IS #1

The binary number system is the most important one in digital systems, but several others also are important. The decimal system is important because it is universally used to represent quantities outside a digital system. This means that there will be situations where decimal values have to be converted to binary values before they are entered into the digital system. For example, when you punch a decimal number into your hand calculator (or computer), the circuitry inside the device converts the decimal number to a binary value.

Likewise, there will be situations where the binary values at the outputs of a digital circuit have to be converted to decimal values for presentation to the outside world. For example, your calculator (or computer) uses binary numbers to calculate answers to a problem, then converts the answers to a decimal value before displaying them.

In addition to binary and decimal, two other number systems find widespread applications in digital systems. The *octal (base-8)* and *hexadecimal (base-16)* number systems are both used for the same purpose - to provide an efficient means for representing large binary numbers. As we shall see, both these number systems have the advantage that they can be easily converted to and from binary.

In a digital system, three or four of these number systems may be in use at the same time, so that an understanding of the system operation requires the ability to convert from one number system to another.

BINARY-TO-DECIMAL CONVERSIONS

The binary number system is a positional system where each binary digit (bit) carries a certain weight based on its position relative to the binary point. Any binary number can be converted to its decimal equivalent simply by summing together the weights of the various positions in the binary number which contain a 1.

Example:

$$\begin{array}{ccccccccccc} 1 & & 1 & & 0 & & 1 & & 1 & & \textit{(binary)} \\ 2^4 & + & 2^3 & + & 0 & + & 2^1 & + & 2^0 & = 16 + 8 + 2 + 1 & \\ & & & & & & & & & = 27_{10} & \textit{(decimal)} \end{array}$$

Another way to organize problem:

16	8	4	2	1	Weights
1	1	0	1	1	Bits

$16 + 8 + 2 + 1 = 27$

IS #1

Another example with a greater number of bits.

$$1\quad 0\quad 1\quad 1\quad 0\quad 1\quad 0\quad 1_2 =$$
$$2^7 + 2^5 + 2^4 + 2^2 + 2^0 = 181_{10}$$

Note that the procedure is to find the weights (i.e. powers of 2) for each bit position that contains a 1, then add them up.

For binary numbers, the rightmost digit is referred to as the least significant digit, or ***LSD***, because its positional value, or weight, is the lowest. The left-most digit is the most significant digit, or ***MSD***, because its positional value, or weight, is the highest.

Typical binary numbers are often written in groups of four or eight digits. Examples are 1001 and 10010110. Each digit, either 0 or 1, is referred to as a ***bit.*** A string of four bits is called a ***nibble,*** and eight bits make a ***byte.*** 1001 is a nibble, and 10010110 is a byte.

EXERCISES:

1. Convert the binary number 10011 to its decimal equivalent.

---

2. Convert the binary number 1110011 to its decimal equivalent.

3. What is the weight of the MSB in the binary number 1000011?

4. What is the weight of the MSB in a 10-bit binary number?

1 2 4 8 16 32 64 128 256 512
1 2 3 4 5 6 7 8 9 10

5. Find the decimal equivalent of these binary numbers:

1 = 1	11111 = 16+15 = 31
11 = 2+1 = 3	111111 = 32+31 = 63
111 = 4+2+1 = 7	1111111 = 64+63 = 127
1111 = 8+4+2+1 = 15	11111111 = 128+127 = 255

KEY: 1. 19 2. 115 3. 64 4. 512

5. 1, 3, 7, 15, 31, 63, 127, 255

IS #1

DECIMAL-TO-BINARY CONVERSIONS

A method to convert a number written in decimal to a number in binary is to use the powers of 2 chart. Use of the chart enables you to find the powers of 2 contained in the decimal number.

For example, convert the decimal number 153 to binary. The largest power of 2 in 153 is 128, 2^7. Subtract this factor from the number and find the power of 2 contained in the difference. 153 - 128 = 25. 25 contains the factor 16, 2^4 and 25 - 16 = 9. 9 contains the factor 8, 2^3 and 9 - 8 = 1. 1 is 2^0. So the binary number contains 2^7, 2^4, 2^3, 2^0. Starting from the most significant place value, write the binary number with a digit of 1 for each place with a factor and a 0 to hold the other places. The final answer: $10011001_2 = 153_{10}$.

```
  153
- 128
   25
-  16
    9
-   8
    1
```

128	64	32	16	8	4	2	1
1	0	0	1	1	0	0	1

EXERCISE: Convert the decimal number 86 to a binary number.

EXERCISE: Convert the decimal number 301 to a binary number.

1. 1010110 2. 100101101

Alphanumeric data is data consisting of letters or numbers or both, such as words, addresses, equations, and tables. Alphanumeric characters that are entered into a computer from a keyboard are converted into unique binary patterns representing each keyboard character. Several binary codes have been used throughout the development of computers. One code is the ASCII (pronounced "AS-KEY"), which stands for American Standard Code for Information Interchange. The computer code ASCII uses three zone bits (more significant bits) and four numeric bits (least significant bits), to code a character.

Zone bits Numeric bits

For example, in ASCII, E is coded as 100 0101. The zone bits are 0100 and the numeric bits are 0101. A code of n bits contains 2^n combinations of bits. An eight-bit code such as ASCII can therefore accommodate $2^8 = 256$ characters. This includes the 10 digits, 26 letters, and characters such as *, #, % etc.

Partial Listing of ASCII Code
American Standard Code for Information Interchange

		Zone Bits		
Hex	Binary	011	100	101
0	0000	0	@	P
1	0001	1	A	Q
2	0010	2	B	R
3	0011	3	C	S
4	0100	4	D	T
5	0101	5	E	U
6	0110	6	F	V
7	0111	7	G	W
8	1000	8	H	X
9	1001	9	I	Y
A	1010	:	J	Z
B	1011	;	K	[
C	1100	<	L	\
D	1101	=	M	]
E	1110	>	N	^
F	1111	?	O	_

Numeric Bits

IS #2

DECIMAL-TO-BINARY CONVERSION BY REMAINDER METHOD

A number expressed in the decimal system can be converted to binary by successive divisions by 2. The remainder of each division is retained as a bit of the binary number, with the first remainder as the least significant bit.

Example: Convert the decimal number 39 to binary by the Remainder Method.

$\frac{39}{2}$ = 19 and remainder of 1

$\frac{19}{2}$ = 9 and remainder of 1

$\frac{9}{2}$ = 4 and remainder of 1

$\frac{4}{2}$ = 2 and remainder of 0

$\frac{2}{2}$ = 1 and remainder of 0

$\frac{1}{2}$ = 0 and remainder of 1

Therefore, $39_{10} = 100111_2$.

EXERCISE:

Using the Remainder Method, convert the following decimal system numbers to their binary equivalents.

1._____ 25

2._____ 54

3._____ 432

KEY:
1. 11001
2. 110110
3. 110110000

IS #2

BINARY-TO-DECIMAL CONVERSION BY DOUBLE-DABBLE METHOD

A method of converting binary numbers to decimal equivalents (referred to as the double-dabble or double-dibble method) is performed as follows.

Write a 1 over the 1 farthest left in the binary number to be converted. Moving to the right, write a number over each bit according to this rule: If writing over a 0 bit, double the preceding number. If writing over a 1 bit, double the preceding number and add 1. The number written over the bit farthest right is the decimal equivalent being sought.

EXAMPLE: **Convert the binary number 1000111 to decimal.**

1	**2**	**4**	**8**	**17**	**35**	**71**
1	**0**	**0**	**0**	**1**	**1**	**1**

Therefore, $1000111_2 = 71_{10}$

EXERCISE:

Using the double-dabble method, convert the following binary numbers to decimal equivalents.

1._____ 100001

2._____ 1100011

3._____ 101010101

4._____ 11110000

KEY:
1. 33
2. 99
3. 341
4. 240

IS #2

BINARY FRACTIONS

The value of a number is determined by the digits in the number, the base of the number system, and the positions of the digits.

EXAMPLE: Binary 100 means "four".
Decimal 100 means "one hundred"

So far, only integers have been considered; but the fractional portion of a number can also be written in terms of the base raised to a power. Only here, the exponents are negative. Thus, we have the following fractions:

10^3	10^2	10^1	10^0	.	10^{-1}	10^{-2}	10^{-3}
1000	100	10	1	.	1/10	1/100	1/1000

2^3	2^2	2^1	2^0	.	2^{-1}	2^{-2}	2^{-3}
8	4	2	1	.	½	¼	⅛

EXERCISE:

Name the value of the bits in the following binary numbers.

.1 _____ .01 _____ .001 _____
.0001 _____ .00001 _____

KEY:

1. 1/2
2. 1/4
3. 1/8
4. 1/16
5. 1/32

IS #2

CONVERSION OF FRACTIONAL BINARY NOTATION TO DECIMAL SYSTEM

Two methods of converting fractional binary fractions to decimal equivalents are described below.

Method 1:

A fractional binary number can be converted by adding the weights corresponding to the 1's in the binary number.

These values or weights are listed below:

2^{-1}	=	1/2	2^{-4}	=	1/16
2^{-2}	=	1/4	2^{-5}	=	1/32
2^{-3}	=	1/8		etc.	

Example: Convert the binary number .1101 to decimal.

$$.1101_2 = 1(2^{-1}) + 1(2^{-2}) + 0(2^{-3}) + 1(2^{-4})$$

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

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

$$= \frac{13}{16} = .8125_{10}$$

Method 2:

Move the binary point to the right of the least significant bit. Convert the resulting number to decimal notation and then divide by 2^x, where "x" is the number of places the binary point was moved.

Example: Convert the binary number .1101 to decimal.

$$.1101_2 = \frac{1101_2}{2^4} = \frac{13}{2^4} = \frac{13}{16} = .8125_{10}$$

IS #2

Exercises:

Convert the following binary numbers to decimal.

1._____ .101

2._____ .001

3._____ .1100

4._____ .0101

5._____ .00001

6._____ .10101

KEY:

1. 5/8 = .625
2. 1/8 = .125
3. 3/4 = .750
4. 5/16 = .3125
5. 1/32 = .03125
6. 21/32 = .65625

IS #2

CONVERSION OF FRACTIONAL DECIMAL NOTATION TO BINARY SYSTEM

Fractional quantities expressed in the decimal system can be converted to the binary system by repeated multiplication by 2. In the result of each multiplication, the digits to the right of the decimal point are used for the next multiplication. The digit to the left of the decimal point is retained as one of the bits of the binary number being sought, the first bit so obtained being the most significant bit.

Example: Convert the decimal number 0.625 to binary.

.625 x 2 = 1.250
.250 x 2 = 0.500
.500 x 2 = 1.000

Therefore, $0.625_{10} = .101_{2}$.

Exercise:

Using the method of repeated multiplication by 2, convert the following decimal system quantities to their binary equivalents.

1._____ .125

2._____ .75

3._____ .3125

4._____ .0625

5._____ .8125

KEY:

1. .001
2. .11
3. .0101
4. .0001
5. .1101

IS #2

PARITY METHOD FOR ERROR DETECTION

The process of transferring data is subject to error, although modern equipment has been designed to reduce the probability of error. However, even relatively infrequent errors can cause useless results, so it is desirable to detect errors whenever possible. A widely used method for detecting errors is the "parity method".

A parity bit is an extra bit that is attached to a code group that is being transferred from one location to another. The parity bit is made either 0 or 1, depending on the number of 1's that are contained in the code group. Two different methods are used.

In the even parity method, the value of the parity bit is chosen so that the total number of 1's in the group (including the parity bit) is an even number. For example, suppose that the code group is 0100 1001. This is the ASCII character I. The group has three 1's. Therefore, a parity bit of 1 is added to make the total number of 1's an even number. The new code group, including the parity bit, becomes: 1 0100 1001.

If the code group contains an even number of 1's to start with, the parity bit is given a value of 0. For example, if the group were 0100 1110, the ASCII code for N, the parity bit would be 0, so the new code would be 0 0100 1110, which has four 1's, which is an even number of 1's.

The odd parity method is used in the same way except that the parity bit is chosen so the total number of 1's, including the parity bit, is an odd number.

The parity bit is used to detect any single bit errors occurring in the transmission of data. For example, if the letter C is being transmitted and odd parity is being used, the transmitted code would be: 0 0100 0011. The receiver will check to see that the code contains an odd number of bits. If the receiver receives the following code: 0 0100 0010, it will find that there is not an odd number of 1's, and an error will be detected, and the data can be transmitted again. The receiver can not identify which bit is in error, for it does not know what the code is supposed to be.

This parity method will not work if two bits are in error, because two errors would not change the "oddness" or "evenness" of the number of 1's in the code. In practice, this parity method is used in situations where the probability of a single error is low and the probability of double errors is practically zero.

Dunwoody

IS #3

BINARY ARITHMETIC

Addition of Binary Numbers

Because the binary system employs only two symbols, addition is a simple process. Several examples are shown below.

$$\begin{array}{r} 0 \\ +\,0 \\ \hline 0 \end{array} \qquad \begin{array}{r} 0 \\ +\,1 \\ \hline 1 \end{array} \qquad \begin{array}{r} 1 \\ +\,1 \\ \hline 10 \end{array} \qquad \begin{array}{r} 1 \\ 1 \\ +\,1 \\ \hline 11 \end{array}$$

Note that 1 plus 1 yields a sum of 0 and a carry of 1 into the next column, producing an answer of 10 (this a binary 10, not a decimal 10). Similarly, 1 plus 1 plus 1 yields a sum of 1 and a carry of 1 into the next column.

Binary numbers, like decimal numbers, are added column by column from right to left. The radix point in binary arithmetic is handled the same way as it is handled in decimal arithmetic. Addition requires that the radix point be aligned.

The most likely mistake that can be made in binary addition is losing track of the carries. Carefully indicating each carry as it is propagated will lessen the possibility of such an error.

Example:

$$\begin{array}{rcccccccc} & & 1 & 1 & & 1 & & & \\ & & 1 & 1 & 1 & 0 & 1 & 1 & 0 \\ + & & & 1 & 1 & 0 & 1 & 0 & 1 \\ \hline & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \end{array}$$

Exercise: **Perform the following binary additions.**

1.

$$\begin{array}{rcccc} & 1 & 0 & 1 & 0 \\ + & & 1 & 0 & 1 \\ \hline & 1 & 1 & 1 & 1 \end{array}$$

2.

$$\begin{array}{rcccccc} & 1 & 1 & 1 & 1 & 1 & 1 \\ + & & & & & & 1 \\ \hline 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}$$

3.

$$\begin{array}{rcccccccc} & 1 & 0 & 1 & 1 & 0 & 0 & 1 \\ + & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ \hline 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \end{array}$$

4.

$$\begin{array}{rccccc} & 1 & 1 & 0. & 1 & 1 \\ + & & & 1. & 1 & 1 \\ \hline 1 & 0 & 0 & 0. & 1 & 0 \end{array}$$

Key:

1. 1111

2. 1000000

3. 11011000

4. 1000.10

IS #3

BINARY SUBTRACTION BY DIRECT METHOD

Because the binary system employs only two symbols, there are only four basic subtractions.

0	1	1	10
-0	-1	-0	-1
0	0	1	1

Note that a *borrow* is required in the case "0 - 1". This borrow is obtained from the next higher-order column. Since a borrow brings a 1 into the column,

$0 - 1$ becomes $10_2 - 1_2$ or $2 - 1 = 1$.

Binary numbers are subtracted column by column, progressing from right to left. The operation of subtraction requires that the binary points, or radix points, be aligned.

Examples:

1 1 0 1	0 2 1 0 1	0 1 1 2 1 0 0 0 1
- 0 1 0 0	- 0 1 1	- 1 0
1 0 0 1	0 1 0	0 1 1 1 1

Exercise:

1. 1 1 1 0 − 1 0 1 0

2. 1 1 1 − 1

3. 1 1 0 1 1 − 1 0 1 1 1

4. 1 0 1 0 1 − 1 1 0

5. 1 0 0 0 0 − 1

6. 1 0 1 1. 1 1 0 − 1 0 0 0. 0 1 1

Key: 1. 1 0 0 2. 1 1 0 3. 1 0 0

4. 1 1 1 1 5. 1 1 1 1 6. 1 1. 0 1 1

IS #3 **BINARY MULTIPLICATION AND DIVISION**

The procedure for multiplying binary is essentially the same as that for decimal multiplication. Each bit of the multiplier, starting with the least significant bit (bit farthest to the right), is multiplied by the multiplicand. The partial products then are added to obtain the final product.

There are only four possible multiplication facts to know.

0	0	1	1
x0	x1	x0	x1
0	0	0	1

Example:

```
    1 0 0 1
  x   1 0 1
  -------
    1 0 0 1
  0 0 0 0
1 0 0 1
---------
1 0 1 1 0 1
```

The procedure for dividing binary numbers is also essentially the same as the decimal system procedure.

Example:

```
                1 1 1
      ---------------
1 1 0 / 1 0 1 0 1 0
        1 1 0
        -----
          1 0 0 1
          1 1 0
          -----
            1 1 0
            1 1 0
            -----
                0
```

Exercise:

1.
```
    1 1 1
  x 1 0 1
  -------
```

2.
```
    1 1.0 1
  x 1 0.1
  -------
```

3. $100\overline{)10100}$

4. $10.01\overline{)1101.1}$

Key:

1. 1 0 0 0 1 1
2. 1 0 0 0. 0 0 1
3. 1 0 1
4. 1 1 0

THE BCD CODE

If each digit of a decimal number is represented by its binary equivalent, this produces a code called binary-coded-decimal, abbreviated BCD. Since a decimal digit can be as large as 9, 4 bits are required to code each digit.

Take a decimal number such as 563. Each digit is changed to its binary equivalent as follows:

5	6	3 (decimal)
0101	0110	0011 (BCD)

Each decimal digit is changed to its straight binary equivalent. Note that 4 bits are always used for each digit. The BCD code represents each digit of the decimal number by a 4-bit binary number. The BCD code does not use the numbers 1010, 1011, 1100, 1101, 1110, and 1111. Only 10 of the possible 16 4-bit binary code groups are used.

Example: Convert the BCD number 100100010110 to decimal. Divide the BCD number into 4-bit groups and convert each to decimal.

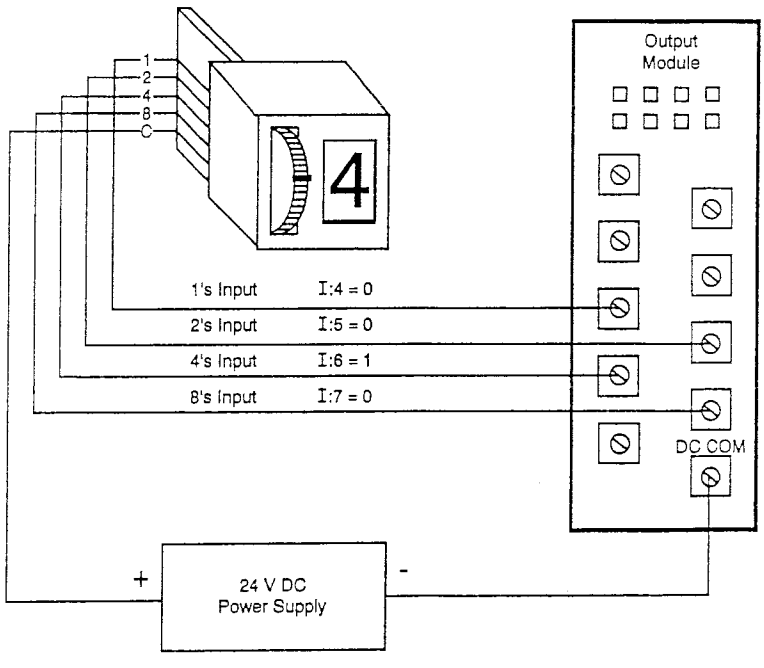
1001	0001	0110
9	1	6

The BCD system is not another number system like binary, octal, hexadecimal. A BCD number is not the same as a straight binary number.

BCD NUMERICAL REPRESENTATIONS

Decimal	Binary	BCD
0	0	0000
1	1	0001
2	10	0010
3	11	0011
4	100	0100
5	101	0101
6	110	0110
7	111	0111
8	1000	1000
9	1001	1001

Typical input applications for PLCs include the entry of decimal data such as count, volume, and weight. An example of a traditional input device is the BCD thumbwheel switch. Each *single*-digit thumbwheel has a internal wheel with the numbers 0 through 9. Internal to the thumbwheel is a series of four switches, one each for the 8s, 4s, 2s and 1s binary positions. Each switch is connected as an input to a PLC input module.



IS #4

COMPLEMENT ARITHMETIC

In the 2s complement number system, positive and negative numbers can be expressed. In this system the MSB of a 2s complement number denotes the sign. 0 means the number is positive. 1 means the number is negative. In 2s complement notation, positive numbers are represented as simple binary numbers with the restriction that the MSB is 0. Negative numbers are some what different. To obtain the representation of a negative number, use the following rule:

1. Represent the number as a positive binary number.
2. Complement the number. Change the 1's to 0's, and the 0's to 1's.
3. Add 1.
4. Ignore any carries out of the MSB.

Ex. Given 8-bit words, find the 2s complement of

a) 25 00011001

b) -25 complement +25 11100110

add 1 + 1

-25 = 11100111

Note the MSB is 1.

To determine the magnitude of an unknown negative number, take its 2s complement. The result is a positive number whose magnitude equals that of the original number.

Ex. What decimal number does 11110100 represent?

This number must be negative because its MSB is 1. Its positive equivalent is obtained by 2s complementing the given number.

Given number	11110100
Complement	00001011
Add 1	+ 1
	00001100

This is the equivalent of +12. Therefore 11110100 = -12

IS #4

ADDING 2S COMPLEMENT NUMBERS

If an addition operation is to be performed, the numbers are added regardless of their sign. The answer is in 2s complement form with the correct sign. Any carries out of the MSB are meaningless and should be ignored.

Ex. Add 19 and -11.

19 is simply 00010011.
-11 (by complementing) is 11110101

```
          00010011
        + 11110101
    Sum   00001000   = +8
```

Note that there is a carry out of the MSB that is ignored.

Ex. Add -19 and -11

```
     -19  11101101          11100010
  +  -11  11110101          00011101
     -30  11100010        +        1
                            00011110   = 30
```

SUBTRACTION OF BINARY NUMBERS

The 2s complement of the subtrahend (number subtracted) is taken and added to the minuend. This is subtraction by changing the sign and adding. As in addition, the signs of the operands and carries out of the MSB are ignored.

Ex. 53 - 30 = 23

```
            53         00110101
  2s comp of 30     +  11100010
                       00010111   =   23
```

Ex. 15 - 8 = 7

```
            15         00001111
  2s comp of 8      +  11111000
                       00000111   =    7
```

IS #5

THE OCTAL NUMBER SYSTEM

The octal number system is very important in digital computer work. The octal number system has a base of eight, meaning that it has eight possible digits: 0, 1, 2, 3, 4, 5, 6, and 7. Thus, each digit of an octal number can have any value from 0 to 7. The digit positions in an octal number have weights as follows:

8^4	8^3	8^2	8^1	8^0	8^{-1}	8^{-2}	8^{-3}	8^{-4}	8^{-5}

octal point

Example of an octal number:

	512s	64s	Eights	Ones
Number	1	4	2	3_8
Weights	8^3	8^2	8^1	8^0

OCTAL TO DECIMAL CONVERSION

An octal number can be easily converted to its decimal equivalent by multiplying each octal digit by its positional weight.

Examples:

$$372_8 = 3 \times (8^2) + 7 \times (8^1) + 2 \times (8^0)$$
$$= 3 \times 64 + 7 \times 8 + 2 \times 1$$
$$= 250_{10}$$

$$24.6_8 = 2 \times (8^1) + 4 \times (8^0) + 6 \times (8^{-1})$$
$$= 20.75^{10}$$

DECIMAL TO OCTAL CONVERSION

The methods for converting a decimal number to its octal equivalent are the same as those used to convert from decimal to binary. One method is using the definition of being an octal number and determining the number of times a power of 8 can be divided into the decimal number and what the remainder would be. This process would continue until a value less than 8 is used for the units column.

Example: Convert the decimal number 266 to an octal number.

$$64\overline{)266} = 4,\quad 266 - 256 = 10 \qquad 8\overline{)10} = 1,\quad 10 - 8 = 2 \qquad 1\overline{)2} = 2$$

64	8	1
4	1	2

Remainder Method:

To convert a decimal integer to octal, progressively divide the decimal number by 8, writing down the remainders after each division. The remainder represent the digits of the octal number, the first remainder being the LSD.

Example:

$$\frac{266}{8} = 33 + \text{remainder of } 2$$

$$\frac{33}{8} = 4 + \text{remainder of } 1$$

$$\frac{4}{8} = 0 + \text{remainder of } 4$$

$$266_{10} = \boxed{412_8}$$

IS #5 **HEXADECIMAL NUMBERS**

The hexadecimal system uses base 16. Thus, it has 16 possible digit symbols. It uses the digits 0 through 9 plus the letters A, B, C, D, E, and F as the 16 digit symbols. The table shows the relationship between the decimal and hexadecimal symbols.

Decimal:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
Hexidec:	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10	11	12

Hexadecimal to Decimal Conversion:

To convert from a hexadecimal number to its equivalent decimal number remember that the various digit positions in a hex number have weights that are powers of 16.

Example:

$$356_{16} = 3 \times 16^2 + 5 \times 16^1 + 6 \times 16^0$$
$$= 768 + 80 + 6$$
$$= 854_{10}$$

$$2AF_{16} = 2 \times 16^2 + 10 \times 16^1 + 15 \times 16^0$$
$$= 512 + 160 + 15$$
$$= 687_{10}$$

Note that in the second example the value 10 was substituted for A and the value 15 for F in the conversion to decimal.

Decimal to Hexadecimal Conversion:

Example: Convert the decimal number 423 to hexadecimal.

```
       1            10      7
    ____          ____    __
 256/423        16/167   1/7
     256           160
     167             7
```

256	16	1
1	A	7

Example: Convert the decimal number 423 to hexadecimal using the Remainder Method.

$$\frac{423}{16} = 26 + R7$$

$$\frac{26}{16} = 1 + R10 = A$$

$$\frac{1}{16} = 0 + R1$$

IS #5

BINARY-OCTAL-HEXADECIMAL CONVERSIONS

The ease with which conversions can be made between octal, hexadecimal and binary numbers make the octal and hexadecimal numbers attractive as a "shorthand" means of expressing large binary numbers. In computer work, binary numbers with many bits are not uncommon. These binary numbers do not always represent a numerical quantity but are often some type of code which conveys nonnumerical information. In computers, binary numbers might represent 1) actual numerical data; 2) numbers corresponding to a location (address) in memory; 3) an instruction code; 4) a code representing alphabetic and other nonnumerical characters; or 5) a group of bits representing the status of devices internal or external to the computer. When dealing with a large quantity of binary numbers of many bits, it is convenient and efficient to write the numbers in octal or hexadecimal rather than binary. However, digital circuits and systems work strictly in binary. Octal and hexadecimal are used as a convenience for the operators of the system.

OCTAL-BINARY CONVERSION BY SUBSTITUTION

A method of converting an octal number to its binary equivalent is to substitute a group of three binary bits for each octal digit. These substitutions are mode according to the table shown below.

Octal Digit	0	1	2	3	4	5	6	7
Binary Equivalent	000	001	010	011	100	101	110	111

Using these conversions, any octal number is converted to binary by individually converting each digit. For example, the octal number 472 can be converted to binary as follows:

4	7	2
↓	↓	↓
100	111	010

Octal 472 is equivalent to binary 100111010.

Example: Convert octal 54.31 to binary.

5	4	.	3	1
↓	↓	↓	↓	↓
101	100	.	011	001

IS #5

Converting from binary to octal is simply the reverse of the foregoing process. The binary digits are grouped into groups of three on each side of the binary point with zeros added on either side if needed to complete a group of three. Then each group of three bits is converted in its octal equivalent.

Example: Convert binary 11010.1011 to octal.

$$\underbrace{011}_{3}\ \underbrace{010}_{2}\ .\ \underbrace{101}_{5}\ \underbrace{100}_{4}$$

Note the 0's were added on each end to complete the groups of three.

Here are several more examples.

$$10110_2 = 26_8$$

$$10011.011 = 23.3_8$$

$$111011.1111 = 73.74_8$$

HEXADECIMAL-BINARY CONVERSIONS

The table on the right shows the relationships among hexadecimal, decimal, and binary numbers. Note that each hexadecimal digit represents a group of four binary digits. The conversions between hexadecimal and binary are done in exactly the same manner as octal and binary except that groups of four bits are used.

Hexadecimal	Decimal	Binary
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111

Example:

$$1\ 1\ 1\ 0\ 1\ 0\ 0\ 1\ 1\ 0_2 = \underbrace{0\ 0\ 1\ 1}_{3}\ \underbrace{1\ 0\ 1\ 0}_{A}\ \underbrace{0\ 1\ 1\ 0}_{6}$$

$$= 3\ A\ 6_{16}$$

$$9\ F\ 2_{16} = \underset{\downarrow}{9}\quad \underset{\downarrow}{F}\quad \underset{\downarrow}{2}$$

$$1\ 0\ 0\ 1\ \ 1\ 1\ 1\ 1\ \ 0\ 0\ 1\ 0$$

$$= 100111110010_2$$

IS #5

HEXADECIMAL-OCTAL CONVERSIONS

A conversion from an octal number to a hexadecimal number can be done through a binary number.

Example: Convert the octal 346 to hexadecimal.

346 = 011 100 110 = 1110 0110 = E6
(oct) (binary) (binary) (hex)

Example: Convert the hexadecimal A7.41 to octal.

A7.41 = 1010 0111 . 0100 0001 = 010 100 111 . 010 000 010 = 247.202
(hex) (binary) (binary) (oct)

Exercise:

1. Convert binary 1111.111 to octal.

2. Convert binary 1111.111 to hexadecimal.

3. Convert octal 371.4 to binary.

4. Convert hexadecimal 7C2.6 to binary.

Answers:
1. 17.7 (8) 2. F.E (16) 3. 011 111 001.100 (2)
4. 0111 1100 0010.0100 (2)

IS #6

Converting between Decimal, Octal and Hexadecimal Systems Using the Binary Number System

Ex. 1
Convert 27_{10} to an octal number.

$27_{10} = 11011_2 = 011\ 011_2 = 33_8$

Check: $33_8 = 3 \times 8 + 3 \times 1 = 24 + 3 = 27$

Ex. 2
Convert 70.5_{10} to an octal number.

$70.5_{10} = 1000110.1_2 = 001\ 000\ 110.\ 100_2 = 106.4_8$

Check: $106.4_8 = 1 \times 64 + 0 \times 8 + 6 \times 1 + 4 \times .125 = 70.5_{10}$

Ex. 3
Convert 35_8 to a decimal number.

$35_8 = 011\ 101_2 = 16 + 8 + 4 + 1 = 29_{10}$

Check: $35_8 = 3 \times 8 + 5 \times 1 = 24 + 5 = 29_{10}$

Ex. 4
Convert 61.4_8 to a decimal number.

$61.4_8 = 110\ 001.\ 100_2 = 32 + 16 + 1 + .5 = 49.5_{10}$

Check: $61.4_8 = 6 \times 8 + 1 \times 1 + 4 \times \frac{1}{8} = 49.5_{10}$

IS #6

Ex. 5
Convert 43_{10} to a hexadecimal number.

$43_{10} = 101011_2 = 0010\ 1011_2 = 2B_{16}$

Check: $2B_{16} = 2 \times 16 + 11 \times 1 = 32 + 11 = 43_{10}$

Ex. 6
Convert 160.25_{10} to a hexadecimal number.

$160.25_{10} = 10100000.01_2 = 1010\ 0000.\ 0100_2 = A0.4_{16}$

Check: $A0.4_{16} = 10 \times 16 + 0 \times 1 + 4 \times 1/16 = 160¼ = 160.25_{10}$

Ex. 7
Convert $2C3_{16}$ to a decimal number.

$2C3_{16} = 0010\ 1100\ 0011_2 = 512 + 64 + 128 + 2 + 1 = 707_{10}$

Check: $2C3_{16} = 2 \times 256 + 12 \times 16 + 3 \times 1 = 512 + 192 + 3 = 707_{16}$

Ex. 8
Convert $76.C_{16}$ to a decimal number.

$76.C_{16} = 0111\ 0110.1100_2 = 64 + 32 + 16 + 4 + 2 + .5 + .25 = 118.75_{10}$

Check: $76.C_{16} = 7 \times 16 + 6 \times 1 + 12 \times 1/16 = 112 + 6 + ¾ = 118.75_{10}$

IS #7

Excess-3 Code

The excess-3 code is related to the BCD code and is sometimes used instead because it possesses advantages in certain operations. The excess-3 code for a decimal number is obtained in the same manner as BCD except that *3* is added to each decimal digit before encoding it in binary.

Example: Convert 52_{10} to its excess-3 representation.

$$\begin{array}{rr} 5 & 2 \\ +3 & +3 \\ \hline 8 & 5 \end{array}$$ add 3 to each digit

1000 0101 convert to 4-digit binary

$52_{10} = 1000\ 0101_{XS\text{-}3}$

Example: Convert this XS-3 representation to decimal.

1001 0100

$$\begin{array}{rr} 9 & 4 \\ -3 & -3 \\ \hline 6 & 1 \end{array}$$

$1001\ 0100_{XS\text{-}3} = 61_{10}$

This table lists the BCD and XS-3 code representations for the decimal digits. Notice that both codes use only 10 of the possible 4-bit code groups. The invalid code groups for XS-3 are: 0000, 0001, 0010, 1101, 1110, and 1111.

Decimal	BCD	Excess-3
0	0000	0011
1	0001	0100
2	0010	0101
3	0011	0110
4	0100	0111
5	0101	1000
6	0110	1001
7	0111	1010
8	1000	1011
9	1001	1100

IS #8

THE GRAY CODE

The Gray Code belongs to a class of codes called minimum exchange codes, in which only one bit in the code group changes when going from one step to the next. The Gray code is an unweighted code, meaning that the bit positions in the code groups do not have specific weight assigned to them. Because of this, the Gray code is not suited for arithmetic operations but finds application in input/output devices and some types of analog-to-digital convertors.

The table below shows the Gray-code representation for the decimal numbers 0 through 15, together with the straight binary code. Examining the Gray code groups for each decimal number, it can be seen that in going from any one decimal number to the next, only one bit of the Gray code changes. For example, in going from 3 to 4, the Gray code changes for 0010 to 0110, with only the second bit from the left changing. Going form 14 to 15 the Gray code bits change from 1001 to 1000, with only the last bit changing. This is the principal characteristic of the Gray code. Compare this with the binary code, where anywhere from one to all of the bits change in going from one step to the next.

Decimal	Binary Code	Gray Code
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100
8	1000	1100
9	1001	1101
10	1010	1111
11	1011	1110
12	1100	1010
13	1101	1011
14	1110	1001
15	1111	1000

The Gray code is often used in situations where other codes, such as binary, might produce erroneous or ambiguous results during those transitions in which more than one bit of the code is changing. For instance, using binary code and going from 0111 to 1000 requires that all four bits change simultaneously. Depending on the device or circuit that is generating the bits, there may be a significant difference in the transition times of the different bits. If so, the transition form 0111 to 1000 could produce one or more intermediate states. For example, if the most-significant bit changes faster than the rest, the following transitions will occur:

The occurrence of 1111 is only momentary but it could conceivably produce erroneous operation of the elements that are being controlled by the bits. Obviously, using the Gray code would eliminate this problem, since only one bit change occurs per transition and no "race" between bits can occur.

Converting from Binary to Gray Code

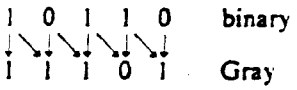
Any binary number can be converted to its Gray-code representation as follows:

1. The first bit of the Gray code is the same as the first bit of the binary number.

2. The second bit of the Gray code equals the exclusive-OR of the first and second bits of the binary number; that is, it will be 1 if these binary-code bits are different, 0 if they are the same.

3. The third Gray-code bit equals the exclusive-OR of the second and third bits of the binary number, and so on.

Example:
Convert the binary number 10110 to Gray Code.



Converting from Gray to Binary

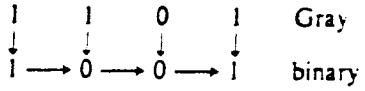
To convert from Gray to binary requires the opposite procedure to that given above.

1. The first binary bit is the same as the first Gray-code bit.

2. If the second Gray bit is 0, the second binary bit is the same as the first; if the second Gray bit is 1, the second binary bit it the inverse of the first binary bit.

3. Step 2 is repeated for each successive bit.

Example: Convert 1101 from Gray to binary:



The first Gray bit is a 1, so the first binary bit is written as a 1. The second Gray bit is a 1, so the second binary bit is made a 0 (inverse of the first binary bit). The third Gray bit is a 0, so the third binary bit is made a 0 (same as the second binary bit). The fourth Gray bit is 1, making the fourth binary bit a 1 (inverse of the third binary bit.)

IS #8

A method for converting from binary to Gray code is "the add no-carry" method. In this method, the first bit of the Gray code number is the same as the first bit of the binary number. The successive bits of the Gray code number are derived by adding (with no carry) successive bits of the binary number.

Table for "add no-carry"

$0 + 0 = 0$
$0 + 1 = 1$
$1 + 0 = 1$
$1 + 1 = 0$

Example: Convert 00010111_2 to Gray code.

00010111_2

$00011100_{\text{Gray code}}$

Copy MSB 0
$0 + 0 = 0$
$0 + 0 = 0$
$0 + 1 = 1$
$1 + 0 = 1$
$0 + 1 = 1$
$1 + 1 = 0$
$1 + 1 = 0$

To convert from Gray code to binary:
The first bit of the binary is the same as the first bit of the Gray code.
Successive bits of the binary are derived by add no-carry of the last binary bit and the next Gray code bit.

Example:
Convert $00011100_{\text{Gray code}}$ to binary.

00011100

00010111

Copy MSB 0
$0 + 0 = 0$
$0 + 0 = 0$
$1 + 0 = 1$
$1 + 1 = 0$
$1 + 0 = 1$
$0 + 1 = 1$
$0 + 1 = 1$

Arithmetic Review Sheet

1. A _____ or base ten system of numeration has 10 symbols to use as digits, and the place values are powers of _____.

2 A _____ or base two system of numeration has 2 symbols to use as digits, and the place values are powers of _____.

3. The word _____ is a contraction for "binary digit".

4. An _____ or base eight system of numeration has 8 symbols to use as digits, and the place values are powers of _____.

5. A _____ or base sixteen system of numeration has 16 symbols to use as digits, and the place values are powers of _____.

6. In base sixteen, the number twelve is represented by the digit _____.

7. Any octal digit can be represented by _____ binary digits.

8. Any hexadecimal digit can be represented by _____ binary digits.

Convert each number to the indicated base.

9. 463_8 = _____$_{10}$

10. 11000111.001_2 = _____$_{16}$

11. $2BC_{16}$ = _____$_{10}$

12. $1A7.E6_{16}$ = _____$_2$

13 211.75_{10} = _____$_8$

14. $D2.7C_{16}$ = _____$_8$

15. 211.75_{10} = _____$_{16}$

16. 111001.01_2 = _____$_{10}$

17. 11000111.001_2 = _____$_8$

18. 57.34_8 = _____$_2$

Perform each operation using binary arithmetic.

19. 110.101 + 10.11 = _____

20. 1101 + 101 + 111 = _____

21. 110.110 - 10.101 = _____

22. 101.11 x 1.01 = _____

23. 100011 ÷ 100 = _____

Arithmetic Review Sheet

Find the two's complement of each number.

24. 00101101

25. 11011011

26. Convert 78_{10} to a BCD number.

27. Convert 63_{10} to an Excess-3 number.

28. Convert 131_{10} to a Gray code number.

KEY

1. decimal, ten
2. binary, two
3. bit
4. octal, eight
5. hexadecimal, sixteen
6. C
7. Three
8. Four
9. 307_{10}
10. $C7.2_{16}$
11. 700_{10}
12. $0001\ 1010\ 0111.1110\ 0110_2$
13. 323.6_8
14. 322.37_8
15. $D3.C_{16}$
16. 57.25_{10}
17. 307.1_8
18. 101111.0111_2
19. 1001.011_2
20. 11001_2
21. 100.001_2
22. 111.0011_2
23. 1000.11_2
24. 11010011_2
25. 00100101_2
26. 0111 1000
27. 1001 0110
28. 11000010

IS #9

Boolean algebra differs in a major way from ordinary algebra in that Boolean constants and variables are allowed to have only two possible values, 0 or 1. A Boolean variable is a quantity that may, at different times, be equal to either 0 or 1. The Boolean variables can represent propositions.

A proposition is a declarative sentence that is either true or false.
False = 0 True = 1

The Boolean variables are often used to represent the voltage level present on a wire or at the input-output terminals of a circuit. For example, in a certain digital system the Boolean value ot 0 might be assigned to any voltage in the range from 0 to 0.8 V, while the Boolean value of 1 might be assigned to any voltage in the range 2 to 5 V. Thus, Boolean 0 and 1 do not represent actual numbers but instead represent the truth or falseness of a proposition, or the state of a voltage variable or what is called its *logic level.* A voltage in a digital circuit is said to be at the logic level 0 or the logic level 1, depending on its actual numerical value.

Logic 0	Logic 1
False	True
Off	On
Low	High
No	Yes
Open switch	Closed switch

Boolean algebra is used to express the effects that various digital circuits have on logic inputs, and to manipulate logic variables for the purpose of determining the best method for performing a given circuit function. Letter symbols will be used to represent logic variables. For example, A might represent a certain digital circuit input or output, and at any time either A = 0 or A = 1, if not one, then the other.

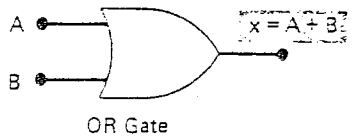
Because only two values are possible, Boolean algebra is relatively easy to work with as compared to ordinary algebra. In Boolean algebra there are no fractions, decimals, negative numbers, square roots, and so on. In fact, in Boolean algebra there are only three basic operations.

1. Logical addition, also called OR addition or simply the OR operation. The common symbol for this operation is the plus sign (+).
2. Logical multiplication, also called AND multiplication or simply the AND operation. The common notation for this operation is multiplication notation.

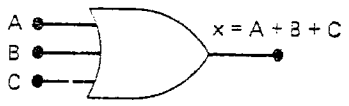
IS #9

3. Logical complementation or inversion, also called the NOT operation. The common symbol for this operation is the overbar ($\overline{\ }$).

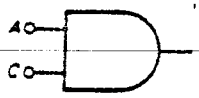
Circuit symbol for a two variable OR gate



Circuit symbol for a three variable OR gate

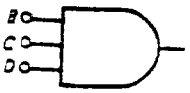


Circuit symbol for a two variable AND gate



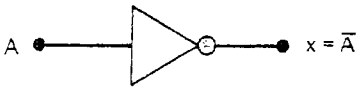
X = AC

Circuit symbol for a three variable AND gate



X = BCD

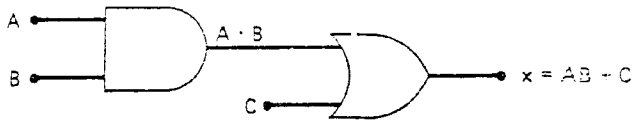
Circuit symbol for the NOT circuit



IS #9

Any logic circuit, no matter how complex, may be described using Boolean operations previously defined. The OR gate, AND gate, and NOT circuit are the basic building of digital systems.

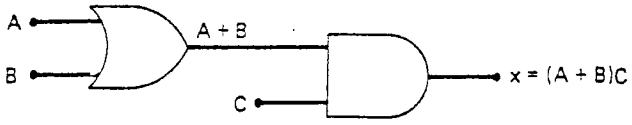
Logic circuit with its Boolean expression



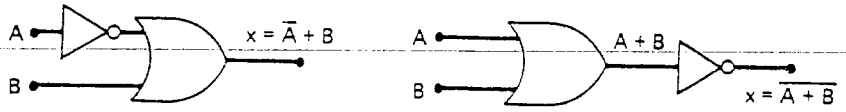
In Boolean algebra it will be understood that if an expression contains both AND or OR operations, the AND operations are performed first, unless there are parentheses in the expression. Then the operation inside the parentheses is performed first. This is the same rule that is used in ordinary algebra to determine the order of operations.

Ordinary algebra: Parentheses - Multiply - Add
Boolean algebra: Parentheses - AND - OR

Logic circuit whose expression requires parentheses

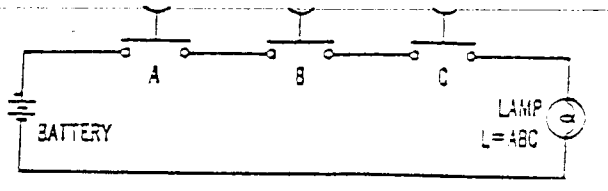


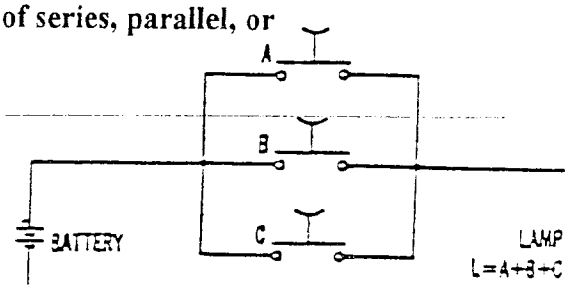
Circuits using INVERTERS



NOTE: $\overline{A+B} \neq \bar{A}+\bar{B}$ and $\overline{AB} \neq \bar{A}\bar{B}$

Another use for Boolean algebra is in the simplification of series, parallel, or combinations circuits.





Information Sheet for WS #9

Match the number of each figure to the number of the corresponding equation. Place answers in the table provided.

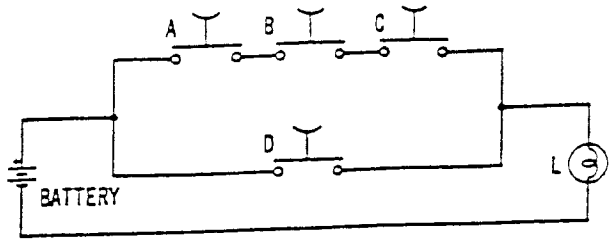


Fig. 1

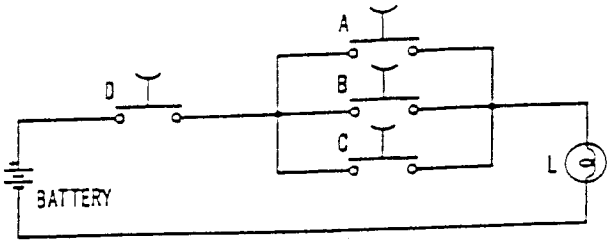


Fig. 2

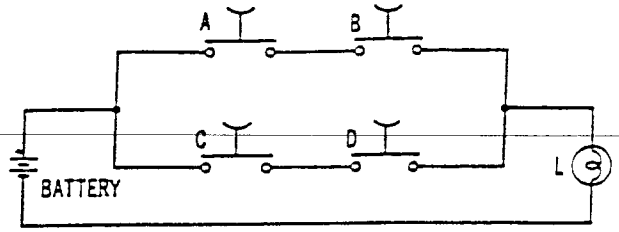


Fig. 3

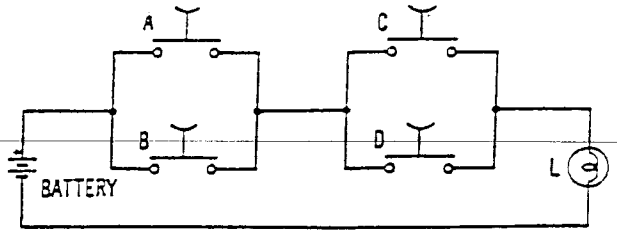


Fig. 4

Equations:

I. $L = D(A+B+C)$
II. $L = AB+CD$
III. $L = ABC+D$
IV. $L = (A+B)(C+D)$

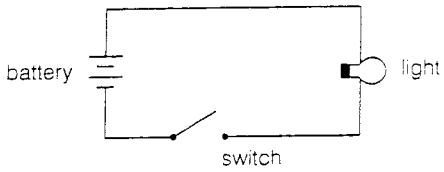
EQUATION	FIGURE
I	
II	
III	
IV	

IS #10

AND/OR LOGIC

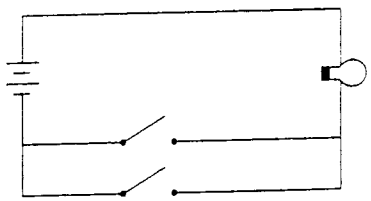
Boolean Algebra will study the logic properties of the circuits from which computers and all similar digital electronic devices are constructed. Early computers were constructed out of thousands of electric switches, or relays. When transistors and integrated circuitry became available, switches gave way to gated networks. The logical properties of switches and gates are very similar.

A switch turns the current in a circuit on and off. A switch is a binary device having *two* values: on and off.



Switch	Light
off	off
on	on

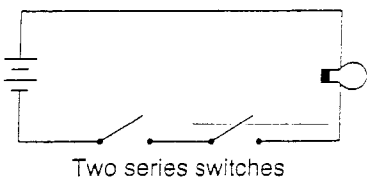
Often circuits have several switches to enable more complex control. In this diagram the light can be controlled by either one of two switches. Current will flow through either switch to the light.


Two parallel switches

Parallel switched circuit

Switch 1	Switch 2	Light
on	on	on
on	off	on
off	on	on
off	off	off

Here is a circuit example from the auto world. If a car has an automatic transmission, to start the car two conditions must be met. The car must be out of gear (in either neutral or park) *and* the key must be turned to "start". Switches that *both* must be *on* in order for the circuit to be on are said to be *in series*. No matter how many switches are linked in series, all must be on for the circuit to be on.


Two series switches

Series switched circuit

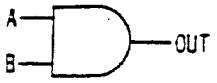
X	Y	Circuit
on	on	on
on	off	off
off	on	off
off	off	off

IS #10

AND/OR LOGIC

Digital computers employ high-speed electronic switches known as *logic or gate* circuits. In these circuits, two different voltage levels represent the binary symbols 0 and 1. One such circuit, the AND gate, is so designed that its output terminal will be at the voltage level which represents binary 1 when *all* of its input terminals are at the binary 1 level. If any input terminal is at the binary 0 level, the output will be binary 0. A block diagram of an AND gate is shown below with a table indicating the output for all combinations of input.

A	B	OUT
0	0	0
0	1	0
1	0	0
1	1	1



The AND gate represented above has two input terminals, A and B, but AND gates can be designed for a greater number of inputs. In every case, however, all inputs must be 1 to produce a 1 output.

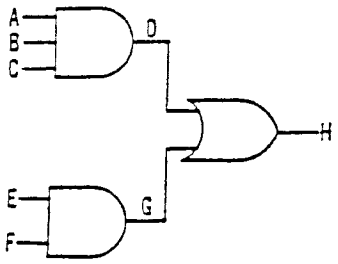
An OR gate is a circuit designed so that its output terminal will be at the voltage level representing binary 1 when *at least* one of its input terminals is at the binary 1 level.

A	B	OUT
0	0	0
0	1	1
1	0	1
1	1	1

IS #10

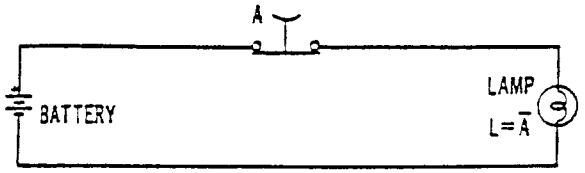
Assignment: Circle the Boolean equations which are correct for the logic diagram shown.

1. D = A + B + C
2. D = AB + C
3. D = ABC
4. G = E + F
5. G = EF
6. H = D + G
7. H = DG
8. H = D + EF
9. H = ABCEF
10. H = ABC + EF
11. H = EF + ABC

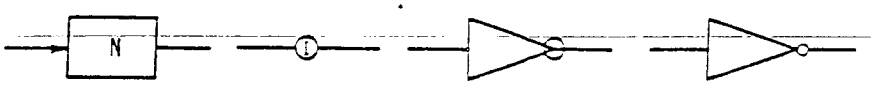


THE NOT CONCEPT

The NOT concept is illustrated by the electrical circuit below. Because switch A has a normally closed contact, actuating the switch will break the circuit and turn off the lamp. The lamp is on only when the switch is NOT actuated. This is expressed by the Boolean equation $L = \overline{A}$ (sometimes written L = A') which is read "L equals NOT A."



Electronic circuits which perform the NOT function are designed so that the output terminal is at the voltage level that represents binary 1 when the input terminal is at the binary 0 level. Also, the output level is binary 0 when the input level is binary 1. Such circuits are also known as *inverters*. Some commonly used schematic symbols are shown below.



IS #10

Assignment: Match the number of the equation to the number of the corresponding diagram.

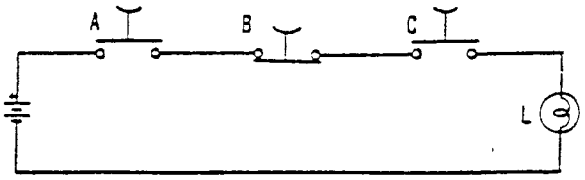


Fig. 1

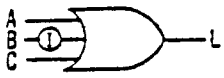


Fig. 2

Fig. 3

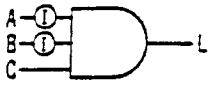


Fig. 4

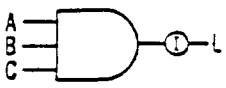


Fig. 5

Equations:

I. $L = \overline{A}\,\overline{B}C$

II. $L = \overline{A} + B + C$

III. $L = A\overline{B}C$

IV. $L = A + \overline{B} + C$

V. $L = \overline{ABC}$

EQUATION	FIGURE
I	
II	
III	
IV	
V	

IS #10

THE TRUTH TABLE

The variables of a Boolean expression can occur in either inverted or noninverted forms, e.g., $\bar{A}$ or A. The binary symbols 1 and 0 are employed to represent the two forms of the variable. Thus, if A = 1, then $\bar{A}$ = 0.

The *truth table* is a tabulation of the value of a Boolean expression for all possible combinations of its variables. A truth table for the expression AB + C is shown below. As there are three variables (A, B, and C) there are eight (2^3) possible combinations of these variables. The truth table therefore has eight rows. For each combination of truth values of variables, the value of the expression AB + C is given in the right-hand column. Note that the value of the expression is 1 whenever both A AND B are 1, or whenever C is 1.

TRUTH TABLE

A	B	C	AB + C
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

A	B	C	$A\bar{B}C$
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

Another truth table, for the expression $A\bar{B}C$, is shown. Note that the value of the expression is 1 only when A = 1 and B = 0 and C = 1.

IS #10

Examples:

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

A	B	C	$\bar{A}BC$
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

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

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

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

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

Key:

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

A	B	C	$\bar{A}BC$
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

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

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

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

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

IS #11

Theorems of Boolean algebra, numbered for reference, are shown below. Also shown are block diagrams (figures) numbered to correspond to the theorems. Theorem 2, for example, is clarified by Fig. 2. Because an OR circuit will produce a 1 output when *at least* one of its inputs is logical 1, the output of Fig. 2 always will be a 1 (whether terminal A is 1 or 0). This constant output is represented by the 1 in the theorem: $A + \overline{A} = 1$.

Fig. 5 clarifies theorem 5. Since a constant 1 is applied to one of the terminals of the AND circuit, the output always will be the same as the input to terminal A. Hence $A \cdot 1 = A$.

(1) $A + A = A$
(2) $A + \overline{A} = 1$
(3) $A + 1 = 1$
(4) $A + 0 = A$
(5) $A \cdot 1 = A$
(6) $A \cdot 0 = 0$
(7) $A \cdot A = A$
(8) $A \cdot \overline{A} = 0$
(9) $1 + 0 = 1$
(10) $1 + 1 = 1$
(11) $1 \cdot 0 = 0$
(12) $1 \cdot 1 = 1$

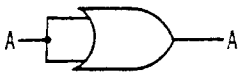


Fig. 1

Fig. 4

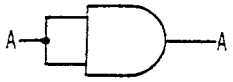


Fig. 7

Fig. 10

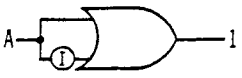


Fig. 2

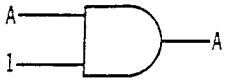


Fig. 5

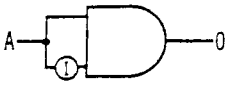


Fig. 8

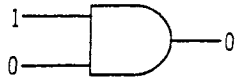


Fig. 11

Fig. 3

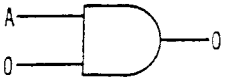


Fig. 6

Fig. 9

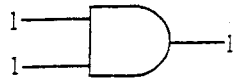


Fig. 12

The above theorems often permit simplification of Boolean expressions.

Example: $B + \overline{B} + 0 = 1 + 0$ (by theorem 2)
$= 1$ (by theorem 9)

1. $1 + 1 + 0$
2. $1 \cdot 1 \cdot A$
3. $M \cdot \overline{M} \cdot 1$
4. $X \cdot 0 + 1$
5. $C \cdot 1 + \overline{D}D$
6. $A + 0 + A + 0$
7. $A + B + 1$
8. $1(E + \overline{E})$
9. $H + H + H + \overline{H}$
10. $1 \cdot 0 \cdot A$

IS #11

Examples

1. $A(AB + C) =$
$AAB + AC =$
$AB + AC$

2. $A(AB + B) =$
$AAB + AB =$
$AB + AB =$
AB

3. $A(\bar{A}B + B) =$
$A\bar{A}B + AB =$
$0B + AB =$
$0 + AB =$
AB

4. $A + B + C + \bar{B} =$
$A + B + \bar{B} + C =$
$A + 1 + C =$
1

5. $A(\bar{B} + B) =$
$A1 =$
A

6. $AB + 1 =$
1

7. $A + B\bar{B} =$
$A + 0 =$
A

8. $(A + 0)(B + 1) =$
$A1 =$
A

9. $(AB)(A\bar{B}) =$
$AAB\bar{B} =$
$A0 =$
0

10. $AC + AC =$
AC

11. $AA + BC + BB + \bar{B} =$
$A + BC + B + \bar{B} =$
$A + BC + 1 =$
1

12. $(A + \bar{B})AB =$
$AAB + \bar{B}AB =$
$AB + A0 =$
$AB + 0 =$
AB

13. $1 + B + \bar{B} =$
$1 + 1 =$
1

14. $AB(\bar{A} + \bar{B}) =$
$AB\bar{A} + AB\bar{B} =$
$B0 + A0 =$
$0 + 0 =$
0

IS #12

BASIC LAWS OF BOOLEAN ALGEBRA

Symbols used: A, B inputs

AB: A and B

A + B: A or B

$\overline{A}$: not A

IDENTITY:	$A = A$	$\overline{A} = \overline{A}$
REDUNDANCY:	$AA = A$	$A + A = A$
COMMUTATIVE:	$AB = BA$	$A + B = B + A$
ASSOCIATIVE:	$A(BC) = ABC$	$A + (B + C) = A + B + C$
DOUBLE NEGATIVE:	$\overline{\overline{A}} = A$	
COMPLEMENTARY:	$A\overline{A} = 0$	$A + \overline{A} = 1$
INTERSECTION:	$A1 = A$	$A0 = 0$
UNION:	$A + 1 = 1$	$A + 0 = A$
DE MORGAN:	$\overline{AB} = \overline{A} + \overline{B}$	$\overline{A + B} = \overline{A}\,\overline{B}$
DISTRIBUTIVE:	$A(B + C) = AB + AC$	$A + BC = (A + B)(A + C)$
ABSORPTION:	$A + AB = A$	
COMMON IDENTITIES:	$A + \overline{A}B = A + B$	

IS #12

Examples:

1.

$AB + A(BC + AB) =$	
$AB + ABC + AAB =$	Distributive
$AB + ABC + AB =$	Redundant A
$AB + ABC =$	Redundant AB
AB	Absorption

2.

$A + AB + AC + ABC =$	
A	Absorption

3.

$A + AB + ABC + \bar{A}B =$	
$A + \bar{A}B =$	Absorption
$A + B$	Common Identity

4.

$A + BC + \bar{A}BC + ABC =$	
$A + BC + \bar{A}BC =$	Absorption
$A + BC + BC =$	Common Identity
$A + BC$	Redundant

5.

$A + ABC + 0 =$	
$A + ABC =$	Union
A	Absorption

6.

$(A + B)(A + C) =$	
$AA + AC + AB + BC =$	Foil
$A + AC + AB + BC =$	Redundant
$A + BC$	Absorption

7.

$(A + 1)(B + 0)(C + \bar{C})(B + C) =$	
$(1)(B)(1)(B + C) =$	Union, Complement
$B(B + C) =$	Intersection
$BB + BC =$	Distribute
$B + BC =$	Redundant
B	Absorption

Dunwoody College of Technology

KARNAUGH MAPS

A Karnaugh Map is a graphical means for simplifying Boolean expressions. With these maps, the redundant terms in an expression can be quickly eliminated.

Boolean expressions can be written in one of two ways. The <u>minterm form</u> consists of the ORing of single terms or And functions.

Example: ABC + ABD + BC + E

The <u>maxterm form</u> consists of the ANDing of single terms or Or functions.

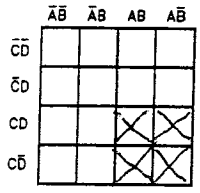
Example: (A + C)(B + D + E)A

The rules of Boolean Algebra permit us to convert from one form to the other. The distributive lay and DeMorgan's Theorem are especially helpful in these conversions.

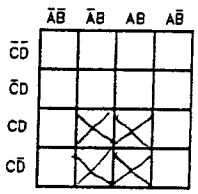
<u>When terms are to be plotted on a Karnaugh Map, they should be in the Minterm form.</u>

Plot of AC + BC + AD + BD on a Karnaugh Map.

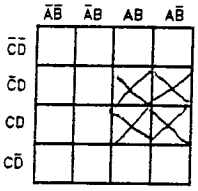
Plot of AC



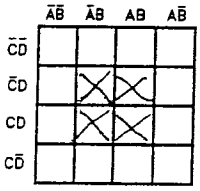
Plot of BC



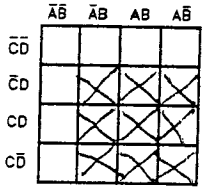
Plot of AD



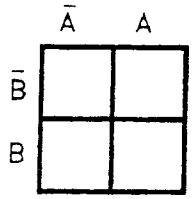
Plot of BD



Complete Map

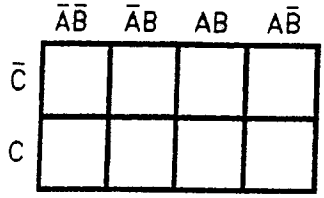


MAPPING



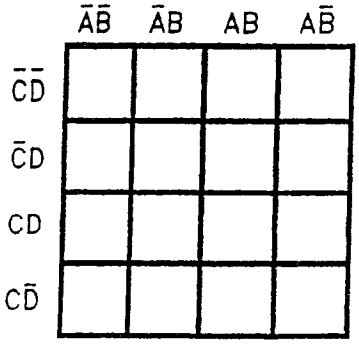
2 VARIABLE DIAGRAM

1 Variable Term = 2 Squares Occupied
2 Variable Term = 1 Square Occupied



3 VARIABLE DIAGRAM

1 Variable Term = 4 Squares Occupied
2 Variable Term = 2 Squares Occupied
3 Variable Term = 1 Square Occupied



4 VARIABLE DIAGRAM

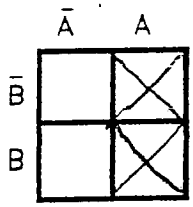
1 Variable Term = 8 Squares Occupied
2 Variable Term = 4 Squares Occupied
3 Variable Term = 2 Squares Occupied
4 Variable Term = 1 Square Occupied

Expression to be plotted must be in Minterm form.
Parantheses removed – Written in OR gates

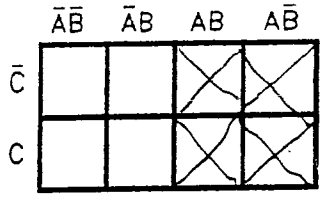
IS #13

Examples: Map the following Boolean expressions.

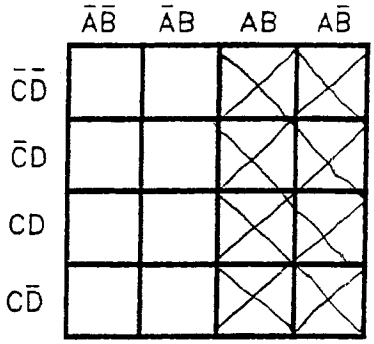
1. A



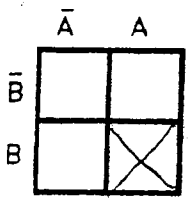
2. A



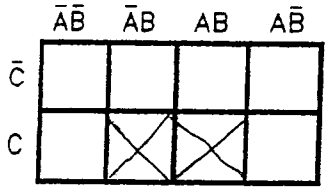
3. A



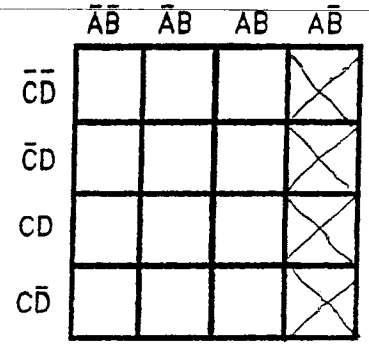
4. AB



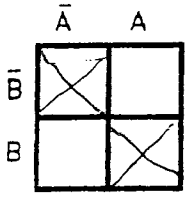
5. BC



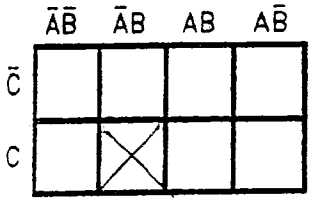
6. $A\bar{B}$



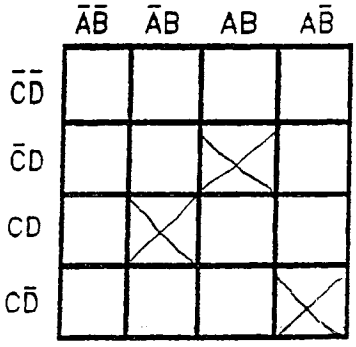
7. $AB + \bar{A}\bar{B}$



8. $\bar{A}BC$



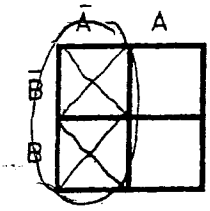
9. $AB\bar{C}D + A\bar{B}C\bar{D} + \bar{A}BCD$



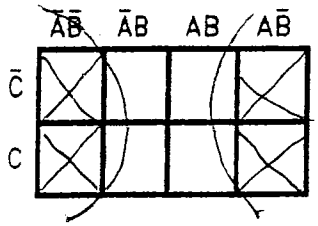
IS #13

Examples: Read the following maps.

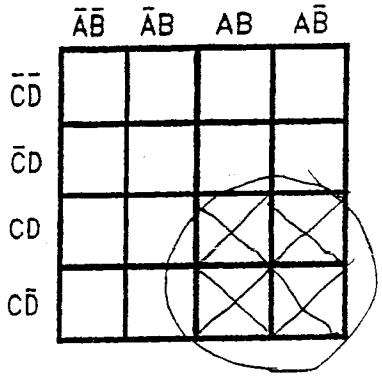
10. $\bar{A}$



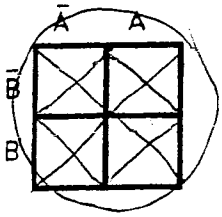
11. $\bar{B}$



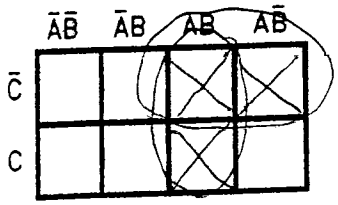
12. AC



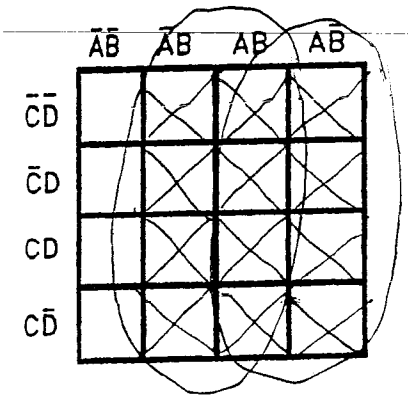
13. 1



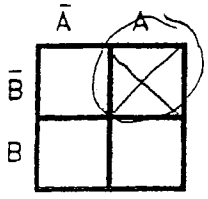
14. $AB + A\bar{C}$



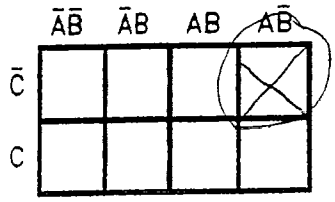
15. A + B



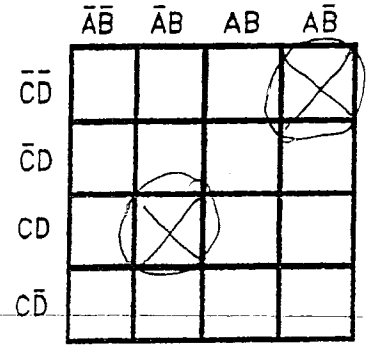
16. $A\bar{B}$



17. $A\bar{B}\bar{C}$



18. $A\bar{B}\bar{C}\bar{D} + \bar{A}BCD$



IS #14

REMOVING COMMON FACTORS

Boolean expressions and equations often can be simplified by removing common factors and applying the basic theorems

Example: Simplify the expression $A + AB$.

$$\begin{aligned} A + AB &= A(1 + B) && \text{(removing A as a common factor)} \\ &= A(1) && \text{(because } 1 + B = 1\text{)} \\ &= A && \text{(because } A \cdot 1 = A\text{)} \end{aligned}$$

Example: Simplify the expression $AB + A\overline{B}$

$$\begin{aligned} AB + A\overline{B} &= A(B + \overline{B}) && \text{(removing A from first two terms)} \\ &= A(1) && \text{(because } B + \overline{B} = 1\text{)} \\ &= A && \text{(because } A \cdot 1 = A\text{)} \end{aligned}$$

$\overline{AB} \neq \overline{A}\overline{B}$

It is important to note that an expression with a single NOT bar extending over several variables is not equal to the same expression with individual NOT bars over the variables. Thus $\overline{AB}$ is not the same as $\overline{A}\overline{B}$. That they are not equal is apparent from the truth table below.

A	B	AB	$\overline{AB}$	$\overline{A}\overline{B}$
0	0	0	1	1
0	1	0	1	0
1	0	0	1	0
1	1	1	0	0

A 1 appears in the AB column only when A AND B are both 1. Because $\overline{AB}$ is the exact opposite (negation) of AB, a 1 appears in the $\overline{AB}$ column wherever there is a 0 in the AB column, i.e., if $AB = 1$, then $\overline{AB} = 0$. The $\overline{A}\overline{B}$ column shows a 1 only when $A = 0$ AND $B = 0$. Comparison of the $\overline{AB}$ and the $\overline{A}\overline{B}$ columns reveals their lack of equivalence.

	B	$\overline{B}$
A		
$\overline{A}$		X

$\overline{A}\overline{B}$

	B	$\overline{B}$
A		X
$\overline{A}$	X	X

$\overline{AB} = \overline{A} + \overline{B}$

Algebra Review Sheet

1. A _____ is a declarative sentence that is either true or false, but not both.

2. "And", "or" and "not" are called _____.

3. In Boolean algebra, the notation _____ corresponds to "and".

4. In Boolean algebra, the notation _____ corresponds to "or".

5. In Boolean algebra, the notation _____ corresponds to "true".

6. In Boolean algebra, the notation _____ corresponds to "false".

7. Two propositions are _____ _____ if they have the same truth values for every combination of truth values for the simple propositions used as components.

8. Construct a truth table for: $A\overline{B} + \overline{C}$

9. Construct a truth table for: $A + \overline{B}$

10. Construct a truth table for $\overline{A} + \overline{B}\overline{C}$

11. Evaluate $A\overline{B} + C$ if A = 1, B = 0 and C = 1.

12. Evaluate $\overline{AB + BC}$ if A = 0, B = 1, and C = 1.

Simplify the following Boolean statements.

13. $AB(A + \overline{B})$ _____

14. $(A + \overline{A})B$ _____

Algebra Review Sheet

Simplify the following Boolean expressions.

15. $A + AB + 1$ _____

16. $A(1 + \bar{A})$ _____

17. $(B + 1)(A + 0)$ _____

18. $(A + B)(\bar{A}C + B)(C + \bar{B})$ _____

19. $\overline{AB} + \overline{A + B}$ _____

20. $AB + \overline{AB}D + A\bar{B} + CD$ _____

Draw a circuit to illustrate each Boolean expression.

21. $B\bar{C} + \bar{B}C$

22. $(B + \bar{C})\ (\bar{B} + C)$

23. Simplify the circuit $(A + \bar{B}C)B$. _____

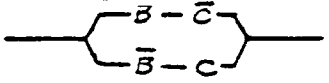
24. Simplify the circuit $(AB + C)AB$. _____

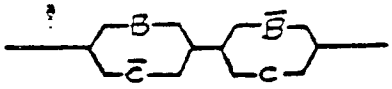
Algebra Review Sheet

1. proposition/statement
2. connectives
3. $\cdot$
4. $+$
5. 1
6. 0
7. logically equivalent
8.
A	B	C	$A\bar{B} + \bar{C}$
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

9.
A	B	$A + \bar{B}$
0	0	1
0	1	0
1	0	1
1	1	1

10.
A	B	C	$\bar{A} + \bar{B}C$
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

11. 1
12. 0
13. AB
14. B
15. 1
16. A
17. A
18. BC
19. $\bar{A} + \bar{B}$
20. A + D
21. 

22. 

23. AB
24. AB