequently, the instructions necessary to accomplish the first addition and modify the addresses for subsequent operations are shown in Figure 6-9.

Instruction Location	Operation Word	Code	Address	Accumulator
[0000]	RESET ADD	R	0500	(0500)
[0001]	ADD	A	1500	(0500) + (1000)
[0002]	STORE	S	1500	(0500) + (1000)
0003	RESET ADD	R	0000	R0500
0004	ADD	A	2001	R0501
0005	STORE	S	0000	R0501
0006	RESET ADD	R	0001	A1000
0007	ADD	A	2001	A1001
0008	STORE	S	0001	A1001
0009	RESET ADD	R	0002	S1500
0010	ADD	A	2001	S1501
0011	STORE	S	0002	S1501
2001		O	0001	

Figure 6-9. Program of Addition, Storage and Modification of Address

The 12 instructions make the initial addition and storage of the sum and modify the address of the first three instructions so that at the end of the steps the first three instructions are in coded characters, R0501, A1001, and S1501. This is the exact requirement for adding the next pair of figures and storing the result. All that remains now is to instruct the machine to return to the beginning -- that is to address 0000 -- for its next instruction.

After completing the last instruction, the one in location 0011, the address counter contains 0012. If in this memory location an instruction can be placed, which will tell the machine to return to the location 0000 for its next instruction, instead of the normal progression to location 0013, the objective will be accomplished. This instruction is called TRANSFER and it means: do nothing with this instruction, but go to location shown in the address for the next one. The address part of the TRANSFER instruction is always the location in which the machine will find its next instruction. It constitutes a means of breaking the normal progression or sequence of instruction execution. In effect, all that happens during the execution phase of instruction is the resetting of the address counter to the specified address -- in this case 0000 -- wiping out the sequential address normally there. Therefore, the instruction at

0012 is:

0012 TRANSFER T 0000 Transfers back to 0000.

This accomplishes the objective desired; the 1500 instructions normally required have been reduced to 13. Of course each addition requires 13 steps instead of three or four times as much time to complete the addition. However, arithmetic operations are performed at high rates of speed in the ALU and it is almost always more effective overall to perform the address modification than it is to take most of the space in memory for instructions and thereby be forced to move in small blocks of data several times to complete the operation.

A program of this kind in which a sequence of operations is performed on one set of data and then the program transferred back to the beginning for performance of another set and so on is called a loop. There is only one thing missing in the loop developed above. The machine would never stop repeating it. There are only 500 numbered pairs to be added, but the computer doesn't know that. After completing the last addition required (the addresses are 0999, 1499, and 1999) it would modify the addresses to 1000, 1500, and 2000 and continue adding. It will not continue forever, sooner or later an address in an instruction will exceed the 2047 or the program may replace a good instruction with gibberish which will have as its first character an invalid operation code either of which will cause the equipment to stop. Obviously this is an undesirable state of affairs, because good data which may be required for subsequent operations may be replaced by "garbage". Being "caught in a loop" is to be avoided.

The way out is to insert a test in the program by means of which the computer can determine when it has completed the required number of cycles around the loop. This test condition is easy to develop. It is known in this example that when STORE 1999 has been executed, the problem is completed. The equipment however, will continue operating on instructions and performing modifications of instructions specified by the instructions in 0003 through 0011. After completing the instruction located in 0011, the accumulator contains STORE 2000, in code S2000, and at this point it is known that the operation is completed. (Note that it is not complete when the accumulator contains S1999 after instruction 0011 -- the last pair of figures is still to be added). Now suppose that after this instruction the contents of the accumulator is compared with a constant S2000 stored somewhere in memory just for this purpose. As long as the contents of the accumulator do not equal this constant, the problem is to continue. As soon as equality is reached the machine is to be stopped. This requires two new instructions, which are defined as

follows:

- (1) The instruction COMPARE m in code "Cm" means: compare the contents of the accumulator with the contents of memory location "m" then go to the next instruction to find out what to do.
- (2) The instruction TRANSFER IF EQUAL means: as the result of the previous comparison if the accumulator contents are equal to the specified memory location contents go to the location "m" which is the address part of this instruction, for the next instruction; if the contents of the accumulator and memory are not equal continue in normal sequence for the next instruction. In the example the instructions are inserted after 0011, the TRANSFER instruction being renumbered; S2000 being stored in memory location 2002. (See Figure 6-10.)

0012	COMPARE	C	2002	Compares contents of Accumulator with S2000, C (2002)
0013	TRANSFER IF EQUAL	E	0015	Loop exit; equal after last number pair is added.
0014	TRANSFER	T	0000	Loop repeats as long as accumulator is not equal to S2000.
0015	MACHINE STOP			

Figure 6-10. COMPARE and TRANSFER IF EQUAL Instructions

Thus 15 steps are repeated in sequence by the computer for every pair of additions. The last step is reached only when the problem is completed. It is not necessary, of course, that this be a machine stop, it could just as well be the beginning of another phase of the overall problem to be solved, the additions being just one portion.

Loops are common in computer programming techniques and all must meet two requirements:

- a. The addresses of instructions must be correctly modified, and
- b. An exit in the form of a test, must be provided to stop the operation when the problem is complete.

This program is typical of a loop and with the instructions used it is as short as possible. Because of the commonness of the loop instructions, computers usually have one or two instructions specifically designed for the modifications of addresses. Use of the instructions would permit this loop to be shortened by two instructions.

## MEANINGS OF THE ADDRESS PORTION OF INSTRUCTIONS

It will be observed from the instructions, which were developed for the solution of the multiple addition problem, that the address portion of the instruction may have different meanings depending upon what the operation part of the instruction calls for. For example RESET ADD, ADD, and COMPARE instructions the equipment is instructed to take the contents from a specified memory location and do something with it. In the STORE instruction it is told to put the contents of the accumulator into the specified memory location.

It is thus apparent that the meaning of the address part must be considered in conjunction with the operation part. Because this meaning is not always obvious, instruction manuals explaining the operation codes available for a specific equipment always define very carefully and precisely exactly what happens when an instruction is carried out and what the address part means. In operation, the equipment interprets the address part of an instruction in conjunction with the operation part, i. e., it "knows" that in the TRANSFER instruction the address part is the location where it will find its next instruction, and not a place in memory where it is to obtain data for execution of the TRANSFER instruction. In general, the address part of an instruction can be considered as falling into one of 8 categories:

### Address of Data to be Taken From Memory

The operation part of some instructions require data to be taken from memory and some specified operation accomplished on that data. Such instructions as RESET ADD, ADD and COMPARE have been described. Others are MULTIPLY, DIVIDE, SUBTRACT and RESET SUBTRACT which are arithmetic operations quite similar to those already discussed. Data can also be taken from memory and be put into some other type of storage. For example, in taking information from memory and putting it on the magnetic tape, one instruction would select a tape unit and the next might be an instruction like WRITE m, which means; "take the contents of memory location "m" and put it in the magnetic tape unit just hooked up".

### Address of Data to be Put Into Memory

This is the reverse condition of that above; the STORE instruction previously defined is an example. Instructions of this type tell the machine to take the information from some specified place which depends upon the operation part and store it in the memory location given by the address part. In addition to the STORE instruction others of this type involve taking data from some other storage device such as a magnetic drum or from an input unit and storing the data in designated memory locations.

### Address of Instructions

The address part of the instruction may be the location in memory where the next instruction will be found. The instructions are used to break the normal sequence of executing operations. There are two types.

### Unconditional Transfer Addresses

If the operation part of an instruction is an unconditional transfer, the machine always goes to the memory location shown as the address part for its next instruction. Every computer has one instruction of this type. That is all that is needed. The TRANSFER instruction defined previously is an unconditional one -- the machine has no option, but always breaks the sequence of execution of operation.

### Conditional Transfer Addresses

If the operation part of an instruction is a conditional transfer, the machine may or may not go to the memory location shown as the address part of its next instruction. It goes to this location only if the conditions of the transfer are met, otherwise it takes its next instruction in the normal sequence (i. e., the one in the memory location next following the conditional transfer instruction). The TRANSFER IF EQUAL instruction used in the addition example is typical, if the two words compared are equal, the machine transfers to the address shown for its next instruction. If not equal, it takes the next instruction in sequence and ignores the transfer operation. Every computer has several conditional transfer instructions and examples are "transfer only if one number is larger than another", "transfer if the accumulator contains 0", and others.

### Address Which are Absolute Numbers

Some instructions tell the machine to perform operations which have nothing to do with the memory location or other components, in which case the number shown in the address portion serves a special purpose. As an example, multiplication of money values frequently result in an answer in the accumulator four or five figures to the right of the decimal point. Suppose that there are five digits which are to be cut down to two by dropping the last three digits. This can be accomplished by telling the equipment to shift the contents to the accumulator three places to the right and to discard the figures moved out during the shift out operation. An instruction such as SHIFT 0003 will do this. The operation part instructs the equipment to shift and the address part instructs how many places to shift. Shift operations are usually necessary in multiplication and division operations in a computer to be sure that decimal points for units positions of answers are properly positioned.

### Address as Identification of Input/Output Units

Obviously, if the ALU and the control unit are to use the various input/output units, some means must be provided to identify the units so that the proper electronic circuits can be set up to connect the desired unit to memory. All such units are therefore, assigned address numbers which may be the same as some locations in memory but the operation part of the instruction is the machine signal that memory is not involved. For example a tape unit may be address 0200 which is also a memory location. The operation part of the instruction to connect an input/output unit may be to SELECT. The complete instruction SELECT 0200 tells the equipment to set up communication lines to connect memory with tape unit 0200.

### Address as Identification of Indicator Units

All computers have a number of special indicators which perform various functions. For example, in adding numbers it may have been determined that the maximum size of the answer would never exceed five digits. A larger answer would turn on "overflow indicator" and this indicator could be checked by an

instruction to determine if it was on or not. By the use of a conditional transfer instruction special instructions would be followed by the machine to handle the overflow condition. As an example suppose the "overflow indicator" was assigned 1000. The instruction CHECK 1000 would mean determine the condition of the "overflow indicator". The next instruction would be a conditional transfer such as TRANSFER IF ON to "m"; if the indicator were ON, the equipment would go to location "m" for its next instruction (this might be the first location of a series of special instructions to handle the overflow condition) and if not ON it would proceed to the next instruction in the normal sequence. Similar indicators are provided to signal such factors as end of magnetic tape, end of paper typewriter, no more cards in the card punch, or an error in a printed line, and all can be interrogated automatically by the machine in a manner similar to the example given.

This ends our discussion of the general subject "How a Computer Works". It was not the intention to give any detailed information with regard to any specific computer, but rather to give a general understanding of computers from a very basic point of view.

## SECTION 7. STORAGE SECTIONS

### MAGNETIC CORE MEMORY

The most common high speed storage used in today's digital computers is the magnetic core memory. This type of data storage is based on the use of one core element for each "bit" of information storage capability of core memory.

Before describing the memory operation in detail, a review of the theory of core operation is presented. The kind of core memory most widely used is the coincident current variety, and only this variety will be discussed.

#### Basic Element of a Core Memory

The basic element of a core memory is the core itself. A core is a "doughnut" formed of ferrous material and a bonding agent. The core is extremely small and appears as follows:

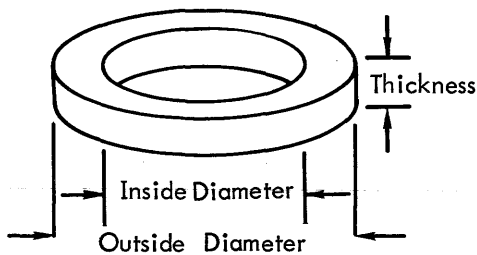


Figure 7-1. Core

The sizes range from an outer diameter of about 1/8 inch to sizes which require magnification to work with them.

A core made primarily of ferrous material is highly magnetic, and tends to hold a magnetized state once it has been subjected to a magnetizing field. A core, just manufactured, that is subjected to a magnetizing force (H) develops a magnetic field (B). Assuming the magnetizing force can be polarized in either direction, we would be able to change the direction of the magnetic field (B) in the core.

If we were to wrap the core with a few turns of wire and pass current through the looped wire in either direction, the magnetic field about the loops of wire would be the magnetizing force exerted on the core. The following diagram shows the result of such an operation.

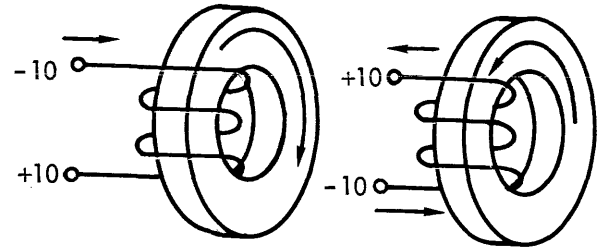


Figure 7-2. Magnetized Core

Notice that as the direction of current changes the direction of (B), the magnetization within the core changes. This can be displayed using graphs of (B) and (H) for a given core material. Assuming that the core was unmagnetized when the process started, the following will explain the graph.

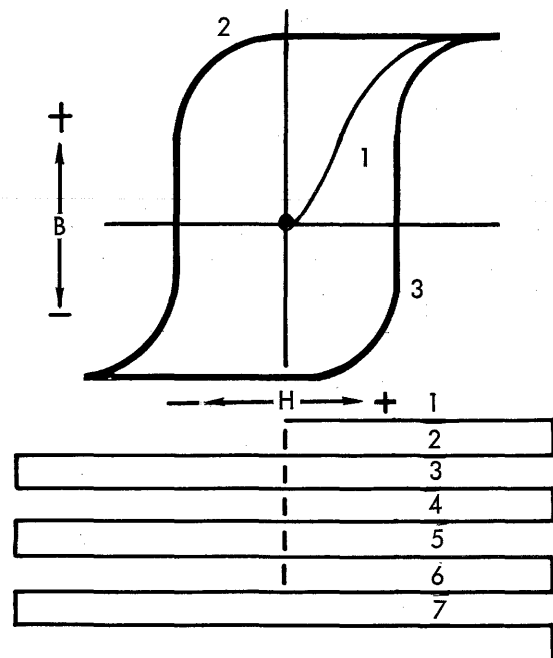


Figure 7-3. Magnetization Curve

#### Magnetic Field of a Core

The core, when starting with no magnetic field (B), will begin from the 0 position, or the crossing of the B - H lines at their 0 points. Once subjected to a magnetizing force, the curve of a core's magnetic field will never again get to that particular point on the graph without having special treatment to get it there.



When current is applied to the magnetizing coil in the direction shown by the current waveforms (path 1), the magnetizing force will force the field within the core along path 1 of the graph. The field within the core will remain at, or near, the saturated level shown until the magnetizing force drive current is reversed. We call this saturation because no matter how much more we increase the magnetizing force (H), the magnetic field within the core does not rise in proportion.

Now consider path 2 of the current waveform. The current through the coil is decreasing and the strength of the magnetic field within the core follows path 2. When we get to the point where (H) is zero, the (B) curve still shows near saturation. This indicates that once a core has been saturated by a magnetizing force and the magnetizing force is removed, the core will hold some magnetic field and not go to zero. This feature of a core is called remanence and the field it contains is called the residual magnetic field.

If we now continue along the current waveform path 2 from the zero point toward the left, the magnetic field in the core will decrease from its saturated condition and go into saturation in the opposite direction with a completely reversed field.

In considering current waveform path 3, the core field will follow curve 3 and go into saturation with a full reversal of the field within the core.

The amount of current in the winding to produce the magnetizing force field is dependent upon the number of turns of the coil (N) and the current passing through the coil (I) in the relationship.

$$\phi = NI$$

It is conceivable that we could have a coil of one turn and with increased current the same core saturation result could be obtained. The configuration of such a device would be as shown in the following diagram.

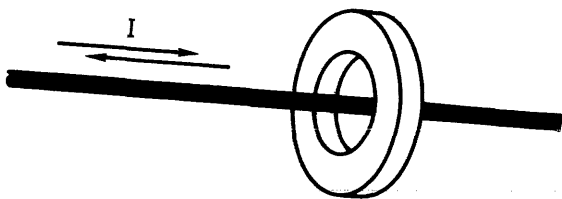


Figure 7-4. Coil with One Turn

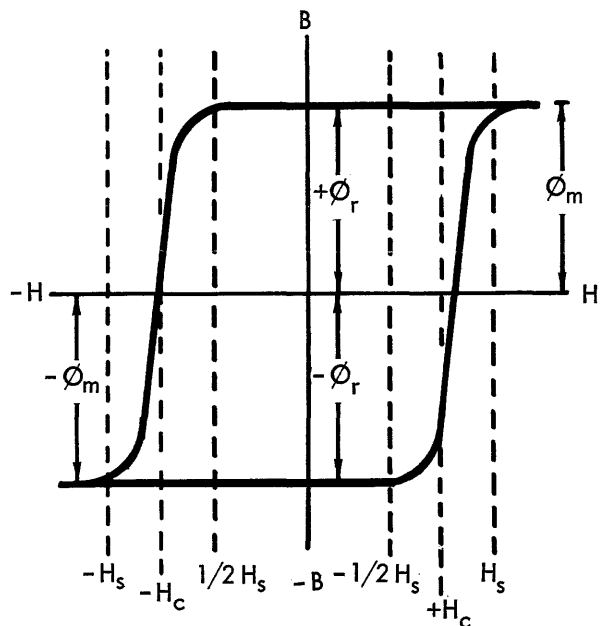


Figure 7-5. Magnetization Curve

$H_s$  = Magnetizing force required to saturate the core to saturation field strength

$\phi_m$  = Saturation core field strength.

$1/2 H_s$  = Residual or remanent core field strength in the absence of magnetizing force after having been saturated.

Note that with  $1/2 H_s$  magnetizing force the core field strength does not change appreciably regardless of which direction the core is saturated or whether the magnetizing force ( $1/2 H_s$ ) is positive or negative.

From Figure 7-5, it is seen that if the core had been saturated in the positive direction by a positive magnetizing force,  $H_s$ , and then the magnetizing force dropped to 0, there would remain in the core a field of strength  $\phi_r$ , very near the saturation level. If, now, a negative magnetizing force is caused to be present to the level  $-1/2 H_s$  the field within the core would remain polarized in the same direction and decrease slightly in amplitude. If this negative magnetizing force is removed, the core field will return to the approximate level  $\phi_r$ . Further, if a negative magnetizing force of  $-H_s$  is applied, the field within the core will go through the critical point at the knee of the curve when the magnetizing force passes  $-H_c$  and as the magnetizing force increases in amplitude, the core field will change polarity very rapidly, go beyond the lower knee of the curve and will saturate again in the opposite, or negative direction.

When the negative magnetizing force is removed, or reduced to 0, the core field will settle to a level of  $-\phi_r$ . This same reasoning process can be applied to

Again let us go to the graph of magnetizing force (H) and core field strength (B) in Figure 7-3. From further amplification of the graph we may gain further information.

the graph to now get the core saturated in a positive direction. This is left as an exercise for the students. The major points to note in Figure 7-3 and discussion are as follows:

- a. Once saturated, a core field will return to  $\pm \phi_r$  when the magnetizing force is removed.
- b.  $\pm 1/2 H_s$  will not change the core field appreciably from  $\pm \phi_r$ .
- c.  $\pm H_c$  is of greater amplitude than  $\pm 1/2 H_s$  and as a result there is little likelihood that  $1/2 H_s$  will disturb the  $\pm \phi_r$  field strength amplitude of a core.
- d.  $\pm H_s$  drives the core well beyond the knee of the saturation level of a core.

### Using Two Wires to Pass Current

Thus far we have talked of current in one wire only but it is also possible that the current could be divided into two equal parts and passed through two separate wires thus creating the same effects as was accomplished with one wire. The configuration would be as shown in Figure 7-6.

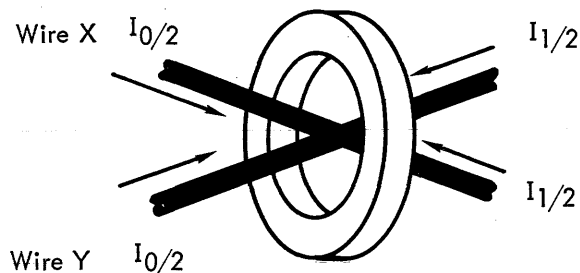


Figure 7-6. Two Wire Coil

Note that the current passing through each wire from either side of the core considered separately, is designated as  $1/2$  the current required to saturate the core to  $0(I_0)$  or  $1/2$  the current required to saturate the core to  $1(I_1)$ . The "0" and "1" field directions were arbitrarily chosen in this case.

The fields of magnetizing force created by currents flowing through wires "X" and "Y" are additive. Thus if we were to consider the field resulting from current flowing from left-to-right in both X and Y, that field would be approximately twice the field that would be caused by current flowing in only one of the wires, either X or Y.

Further, if current were flowing in X from left-to-right and in Y from right-to-left, the resulting field of magnetizing force would be approximately zero.

### Inhibiting Current

With this information we can now insert another wire through the core and use it to prevent magnetizing the core in the 1 direction by causing current to pass through that third wire always in a direction which would prevent 1 core field saturation when desired, by merely turning the third wire current off when 1 core field is desired. We would have a system of preventing or inhibiting 1 when desired for the core. The configuration would be:

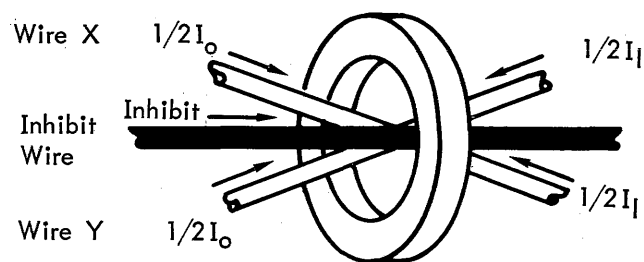


Figure 7-7. Inhibit Current Scheme

It must be explained that inhibit current will be equal to about the value of  $1/2 I_0$ . If we now wish to record a zero magnetic field in the core, assuming it is already saturated and at zero, and we know that  $1/2 I_1$  will flow in both X and Y wires, we must do something to prevent the cumulative action of the  $I_1$  current from switching the core field. To do this we must turn on the inhibit wire current before the X and Y wire currents are permitted to flow.

The following diagrams show the conditions with inhibit and no inhibit current. The cores are assumed to be polarized to 0 at the time the currents flow and the currents start and stop at the same instant.

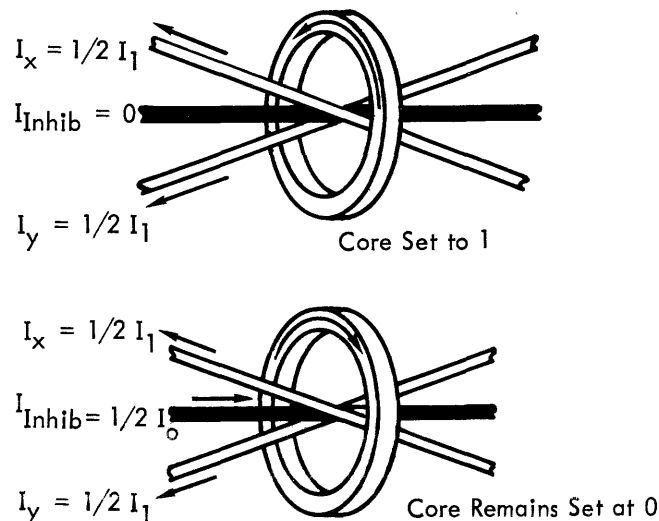


Figure 7-8. Conditions of Inhibit and No-Inhibit Current

What has been discussed thus far is the writing process that takes place in magnetic core systems. Cores are set at 0 prior to the "write" series of events which will provide X and Y half currents to the selected cores. Those cores which are to remain at 0 will always have the inhibit half current flowing in the wire associated with those individual cores.

### The Read Operation

Once the information is in the core, how is the information obtained out of the core for use when desired and then restored to the core for retention in memory? We have said that all cores are set to zero, after which selected 1's are "written" back into the memory cores. It should be obvious now that when cores are set to zero, only those which had a 1 polarization will have a complete reversal of core field. It is this changing core field which indicates that the core had a 1 in it when it was set to 0. To gather this information or to "sense" the change, a conductor in the presence of the rapidly changing field is required. The conductor in the changing magnetic field will have a voltage induced in it and this voltage can be amplified and used as a 1 indicator. Therefore another wire must be placed through the core for sensing the change in polarity when the core is set to 0.

Setting the selected cores to 0 is known as the read operation. It is only during this period that a change of core polarity contains usable information and therefore, only during this period are the "sensing amplifiers" activated and able to amplify a 1 voltage from the sense wire, if it is present. The core configuration now appears as in Figure 7-9:

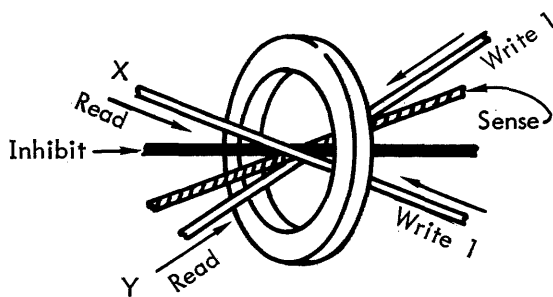


Figure 7-9. Core Lacing and Various Currents

It should be noted that half currents are reversed at X and Y for read and write. Inhibit current flows only during write and always in the same direction.

In the case of the core being polarized to 1 the read half currents flowing in X and Y will cause the field of the core to rapidly change polarity and the collapsing and building core field will occur when a 1 is written into a core which was previously reset to zero by the read half currents. However, at this time the voltage induced in the sense wire is ignored.

Thus far we have talked about single cores. However single cores are not used alone, but are gathered together to handle multiple bits of storage. An example of how they are arranged for storage of 16 bits of information is given in Figure 7-10.

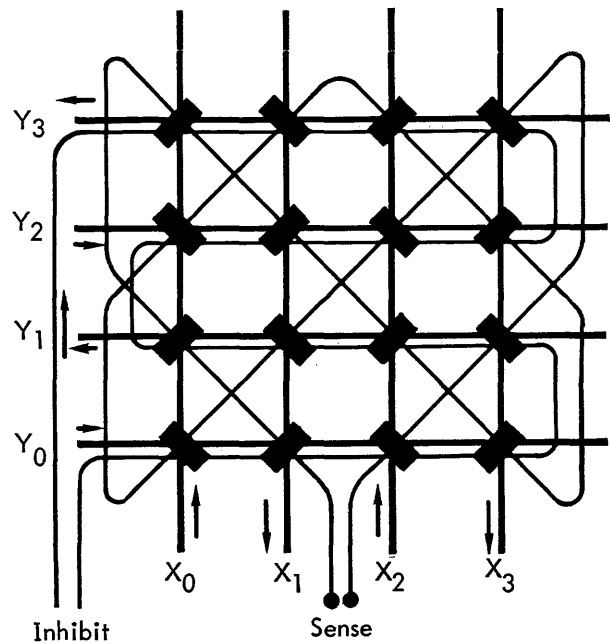


Figure 7-10. Sixteen-Bit Core Storage Arrangement

Figure 7-10 points up pertinent information with regard to core memories as follows:

- a. Adjacent X lines have current flowing through them in opposite directions.
- b. Adjacent Y lines have current flowing through them in opposite directions.
- c. The inhibit lines parallel the Y lines and the current will flow only in one direction, i. e., in the direction of Y read half-current.
- d. One X and one Y line can address only one core with full current, and this is the core at the crossover point. All other cores on the X line and Y line in use are subjected to only half current.
- e. It is possible to get either polarity of signal from the sense line as input to the sense amplifier circuits.
- f. Cores on the Y drive line and X drive line, not at the crossover point have noise generated by the half current disturbing the core, which induces a small voltage in the sense winding. The amplitude of the voltage is dependent upon the flux change and the

polarity is determined by the direction of flux change with respect to the sense winding.

g. All voltages induced in the sense winding are algebraically additive.

That the currents in adjacent X lines and adjacent Y lines are in opposite directions is a matter of core plane design. What is shown in Figure 7-10 is one core plane capable of holding 16 discrete bits of 1 or 0 information. The reason for wiring the core plane in this way will become clear as we begin to stack core planes and as we go on with the discussion of the points mentioned previously.

### Core Addressing

Whether we read or write in the core memory, only one X drive line and one Y drive line are used to address a single core in a core plane. For instance if we use  $X_0$  and  $Y_0$  drive lines, we selected the core which is subjected to the cumulative magnetic fields created by half current in  $X_0$  and half current in  $Y_0$ . All other cores on line  $X_0$  and line  $Y_0$  are subjected to the magnetic force caused by half currents in lines  $X_0$  and  $Y_0$ . These cores not selected are called "half selected." This applies to either read or write operations.

It will be noted that the inhibit winding goes through every core in the plane and the current in this winding has been specified to be always in the same direction as current in all Y lines which would cause a 0 to be written in the selected core. In other words, the inhibit current cancels the effect of the Y current for all cores in the selected Y line. Assuming that the arrows alongside X and Y drive lines in Figure 7-10 are shown in the 1 direction of current, the current in the inhibit winding would be as shown in Figure 7-10.

From previous discussions on core theory for a single core, and the above discussion on single core selection in a core plane, further discussion of how information is stored in a unique location within the core plane would be redundant. All that is needed is to know how a core is "selected" and then the theory of a single core applies to that selected core for information storage. The only lines of concern during this operation are selected X and Y drive lines and the inhibit winding.

### Reading from Core

Reading from core memory requires more discussion since there are actions here which are not apparent without investigation. The only lines used in reading are the selected X and Y drive lines and the sense winding.

From preceding single core theory it is obvious that when the selected X and Y lines have currents in each

of them in such directions to drive the selected core to zero a large change of flux will cause a large voltage in the sense winding. Further, if the selected core was at zero, a much smaller voltage will be induced in the sense winding since the resultant change of fields strength in the selected core is smaller.

In addition to the effect of the read half currents in the selected X and Y lines on the selected core, there is the effect of the half currents in both X and Y on half selected cores. These half selected cores will also undergo a small change in flux and cause a correspondingly small voltage to be induced in the sense winding.

Regardless of the source of the induced voltages in the sense winding the voltage appearing at the sense winding terminals is the algebraic sum of the induced voltages.

Because of the way a core plane is wired and the direction of currents in each X and Y line, the voltages induced in the sense winding by individual half selected cores tend to cancel one another. In other words the voltage induced in the sense winding by one core half-current-flux change will tend to force current in the sense winding in one direction and the effect of another core half-current-flux change will tend to force sense winding current in the opposite direction. Figure 7-11 is a segment of Figure 7-10 using  $X_0$  and  $Y_0$  to explain this point.

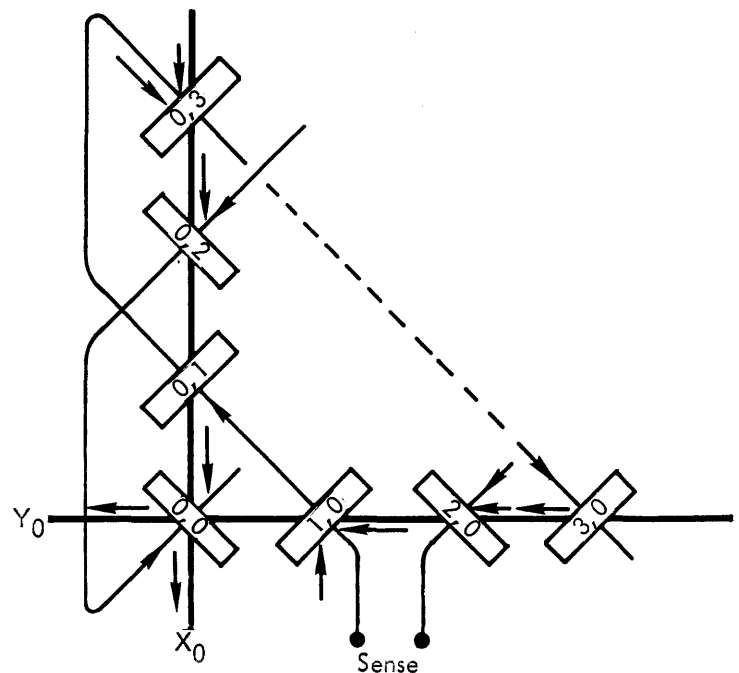


Figure 7-11. Effects of One Core Half-Current-Flux Change

Referring to Figure 7-11, we see the cores are numbered with two numbers. These are in the form X,Y, with the numbers representing the subscript numbers of the X and Y drive lines that lace through the individual cores. Also, the sense winding is traced in only one direction. The arrow associated with each segment of the sense winding as it passes through a core indicates the direction of tracing the sense winding.

The fully selected core (0,0) with X and Y half currents going in to the core in one direction and the direction of sense-winding tracing in the opposite direction produces a zero going indication at the sense wire terminals. Knowing this we can then say that for the half selected cores:

a. If the sense-winding tracing arrow and the direction of half current are opposite, a small 0 signal is produced.

b. If the sense-winding tracing arrow and the direction of half current are the same, a small 1 signal is produced.

Therefore, we can now count these half selected cores which produce small 1's and 0's.

<u>Core #</u>	<u>1</u>	<u>0</u>
0,1		X
0,2	X	
0,3	X	
1,0	X	
2,0	X	
3,0		X
	4	2

Thus the effect of the 1 signal which could reduce the amplitude of the fully selected signal is offset by cores (0,1) and (3,0) generating in the sense winding 0 signals to offset the 1 effect of cores (0,2), (0,3), (1,0) and (2,0). This effect does not seem significant in this case since the core plane is very small and only 50% reduction in cancelling effect from half selected cores result. However, as core planes increase in size to 64 x 64 or 4096 cores in a plane, this reduction becomes significant.

It is left as a student exercise to draw out a simple 8 x 8 plane as in Figure 7-12 and prove to himself that in this case the opposing induced voltages from half currents cause greater than 50% core noise reduction due to cancellation.

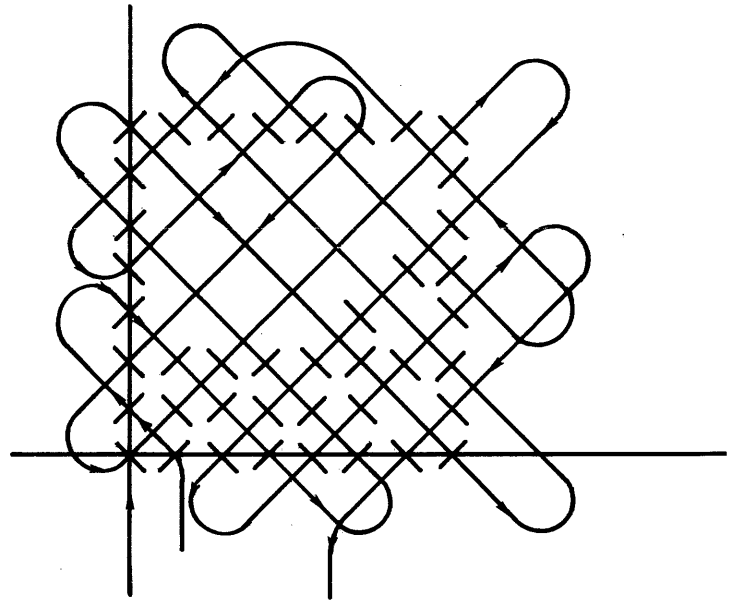


Figure 7-12. 8 x 8 Core Plane

### Core Arrays

Thus far we have been concerned with single cores and single core planes. The next step in logical sequence is to use the core planes in arrays in order to be able to store and retrieve information in a manner that we can handle more than one bit of information at a time.

The core plane we discussed previously is also called a "bit-plane". The name bit-plane comes from the fact that in digital computers using core memory there is one such plane for every bit in a computer word. Thus, a 6-bit computer word would require 6-bit planes, and a 24-bit computer word would require 24 bit-planes.

The number of words that can be stored in a memory is a function of the number of X and Y drive lines per bit-plane. As an example, a 4096 word memory for 24-bit word length would require 24 bit-planes, each of which has 64 X and 64 Y drive lines.

The diagram of Figure 7-13 shows the X and Y drive lines for 4-bit words in core memory. Note particularly in Figure 7-13:

a. Each end of an  $X_0$  drive line of each plane, except the top and bottom, is attached to the end of a  $X_0$  drive line of an adjacent plane by a connecting wire external to the core planes. Therefore, all  $X_0$  drive lines are in series.

b. Each end of a  $Y_0$  drive line of each plane, except the top and bottom, is attached to the end of a  $Y_0$  drive line of an adjacent plane by a connecting wire external to the core plane. Therefore, all  $Y_0$  drive lines are in series.

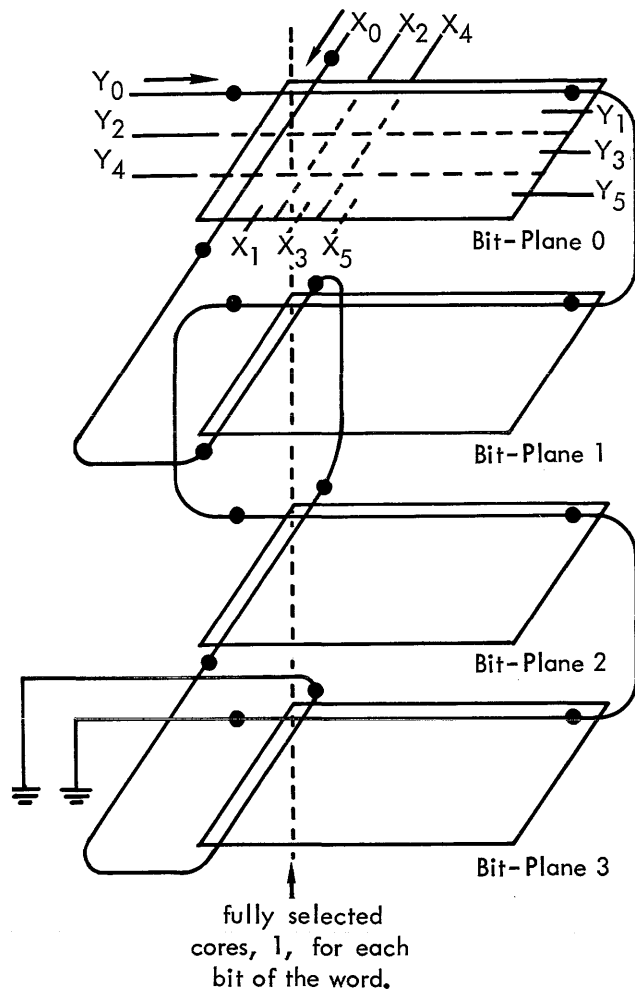


Figure 7-13. X and Y Drive Lines for 4-bit Words in Core Memory.

c. The driven end of the  $X_0$  and  $Y_0$  drive lines are attached to current drivers which can supply current in either direction. The ends of the series connections of  $X_0$  and  $Y_0$  drive lines, opposite the driven ends, are grounded. Therefore the same  $X_0$  half current flows through all plane  $X_0$  drive lines and the same  $Y_0$  half current flows through all plane  $Y_0$  drive lines.

d. In each bit-plane, only one core is fully selected. The selected cores in all planes are in the same relative physical location in each bit-plane.

e. All even numbered X drive lines are attached to drivers on the same side of the core plane, and all odd numbered X drive lines are attached to drivers on the opposite side of the core plane.

f. All even numbered Y drive lines are attached to drivers on the same side of the core plane, and all odd numbered Y drive lines are attached to drivers on the opposite side of the core plane.

g. There is required for each X line a separate current driver, and for each Y line a separate current driver.

Since we are interested in storing individual bits in each bit-plane, and reading individual bits from each bit-plane, the sense winding and inhibit winding of each bit-plane are exclusive to their bit-plane. Therefore, a separate inhibit current driver is required for each plane, and a separate sense amplifier is needed for each plane.

There are many different ways of accomplishing plane wiring, current driving, and sensing. Those presented here are only intended to give a reader a basic understanding of core memory theory.



## APPENDIX A. CONVERSION TABLES

### Octal-Decimal Integer Conversion Table

		0 1 2 3 4 5 6 7								0 1 2 3 4 5 6 7								
0000 to 0777 (Octal)	0000 to 0511 (Decimal)	0000	0001	0002	0003	0004	0005	0006	0007	0400	0256	0257	0258	0259	0260	0261	0262	0263
		0010	0008	0009	0010	0011	0012	0013	0014	0410	0264	0265	0266	0267	0268	0269	0270	0271
		0020	0016	0017	0018	0019	0020	0021	0022	0420	0272	0273	0274	0275	0276	0277	0278	0279
		0030	0024	0025	0026	0027	0028	0029	0030	0430	0280	0281	0282	0283	0284	0285	0286	0287
		0040	0032	0033	0034	0035	0036	0037	0038	0440	0288	0289	0290	0291	0292	0293	0294	0295
		0050	0040	0041	0042	0043	0044	0045	0046	0450	0296	0297	0298	0299	0300	0301	0302	0303
		0060	0048	0049	0050	0051	0052	0053	0054	0460	0304	0305	0306	0307	0308	0309	0310	0311
		0070	0056	0057	0058	0059	0060	0061	0062	0470	0312	0313	0314	0315	0316	0317	0318	0319
Octal	Decimal	0100	0064	0065	0066	0067	0068	0069	0070	0500	0320	0321	0322	0323	0324	0325	0326	0327
		0110	0072	0073	0074	0075	0076	0077	0078	0510	0328	0329	0330	0331	0332	0333	0334	0335
		0120	0080	0081	0082	0083	0084	0085	0086	0520	0336	0337	0338	0339	0340	0341	0342	0343
		0130	0088	0089	0090	0091	0092	0093	0094	0530	0344	0345	0346	0347	0348	0349	0350	0351
		0140	0096	0097	0098	0099	0100	0101	0102	0540	0352	0353	0354	0355	0356	0357	0358	0359
		0150	0104	0105	0106	0107	0108	0109	0110	0550	0360	0361	0362	0363	0364	0365	0366	0367
		0160	0112	0113	0114	0115	0116	0117	0118	0560	0368	0369	0370	0371	0372	0373	0374	0375
		0170	0120	0121	0122	0123	0124	0125	0126	0570	0376	0377	0378	0379	0380	0381	0382	0383
		0200	0128	0129	0130	0131	0132	0133	0134	0600	0384	0385	0386	0387	0388	0389	0390	0391
		0210	0136	0137	0138	0139	0140	0141	0142	0610	0392	0393	0394	0395	0396	0397	0398	0399
		0220	0144	0145	0146	0147	0148	0149	0150	0620	0400	0401	0402	0403	0404	0405	0406	0407
		0230	0152	0153	0154	0155	0156	0157	0158	0630	0408	0409	0410	0411	0412	0413	0414	0415
		0240	0160	0161	0162	0163	0164	0165	0166	0640	0416	0417	0418	0419	0420	0421	0422	0423
		0250	0168	0169	0170	0171	0172	0173	0174	0650	0424	0425	0426	0427	0428	0429	0430	0431
		0260	0176	0177	0178	0179	0180	0181	0182	0660	0432	0433	0434	0435	0436	0437	0438	0439
		0270	0184	0185	0186	0187	0188	0189	0190	0670	0440	0441	0442	0443	0444	0445	0446	0447
		0300	0192	0193	0194	0195	0196	0197	0198	0700	0448	0449	0450	0451	0452	0453	0454	0455
		0310	0200	0201	0202	0203	0204	0205	0206	0710	0456	0457	0458	0459	0460	0461	0462	0463
		0320	0208	0209	0210	0211	0212	0213	0214	0720	0464	0465	0466	0467	0468	0469	0470	0471
		0330	0216	0217	0218	0219	0220	0221	0222	0730	0472	0473	0474	0475	0476	0477	0478	0479
		0340	0224	0225	0226	0227	0228	0229	0230	0740	0480	0481	0482	0483	0484	0485	0486	0487
		0350	0232	0233	0234	0235	0236	0237	0238	0750	0488	0489	0490	0491	0492	0493	0494	0495
		0360	0240	0241	0242	0243	0244	0245	0246	0760	0496	0497	0498	0499	0500	0501	0502	0503
		0370	0248	0249	0250	0251	0252	0253	0254	0770	0504	0505	0506	0507	0508	0509	0510	0511
1000 to 1777 (Octal)	0512 to 1023 (Decimal)	1000	0512	0513	0514	0515	0516	0517	0518	1400	0768	0769	0770	0771	0772	0773	0774	0775
		1010	0520	0521	0522	0523	0524	0525	0526	1410	0776	0777	0778	0779	0780	0781	0782	0783
		1020	0528	0529	0530	0531	0532	0533	0534	1420	0784	0785	0786	0787	0788	0789	0790	0791
		1030	0536	0537	0538	0539	0540	0541	0542	1430	0792	0793	0794	0795	0796	0797	0798	0799
		1040	0544	0545	0546	0547	0548	0549	0550	1440	0800	0801	0802	0803	0804	0805	0806	0807
		1050	0552	0553	0554	0555	0556	0557	0558	1450	0808	0809	0810	0811	0812	0813	0814	0815
		1060	0560	0561	0562	0563	0564	0565	0566	1460	0816	0817	0818	0819	0820	0821	0822	0823
		1070	0568	0569	0570	0571	0572	0573	0574	1470	0824	0825	0826	0827	0828	0829	0830	0831
		1100	0576	0577	0578	0579	0580	0581	0582	1500	0832	0833	0834	0835	0836	0837	0838	0839
		1110	0584	0585	0586	0587	0588	0589	0590	1510	0840	0841	0842	0843	0844	0845	0846	0847
		1120	0592	0593	0594	0595	0596	0597	0598	1520	0848	0849	0850	0851	0852	0853	0854	0855
		1130	0600	0601	0602	0603	0604	0605	0606	1530	0856	0857	0858	0859	0860	0861	0862	0863
		1140	0608	0609	0610	0611	0612	0613	0614	1540	0864	0865	0866	0867	0868	0869	0870	0871
		1150	0616	0617	0618	0619	0620	0621	0622	1550	0872	0873	0874	0875	0876	0877	0878	0879
		1160	0624	0625	0626	0627	0628	0629	0630	1560	0880	0881	0882	0883	0884	0885	0886	0887
		1170	0632	0633	0634	0635	0636	0637	0638	1570	0888	0889	0890	0891	0892	0893	0894	0895
		1200	0640	0641	0642	0643	0644	0645	0646	1600	0896	0897	0898	0899	0900	0901	0902	0903
		1210	0648	0649	0650	0651	0652	0653	0654	1610	0904	0905	0906	0907	0908	0909	0910	0911
		1220	0656	0657	0658	0659	0660	0661	0662	1620	0912	0913	0914	0915	0916	0917	0918	0919
		1230	0664	0665	0666	0667	0668	0669	0670	1630	0920	0921	0922	0923	0924	0925	0926	0927
		1240	0672	0673	0674	0675	0676	0677	0678	1640	0928	0929	0930	0931	0932	0933	0934	0935
		1250	0680	0681	0682	0683	0684	0685	0686	1650	0936	0937	0938	0939	0940	0941	0942	0943
		1260	0688	0689	0690	0691	0692	0693	0694	1660	0944	0945	0946	0947	0948	0949	0950	0951
		1270	0696	0697	0698	0699	0700	0701	0702	1670	0952	0953	0954	0955	0956	0957	0958	0959
		1300	0704	0705	0706	0707	0708	0709	0710	1700	0960	0961	0962	0963	0964	0965	0966	0967
		1310	0712	0713	0714	0715	0716	0717	0718	1710	0968	0969	0970	0971	0972	0973	0974	0975
		1320	0720	0721	0722	0723	0724	0725	0726	1720	0976	0977	0978	0979	0980	0981	0982	0983
		1330	0728	0729	0730	0731	0732	0733	0734	1730	0984	0985	0986	0987	0988	0989	0990	0991
		1340	0736	0737	0738	0739	0740	0741	0742	1740	0992	0993	0994	0995	0996	0997	0998	0999
		1350	0744	0745	0746	0747	0748	0749	0750	1750	1000	1001	1002	1003	1004	1005	1006	1007
		1360	0752	0753	0754	0755	0756	0757	0758	1760	1008	1009	1010	1011	1012	1013	1014	1015
		1370	0760	0761	0762	0763	0764	0765	0766	1770	1016	1017	1018	1019	1020	1021	1022	1023





## APPENDIX A. CONVERSION TABLES (Cont'd)

### Octal-Decimal Integer Conversion Table (Cont'd)

			0	1	2	3	4	5	6	7			0	1	2	3	4	5	6	7
4000 to 4777 (Octal)	2048 to 2559 (Decimal)	4000	2048	2049	2050	2051	2052	2053	2054	2055	4400	2304	2305	2306	2307	2308	2309	2310	2311	
		4010	2056	2057	2058	2059	2060	2061	2062	2063	4410	2312	2313	2314	2315	2316	2317	2318	2319	
		4020	2064	2065	2066	2067	2068	2069	2070	2071	4420	2320	2321	2322	2323	2324	2325	2326	2327	
		4030	2072	2073	2074	2075	2076	2077	2078	2079	4430	2328	2329	2330	2331	2332	2333	2334	2335	
		4040	2080	2081	2082	2083	2084	2085	2086	2087	4440	2336	2337	2338	2339	2340	2341	2342	2343	
		4050	2088	2089	2090	2091	2092	2093	2094	2095	4450	2344	2345	2346	2347	2348	2349	2350	2351	
		4060	2096	2097	2098	2099	2100	2101	2102	2103	4460	2352	2353	2354	2355	2356	2357	2358	2359	
		4070	2104	2105	2106	2107	2108	2109	2110	2111	4470	2360	2361	2362	2363	2364	2365	2366	2367	
		4100	2112	2113	2114	2115	2116	2117	2118	2119	4500	2368	2369	2370	2371	2372	2373	2374	2375	
		4110	2120	2121	2122	2123	2124	2125	2126	2127	4510	2376	2377	2378	2379	2380	2381	2382	2383	
4120	2128	2129	2130	2131	2132	2133	2134	2135	4520	2384	2385	2386	2387	2388	2389	2390	2391			
4130	2136	2137	2138	2139	2140	2141	2142	2143	4530	2392	2393	2394	2395	2396	2397	2398	2399			
4140	2144	2145	2146	2147	2148	2149	2150	2151	4540	2400	2401	2402	2403	2404	2405	2406	2407			
4150	2152	2153	2154	2155	2156	2157	2158	2159	4550	2408	2409	2410	2411	2412	2413	2414	2415			
4160	2160	2161	2162	2163	2164	2165	2166	2167	4560	2416	2417	2418	2419	2420	2421	2422	2423			
4170	2168	2169	2170	2171	2172	2173	2174	2175	4570	2424	2425	2426	2427	2428	2429	2430	2431			
4200	2176	2177	2178	2179	2180	2181	2182	2183	4600	2432	2433	2434	2435	2436	2437	2438	2439			
4210	2184	2185	2186	2187	2188	2189	2190	2191	4610	2440	2441	2442	2443	2444	2445	2446	2447			
4220	2192	2193	2194	2195	2196	2197	2198	2199	4620	2448	2449	2450	2451	2452	2453	2454	2455			
4230	2200	2201	2202	2203	2204	2205	2206	2207	4630	2456	2457	2458	2459	2460	2461	2462	2463			
4240	2208	2209	2210	2211	2212	2213	2214	2215	4640	2464	2465	2466	2467	2468	2469	2470	2471			
4250	2216	2217	2218	2219	2220	2221	2222	2223	4650	2472	2473	2474	2475	2476	2477	2478	2479			
4260	2224	2225	2226	2227	2228	2229	2230	2231	4660	2480	2481	2482	2483	2484	2485	2486	2487			
4270	2232	2233	2234	2235	2236	2237	2238	2239	4670	2488	2489	2490	2491	2492	2493	2494	2495			
4300	2240	2241	2242	2243	2244	2245	2246	2247	4700	2496	2497	2498	2499	2500	2501	2502	2503			
4310	2248	2249	2250	2251	2252	2253	2254	2255	4710	2504	2505	2506	2507	2508	2509	2510	2511			
4320	2256	2257	2258	2259	2260	2261	2262	2263	4720	2512	2513	2514	2515	2516	2517	2518	2519			
4330	2264	2265	2266	2267	2268	2269	2270	2271	4730	2520	2521	2522	2523	2524	2525	2526	2527			
4340	2272	2273	2274	2275	2276	2277	2278	2279	4740	2528	2529	2530	2531	2532	2533	2534	2535			
4350	2280	2281	2282	2283	2284	2285	2286	2287	4750	2536	2537	2538	2539	2540	2541	2542	2543			
4360	2288	2289	2290	2291	2292	2293	2294	2295	4760	2544	2545	2546	2547	2548	2549	2550	2551			
4370	2296	2297	2298	2299	2300	2301	2302	2303	4770	2552	2553	2554	2555	2556	2557	2558	2559			
5000	2560	2561	2562	2563	2564	2565	2566	2567	5400	2816	2817	2818	2819	2820	2821	2822	2823			
5010	2568	2569	2570	2571	2572	2573	2574	2575	5410	2824	2825	2826	2827	2828	2829	2830	2831			
5020	2576	2577	2578	2579	2580	2581	2582	2583	5420	2832	2833	2834	2835	2836	2837	2838	2839			
5030	2584	2585	2586	2587	2588	2589	2590	2591	5430	2840	2841	2842	2843	2844	2845	2846	2847			
5040	2592	2593	2594	2595	2596	2597	2598	2599	5440	2848	2849	2850	2851	2852	2853	2854	2855			
5050	2600	2601	2602	2603	2604	2605	2606	2607	5450	2856	2857	2858	2859	2860	2861	2862	2863			
5060	2608	2609	2610	2611	2612	2613	2614	2615	5460	2864	2865	2866	2867	2868	2869	2870	2871			
5070	2616	2617	2618	2619	2620	2621	2622	2623	5470	2872	2873	2874	2875	2876	2877	2878	2879			
5100	2624	2625	2626	2627	2628	2629	2630	2631	5500	2880	2881	2882	2883	2884	2885	2886	2887			
5110	2632	2633	2634	2635	2636	2637	2638	2639	5510	2888	2889	2890	2891	2892	2893	2894	2895			
5120	2640	2641	2642	2643	2644	2645	2646	2647	5520	2896	2897	2898	2899	2900	2901	2902	2903			
5130	2648	2649	2650	2651	2652	2653	2654	2655	5530	2904	2905	2906	2907	2908	2909	2910	2911			
5140	2656	2657	2658	2659	2660	2661	2662	2663	5540	2912	2913	2914	2915	2916	2917	2918	2919			
5150	2664	2665	2666	2667	2668	2669	2670	2671	5550	2920	2921	2922	2923	2924	2925	2926	2927			
5160	2672	2673	2674	2675	2676	2677	2678	2679	5560	2928	2929	2930	2931	2932	2933	2934	2935			
5170	2680	2681	2682	2683	2684	2685	2686	2687	5570	2936	2937	2938	2939	2940	2941	2942	2943			
5200	2688	2689	2690	2691	2692	2693	2694	2695	5600	2944	2945	2946	2947	2948	2949	2950	2951			
5210	2696	2697	2698	2699	2700	2701	2702	2703	5610	2952	2953	2954	2955	2956	2957	2958	2959			
5220	2704	2705	2706	2707	2708	2709	2710	2711	5620	2960	2961	2962	2963	2964	2965	2966	2967			
5230	2712	2713	2714	2715	2716	2717	2718	2719	5630	2968	2969	2970	2971	2972	2973	2974	2975			
5240	2720	2721	2722	2723	2724	2725	2726	2727	5640	2976	2977	2978	2979	2980	2981	2982	2983			
5250	2728	2729	2730	2731	2732	2733	2734	2735	5650	2984	2985	2986	2987	2988	2989	2990	2991			
5260	2736	2737	2738	2739	2740	2741	2742	2743	5660	2992	2993	2994	2995	2996	2997	2998	2999			
5270	2744	2745	2746	2747	2748	2749	2750	2751	5670	3000	3001	3002	3003	3004	3005	3006	3007			
5300	2752	2753	2754	2755	2756	2757	2758	2759	5700	3008	3009	3010	3011	3012	3013	3014	3015			
5310	2760	2761	2762	2763	2764	2765	2766	2767	5710	3016	3017	3018	3019	3020	3021	3022	3023			
5320	2768	2769	2770	2771	2772	2773	2774	2775	5720	3024	3025	3026	3027	3028	3029	3030	3031			
5330	2776	2777	2778	2779	2780	2781	2782	2783	5730	3032	3033	3034	3035	3036	3037	3038	3039			
5340	2784	2785	2786	2787	2788	2789	2790	2791	5740	3040	3041	3042	3043	3044	3045	3046	3047			
5350	2792	2793	2794	2795	2796	2797	2798	2799	5750	3048	3049	3050	3051	3052	3053	3054	3055			
5360	2800	2801	2802	2803	2804	2805	2806	2807	5760	3056	3057	3058	3059	3060	3061	3062	3063			
5370	2808	2809	2810	2811	2812	2813	2814	2815	5770	3064	3065	3066	3067	3068	3069	3070	3071			



## APPENDIX A. CONVERSION TABLES (Cont'd)

### Octal-Decimal Fraction Conversion Table

OCTAL	DEC.	OCTAL	DEC.	OCTAL	DEC.	OCTAL	DEC.
.000	.000000	.100	.125000	.200	.250000	.300	.375000
.001	.001953	.101	.126953	.201	.251953	.301	.376953
.002	.003906	.102	.128906	.202	.253906	.302	.378906
.003	.005859	.103	.130859	.203	.255859	.303	.380859
.004	.007812	.104	.132812	.204	.257812	.304	.382812
.005	.009765	.105	.134765	.205	.259765	.305	.384765
.006	.011718	.106	.136718	.206	.261718	.306	.386718
.007	.013671	.107	.138671	.207	.263671	.307	.388671
.010	.015625	.110	.140625	.210	.265625	.310	.390625
.011	.017578	.111	.142578	.211	.267578	.311	.392578
.012	.019531	.112	.144531	.212	.269531	.312	.394531
.013	.021484	.113	.146484	.213	.271484	.313	.396484
.014	.023437	.114	.148437	.214	.273437	.314	.398437
.015	.025390	.115	.150390	.215	.275390	.315	.400390
.016	.027343	.116	.152343	.216	.277343	.316	.402343
.017	.029296	.117	.154296	.217	.279296	.317	.404296
.020	.031250	.120	.156250	.220	.281250	.320	.406250
.021	.033203	.121	.158203	.221	.283203	.321	.408203
.022	.035156	.122	.160156	.222	.285156	.322	.410156
.023	.037109	.123	.162109	.223	.287109	.323	.412109
.024	.039062	.124	.164062	.224	.289062	.324	.414062
.025	.041015	.125	.166015	.225	.291015	.325	.416015
.026	.042968	.126	.167968	.226	.292968	.326	.417968
.027	.044921	.127	.169921	.227	.294921	.327	.419921
.030	.046875	.130	.171875	.230	.296875	.330	.421875
.031	.048828	.131	.173828	.231	.298828	.331	.423828
.032	.050781	.132	.175781	.232	.300781	.332	.425781
.033	.052734	.133	.177734	.233	.302734	.333	.427734
.034	.054687	.134	.179687	.234	.304687	.334	.429687
.035	.056640	.135	.181640	.235	.306640	.335	.431640
.036	.058593	.136	.183593	.236	.308593	.336	.433593
.037	.060546	.137	.185546	.237	.310546	.337	.435546
.040	.062500	.140	.187500	.240	.312500	.340	.437500
.041	.064453	.141	.189453	.241	.314453	.341	.439453
.042	.066406	.142	.191406	.242	.316406	.342	.441406
.043	.068359	.143	.193359	.243	.318359	.343	.443359
.044	.070312	.144	.195312	.244	.320312	.344	.445312
.045	.072265	.145	.197265	.245	.322265	.345	.447265
.046	.074218	.146	.199218	.246	.324218	.346	.449218
.047	.076171	.147	.201171	.247	.326171	.347	.451171
.050	.078125	.150	.203125	.250	.328125	.350	.453125
.051	.080078	.151	.205078	.251	.330078	.351	.455078
.052	.082031	.152	.207031	.252	.332031	.352	.457031
.053	.083984	.153	.208984	.253	.333984	.353	.458984
.054	.085937	.154	.210937	.254	.335937	.354	.460937
.055	.087890	.155	.212890	.255	.337890	.355	.462890
.056	.089843	.156	.214843	.256	.339843	.356	.464843
.057	.091796	.157	.216796	.257	.341796	.357	.466796
.060	.093750	.160	.218750	.260	.343750	.360	.468750
.061	.095703	.161	.220703	.261	.345703	.361	.470703
.062	.097656	.162	.222656	.262	.347656	.362	.472656
.063	.099609	.163	.224609	.263	.349609	.363	.474609
.064	.101562	.164	.226562	.264	.351562	.364	.476562
.065	.103515	.165	.228515	.265	.353515	.365	.478515
.066	.105468	.166	.230468	.266	.355468	.366	.480468
.067	.107421	.167	.232421	.267	.357421	.367	.482421
.070	.109375	.170	.234375	.270	.359375	.370	.484375
.071	.111328	.171	.236328	.271	.361328	.371	.486328
.072	.113281	.172	.238281	.272	.363281	.372	.488281
.073	.115234	.173	.240234	.273	.365234	.373	.490234
.074	.117187	.174	.242187	.274	.367187	.374	.492187
.075	.119140	.175	.244140	.275	.369140	.375	.494140
.076	.121093	.176	.246093	.276	.371093	.376	.496093
.077	.123046	.177	.248046	.277	.373046	.377	.498046

## APPENDIX A. CONVERSION TABLES (Cont'd)

### Octal-Decimal Fraction Conversion Table (Cont'd)

OCTAL	DEC.	OCTAL	DEC.	OCTAL	DEC.	OCTAL	DEC.
.000000	.000000	.000100	.000244	.000200	.000488	.000300	.000732
.000001	.000003	.000101	.000247	.000201	.000492	.000301	.000736
.000002	.000007	.000102	.000251	.000202	.000495	.000302	.000740
.000003	.000011	.000103	.000255	.000203	.000499	.000303	.000743
.000004	.000015	.000104	.000259	.000204	.000503	.000304	.000747
.000005	.000019	.000105	.000263	.000205	.000507	.000305	.000751
.000006	.000022	.000106	.000267	.000206	.000511	.000306	.000755
.000007	.000026	.000107	.000270	.000207	.000514	.000307	.000759
.000010	.000030	.000110	.000274	.000210	.000518	.000310	.000762
.000011	.000034	.000111	.000278	.000211	.000522	.000311	.000766
.000012	.000038	.000112	.000282	.000212	.000526	.000312	.000770
.000013	.000041	.000113	.000286	.000213	.000530	.000313	.000774
.000014	.000045	.000114	.000289	.000214	.000534	.000314	.000778
.000015	.000049	.000115	.000293	.000215	.000537	.000315	.000782
.000016	.000053	.000116	.000297	.000216	.000541	.000316	.000785
.000017	.000057	.000117	.000301	.000217	.000545	.000317	.000789
.000020	.000061	.000120	.000305	.000220	.000549	.000320	.000793
.000021	.000064	.000121	.000308	.000221	.000553	.000321	.000797
.000022	.000068	.000122	.000312	.000222	.000556	.000322	.000801
.000023	.000072	.000123	.000316	.000223	.000560	.000323	.000805
.000024	.000076	.000124	.000320	.000224	.000564	.000324	.000808
.000025	.000080	.000125	.000324	.000225	.000568	.000325	.000812
.000026	.000083	.000126	.000328	.000226	.000572	.000326	.000816
.000027	.000087	.000127	.000331	.000227	.000576	.000327	.000820
.000030	.000091	.000130	.000335	.000230	.000579	.000330	.000823
.000031	.000095	.000131	.000339	.000231	.000583	.000331	.000827
.000032	.000099	.000132	.000343	.000232	.000587	.000332	.000831
.000033	.000102	.000133	.000347	.000233	.000591	.000333	.000835
.000034	.000106	.000134	.000350	.000234	.000595	.000334	.000839
.000035	.000110	.000135	.000354	.000235	.000598	.000335	.000843
.000036	.000114	.000136	.000358	.000236	.000602	.000336	.000846
.000037	.000118	.000137	.000362	.000237	.000606	.000337	.000850
.000040	.000122	.000140	.000366	.000240	.000610	.000340	.000854
.000041	.000125	.000141	.000370	.000241	.000614	.000341	.000858
.000042	.000129	.000142	.000373	.000242	.000617	.000342	.000862
.000043	.000133	.000143	.000377	.000243	.000621	.000343	.000865
.000044	.000137	.000144	.000381	.000244	.000625	.000344	.000869
.000045	.000141	.000145	.000385	.000245	.000629	.000345	.000873
.000046	.000144	.000146	.000389	.000246	.000633	.000346	.000877
.000047	.000148	.000147	.000392	.000247	.000637	.000347	.000881
.000050	.000152	.000150	.000396	.000250	.000640	.000350	.000885
.000051	.000156	.000151	.000400	.000251	.000644	.000351	.000888
.000052	.000160	.000152	.000404	.000252	.000648	.000352	.000892
.000053	.000164	.000153	.000408	.000253	.000652	.000353	.000896
.000054	.000167	.000154	.000411	.000254	.000656	.000354	.000900
.000055	.000171	.000155	.000415	.000255	.000659	.000355	.000904
.000056	.000175	.000156	.000419	.000256	.000663	.000356	.000907
.000057	.000179	.000157	.000423	.000257	.000667	.000357	.000911
.000060	.000183	.000160	.000427	.000260	.000671	.000360	.000915
.000061	.000186	.000161	.000431	.000261	.000675	.000361	.000919
.000062	.000190	.000162	.000434	.000262	.000679	.000362	.000923
.000063	.000194	.000163	.000438	.000263	.000682	.000363	.000926
.000064	.000198	.000164	.000442	.000264	.000686	.000364	.000930
.000065	.000202	.000165	.000446	.000265	.000690	.000365	.000934
.000066	.000205	.000166	.000450	.000266	.000694	.000366	.000938
.000067	.000209	.000167	.000453	.000267	.000698	.000367	.000942
.000070	.000213	.000170	.000457	.000270	.000701	.000370	.000946
.000071	.000217	.000171	.000461	.000271	.000705	.000371	.000949
.000072	.000221	.000172	.000465	.000272	.000709	.000372	.000953
.000073	.000225	.000173	.000469	.000273	.000713	.000373	.000957
.000074	.000228	.000174	.000473	.000274	.000717	.000374	.000961
.000075	.000232	.000175	.000476	.000275	.000720	.000375	.000965
.000076	.000236	.000176	.000480	.000276	.000724	.000376	.000968
.000077	.000240	.000177	.000484	.000277	.000728	.000377	.000972

APPENDIX A. CONVERSION TABLES (Cont'd)

Octal-Decimal Fraction Conversion Table (Cont'd)

OCTAL	DEC.	OCTAL	DEC.	OCTAL	DEC.	OCTAL	DEC.
.000400	.000976	.000500	.001220	.000600	.001464	.000700	.001708
.000401	.000980	.000501	.001224	.000601	.001468	.000701	.001712
.000402	.000984	.000502	.001228	.000602	.001472	.000702	.001716
.000403	.000988	.000503	.001232	.000603	.001476	.000703	.001720
.000404	.000991	.000504	.001235	.000604	.001480	.000704	.001724
.000405	.000995	.000505	.001239	.000605	.001483	.000705	.001728
.000406	.000999	.000506	.001243	.000606	.001487	.000706	.001731
.000407	.001003	.000507	.001247	.000607	.001491	.000707	.001735
.000410	.001007	.000510	.001251	.000610	.001495	.000710	.001739
.000411	.001010	.000511	.001255	.000611	.001499	.000711	.001743
.000412	.001014	.000512	.001258	.000612	.001502	.000712	.001747
.000413	.001018	.000513	.001262	.000613	.001506	.000713	.001750
.000414	.001022	.000514	.001266	.000614	.001510	.000714	.001754
.000415	.001026	.000515	.001270	.000615	.001514	.000715	.001758
.000416	.001029	.000516	.001274	.000616	.001518	.000716	.001762
.000417	.001033	.000517	.001277	.000617	.001522	.000717	.001766
.000420	.001037	.000520	.001281	.000620	.001525	.000720	.001770
.000421	.001041	.000521	.001285	.000621	.001529	.000721	.001773
.000422	.001045	.000522	.001289	.000622	.001533	.000722	.001777
.000423	.001049	.000523	.001293	.000623	.001537	.000723	.001781
.000424	.001052	.000524	.001296	.000624	.001541	.000724	.001785
.000425	.001056	.000525	.001300	.000625	.001544	.000725	.001789
.000426	.001060	.000526	.001304	.000626	.001548	.000726	.001792
.000427	.001064	.000527	.001308	.000627	.001552	.000727	.001796
.000430	.001068	.000530	.001312	.000630	.001556	.000730	.001800
.000431	.001071	.000531	.001316	.000631	.001560	.000731	.001804
.000432	.001075	.000532	.001319	.000632	.001564	.000732	.001808
.000433	.001079	.000533	.001323	.000633	.001567	.000733	.001811
.000434	.001083	.000534	.001327	.000634	.001571	.000734	.001815
.000435	.001087	.000535	.001331	.000635	.001575	.000735	.001819
.000436	.001091	.000536	.001335	.000636	.001579	.000736	.001823
.000437	.001094	.000537	.001338	.000637	.001583	.000737	.001827
.000440	.001098	.000540	.001342	.000640	.001586	.000740	.001831
.000441	.001102	.000541	.001346	.000641	.001590	.000741	.001834
.000442	.001106	.000542	.001350	.000642	.001594	.000742	.001838
.000443	.001110	.000543	.001354	.000643	.001598	.000743	.001842
.000444	.001113	.000544	.001358	.000644	.001602	.000744	.001846
.000445	.001117	.000545	.001361	.000645	.001605	.000745	.001850
.000446	.001121	.000546	.001365	.000646	.001609	.000746	.001853
.000447	.001125	.000547	.001369	.000647	.001613	.000747	.001857
.000450	.001129	.000550	.001373	.000650	.001617	.000750	.001861
.000451	.001132	.000551	.001377	.000651	.001621	.000751	.001865
.000452	.001136	.000552	.001380	.000652	.001625	.000752	.001869
.000453	.001140	.000553	.001384	.000653	.001628	.000753	.001873
.000454	.001144	.000554	.001388	.000654	.001632	.000754	.001876
.000455	.001148	.000555	.001392	.000655	.001636	.000755	.001880
.000456	.001152	.000556	.001396	.000656	.001640	.000756	.001884
.000457	.001155	.000557	.001399	.000657	.001644	.000757	.001888
.000460	.001159	.000560	.001403	.000660	.001647	.000760	.001892
.000461	.001163	.000561	.001407	.000661	.001651	.000761	.001895
.000462	.001167	.000562	.001411	.000662	.001655	.000762	.001899
.000463	.001171	.000563	.001415	.000663	.001659	.000763	.001903
.000464	.001174	.000564	.001419	.000664	.001663	.000764	.001907
.000465	.001178	.000565	.001422	.000665	.001667	.000765	.001911
.000466	.001182	.000566	.001426	.000666	.001670	.000766	.001914
.000467	.001186	.000567	.001430	.000667	.001674	.000767	.001918
.000470	.001190	.000570	.001434	.000670	.001678	.000770	.001922
.000471	.001194	.000571	.001438	.000671	.001682	.000771	.001926
.000472	.001197	.000572	.001441	.000672	.001686	.000772	.001930
.000473	.001201	.000573	.001445	.000673	.001689	.000773	.001934
.000474	.001205	.000574	.001449	.000674	.001693	.000774	.001937
.000475	.001209	.000575	.001453	.000675	.001697	.000775	.001941
.000476	.001213	.000576	.001457	.000676	.001701	.000776	.001945
.000477	.001216	.000577	.001461	.000677	.001705	.000777	.001949

## APPENDIX A. CONVERSION TABLES (Cont'd)

Table of Powers of Two

$2^n$	$n$	$2^{-n}$
1	0	1.0
2	1	0.5
4	2	0.25
8	3	0.125
16	4	0.0625
32	5	0.03125
64	6	0.015625
128	7	0.0078125
256	8	0.00390625
512	9	0.001953125
1024	10	0.0009765625
2048	11	0.00048828125
4096	12	0.000244140625
8192	13	0.0001220703125
16384	14	0.00006103515625
32768	15	0.000030517578125
65536	16	0.0000152587890625
131072	17	0.00000762939453125
262144	18	0.000003814697265625
524288	19	0.0000019073486328125
1048576	20	0.00000095367431640625
2097152	21	0.000000476837158203125
4194304	22	0.0000002384185791015625
8388608	23	0.00000011920928955078125
16777216	24	0.000000059604644775390625
33554432	25	0.0000000298023223876953125
67108864	26	0.00000001490116119384765625
134217728	27	0.000000007450580596923828125
268435456	28	0.0000000037252902984619140625
536870912	29	0.00000000186264514923095703125
1073741824	30	0.000000000931322574615478515625
2147483648	31	0.0000000004656612873077392578125
4294967296	32	0.00000000023283064365386962890625
8589934592	33	0.000000000116415321826934814453125
17179869184	34	0.0000000000582076609134674072265625
34359738368	35	0.00000000002910383045673370361328125
68719476736	36	0.000000000014551915228366851806640625
137438953472	37	0.0000000000072759576141834259033203125
274877906944	38	0.00000000000363797880709171295166015625
549755813888	39	0.000000000001818989403545856475830078125
1099511627776	40	0.0000000000009094947017729282379150390625
2199023255552	41	0.00000000000045474735088646411895751953125
4398046511104	42	0.000000000000227373675443232059478759765625
8796093022208	43	0.0000000000001136868377216160297393798828125
17592186044416	44	0.00000000000005684341886080801486968994140625
35184372088832	45	0.000000000000028421709430404007434844970703125
70368744177664	46	0.0000000000000142108547152020037174224853515625
140737488355328	47	0.00000000000000710542735760100185871124267578125
281474976710656	48	0.000000000000003552713678800500929355621337890625