



University of Anbar
College of Engineering / Electrical Department

Digital Techniques Lectures

By
Mohammed Khaleel



Number Systems:

Many number systems are in use in digital technology. The most common are the decimal, binary, octal, and hexadecimal systems. The decimal system is clearly the most familiar to us because it is the tool that we use every day. Examining some of its characteristics will help us to better understand of other systems.

Decimal System:

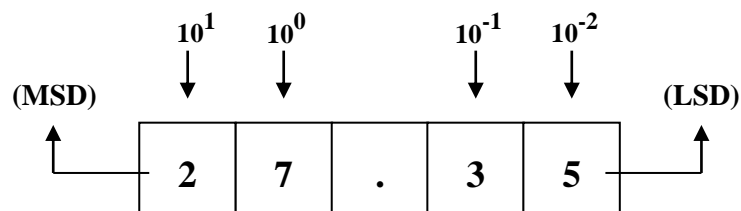
The decimal system is composed of 10 numerals or symbols. These symbols are 0,1,2,3,4,5,6,7,8,9; using these symbols as digits of a number, we can express any quantity. The decimal system is also called the base-10 system. The decimal system is a positional-value system in which the value of a digit depends on its position. For example, consider the decimal number 453. We know that the digit 4 actually represents 4 hundreds (400), the 5 represents 5 tens (50), and 3 represents 3 units.

In essence, the 4 carries the most weight of these digits; it's referred to us the most significant digit (MSD). The 3 carries the least weight and is called the least significant digit (LSD).

Example: consider the decimal number 27.35; this number is actually equal to:

$$2 \times 10 + 7 \times 1 + 3 \times 0.1 + 5 \times 0.01$$

The decimal point is used to separate the integer and the fractional parts of the number.





Binary System:

In the binary system there are only two symbols or possible digit values, 0 and 1. Even so, this base-2 system can be used to represent any quantity that can be represented in decimal or other number systems. The binary number system of Table (1) is nothing more than a code. After some practice, it becomes almost as familiar as the decimal number system.

Table (1)

<i>Quantity</i>	<i>Binary No.</i>	<i>Decimal No.</i>
None	0	0
.	1	1
..	10	2
...	11	3
....	100	4
.....	101	5
.....	110	6
.....	111	7

Binary-to-Decimal Conversion:

The binary number system is a positional system where each binary digit (Bit) carries a certain weight based on its position relative to the LSB (Least Significant Bit). 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. To illustrate:

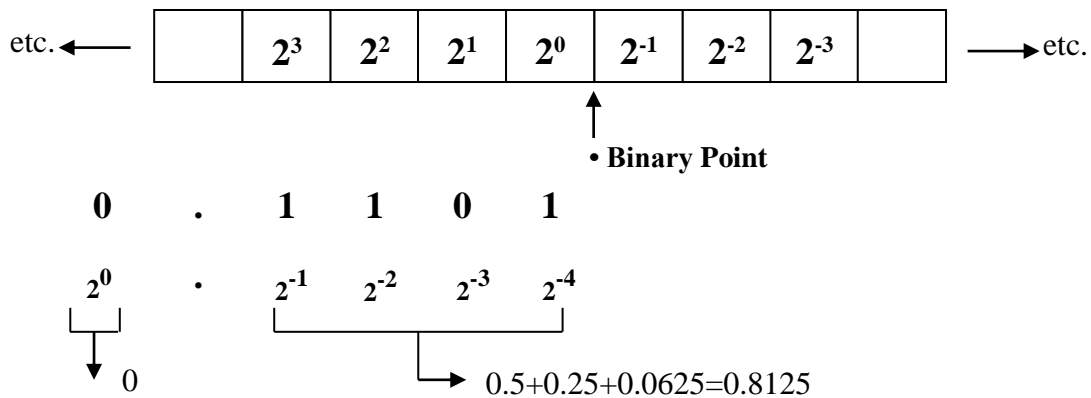
$$\begin{array}{cccccc}
 \mathbf{1} & \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{1} & \mathbf{\hspace{1cm}} & \mathbf{(Binary)} \\
 2^4 + & 2^3 + & 2^2 + & 2^1 + & 2^0 = & 16+8+2+1 & \\
 & & & & = & 27_{(10)} & \mathbf{(Decimal)}
 \end{array}$$

Let's try another example with greater number of bits.

$$\begin{array}{cccccccc}
 \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{\hspace{1cm}} & \mathbf{(Binary)} \\
 2^7 + & 2^6 + & 2^5 + & 2^4 + & 2^3 + & 2^2 + & 2^1 + & 2^0 = & 181_{(10)} & \mathbf{(Decimal)}
 \end{array}$$

Example: find the decimal equivalent of the 0.1101?

As far as mixed numbers are concerned (number that have an integer and fractional part), the weights for a mixed number are:



Hence: $0.1101_{(2)} = 0.8125_{(10)}$

Example: convert binary 110.001 to a decimal number

$$\begin{array}{ccccccc}
 1 & 1 & 0 & . & 0 & 0 & 1 \\
 2^2 & 2^1 & 2^0 & . & 2^{-1} & 2^{-2} & 2^{-3}
 \end{array} \longrightarrow 6.125$$

Decimal-to-Binary Conversion:

One way to convert a decimal number into its binary equivalent is to reverse the process described in the binary to decimal conversion paragraph. For instance, suppose you want to convert decimal 9 into the corresponding binary number. All you need to do is express 9 as a sum of power of 2, and then write 1's and 0's in the appropriate positions.

$$\begin{aligned}
 9 &= 8+1 = 8 + 0 + 0 + 1 \\
 &= 2^3 + 2^2 + 2^1 + 2^0 \\
 &= 1 \quad 0 \quad 0 \quad 1 \quad \text{(Binary)}
 \end{aligned}$$

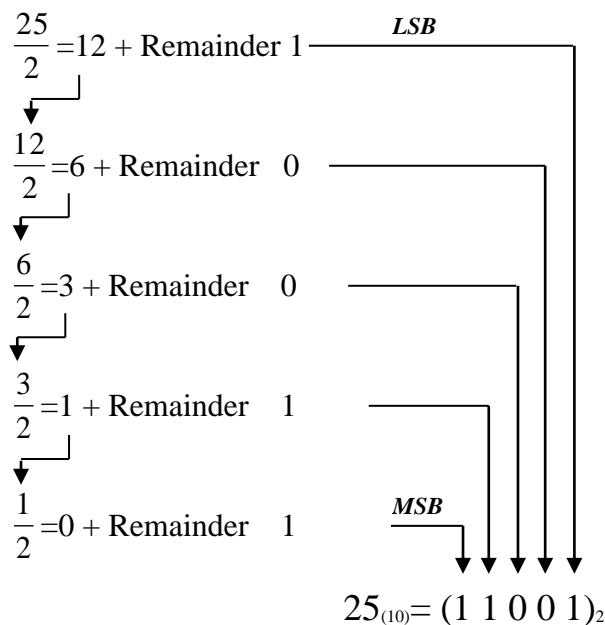


As another example,

$$25 = 16 + 8 + 1 = 16 + 8 + 0 + 0 + 1$$

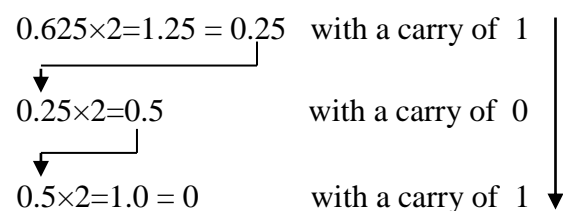
$$1 \quad 1 \quad 0 \quad 0 \quad 1 \text{ (Binary)}$$

Another popular way to convert decimal number to binary numbers is the (repeated division). This method requires repeatedly dividing the decimal number by 2 and writing down the remainders after each division until a quotient of 0 is obtained. Note that the binary result is obtained by writing the first remainder as the LSB and the last remainder as the MSB. Let us convert decimal 25 to its binary equivalent using the repeated division method.



As far as fractions are concerned, it is possible to multiply by 2 and record a carry in the integer position. As an example, convert 0.625 to binary fraction.

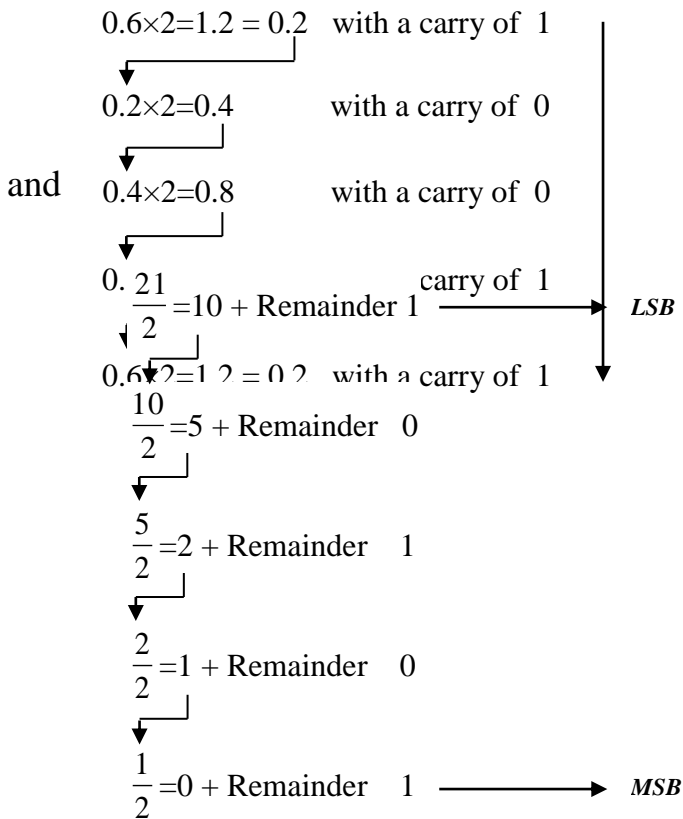
By taking the carries in forward order, we get (0.101) which is the binary equivalent of 0.625





Example: convert $21.6_{(10)}$ to a binary number.

Split 21.6 into an integer of 21 and a fraction of 0.6, and apply repeated division to each part.

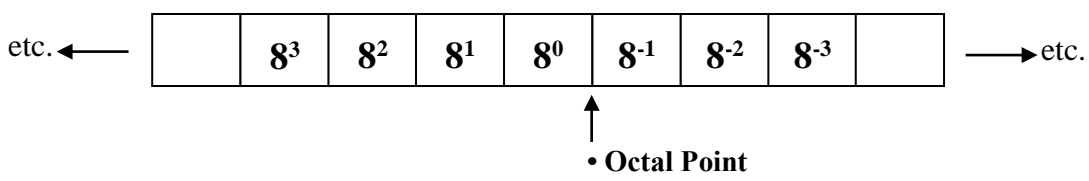


This conversion of fractional part is an approximation value because we terminated the conversion after five bits. If more accuracy is needed, continue multiplying by 2 until you have as many digits as necessary.

hence: $21.6_{(10)} = (10110110011)_2$

Octal Number System:

The octal number system is very important in digital computer work. the octal number system has a base of 8, meaning it has eight possible digits: 0,1,2,3,4,5,6,7. Thus, each digit of an octal number can have any value from 0 to 7. The digital positions in an octal number have weights as follows:





Octal-to-Decimal Conversion:

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

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

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

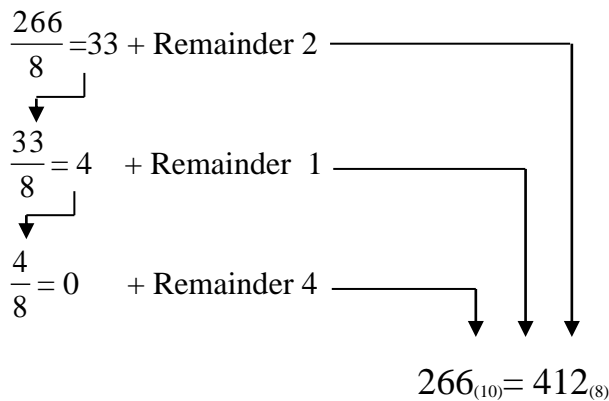
Example: convert $24.6_{(8)}$ to its decimal equivalent.

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

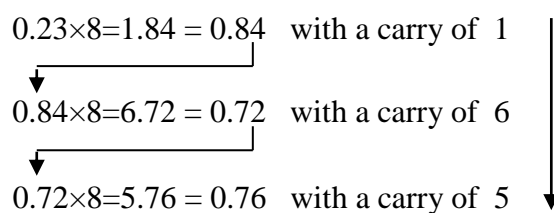
$$= 2 \times 8 + 4 \times 1 + 6 \times 0.125 = 20.75_{(10)}$$

Decimal-to-Octal Conversion:

A decimal integer can be converted to octal by using the same repeated division method that have been used in the decimal-to-binary conversion, but with a division factor of 8 instead of 2. An example is shown below:



For decimal fractions, multiplying instead of dividing, writing the carry into the integers position. An example of this is to convert 0.23 into an octal fraction.





The process is terminated after three places; if more accuracy were required, we continue multiplying to obtain more octal digit.

Octal-to-Binary Conversion:

The primary advantage of the octal number system is the ease with which conversion can be made between binary and octal numbers. The conversion from octal to binary is performed by converting each octal digit to its 3-bit binary equivalent. The eight possible digits are converted as indicated in Table (2).

Table (2)

Octal No.	0	1	2	3	4	5	6	7
Binary Equivalent	000	001	010	011	100	101	110	111

For example, $472_{(8)}$ is converted to its binary equivalent as follows:-

$$472_{(8)} = 100111010_{(2)}$$

4	7	2
↓	↓	↓
100	111	010

Example: convert $34.562_{(8)}$ to its binary equivalent.

$$34.562_{(8)} = 011100.101110010_{(2)}$$

3	4	.	5	6	2
↓	↓		↓	↓	↓
011	100	.	101	110	010

Binary-to-Octal Conversion:

Converting from binary integers to octal integers is done by grouping the binary bits into groups of three bits starting at the LSB. Then each group is converted to its octal equivalent.

Example: convert $100111010_{(2)}$ to octal system.

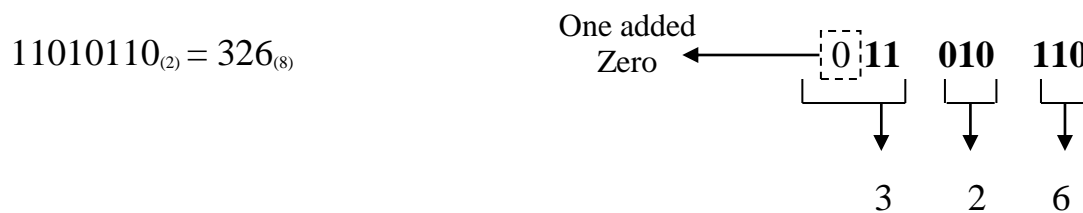
$$100111010_{(2)} = 472_{(8)}$$

	100	111	010
8	┌	┌	┌
	↓	↓	↓
	4	7	2

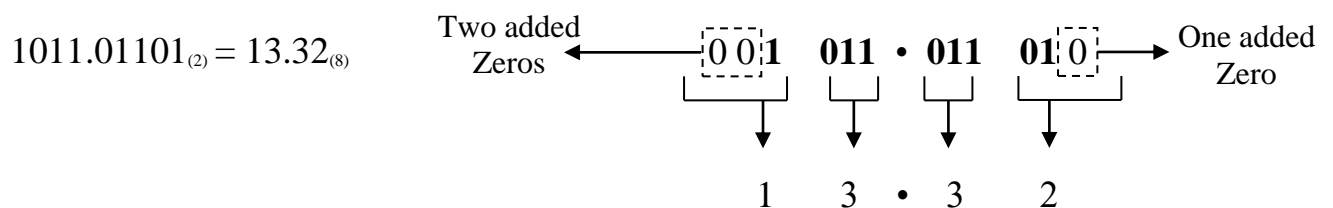


Example: convert $11010110_{(2)}$ to its octal equivalent.

Sometimes the binary number will not have even groups of 3bits. For this case, extra 0's can be added to the left of the MSB of the binary number to fill out the last group as shown below:



Example: convert $1011.01101_{(2)}$ to octal system.



Hexadecimal Number System:

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. Table (3) shows the relationships among hexadecimal, decimal, and binary digits.

Table (3)

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

Hex-to-Decimal Conversion:

a hex number can be converted to its decimal equivalent by using the fact that each hex digit position has a weight that is a power of 16. the LSD has a weight of $16^0 = 1$, the next higher digit has a weight of $16^1 = 16$, the next higher digit has a weight of $16^2 = 256$, and so on. The conversion is demonstrated in the examples below:

$$356_{(16)} = 3 \times 16^2 + 5 \times 16^1 + 6 \times 16^0$$

$$= 768 + 80 + 6 = 854_{(10)}$$

$$2AF_{(16)} = 2 \times 16^2 + 10 \times 16^1 + 15 \times 16^0$$

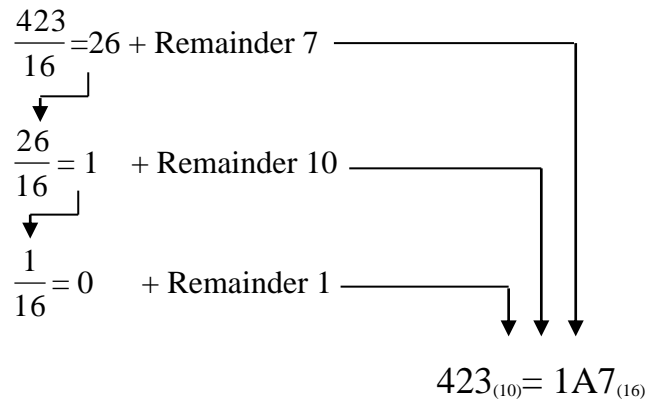
$$= 512 + 160 + 15 = 687_{(10)}$$

Decimal-to-Hex Conversion:

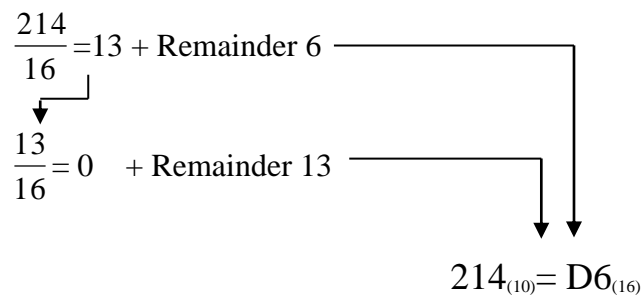
Recall that we did decimal-to-binary conversion using repeated division by 2, and decimal-to-octal conversion using repeated division by 8. Likewise, decimal-to-Hex conversion can be done using repeated division by 16.



Example: convert $423_{(10)}$ to hex.



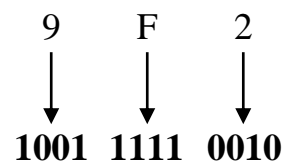
Example: convert $214_{(10)}$ to hex.



Hex-to-Binary Conversion:

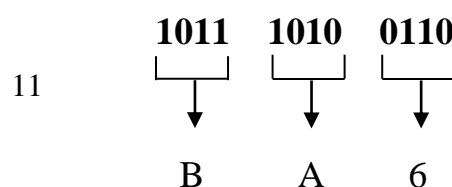
Each Hex digit is converted to its 4-bit binary equivalent. This is illustrated below for $9F2_{(16)}$.

$$9F2_{(16)} = 100111110010_{(2)}$$



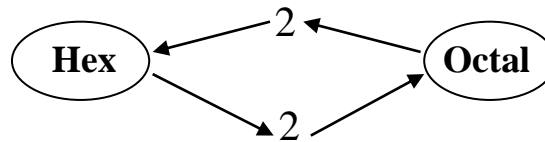
Binary-to-Hex Conversion:

This conversion is just the reverse of the process of Hex-to-Decimal conversion. The binary number is grouped into groups of 4bits, and each group is converted to its equivalent hex digit as in the example below:



$$101110100110_{(2)} = BA6_{(16)}$$

Hex-to-Octal Conversion:



Review Questions:

1. What is the binary equivalent of decimal number 363? convert to octal and then to binary?
2. What is the largest number that can be represented using 8 bits?
3. What is the decimal equivalent of $1101011_{(2)}$?
4. What is the next binary number following $10111_{(2)}$ in the counting sequence?
5. What is the weight of the MSB of a 16-Bit number?
6. Convert $614_{(8)}$ to decimal?
7. Convert $146_{(10)}$ to octal?
8. Convert $24CE_{(16)}$ to decimal?
9. Convert $3117_{(10)}$ to hex, then from hex to binary?
10. Solve for x in the following equation: $1011.11_{(2)} = x_{(10)}$?
11. Solve for x in the following equation: $174.3_{(8)} = x_{(10)}$?
12. Solve for x in the following equation: $10949.8125_{(10)} = x_{(2)}$?
13. Solve for x in the following equation: $2C6B.F2_{(16)} = x_{(2)}$?



Arithmetic Operation

Addition of Binary Numbers:

The addition of two binary numbers is performed in exactly the same manner as the addition of decimal numbers. Only four cases can occur in adding the two binary digits (bits) in any position. They are:

- 0+0=0
- 1+0=1
- 0+1=1
- 1+1=0 with carry 1
- 1+1+1=1 with carry 1

Examples:

$\begin{array}{r} 011 \quad (3) \\ + 110 \quad (6) \\ \hline 1001 \quad (9) \end{array}$	$\begin{array}{r} 1001 \quad (9) \\ + 1111 \quad (15) \\ \hline 11000 \quad (24) \end{array}$	$\begin{array}{r} 11.011 \quad (3.375) \\ + 10.110 \quad (2.750) \\ \hline 110.001 \quad (6.125) \end{array}$	$\begin{array}{r} 1010 \quad (10) \\ + 1101 \quad (13) \\ \hline 10111 \quad (23) \end{array}$
--	---	---	--

Subtraction of Binary Numbers (Using Direct Method):

The four basic rules for subtracting binary digits are:

- 0-0=0
- 1-1=0
- 1-0=1
- 0-1=1 with borrow 1

Examples:

$\begin{array}{r} 11 \quad (3) \\ - 01 \quad (1) \\ \hline 10 \quad (2) \end{array}$	$\begin{array}{r} 11 \quad (3) \\ - 10 \quad (2) \\ \hline 01 \quad (1) \end{array}$	$\begin{array}{r} 101 \quad (5) \\ - 011 \quad (3) \\ \hline 010 \quad (2) \end{array}$
--	--	---

when 1 is borrowed, a 0 is left .

 when 1 is borrowed, making 10 instead of 0.

Subtraction of Binary Numbers (Using Complement Method):

The 1's complement and the 2's complement of a number are important because they permit the representation of negative numbers. The method of 2's complement arithmetic is used in computer to handle negative numbers



- ❖ 1's complement: the 1's complement form of any binary number is obtained simply by changing each 0 in the number to a 1 and each 1 to a 0. In other word, change each bit to its complement. For example:

$\begin{array}{cccccc} 1 & 0 & 1 & 1 & 0 & 1 \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ 0 & 1 & 0 & 0 & 1 & 0 \end{array}$	Binary No.	$\begin{array}{cccccc} 0 & 1 & 1 & 0 & 1 & 0 \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ 1 & 0 & 0 & 1 & 0 & 1 \end{array}$	Binary No.
	1's complement		1's complement

- ❖ 2's complement: the 2's complement form of a binary number is formed simply by taking the 1's complement of the number and adding 1 to the least significant bit position.

$$2's \text{ complement} = (1's \text{ complement}) + 1$$

Example: find 2's complement of 10110010.

$\begin{array}{cccccc} 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ + & & & & & & & 1 \\ \hline 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 \end{array}$	binary number.	1's complement	adding 1	2's complement
--	----------------	----------------	----------	----------------

Example: find $11010_{(2)} - 10000_{(2)}$ using 1's complement method (Case 1).

As long as the carry appear, the number is positive and a carry must be added to the result.

$\begin{array}{r} 11010 \\ + 01111 \\ \hline 101001 \\ + \quad 1 \\ \hline 01010 \end{array}$	1's complement of 10000
---	-------------------------

$$11010_{(2)} - 10000_{(2)} = 01010_{(2)}$$

Example: find $10000_{(2)} - 11010_{(2)}$ using 1's complement method (Case 2).

As long as no carry appear, the number is negative, then 1's complementing of the final result in needed.

$\begin{array}{r} 10000 \\ + 00101 \\ \hline 10101 \\ \downarrow \downarrow \downarrow \downarrow \downarrow \\ 01010 \end{array}$	1's complement of 11010
	1's complement

$$10000_{(2)} - 11010_{(2)} = -01010_{(2)}$$



Example: find $11010_{(2)} - 10000_{(2)}$ using 2's complement method (Case 3).

As long as the carry appear, the number is positive and a carry must be discarded	←	$\begin{array}{r} 11010 \\ + 10000 \\ \hline 101010 \end{array}$	2's complement of 10000
---	---	--	-------------------------

$11010_{(2)} - 10000_{(2)} = 01010_{(2)}$

Example: find $10000_{(2)} - 11010_{(2)}$ using 2's complement method (Case 4).

As long as no carry appear, the number is negative, then 2's complementing of the final result is needed.	←	$\begin{array}{r} 10000 \\ + 00110 \\ \hline 10110 \\ \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \\ 01010 \end{array}$	2's complement of 11010 2's complement
---	---	---	---

$10000_{(2)} - 11010_{(2)} = - 01010_{(2)}$

Multiplication of Binary Numbers:

The numbers in a multiplication are the *multiplicand*, the *multiplier*, and the *product*. These are illustrated in the following decimal multiplication:-

the multiplication rules for binary numbers are:

$0 \times 0 = 0$

$0 \times 1 = 0$

$1 \times 0 = 0$

$1 \times 1 = 1$

8	→	<i>multiplicand</i>
$\times 3$	→	<i>multiplier</i>
24	→	<i>product</i>

Example: find the product of $100_{(2)}$ and $010_{(2)}$.

$100_{(2)} \times 010_{(2)} = 01000_{(2)}$

100	(4)
$\times 010$	(2)
000	
1000	+
$\hline 00000$	+
01000	(8)



Division of Binary Numbers:

The numbers in a division are the *dividend*, the *divisor*, and the *quotient*.

These are illustrated in the following standard.

to illustrate, consider the following division examples:

$$\frac{\text{dividend}}{\text{divisor}} = \text{quotient}$$

$$\begin{array}{r} 110 \\ 100 \overline{) 11000} \\ \underline{100} \\ 0100 \\ \underline{100} \\ 0000 \end{array}$$

$$\begin{array}{r} 0010.1 \\ 100 \overline{) 1010} \\ \underline{100} \\ 00100 \\ \underline{100} \\ 000 \end{array}$$

$$\begin{array}{r} 010.1 \\ 10 \overline{) 101} \\ \underline{10} \\ 0010 \\ \underline{10} \\ 00 \end{array}$$

Addition of Hexadecimal Numbers:

Hex numbers are used extensively in machine-language computer programming and in conjunction with computer memories. When working in these areas, there will be situations where hex numbers have to be added or subtracted. The addition can be done in the same manner as decimal addition. Let's add the hex numbers 58 and 24, 58 and 4B.

$$\begin{array}{r} 58 \\ + 24 \\ \hline 7C \end{array}$$

$$\begin{array}{r} 3AF \\ + 23C \\ \hline 5EB \end{array}$$

$$\begin{array}{r} 58 \\ + 4B \\ \hline 93 \end{array}$$

Examples: add the following hexadecimal numbers.

(a) $23_{(16)} + 16_{(16)}$

$$\begin{array}{r} 23 \\ + 16 \\ \hline 39 \end{array}$$

right column: $3_{(16)} + 6_{(16)} = 3_{(10)} + 6_{(10)} = 9_{(10)} = 9_{(16)}$
 left column: $2_{(16)} + 1_{(16)} = 2_{(10)} + 1_{(10)} = 3_{(10)} = 3_{(16)}$

(b) $58_{(16)} + 22_{(16)}$

$$\begin{array}{r} 58 \\ + 22 \\ \hline 7A \end{array}$$

right column: $8_{(16)} + 2_{(16)} = 8_{(10)} + 2_{(10)} = 10_{(10)} = A_{(16)}$
 left column: $5_{(16)} + 2_{(16)} = 5_{(10)} + 2_{(10)} = 7_{(10)} = 7_{(16)}$

(c) $DF_{(16)} + AC_{(16)}$



$$\begin{array}{r}
 DF \quad \text{right column: } F_{(16)} + C_{(16)} = 15_{(10)} + 12_{(10)} = 27_{(10)} \\
 \phantom{\text{right column: }} = 27_{(10)} - 16_{(10)} = 11_{(10)} = B_{(16)} \text{ with a carry of } 1 \\
 + \underline{AC} \quad \text{left column: } D_{(16)} + A_{(16)} + 1_{(16)} = 13_{(10)} + 10_{(10)} + 1_{(10)} = 24_{(10)} \\
 18B \phantom{\text{left column: }} = 24_{(10)} - 16_{(10)} = 8_{(10)} = 8_{(16)} \text{ with a carry of } 1
 \end{array}$$

Subtraction of Hexadecimal Numbers (Using Direct Method):

Reverse operation of addition may be used as a direct way to subtract hexadecimal numbers as shown in the following examples:

$$\begin{array}{r}
 D3A \quad \text{right column: } A_{(16)} - 4_{(16)} = 10_{(10)} - 4_{(10)} = 6_{(10)} = 6_{(16)} \\
 - \underline{F4} \quad \text{middle column: } 3_{(16)} - F_{(16)} = 3_{(10)} - 15_{(10)} \text{ (need borrow)} \\
 C46 \phantom{\text{middle column: }} = 19_{(10)} - 15_{(10)} = 4_{(10)} = 4_{(16)} \\
 \quad \text{left column: } D_{(16)} - 1_{(16)} = C_{(16)}
 \end{array}$$

$$\begin{array}{r}
 84 \quad \text{right column: } 4_{(16)} - A_{(16)} = 4_{(10)} - 10_{(10)} \text{ (need borrow)} \\
 - \underline{2A} \phantom{\text{right column: }} = 20_{(10)} - 10_{(10)} = 10_{(10)} = A_{(16)} \\
 5A \quad \text{left column: } 8_{(16)} - 2_{(16)} - 1_{(16)} = 5_{(16)}
 \end{array}$$

Subtraction of Hexadecimal Numbers (Using Complement Method):

Remember that hex numbers are just an efficient way to represent binary numbers. Thus we can subtract hex numbers using the same method we used for binary numbers. In order to find the complement of hex numbers, two ways are found

- first way:

7	3	A	→	hex number
0111	0011	1010	→	convert to binary
1000	1100	0101	→	1's complement representation
1000	1100	0110	→	2's complement representation
8	C	6	→	conversion back to hex

- Second way: this procedure is quicker, subtract each hex digit from F, and then add 1. let's try this for the same hex number from the example above:

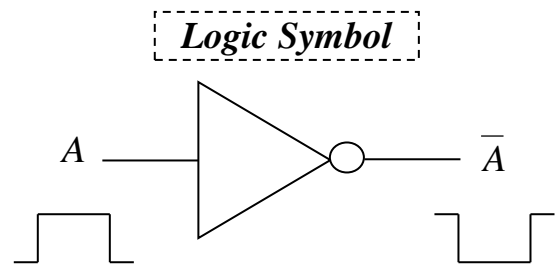
F	F	F	→	Subtract each digit from F
-7	-3	-A		
8	C	5	→	adding 1
		+1		
8	C	6	→	hex. Equivalent of 2's comp.

1- Inverter (NOT Gate):

The inverter performs the operation called inversion or complementation. The purpose of the inverter is to change the one logic level to the opposite level. In terms of bits, it changes a 1 to 0 and a 0 to a 1.

Inverter Truth Table

Input (A)	Output (\bar{A})
0	1
1	0

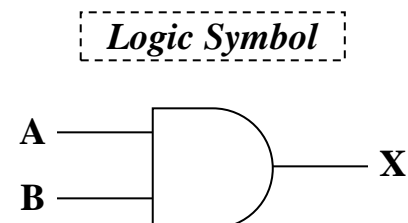


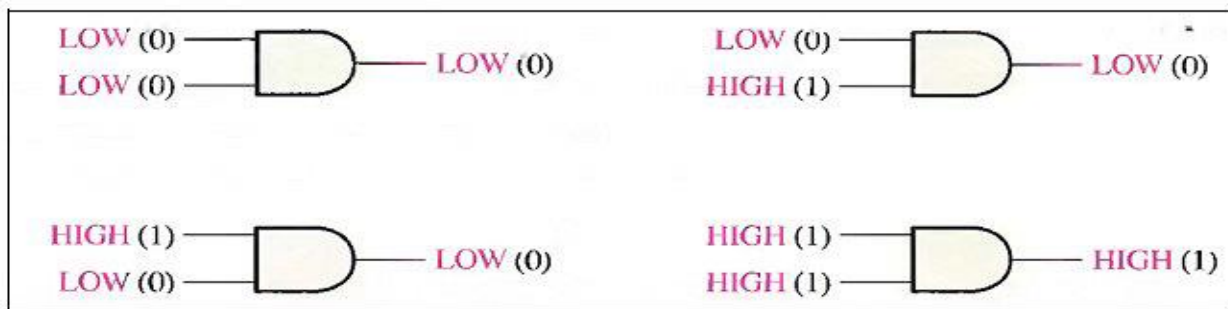
2- AND Gate:

The AND Gate is one of the basic gates from which all logic functions are constructed. An AND gate can have two or more inputs and performs what is known as logical multiplication. Figure below, all possible logic levels for a 2-input AND gate.

AND Gate Truth Table

Inputs		Output
A	B	X
0	0	0
0	1	0
1	0	0
1	1	1





The total number of possible combinations of binary inputs to a gate is determined by the following formula:

$$N = 2^n$$

Where N is the total possible combinations and n is the number of input variables.

To illustrate,

For two input variables: $N=2^2 =4$

For three input variables: $N=2^3 =8$

For four input variables: $N=2^4 =16$

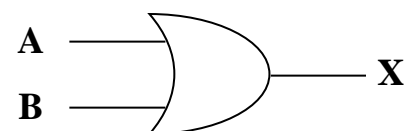
3- OR Gate:

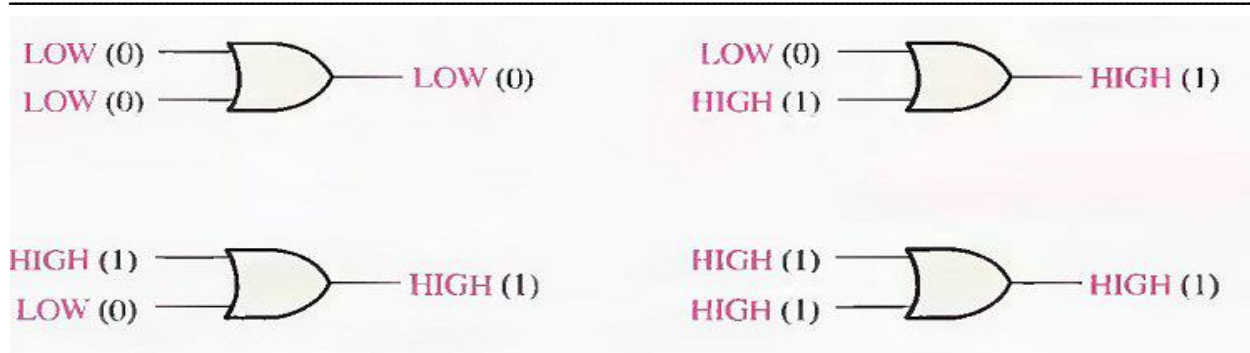
The OR gate is one of the basic gates from which all logic functions are constructed. An OR gate can have two or more inputs and performs what is known as logical addition. Figure below, all possible logic levels for a 2-input OR gate.

OR Gate Truth Table

Inputs		Output
A	B	X
0	0	0
0	1	1
1	0	1
1	1	1

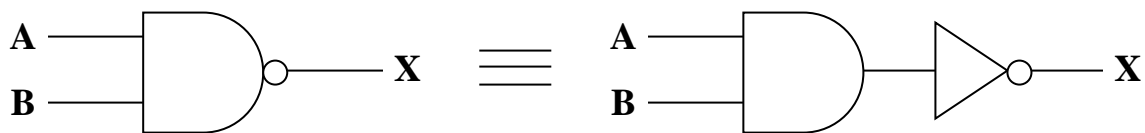
Logic Symbol





4- NAND Gate:

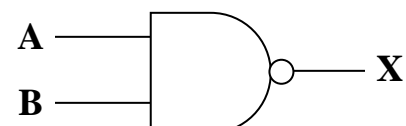
The NAND gate is a popular logic element because it can be used as a universal gate; that is; NAND gate can be used to perform the AND, OR, and Inverter operations, or any combination of these operations. The term NAND is a contraction of NOT-AND and implies an AND function with a complemented (Inverted) output. Figure below Operation of a 2-input NAND gate.

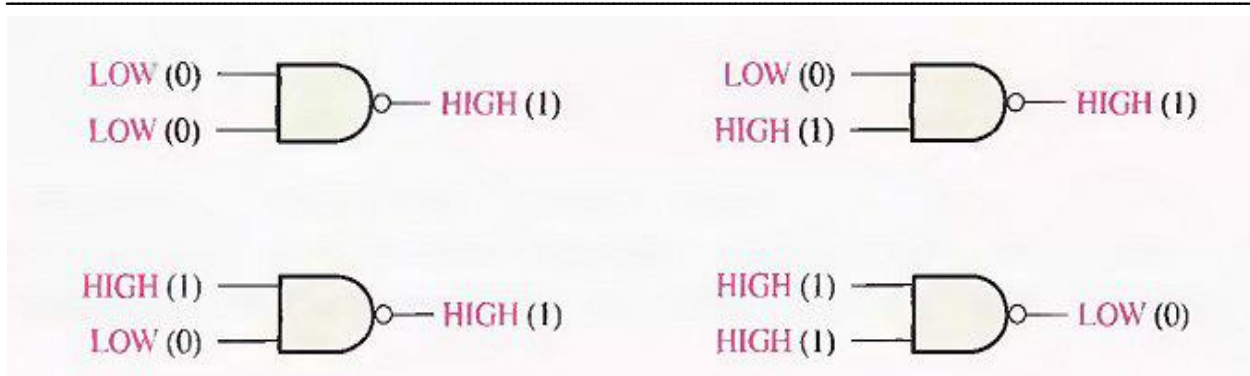


NAND Gate Truth Table

Inputs		Output
A	B	X
0	0	1
0	1	1
1	0	1
1	1	0

Logic Symbol





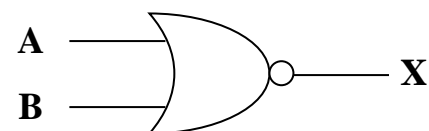
5- NOR Gate:

The NOR gate, like the NAND gate, is a very useful logic element because it can also be used as a universal gate; that is; NOR gate can be used to perform the AND, OR, and Inverter operations, or any combination of these operations.

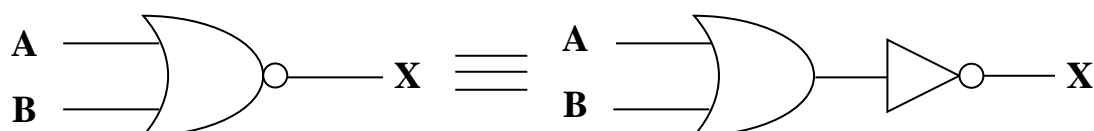
NOR Gate Truth Table

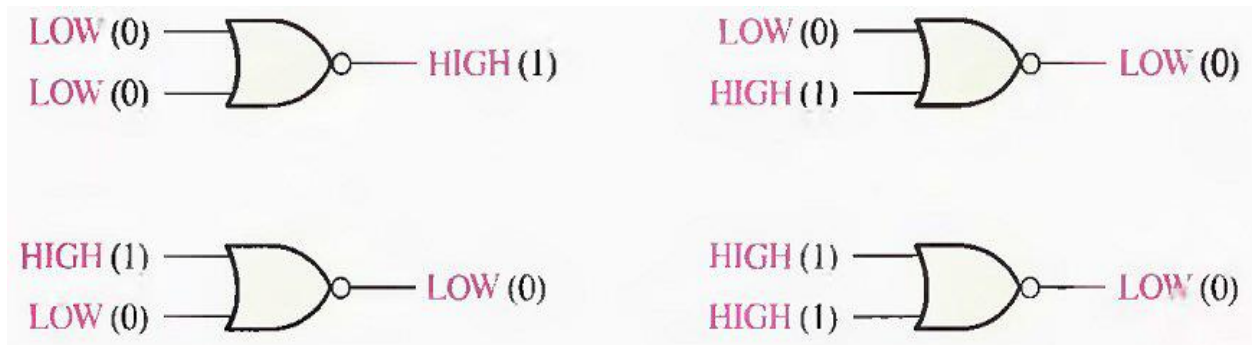
Inputs		Output
A	B	X
0	0	1
0	1	0
1	0	0
1	1	0

Logic Symbol



The term NOR is contraction of **NOT-OR** and implies an OR function with an inverted output. Figure below Operation of a 2-input NOR gate.





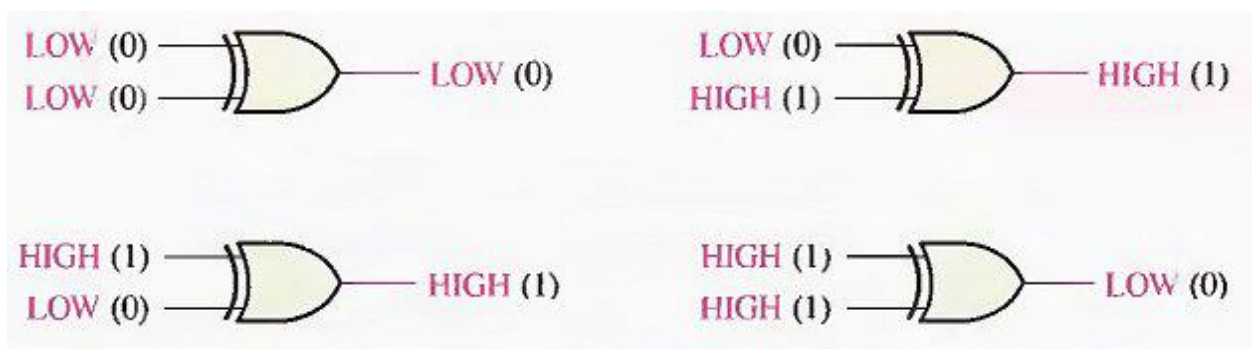
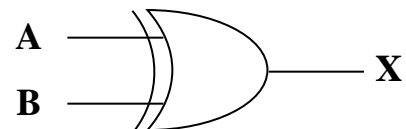
6- Exclusive-OR Gate (XOR):

The Exclusive-OR is actually formed by a combination of other gates. Figure below, all possible logic levels for an exclusive-OR gate.

XOR Gate Truth Table

Inputs		Output
A	B	X
0	0	0
0	1	1
1	0	1
1	1	0

Logic Symbol



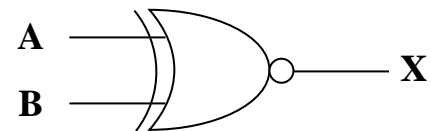
7- Exclusive-NOR Gate (XNOR):

The Exclusive-NOR is actually formed by a combination of other gates.
Figure below, all possible logic levels for an exclusive-NOR gate.

XNOR Gate Truth Table

Inputs		Output
A	B	X
0	0	1
0	1	0
1	0	0
1	1	1

Logic Symbol



Example: (a) Develop the truth table for a 3-input AND gate.

(b) Determine the total number of possible input combinations for a 5-input AND gate.

For branch (a) there are eight possible input combinations for a 3-input AND gate.

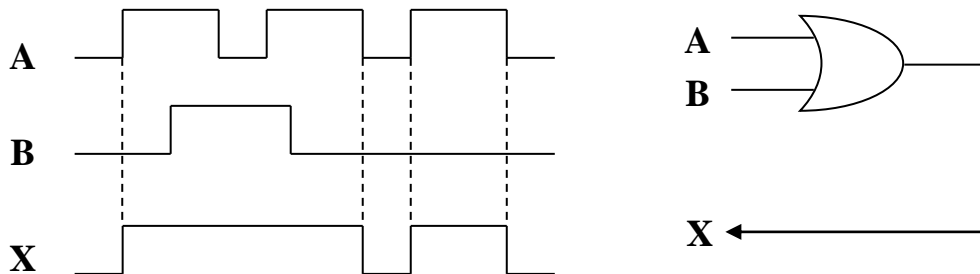
Input			Output
A	B	C	X
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0



1	0	1	0
1	1	0	0
1	1	1	1

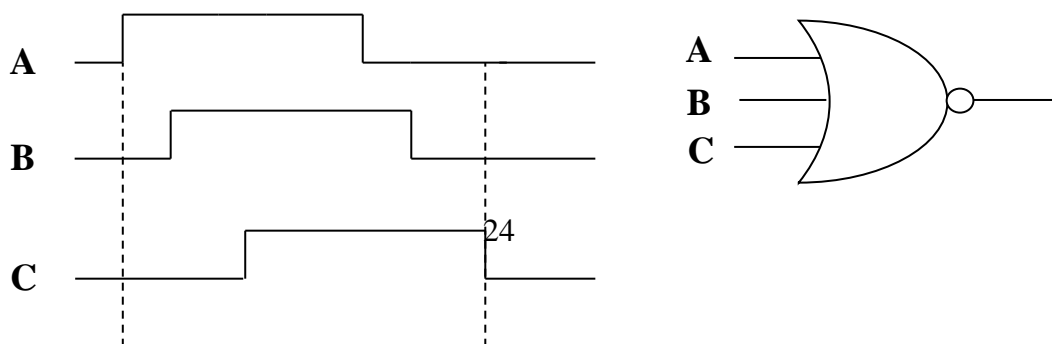
For branch (b), $N=2^5 =32$. There are 32 possible combinations of input bits for a 5-input AND gate.

Example: for the two input waveforms, A and B, sketch the output waveform, showing its proper relation to the inputs.



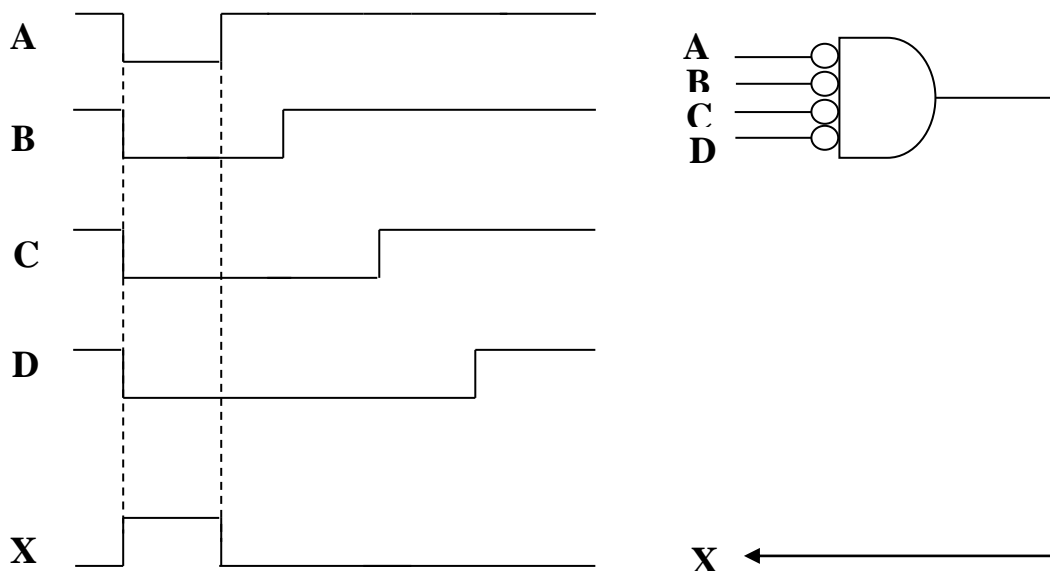
When either or both inputs are **HIGH**, the output is **HIGH** as shown by the output waveform X in the timing diagram.

Example: Sketch the output waveform for the 3-input NOR gate, showing the proper relation to the input.



The output X is **LOW** when any input is **HIGH** as shown by the output waveform X in the timing diagram.

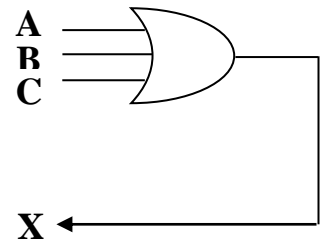
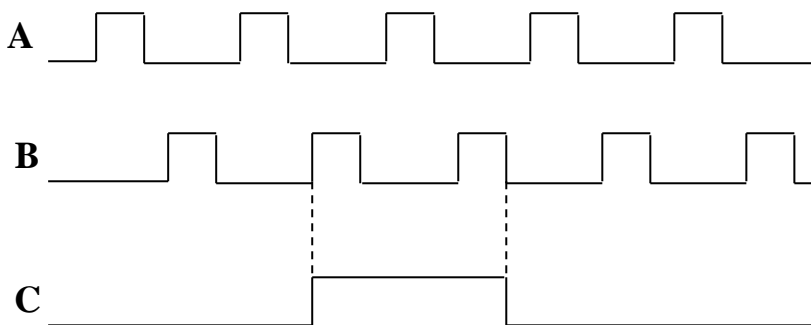
Example: For the 4-input NOR gate operating as a negative-AND. determine the output relative to the inputs.



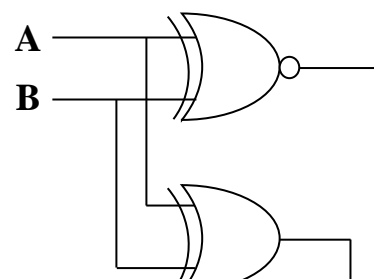
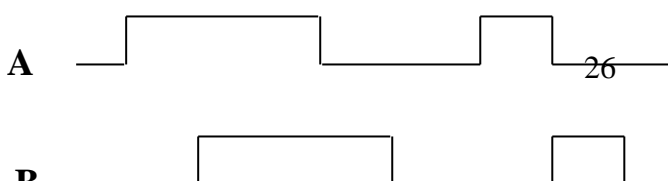


Review Questions:

1. When is the output of an AND gate HIGH?
2. Describe the truth table for a five-input AND gate?
3. How can you use an XOR gate to detect when two bits are different?
4. For the 3-input OR gate, determine the output waveform in proper relation to the inputs?



5. Determine the output waveform for the XOR gate and for the XNOR gate, given the input waveforms, A and B?





Choose the right answer:

1. An inverter performs an operation known as:
 - (a) Complementation
 - (b) Assertion
 - (c) Inversion
 - (d) both a and c
2. The output of an OR gate with inputs A,B, and C is a 1 when:
 - (a) A=1, B=1, C=1
 - (b) A=0, B=0, C=1
 - (c) A=0, B=0, C=0
 - (d) All
 - (e) only a and b
3. A pulse is applied to each input of a 2-input NAND gate. One pulse goes **HIGH** at $t=0$ and goes **LOW** at $t=1ms$, the other pulse goes **HIGH** at $t=0.8ms$ and goes back **LOW** at $t=3ms$. the output pulse can be described as follows:
 - (a) It goes **LOW** at $t=0$ and back **HIGH** at $t=3ms$.
 - (b) It goes **LOW** at $t=0.8ms$ and back **HIGH** at $t=3ms$.
 - (c) It goes **LOW** at $t=0.8ms$ and back **HIGH** at $t=1ms$.
 - (d) It goes **HIGH** at $t=0.8ms$ and back **LOW** at $t=1ms$.
4. when the input to an inverter is **HIGH**, the output is :
 - (a) **HIGH**
 - (b) **LOW**

Example: subtract $3A5_{(16)}$ from $592_{(16)}$.

First, covert $3A5$ to its 2's complement form by using either method presented above. The result is $C5B$. Then add this to 592 .



$$\begin{array}{r}
 592 \\
 + C5B \\
 \hline
 1\ 1ED
 \end{array}$$

← Discarded carry

Example: subtract the following hexadecimal numbers:

- (a) $84 - 2A$ (b) $C3 - 0B$

For branch (a): the 2's complement of $2A = D6$

$$\begin{array}{r}
 84 \\
 + D6 \\
 \hline
 1\ 5A
 \end{array}$$

← Drop carry **the difference is $5A_{(16)}$**

For branch (b): the 2's complement of $0B = F5$

$$\begin{array}{r}
 C3 \\
 + F5 \\
 \hline
 1\ B8
 \end{array}$$

← Drop carry **the difference is $B8_{(16)}$**

Multiplication of Hexadecimal Numbers:

The multiplication of hex numbers is well illustrated in the following example:

$3A$	right column :	$A_{(16)} \times F_{(16)} = 10_{(10)} \times 15_{(10)} = 150_{(10)} = 96_{(16)}$
$\times F$		$= 6$ with carry 9
$\hline 366$	left column :	$3_{(16)} \times F_{(16)} + 9_{(16)} = 3_{(10)} \times 15_{(10)} + 9_{(10)} = 45_{(10)} + 9_{(10)} = 54_{(10)}$
		$= 36_{(16)}$



Review Questions:

- perform the following binary additions:
(a) $1101+1010$ (b) $10111+01101$
- perform the following binary subtractions:
(a) $11101-0100$ (b) $1001-0111$
- perform the indicated binary operation:
(a) 110×111 (b) $1100 \div 011$
- determine the 1's complement of each binary number:
(a) 11010 (b) 001101
- determine the 2's complement of each binary number:
(a) 10111 (b) 010001
- subtract the hexadecimal numbers:
(a) $75_{(16)} - 21_{(16)}$ (b) $94_{(16)} - 5C_{(16)}$
- add the hexadecimal numbers directly:
(a) $18_{(16)} + 34_{(16)}$ (b) $3F_{(16)} + 2A_{(16)}$
- multiply the following pairs of binary numbers:
(a) 101.101×110.010 ***Ans.:100011.00101***
(b) 0.1101×0.1011 ***Ans.:0.10001111***
- perform the following divisions:
(a) $10110.1101 \div 1.1$ ***Ans.:1111.0011***
(b) $111111 \div 1001$ ***Ans.:111***



Digital Codes

1- Binary Coded Decimal (BCD):

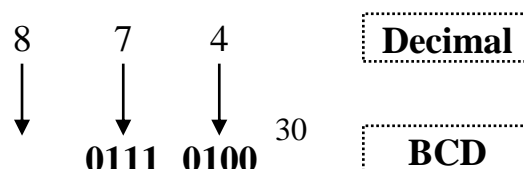
When numbers, letters, or words are represented by a special group of symbols, this is called encoding, and the group of symbols is called a code. Probably one of the familiar codes is the Morse code, where series of dots and dashes represent letters of the alphabet. We have seen that decimal numbers can be represented by an equivalent binary number. The group of 0s and 1s in the binary number can be thought of as a code representing the decimal number. When a decimal number is represented by its equivalent binary number, we call it (straight binary coding). We have seen that conversion between decimal and binary can become long and complicated for large numbers. For this reason, a means of encoding decimal numbers that combines some features of both the decimal and binary systems is used in certain situations.

The 8421 code is a type of binary coded decimal (BCD) code. Binary coded decimal means that each decimal digit, 0 through 9, is represented by a binary code of 4 bits. The designation 8421 indicates the binary weights of the four bits ($2^3, 2^2, 2^1, 2^0$). The ease of conversion between 8421 code numbers and the familiar decimal numbers is the main advantage of this code. All you have to remember are the ten binary combinations that represent the ten decimal digits as shown in Table (1). The 8421 code is the predominant BCD code, and when referring to BCD, it always means the 8421 code unless otherwise stated.

Table (1)

Decimal digit	0	1	2	3	4	5	6	7	8	9
BCD	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001

To illustrate the BCD code, take a decimal number such as 874. Each digit is changed to its binary equivalent as follows:





It's also important to understand that a BCD number is not the same as a straight binary number. a straight binary code takes the complete decimal number and represents it in binary; the BCD code converts each decimal digit to binary individually . To illustrate, take the number 137 and compare its straight binary and BCD codes.

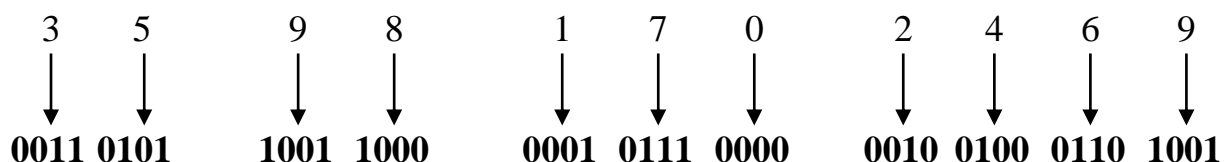
$$137_{(10)} = 10001001_{(2)} \quad \text{(Binary)}$$

$$137_{(10)} = 000100110111 \quad \text{(BCD)}$$

The BCD code requires 12 bits while the straight binary code requires only 8 bits to represent 137. BCD is used in digital machines whenever decimal information is either applied as inputs or displayed as outputs. Digital voltmeter, frequency counters, and digital clocks, all use BCD because they display output information in decimal. BCD is not often used in modern high speed digital computers for the reason that the BCD code for a given decimal number requires more bits that the straight binary code and is therefore less efficient. This is important in digital computers because the number of places in memory where these bits can be stored is limited.

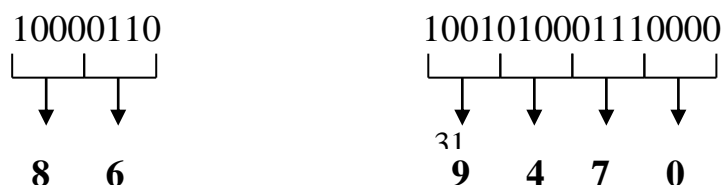
Example: convert each of the following decimal numbers to BCD:

- (a) 35 (b) 98 (c) 170 (d) 2469



Example: convert each of the following BCD codes to decimal.

- (a) 10000110 (b) 1001010001110000





2- Gray Code:

The gray code is un-weighted and is not an arithmetic code; that is, there are no specific weights assigned to the bit positions. The important feature of the Gray code is that it exhibits only a single bit change from one code number to the next. Table (2) is a listing of the four bit gray code for decimal numbers 0 through 15. Notice the single bit change between successive gray code numbers. For instance, in going from decimal 3 to decimal 4, the gray code changes from 0010 to 0110, while the binary code changes from 0011 to 0100, a change of three bits. The only bit change is in the third bit from the right in the gray code; the other remain the same.

Table (2)

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

Binary-to-Gray Conversion:

Conversion between binary code and Gray code is sometimes useful. in the conversion process, the following rules apply:

- ❖ **The most significant bit (left-most) in the gray code is the same as the corresponding MSB in binary number.**



- ❖ Going from left to right, add each adjacent pair of binary code bits to get the next gray code bit. Discard carry.

Example: convert the binary number 10110 to Gray code.

Step 1: the left-most Gray code digit is the same as the left-most binary code bit.

$$\begin{array}{r} 10110 \quad (\text{Binary}) \\ \downarrow \\ 1 \quad (\text{Gray}) \end{array}$$

Step 2: add the left-most binary code bit to the adjacent one:

$$\begin{array}{r} 1 + 0110 \quad (\text{Binary}) \\ \downarrow \\ 1 \quad 1 \quad (\text{Gray}) \end{array}$$

Step 3: add the next adjacent pair:

$$\begin{array}{r} 1 \quad 0 + 110 \quad (\text{Binary}) \\ \downarrow \\ 1 \quad 1 \quad 1 \quad (\text{Gray}) \end{array}$$

Step 4: add the next adjacent pair and discard the carry:

$$\begin{array}{r} 1 \quad 0 \quad 1 + 10 \quad (\text{Binary}) \\ \downarrow \\ 1 \quad 1 \quad 1 \quad 0 \quad (\text{Gray}) \end{array}$$

Step 5: add the last adjacent pair:

$$\begin{array}{r} 1 \quad 0 \quad 1 \quad 1 + 0 \quad (\text{Binary}) \\ \downarrow \\ 1 \quad 1 \quad 1 \quad 0 \quad 1 \quad (\text{Gray}) \end{array}$$

Hence the Gray Code is 11101

Gray-to-Binary Conversion:



To convert from Gray code to binary, a similar method is used, but there are some differences. The following rules apply:

- ❖ **The most significant bit (left-most) in the binary code is the same as the corresponding bit in the Gray code.**
- ❖ **Add each binary code bit generated to the gray code bit in the next adjacent positions. Discard carry.**

Example: convert the Gray code number 11011 to binary.

Step 1: the left-most bits are the same.

$$\begin{array}{r} 11011 \quad \text{(Gray)} \\ \downarrow \\ 1 \quad \text{(Binary)} \end{array}$$

Step 2: add the last binary code bit just generated to the gray code bit in the next position. Discard the carry.

$$\begin{array}{r} 1 \quad 1 \quad 011 \quad \text{(Gray)} \\ + \quad \downarrow \\ 1 \quad 0 \quad \quad \text{(Binary)} \end{array}$$

Step 3: add the last binary code bit generated to the next Gray code bit.

$$\begin{array}{r} 1 \quad 1 \quad 0 \quad 11 \quad \text{(Gray)} \\ + \quad \downarrow \\ 1 \quad 0 \quad 0 \quad \quad \text{(Binary)} \end{array}$$

Step 4: add the last binary code bit generated to the next Gray code bit.

$$\begin{array}{r} 1 \quad 1 \quad 0 \quad 1 \quad 1 \quad \text{(Gray)} \\ + \quad \downarrow \\ 1 \quad 0 \quad 0 \quad 1 \quad \quad \text{(Binary)} \end{array}$$

Step 5: add the last binary code bit generated to the next Gray code bit. discard carry.

$$\begin{array}{r} 1 \quad 1 \quad 0 \quad 1 \quad 1 \quad \text{(Gray)} \\ + \quad \downarrow \\ 1 \quad 0 \quad 0 \quad 1 \quad 0 \quad \text{(Binary)} \end{array}$$



Hence the final binary number is 10010

Example: (a) Convert the binary number 11000110 to Gray-code.
 (b) Convert the Gray-code 10101111 to binary.

(a) Binary to Gray code:-

1	+	1	+	0	+	0	+	0	+	1	+	1	+	0
↓		↓		↓		↓		↓		↓		↓		↓
1		0		1		0		0		1		0		1

(b) Gray code to Binary:-

1	0	1	0	1	1	1	1
↓ ↘	↓ ↘	↓ ↘	↓ ↘	↓ ↘	↓ ↘	↓ ↘	↓ ↘
1	1	0	0	1	0	1	0

3- Excess-3 Code:

This is a digital code related to BCD that is derived by adding 3 to each decimal digit and then converting the result of that addition to 4-bit binary. This code is un-weighted. For instance, the excess-3 code for decimal 2 and 9 are:

2	
+ 3	
5	→ (0101)

9	
+ 3	
12	→ (1100)



The excess-3 code for each decimal digit is found by the same procedure. the entire code is shown in Table (3).

Table (3)

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

Example: convert each of the following decimal number to Excess-3 code:

- (a) 13 (b) 430

First, add 3 to each digit in the decimal number, and then convert each resulting 4-bit sum to its equivalent binary code.

$$\begin{array}{r}
 1 \quad 3 \\
 +3 \quad +3 \\
 \hline
 4 \quad 6 \\
 \downarrow \quad \downarrow \\
 0100 \quad 0110 \quad (\text{Excess-3})
 \end{array}$$

$$\begin{array}{r}
 4 \quad 3 \quad 0 \\
 +3 \quad +3 \quad +3 \\
 \hline
 7 \quad 6 \quad 3 \\
 \downarrow \quad \downarrow \quad \downarrow \\
 0111 \quad 0110 \quad 0011 \quad (\text{Excess-3})
 \end{array}$$

4- Alphanumeric Code:



In order to be very useful, a computer must be capable of handling non-numeric information. In other words, a computer must be able to recognize codes that represent numbers, letters, and special characters. These codes are classified as alphanumeric codes. The most common alphanumeric code, known as the American Standard Code for Information Interchange (ASCII), is used by most minicomputer and microcomputer manufacturers.

The ASCII is a seven-bit code in which the decimal digits are represented by the 8421 BCD code preceded by 011. The letters of the alphabet and other symbols and instructions are represented by other code combinations, shown in Table (4). For instance, the letter **A** is represented by 1000001 (41_{16}), the **comma** by 0101100 ($2C_{16}$) and the **ETX** (end of text) by 0000011 (03_{16}).

Table (4)

LSBs	MSBs							
	000 (0)	001 (1)	010 (2)	011 (3)	100 (4)	101 (5)	110 (6)	111 (7)
0000	NUL	DLE	SP	0	@	P	`	p
0001	SOH	DC1	!	1	A	Q	a	q
0010	STX	DC2	“	2	B	R	b	r
0011	ETX	DC3	#	3	C	S	c	s
0100	EOT	DC4	\$	4	D	T	d	t
0101	ENQ	NAK	%	5	E	U	e	u
0110	ACK	SYN	&	6	F	V	f	v
0111	BEL	ETB	‘	7	G	W	g	w
1000	BS	CAN	(8	H	X	h	x
1001	HT	EM)	9	I	Y	i	y
1010	LF	SUB	*	:	J	Z	j	z
1011	VT	EXC	+	;	K	[k	{
1100	FF	FS	,	<	L	\	l	



1101	CR	GS	-	=	M]	m	}
1110	SO	RS	.	>	N	↑	n	~
1111	SI	US	/	?	O	_	o	DEL

Example: determine the codes that are entered from the computer's keyboard when the following basic program statement is typed in. also express each in hexadecimal notation.

20 PRINT "A=";X

<i>Character</i>	<i>ASCII</i>	<i>Hexadecimal</i>
2	0110010	32
0	0110000	30
Space	0100000	20
P	1010000	50
R	1010010	52
I	1001001	49
N	1001110	4E
T	1010100	54
Space	0100000	20
“	0100010	22
A	1000001	41
=	0111101	3D
“	0100010	22
;	0111011	3B
X	1011000	58



Review Questions:

1. How many bits are required to represent the decimal numbers in the range from 0 to 999 using straight binary code? Using BCD code?
2. What is the binary weight of each 1 in the following BCD numbers?
(a) 0010 (b) 1000 (c) 0001 (d) 0100
3. Convert the following binary numbers to Gray codes?
(a) 1100 (b) 1010 (c) 11010
4. Convert the following Gray codes to binary?
(a) 1000 (b) 1010 (c) 11101
5. Convert the following decimal numbers to Excess-3 code?
(a) 3 (b) 87 (c) 349
6. Convert decimal 928 to Excess-3?

Boolean algebra:

Boolean algebra is the mathematics of digital systems. It is important that you understand its principles thoroughly because a basic knowledge of Boolean algebra is indispensable to the study and analysis of logic circuits. The Boolean expressions for the previously studied gates are as follows:

NOT Gate: if input is A , the output is \bar{A}

AND Gate: if inputs are A and B , the output is $A.B$



OR Gate : if inputs are A and B , the output is $A+B$

NAND Gate: if inputs are A and B , the output is $\overline{A.B}$

A	B	AB	$A+AB$
0	0	0	0

NOR Gate : if inputs are A and B , the output is $\overline{A+B}$

XOR Gate : if inputs are A and B , the output is $\overline{A}B + A\overline{B} = A \oplus B$

XNOR Gate: if inputs are A and B , the output is $\overline{\overline{A}B + A\overline{B}} = A \odot B$

Laws of Boolean algebra:

1. Commutative law: $\begin{cases} \rightarrow A + B = B + A \\ \rightarrow AB = BA \end{cases}$
2. Associative law: $\begin{cases} \rightarrow A + (B + C) = (A + B) + C \\ \rightarrow A(BC) = (AB)C \end{cases}$
3. Distributive law: $\rightarrow A(B + C) = AB + AC$

Rules for Boolean algebra:

Table (1) lists 12 basic rules that are useful in manipulating and simplifying Boolean expressions.

Table (1)

1.	$A+0=A$	7.	$A.A=A$
2.	$A+1=1$	8.	$A.\overline{A}=0$
3.	$A.0=0$	9.	$\overline{\overline{A}}=A$
4.	$A.1=A$	10.	$A+AB=A$
5.	$A+A=A$	11.	$A+\overline{A}B=A+B$
6.	$A+\overline{A}=1$	12.	$(A+B)(A+C)=A+BC$

Example: Prove that $A+AB=A$.



$$A + AB = A(1 + B)$$

$$= A.1 = A$$

0	1	0	0
1	0	0	1
1	1	1	1



Example: Prove that $A + \overline{A}B = A + B$.

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

$$= (AA + AB) + \overline{A}B$$

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

$$= (A + \overline{A})(A + B)$$

$$= A + B$$

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

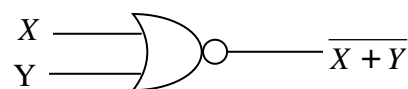


DeMorgan's Theorem:

Two of the most important theorems of Boolean algebra were contributed by a great mathematician named DeMorgan. DeMorgan's theorems are extremely useful in simplifying expressions in which a product of sum of variables is inverted. The two theorems are:

1- $\overline{X + Y} = \overline{X} \cdot \overline{Y}$

X	Y	$\overline{X + Y}$	$\overline{X} \cdot \overline{Y}$
0	0	1	1
0	1	0	0
1	0	0	0
1	1	0	0

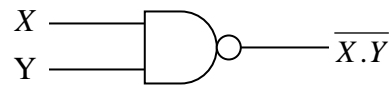


≡

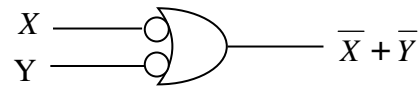


2- $\overline{X \cdot Y} = \overline{X} + \overline{Y}$

X	Y	$\overline{X.Y}$	$\overline{X + Y}$
0	0	1	1
0	1	1	1
1	0	1	1
1	1	0	0



≡



Example: Apply DeMorgan's theorem to the expressions: \overline{WXYZ} and $\overline{W + X + Y + Z}$.

$$\overline{WXYZ} = \overline{W} + \overline{X} + \overline{Y} + \overline{Z}$$

$$\overline{W + X + Y + Z} = \overline{W} \overline{X} \overline{Y} \overline{Z}$$

Example: Apply DeMorgan's theorem to each of the following expressions:

(a) $\overline{(A + B + C)D}$

(b) $\overline{\overline{AB} + \overline{CD} + EF}$

$$\begin{aligned} \overline{(A + B + C)D} &= \overline{(A + B + C)} + \overline{D} \\ &= \overline{A} \cdot \overline{B} \cdot \overline{C} + \overline{D} \end{aligned}$$

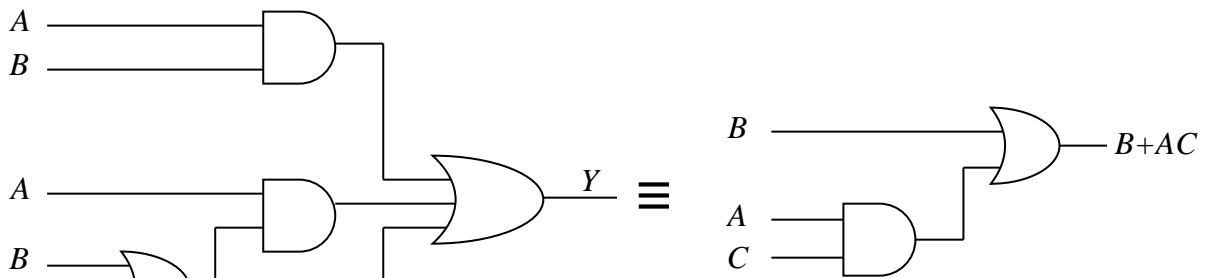
$$\begin{aligned} \overline{\overline{AB} + \overline{CD} + EF} &= \overline{(\overline{AB})(\overline{CD})(EF)} \\ &= \overline{(\overline{A} + B)(\overline{C} + D)(\overline{E} + F)} \end{aligned}$$

Simplification using Boolean algebra:

Many times in the application of Boolean algebra, we have to reduce a particular expression to its simplest form or change its form to a more convenient one to implement the expression most efficiently. The purpose of simplifying Boolean expression is to use the fewest gates possible to implement a given expression.

Example: Simplify the following expression: $Y = AB + A(B + C) + B(B + C)$.

$$\begin{aligned} AB + A(B + C) + B(B + C) &= AB + AB + AC + BB + BC \\ &= AB + AB + AC + B + BC \\ &= AB + AC + B + BC \\ &= AB + AC + B = B + AC \end{aligned}$$



Exa C le: Siy Verify the expression:

$$\overline{AB}(B + \overline{ABC} + \overline{ABC}) + \overline{ABC} + \overline{ABC}$$

$$BC(\overline{A} + A) + \overline{ABC} + \overline{ABC} + \overline{ABC}$$

$$BC.1 + \overline{AB}(C + \overline{C}) + \overline{ABC}$$

$$BC + \overline{AB}.1 + \overline{ABC} = BC + \overline{B}(A + \overline{AC}) = BC + \overline{B}(A + \overline{C}) = BC + \overline{AB} + \overline{BC}$$

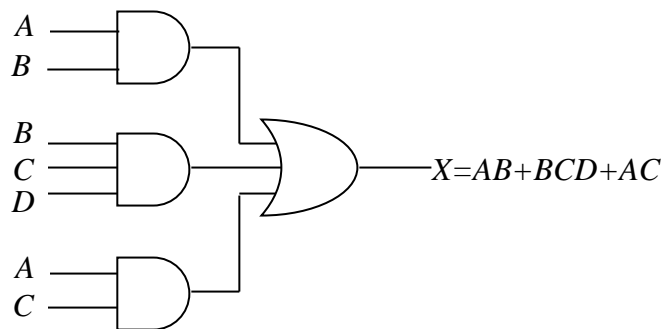
The Sum-of-Product (SOP) form:

When two or more product terms are summed by Boolean addition, the resulting expression is a sum of product (SOP). Some examples are:

$$AB + ABC$$

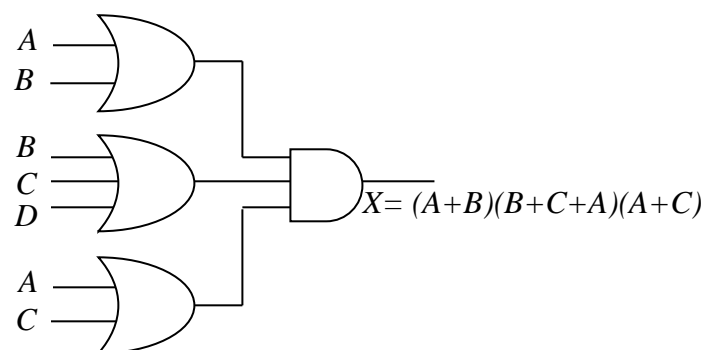
$$ABC + CDE + \overline{BCD}$$

$$\overline{AB} + \overline{ABC} + AC$$



The Product-of-Sum (POS) form:

When two or more sum terms are multiplied, the resulting expression is a product of sum (POS). Some examples are:



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

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

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

Example: Convert each of the following expression to **general** SOP form.

(a) $AB + B(CD + EF)$

(b) $\overline{(A + B)} + C$

(a): $AB + B(CD + EF) = AB + BCD + BEF$

(b): $\overline{(A + B)} + C = \overline{(A + B)}\bar{C} = (A + B)\bar{C} = A\bar{C} + B\bar{C}$

Example: Convert the following expression into **standard** SOP form.

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

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

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

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

$$\text{hence: } \bar{A}\bar{B}\bar{C} + \bar{A}\bar{B} + A\bar{B}\bar{C}D = \bar{A}\bar{B}\bar{C}D + \bar{A}\bar{B}\bar{C}\bar{D} + \bar{A}\bar{B}CD + \bar{A}\bar{B}C\bar{D} + \bar{A}\bar{B}\bar{C}D + \bar{A}\bar{B}\bar{C}\bar{D} + A\bar{B}\bar{C}D$$

Example: Convert the following expression into **standard** POS form.

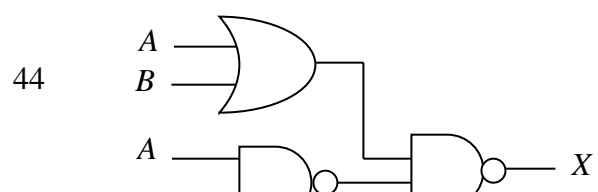
$$(A + \bar{B} + C)(\bar{B} + C + \bar{D})(A + \bar{B} + \bar{C} + D)$$

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

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

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

Example: Determine if the circuit gives XOR, XNOR or neither in the output.

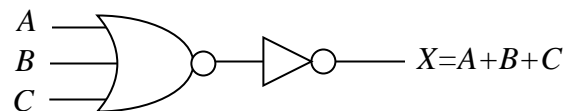


$$\begin{aligned}
 X &= \overline{(A + B) \cdot (\overline{A \cdot B})} \\
 &= \overline{(A + B)} + \overline{(\overline{A \cdot B})} \\
 &= \overline{A} \overline{B} + AB \equiv \text{XNOR Gate}
 \end{aligned}$$

Example: Determine the Boolean expression for a three-input NOR gate followed by an inverter.

The expression at the NOR output is $\overline{(A + B + C)}$, which is then fed through an inverter to produce:

$$X = \overline{\overline{(A + B + C)}} = A + B + C$$



Example: Simplify the expression $Y = \overline{A} \overline{B} D + \overline{A} B \overline{D}$.

$$Y = \overline{A} \overline{B} (D + \overline{D}) = \overline{A} \overline{B} \cdot 1 = \overline{A} \overline{B}$$

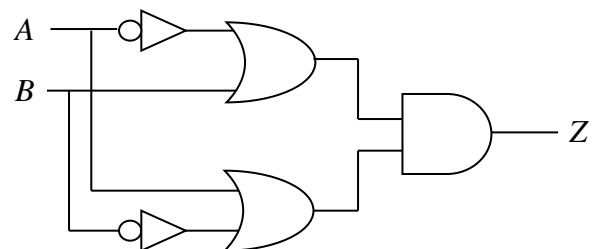
Example: Simplify the expression $X = ACD + \overline{A} BCD$.

$$X = CD(A + \overline{A} B) = CD(A + B) = ACD + BCD$$

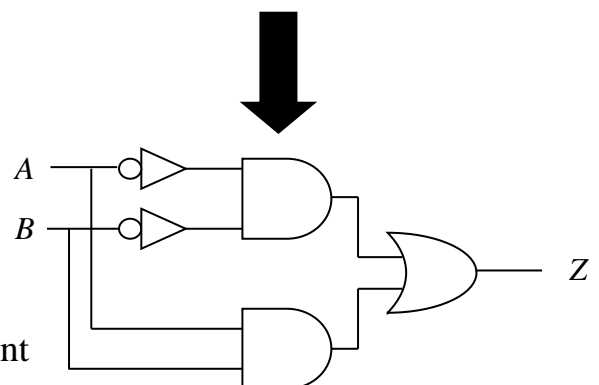
Example: Simplify the logic circuit shown in the Figure below.

The expression for output Z is:

$$\begin{aligned}
 Z &= (\overline{A} + B)(A + \overline{B}) \\
 &= \overline{A} A + \overline{A} \overline{B} + BA + B \overline{B} \\
 &= \overline{A} \overline{B} + AB
 \end{aligned}$$



If we compare the resulting circuit with the original one, we see that both circuits contain the same number of gates and connections. In this case, the simplification process produced an equivalent, but not simpler circuit. Also one could notice that the resulting output is equivalent





to exclusive NOR gate.

Example: Develop truth table for the expression $\overline{ABC} + \overline{ABC} + ABC$.

There are three variables in the domain, so there are eight possible binary values of the variables as listed in the left columns of Table (2).

Table (2)

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

Review Questions:

1. Apply DeMorgan's theorem to the following expressions:

(a) $\overline{ABC} + (\overline{D + E})$

(b) $\overline{A + B + C} + \overline{DE}$

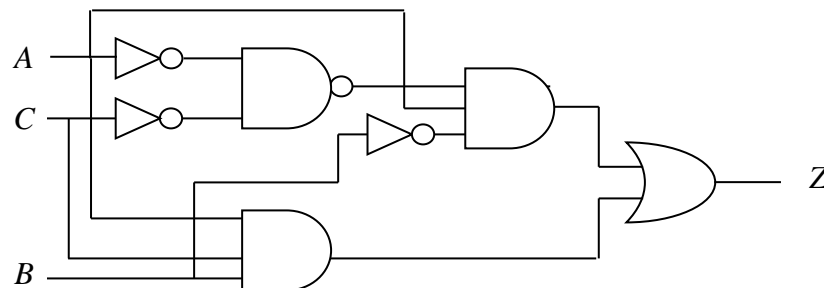
(c) $\overline{\overline{\overline{ABC}}(\overline{\overline{\overline{EFG}}})} + \overline{\overline{\overline{HIJ}}(\overline{\overline{\overline{KLM}}})}$

(d) $\overline{\overline{\overline{A + BC + CD}}} + \overline{BC}$

2. Simplify the following expressions:

- (a) $A + AB + \overline{A}BC$ Ans : A
- (b) $(\overline{A} + B)C + ABC$ Ans : $C(\overline{A} + B)$
- (c) $\overline{A}BC(BD + CDE) + A\overline{C}$ Ans : $A(\overline{C} + \overline{B}DE)$
- (d) $(\overline{A} + B)(A + B)$ Ans : B
- (e) $A\overline{C} + ABC$ Ans : $A\overline{C}$
- (f) $\overline{A}BCD + \overline{A}BC\overline{D}$ Ans : $\overline{A}BD$
- (g) $(\overline{A} + B)(A + B + D)\overline{D}$ Ans : $B\overline{D}$
- (h) $\overline{A}C(\overline{A}BD) + \overline{A}BC\overline{D} + \overline{A}BC$ Ans : $\overline{B}C + \overline{A}D(C + B)$
- (i) $ABC + ABC\overline{C} + A\overline{B}C$ Ans : $A(B + C)$

3. Simplify the logic circuit shown in figure below: (ANS: $A(\overline{B} + C)$)



4. Draw the logic circuit represented by each expression.

- (a) $AB + \overline{A}B + \overline{A}BC$ (b) $A + B[C + D(B + \overline{C})]$

5. Convert the following expressions to sum-of-product form:

- (a) $(A + B)(C + \overline{B})$ (b) $(A + \overline{B}C)C$

6. Develop a truth table for each of the SOP expressions:

- (a) $\overline{A}B + ABC\overline{C} + \overline{A}C + A\overline{B}C$ (b) $\overline{X} + Y\overline{Z} + WZ + X\overline{Y}Z$

7. Prove that $(A+B)(A+C)=A+BC$.



The Karnaugh Map (K-Map):

The Karnaugh map provides a systematic method for simplifying Boolean expression and, if properly used, will produce the simplest SOP or POS expression possible. The K-map, like a truth table, is a means for showing the relationship between logic inputs and the desired output. The K-map is an array of **Cells** in which each cell represents a binary value of the input variables. The cells are arranged in a way so that simplification of a given expression is simply a matter of properly grouping the cells. The number of cells in a Karnaugh map is equal to the total number of possible input variable combinations as is the number of rows in a truth table. For three variables, the number of cells is $2^3=8$. For four variables, the number of cells is $2^4=16$.

Three examples of K-maps for two, three, and four variables, together with the corresponding truth tables are shown in Figure below:

A	B	X
0	0	1
0	1	0
1	0	0
1	1	1

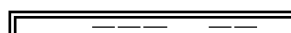
$\overline{A}\overline{B}$
 \longrightarrow
 $\boxed{X = \overline{A}\overline{B} + AB}$
 AB

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

A	B	C	X
0	0	0	1

$\longrightarrow \overline{A}\overline{B}\overline{C}$
 $\longrightarrow \overline{A}\overline{B}C$
 $\longrightarrow \overline{A}B\overline{C}$

$\cdot 8$





0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

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

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

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

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

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

$ABCD$

$\bar{C}\bar{D}$ $\bar{C}D$ CD $C\bar{D}$

$\bar{A}\bar{B}$

$\bar{A}B$

AB

$A\bar{B}$

0	1	0	0
0	1	0	0
0	1	1	0
0	0	0	0

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

Karnaugh Map SOP Minimization:

As stated in the last section, the Karnaugh map is used for simplifying Boolean expressions to their minimum form. A minimized SOP expression contains the fewest possible terms with the fewest possible variables per term. Generally, a minimum SOP expression can be implemented with fewer logic gates than a standard expression and this is the basic purpose in the simplification process.

Example: Map the following SOP expression on a Karnaugh map:

$$\overline{\overline{A}BC} + \overline{A}\overline{B}C + A\overline{B}\overline{C} + ABC$$

$\overline{\overline{A}BC}$	→	001		\overline{AB}		1
$\overline{A}\overline{B}C$	→	010		\overline{AB}	1	
$A\overline{B}\overline{C}$	→	110		AB	1	1
ABC	→	111		AB		
				\overline{AB}		

Example: Map the following SOP expression on a Karnaugh map:

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

\overline{A}	→	$\overline{\overline{A}BC}$	000		\overline{AB}	1	1
	→	$\overline{A}\overline{B}C$	001		\overline{AB}	1	
	→	$\overline{A}BC$	010		AB	1	1
	→	$\overline{A}BC$	011		AB	1	1
\overline{AB}	→	$\overline{A}\overline{B}\overline{C}$	100		\overline{AB}	1	1
	→	$\overline{A}\overline{B}C$	101				
$A\overline{B}\overline{C}$	→	$\overline{A}B\overline{C}$	110				

Looping:

The expression for output X can be simplified by properly combining these squares in the K-map which contains 1's. The process for combining these 1's is called **Looping**. Looping a pair of adjacent 1's in the K-map eliminates the variable that appears in complemented and un-complemented form as shown in the examples below:-

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

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

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

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

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

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

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

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

$$= \bar{A}\bar{B} \quad \bar{C} \quad C \quad \bar{C} \quad C$$

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

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

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



The K-map may contain a group of four 1's that are adjacent to each other. This group is called a **quad**. The simplification of such groups is shown in the examples below:

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

$$X = C$$

	$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$
$\bar{A}\bar{B}$	0	0	0	0
$\bar{A}B$	0	1	1	0
AB	0	1	1	0
$A\bar{B}$	0	0	0	0

$$X = BD$$

	$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$
$\bar{A}\bar{B}$	0	0	0	0
$\bar{A}B$	0	0	0	0
AB	1	1	1	1
$A\bar{B}$	0	0	0	0

$$X = AB$$

	$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$
$\bar{A}\bar{B}$	0	0	0	0
$\bar{A}B$	0	0	0	0
AB	1	0	0	1
$A\bar{B}$	1	0	0	1

$$X = A\bar{D}$$

	$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$
$\bar{A}\bar{B}$	1	0	0	1
$\bar{A}B$	0	0	0	0
AB	0	0	0	0
$A\bar{B}$	1	0	0	1

$$X = \bar{B}\bar{D}$$



Example: Simplify the following equation using Boolean algebra and Karnaugh map:
map: $X = B.(A + \bar{B}).(B + C)$

1- Using Boolean algebra

$$\begin{aligned} X &= B.(A + \bar{B}).(B + C) \\ &= (AB + 0).(B + C) \\ &= AB(B + C) \\ &= AB + ABC \\ &= AB(1 + C) \\ &= AB \end{aligned}$$

2- Using Karnaugh map

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

$$\begin{aligned} X &= AB + ABC \\ \text{But } AB &= ABC + AB\bar{C} \\ X &= ABC + AB\bar{C} + ABC \\ &= ABC + AB\bar{C} \end{aligned}$$

Example: Use a Karnaugh map to minimize the SOP expression:-

$$\bar{A}\bar{B}C + \bar{A}BC + A\bar{B}\bar{C} + A\bar{B}C + A\bar{B}C$$

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

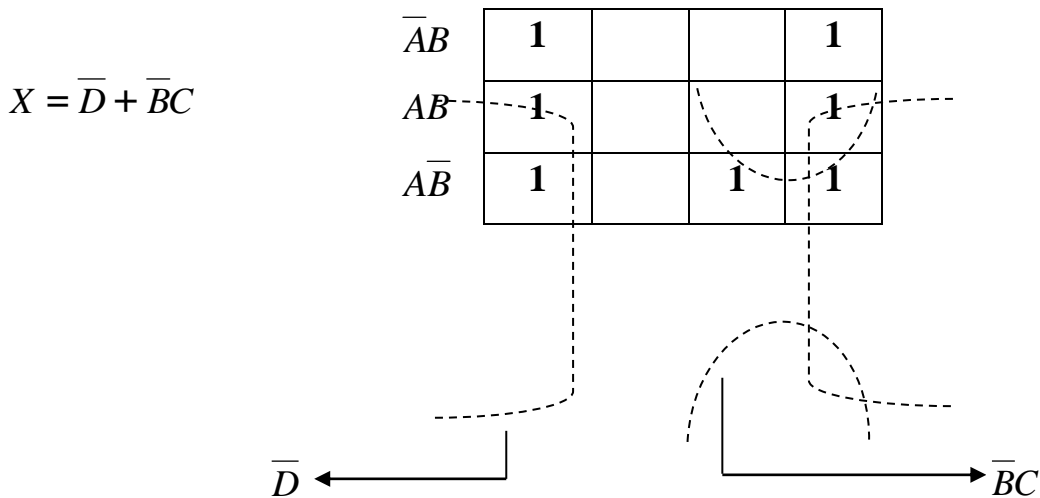
	\bar{C}	C	
$\bar{A}\bar{B}$	1	1	→ $\bar{A}C$
$\bar{A}B$		1	
AB			
$A\bar{B}$	1	1	→ \bar{B}

Example: Use a Karnaugh map to minimize the SOP expression:-

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

The first term $\bar{B}\bar{C}\bar{D}$ must be expanded into $\bar{A}\bar{B}\bar{C}\bar{D}$ and $A\bar{B}\bar{C}\bar{D}$ to get a standard SOP expression which is then mapped and the cells are grouped as shown in figure below:

	$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$
$\bar{A}\bar{B}$	1		1	1



Don't Care Condition:

Some logic circuit can be designed so that there are certain input conditions for which there are no specified output levels, usually because these conditions will never occur. In other words, there will be certain combinations of input levels where we “don't care” whether the output is **HIGH** or **LOW**. This is illustrated in the truth table below:-

A	B	C	Z
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	x
1	0	0	x
1	0	1	1
1	1	0	1
1	1	1	1

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

Whenever “don't care” conditions occur, we have to decide which ones to change to 0 and which to 1 to produce the best K-map looping (i.e. the simplest expression).

Karnaugh Map POS Minimization

In this section, we will focus on POS expressions instead of SOP. The approaches are much the same except that with POS expressions, 0's representing the standard sum terms are placed on the karnaugh map instead of 1's.

Example: Map the following POS expression on a karnaugh map.

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

		1100	1011	0010	1111	0011
	CD	00	01	11	10	
AB	CD	$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$	
00	$\bar{A}\bar{B}$			0	0	
01	$\bar{A}B$					
11	AB	0		0		
10	$A\bar{B}$			0		

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

Example: Use a karnaugh map to minimize the POS expression:

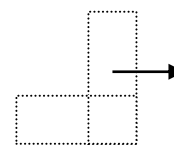
$$(A + B + C)(A + B + \bar{C})(A + \bar{B} + C)(A + \bar{B} + \bar{C})(\bar{A} + \bar{B} + C)$$

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

keep in mind that this minimum POS expression is equivalent to the original standard POS expression grouping the 1's as shown yields a SOP expression that is equivalent to grouping the 0's.

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

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



Review Questions:

1. Use a Karnaugh map to minimize the POS expression:

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

2. Map the following SOP expression on a karnaugh map:

$$\bar{B}\bar{C} + A\bar{B} + ABC\bar{C} + \bar{A}BC\bar{D} + \bar{A}\bar{B}CD + \bar{A}BCD$$

3. What is the difference in mapping a POS expression and an SOP expression?

4. What is the standard sum term for a 0 in cell 1011?

5. What is the standard product term for a 1 in cell 0010?

6. Determine the minimum expression for each K-map in figure below:

	$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$
$\bar{A}\bar{B}$	1	1	1	1
$\bar{A}B$	1	1	0	0
AB	0	0	0	1
$A\bar{B}$	0	1	1	0

	$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$
$\bar{A}\bar{B}$	1	0	1	1
$\bar{A}B$	1	0	0	1
AB	0	0	0	0
$A\bar{B}$	1	0	1	1

7. Use a K-map to simplify each expression to a minimum SOP form:

(a) $\bar{A}\bar{B}\bar{C} + A\bar{B}C + \bar{A}BC + ABC\bar{C}$

Ans : No Simplification

(b) $AC[\bar{B} + B(B + \bar{C})]$

Ans : AC

(c) $DEF + \bar{D}\bar{E}\bar{F} + \bar{D}EF$

Ans : $\bar{D}\bar{F} + E\bar{F}$

8. Reduce the function specified in the truth table in figure below to its minimum SOP form by using K-map.

Ans: $\bar{B} + C$

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

9. Use a k-map to simplify each expression to a minimum POS form:

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

(b) $(X + \bar{Y})(W + \bar{Z})(\bar{X} + \bar{Y} + \bar{Z})(W + X + Y + Z)$

10. Simplify the following equation: $f(A, B, C, D) = \sum(1, 3, 7, 11, 15)$.

11. Simplify the following equation:

$$f(A, B, C, D) = \sum(1, 5, 6, 11, 14, 15)$$

$dm(2, 3, 7)$ where dm is denoted for don't care.

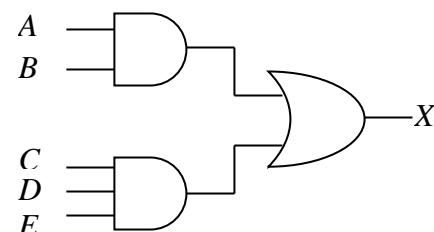
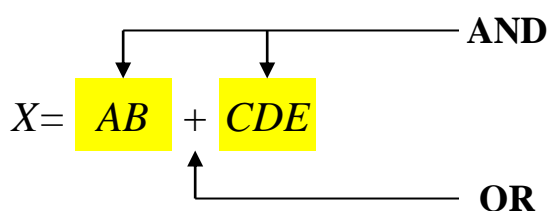
Implementation of Logic Circuit:

In this section, several general examples are used to illustrate how to implement a logic circuit from a Boolean expression or a truth table.

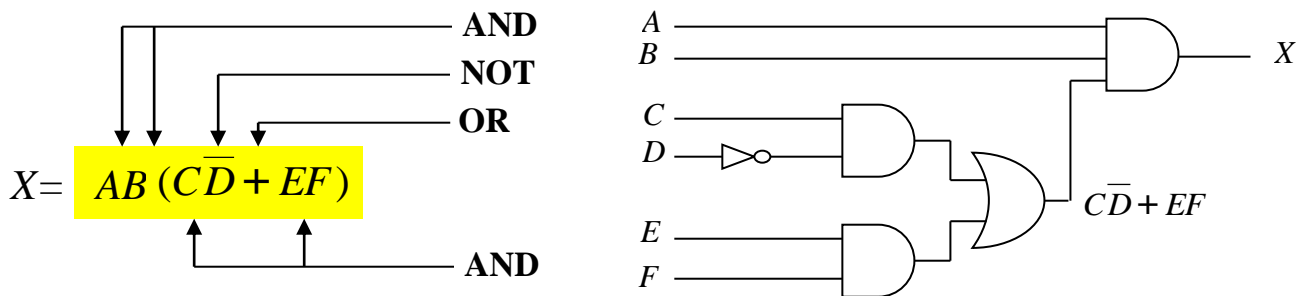
1- From Boolean Expression to a Circuit:

The following expression is the first in the series of examples that will be considered:- $X = AB + CDE$

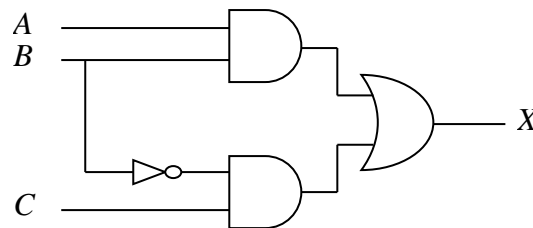
This expression is composed of two terms, AB and CDE , with a total of five variables. The first term is formed by ANDing A with B , and the second term is formed by ANDing C, D , and E . The two terms are then ORed to form the output X . These operations are indicated in the structure of the expression as follows:



Example: Implement the following expression: $X = AB(C\bar{D} + EF)$



Example: Draw the circuit diagram that implements the expression: $X = AB + \bar{B}C$



2- From Truth Table to a Circuit:

Beginning with a truth table, we will write the SOP expression from the truth table and then implement the logic circuit. The table below specifies a logic function. The expression is obtained from a truth table by ORing the product terms for all occurrences of $X=1$. The expression is: $X = \bar{A}BC + A\bar{B}\bar{C}$

Input			Output
A	B	C	X
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1

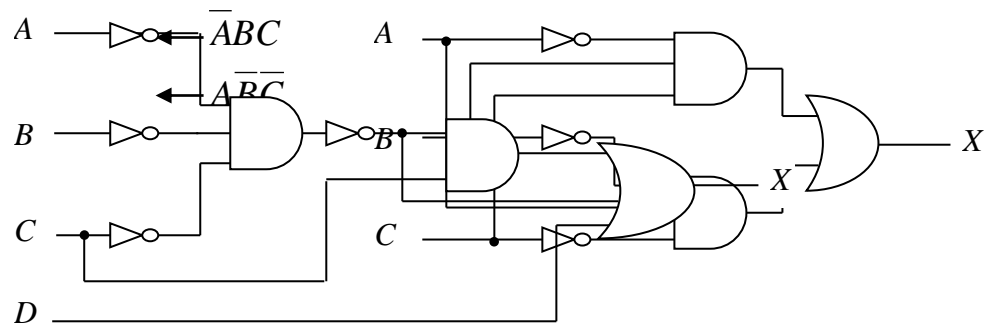
1	0	1	0
1	1	0	0
1	1	1	0

The logic gates required to implement this expression are as follows:

Three inverters to form the \bar{A} , \bar{B} and \bar{C} variables; two 3-input AND gates to form the term $\bar{A}\bar{B}\bar{C}$ and $\bar{A}BC$; and one 2-input OR gate to form the final output.

Example: Reduce the combination

form:



The expression for the output of the circuit is: $X = (\bar{A}\bar{B}\bar{C})C + \bar{A}\bar{B}C + D$

Applying DeMorgan's theorem and Boolean algebra:

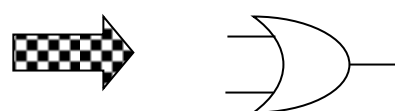
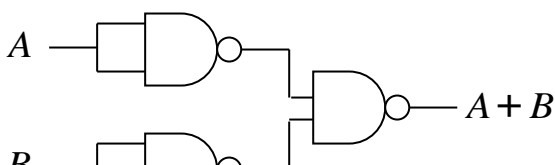
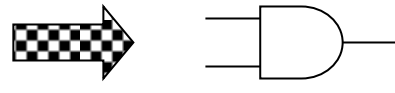
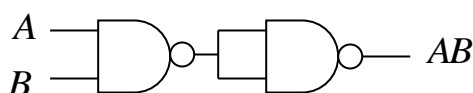
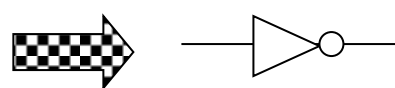
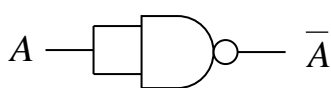
$$\begin{aligned} X &= (\bar{A} + \bar{B} + \bar{C})C + \bar{A} + \bar{B} + \bar{C} + D \\ &= AC + BC + CC + A + B + C + D = AC + BC + C + A + B + C + D \\ &= C(A + B + 1 + 1) + A + B + D = A + B + C + D \end{aligned}$$

Hence, the simplified circuit is a **4-input OR gate**.

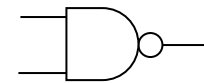
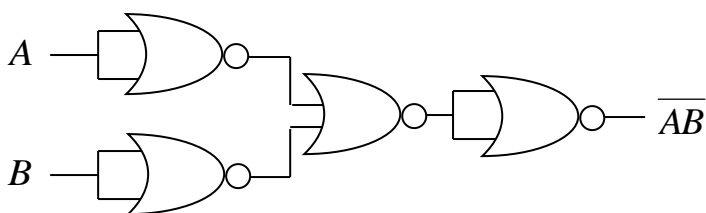
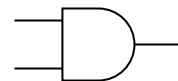
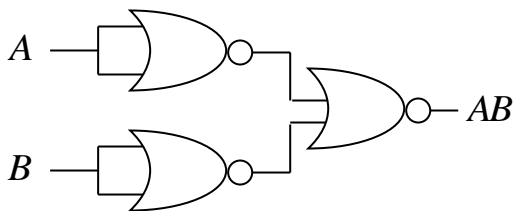
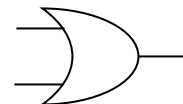
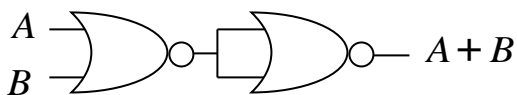
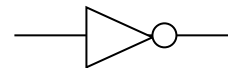
Universal Property of NAND and NOR Gates:

Up to this point, combinational circuits implemented with AND gates, OR gates and inverters have been studied. In this section, the universal property of the NAND gate and the NOR gate is introduced. The universality of the NAND gate means that it can be used as an inverter and that combinations of NAND gates can be used to implement the AND, OR and NOR operations. Similarly, the NOR gate can be used to implement the inverter, AND, OR and NAND operations.

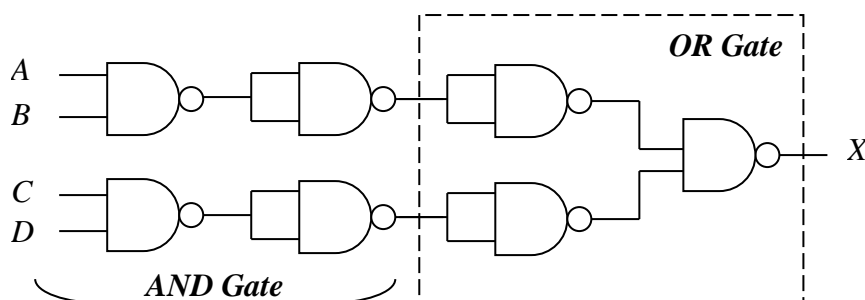
1- Using NAND Gate:

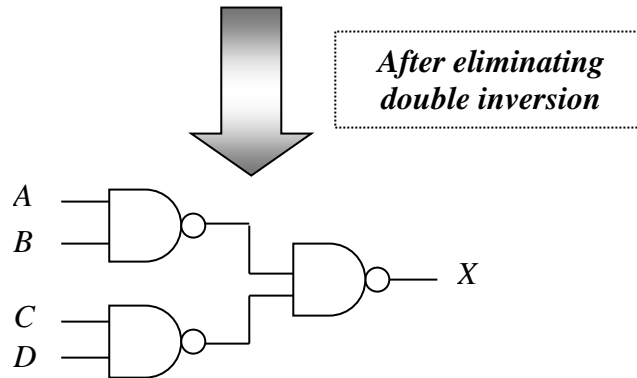


2- Using NOR Gate:



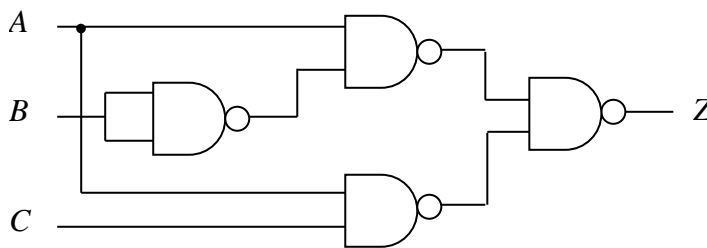
Example: Design using NAND gates the following expression: $X = AB + CD$





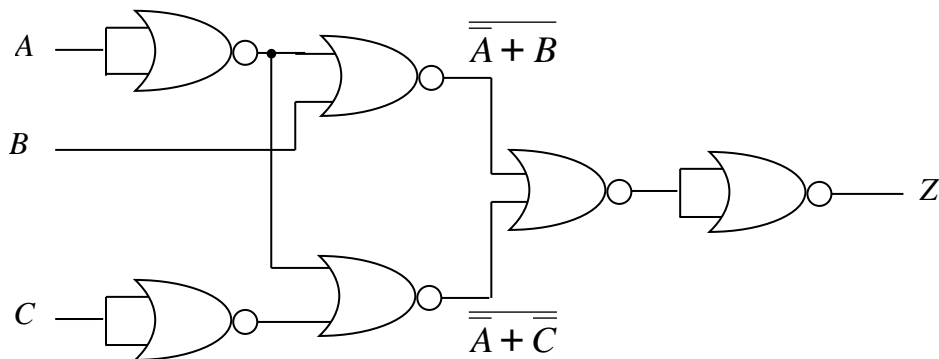
Example: Design $Z = AC + A\bar{B}$ using NAND gates only:

$$Z = \overline{\overline{AC + A\bar{B}}} = \overline{\overline{AC} \cdot \overline{A\bar{B}}} \quad (\text{Conversion to Product form})$$

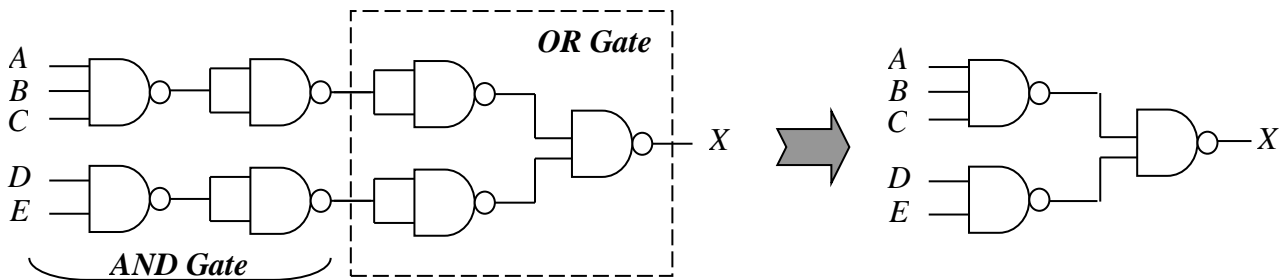


Example: Design $Z = AC + A\bar{B}$ using NOR gates only:

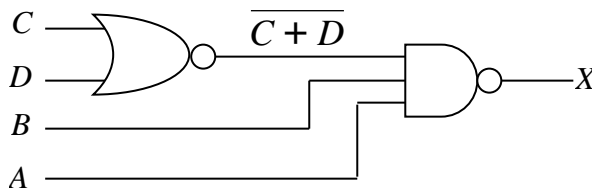
$$Z = \overline{\overline{AC} + \overline{A\bar{B}}} = \overline{\overline{A + C} + \overline{A + B}} \quad (\text{Conversion to Sum form})$$



Example: Implement the following expression with NAND gates only:
 $ABC + DE$



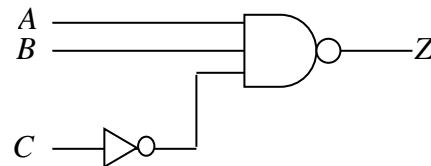
Example: Implement the logic circuit that has the expression
 $X = AB.(C + D)$ using only one NAND and NOR gate.



Example: Implement a circuit having the output expression $Z = \bar{A} + \bar{B} + C$ using only one NAND and an inverter gate.

$Z = \bar{A} + \bar{B} + C$ can be written as:

$$Z = \overline{\overline{\bar{A} + \bar{B} + C}} = \overline{A.B.C}$$



Review Questions:

8. Implement the following logic expressions:

(a) $X = ABC + AB + AC$

(b) $X = AB(C + DE)$

(c) $X = B(C\bar{D}\bar{E} + \bar{E}FG)(\bar{A}\bar{B} + C)$

(d) $X = \overline{ABC} + B(EF + \bar{G})$

9. Use NAND gates to implement::

(a) $X = \bar{A} + B$

(b) $X = \bar{A}\bar{B}$

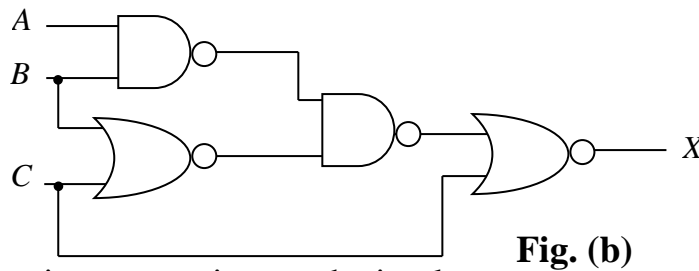
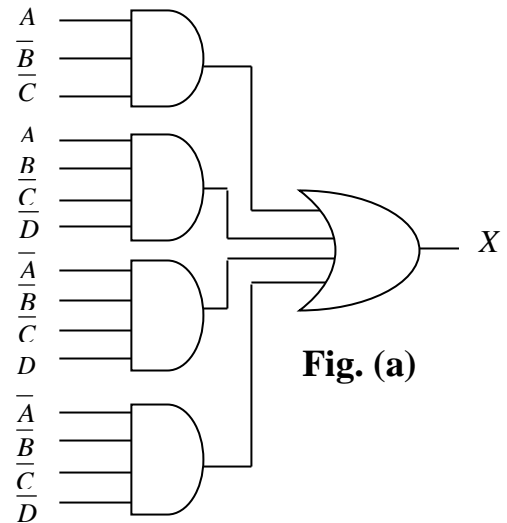
10. Implement the expression $X = \overline{(\bar{A} + \bar{B} + \bar{C})DE}$ by using NAND logic.

11. Implement the expression $X = \overline{\overline{ABC} + (D + E)}$ with NOR logic.

12. Design a logic circuit to implement the operation specified in the truth table indicated by Table (a):
13. Minimize the combinational logic circuit shown in Fig. (a):
14. Implement the logic circuit in Fig. (b) using only NAND gates. Repeat the design using only NOR gates.

Table (a)

Input			Output
A	B	C	X
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0



15. Show how the following expression can be implemented as stated using only NAND gates:
- (a) $X = ABC$
 - (b) $X = \overline{ABC}$
 - (c) $X = A + B$
 - (d) $X = A + B + \overline{C}$
 - (e) $X = \overline{AB} + \overline{CD}$
 - (f) $X = (A + B)(C + D)$
 - (g) $X = AB[C(\overline{DE} + \overline{AB}) + \overline{BCE}]$
16. Implement each function by using only NAND gates:
- (a) $X = \overline{AB} + AB$
 - (b) $X = A[BC(A + B + C + D)]$



17. Minimize the gates required to implement the function in SOP form:

(a) $X = \overline{A}B + AB$

(b) $X = \overline{ABC} + B(EF + \overline{G})$

(c) $X = A[BC(A + B + C + D)]$

(d) $X = B(C\overline{D}E + \overline{E}FG)(\overline{A}B + C)$

18. Design a minimal circuit to detect 31-day months, coded in 8-4-2-1 binary.

19. Obtain a minimal SOP for the function below then implement using NAND

gates only: $f(A, B, C, D) = \sum (0, 7, 8, 10, 12) + d(2, 6, 11)$.

20. Using a karnaugh map, convert the following standard POS expression into a minimum POS expression, a standard SOP expression and a minimum SOP expression:

$f(A, B, C, D) = \Pi(1, 2, 3, 4, 9, 12)$ where Π is denoted for POS

Basic Adders:

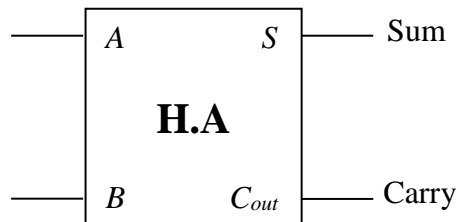
Adders are important not only in computers, but in many types of digital systems in which numerical data are processed. An understanding of the basic adder operation is fundamental to the study of digital systems. In this section, the half-adder and the full-adder are introduced.

3- The Half-Adder (H.A):

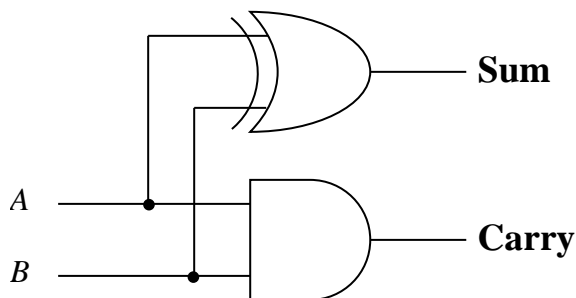
Recall the basic rules for binary addition as stated in the previous lectures:

0 + 0 = 0
0 + 1 = 1
1 + 0 = 1
1 + 1 = 10

These operations are performed by a logic circuit called a half-adder. The half-adder accepts two binary digits on its inputs and produces two binary digits on its outputs, a **Sum** bit and a **Carry** bit.



A	B	C _{out}	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0



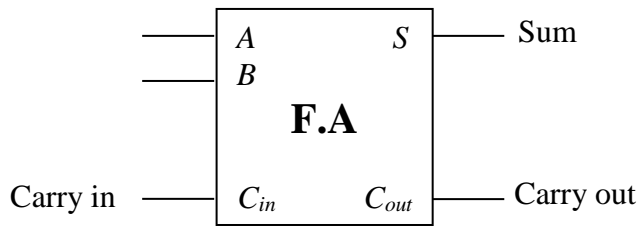
$$S = A \oplus B$$

$$C_{out} = AB$$

4- The Full-Adder(F.A):

The second basic category of adder is the Full-adder. The full-adder accepts three inputs including an input carry and generates a **Sum** output and output **carry**.

A	B	C _{in}	C _{out}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0



1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

From the truth table:-

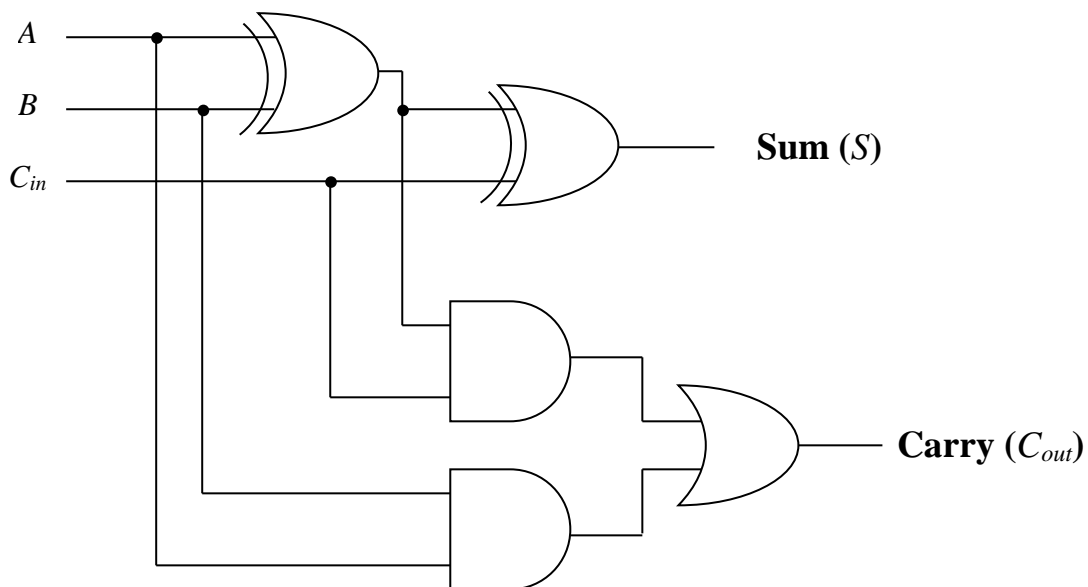
$$\begin{aligned}
 C_{out} &= \overline{A}BC_{in} + A\overline{B}C_{in} + AB\overline{C}_{in} + ABC_{in} \\
 &= (\overline{A}B + A\overline{B})C_{in} + AB \\
 C_{out} &= (A \oplus B)C_{in} + AB \quad \dots\dots\dots Eq.(1)
 \end{aligned}$$

$$\begin{aligned}
 S &= \overline{A}B\overline{C}_{in} + \overline{A}B C_{in} + A\overline{B}\overline{C}_{in} + A\overline{B} C_{in} \\
 &= \overline{A}B\overline{C}_{in} + (\overline{A}B + A\overline{B})C_{in} + ABC_{in} \quad \dots\dots\dots Eq.(2)
 \end{aligned}$$

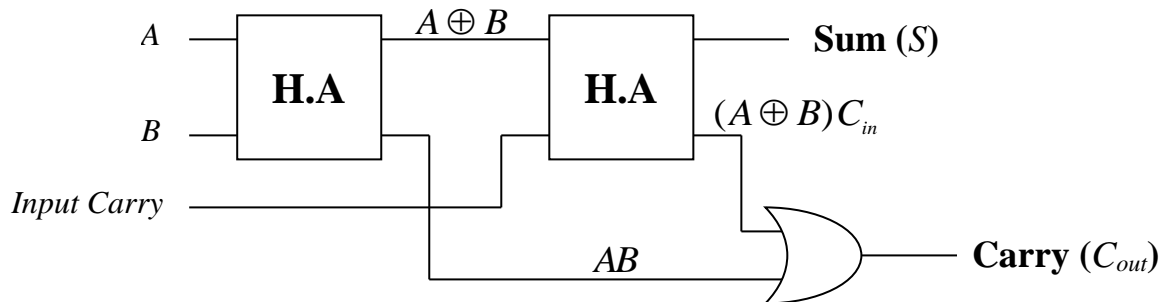
$$\begin{aligned}
 \text{But } \overline{A}B\overline{C}_{in} + ABC_{in} &= \overline{\overline{A}B + AB}C_{in} \\
 &= \overline{(A + B)(\overline{A} + \overline{B})}C_{in} \\
 &= \overline{(\overline{A}B + B\overline{A})}C_{in} \quad \dots\dots\dots Eq.(3)
 \end{aligned}$$

Substituting Eq.3 in Eq.2

$$S = (\overline{A}B + A\overline{B})C_{in} + \overline{(\overline{A}B + AB)}C_{in} = A \oplus B \oplus C_{in}$$

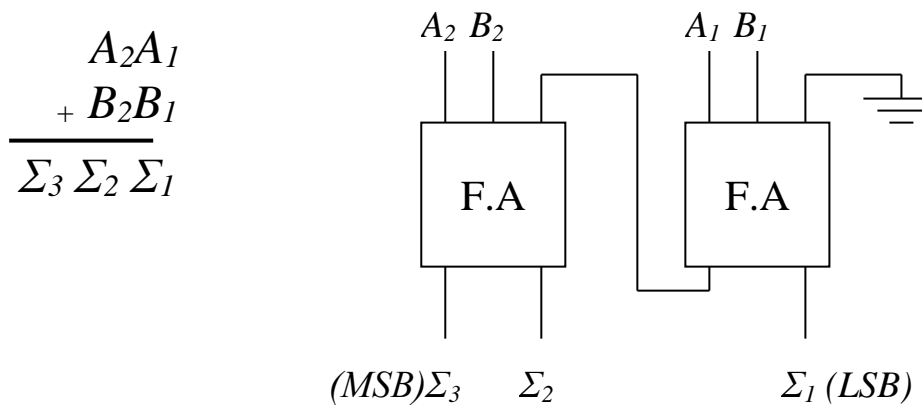


Example: Arrange two half-adders to form full-adder.



Parallel Binary Adders:

Adders that are available in integrated circuits form are parallel binary adders. As you saw in the previous section, a single full-adder is capable of adding two 1-bit numbers and an input carry. To add binary numbers with more than one bit, additional full-adders must be employed. To implement the addition of binary numbers, a full-adder is required for each bit in the numbers. So, for 2-bit numbers, two adders are needed; for 4-bit numbers, four adders are used; and so on.



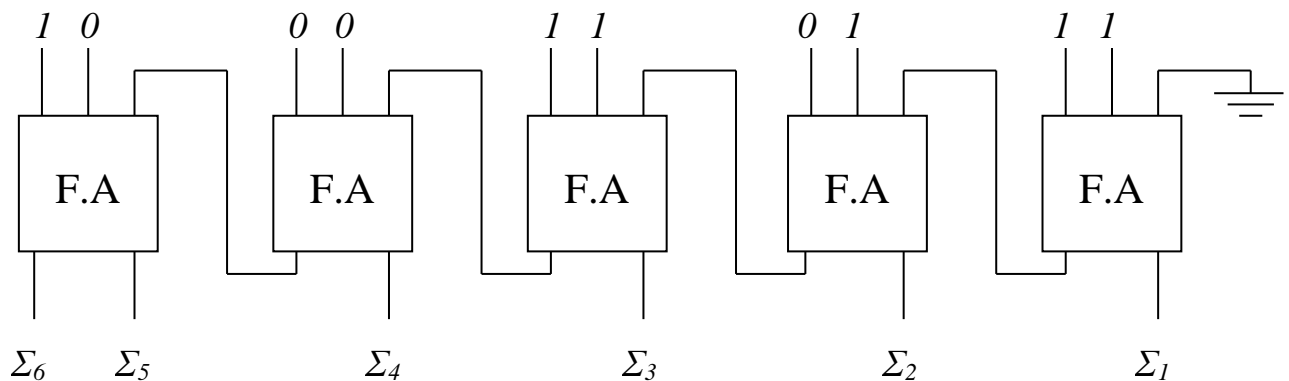
Review Questions:

21. Show how a H.A can be fabricated using only NOR gates.
22. Design a 4-bit parallel adder.
23. Draw a circuit for 2-bit parallel adder.
24. Draw a block diagram for 8-bit parallel adder.

25. Determine an alternative method for implementing the full-adder.

*Hint: write the expressions of the circuit and simplify using Karnaugh Map.
 Then implement using AND-OR gates.*

26. For the parallel adder in figure below, determine the complete sum by analysis of the logical operation of the circuit. Verify your results by long hand addition of the two input number.



Half- and Full- Subtractors:

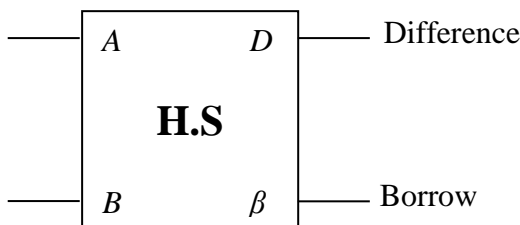
Instead of using complements to subtract, circuits can subtract binary numbers directly.

5- The Half-Subtractor (H.S):

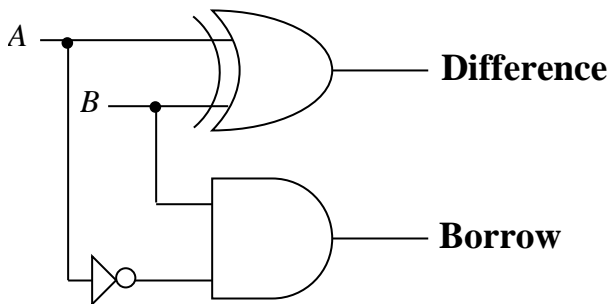
Recall the basic rules for binary subtraction as stated in the previous lectures:

- 0 – 0 = 0 with a borrow of 0
- 0 – 1 = 1 with a borrow of 1
- 1 – 0 = 1 with a borrow of 0
- 1 – 1 = 0 with a borrow of 0

These operations are performed by a logic circuit called a half-subtractor. The half-subtractor accepts two binary digits on its inputs and produces two binary digits on its outputs, a **Difference** bit and a **Borrow** bit.



A	B	Borrow (β)	Difference (D)
0	0	0	0
0	1	1	1
1	0	0	1
1	1	0	0



$$\text{Difference} = D = A \oplus B$$

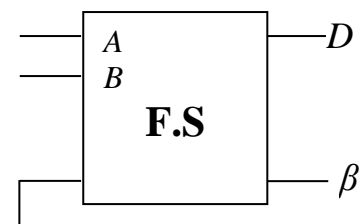
$$\text{Borrow} = \beta = \overline{A}B$$

6- The Full- Subtractor (F.S):

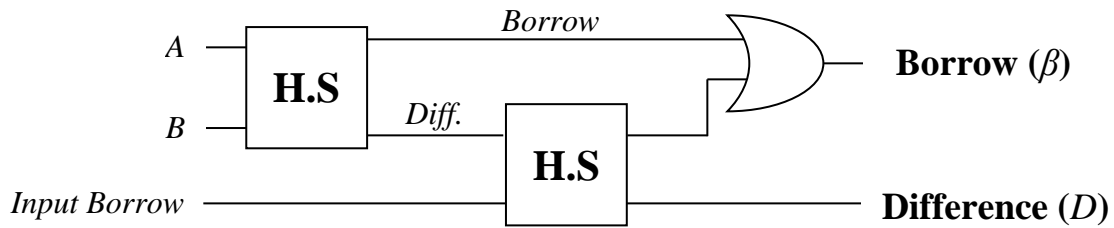
The half-subtractor handles only two bits at a time and can be used for the least significant column of a subtraction problem. To take care of a higher-order column, we need a full-subtractor.

The full-subtractor uses two half-subtractors

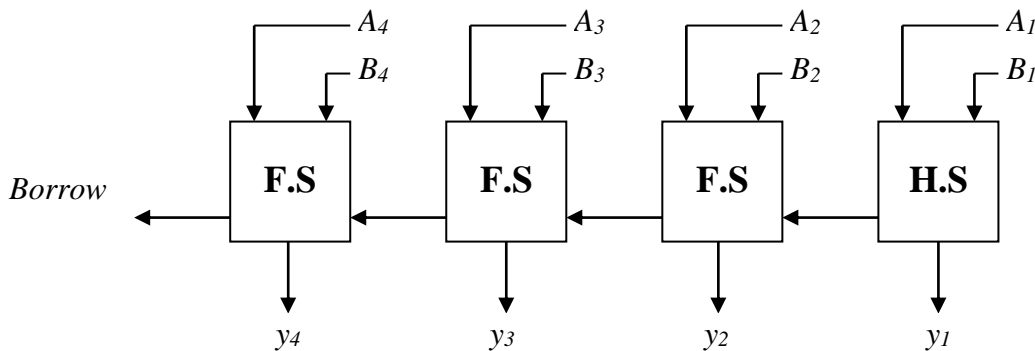
And an OR gate as shown in figure below:



Input Borrow

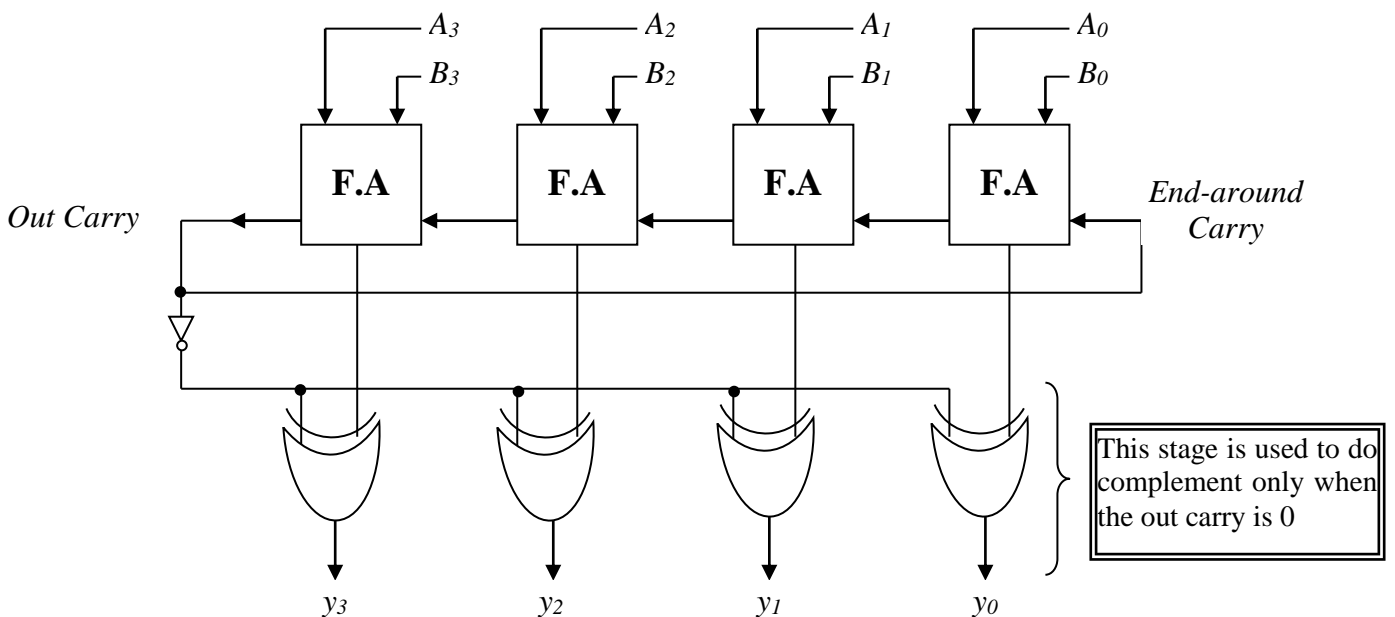


Half- and Full- subtractors are analogous to half- and full- adders; by cascading half- and full- subtractors as shown in figure below, we have a system that directly subtract $B_4B_3B_2B_1$ From $A_4A_3A_2A_1$.

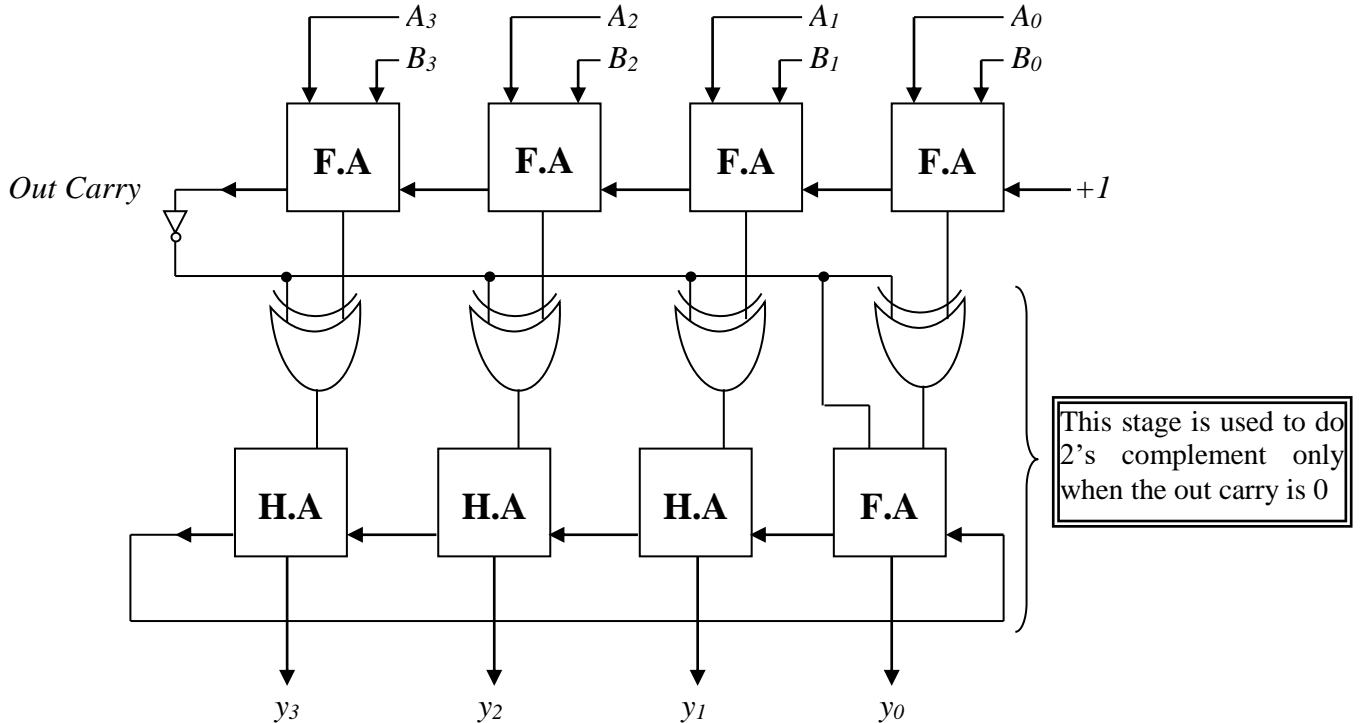


1's Complement Subtractors:

The following figure shows a circuit that subtracts $B_3B_2B_1B_0$ from $A_3A_2A_1A_0$. The first four inverters complement each B bit to get $\bar{B}_3\bar{B}_2\bar{B}_1\bar{B}_0$, the 1's complement of $B_3B_2B_1B_0$. The full-adders add A and B and the end-around carry.

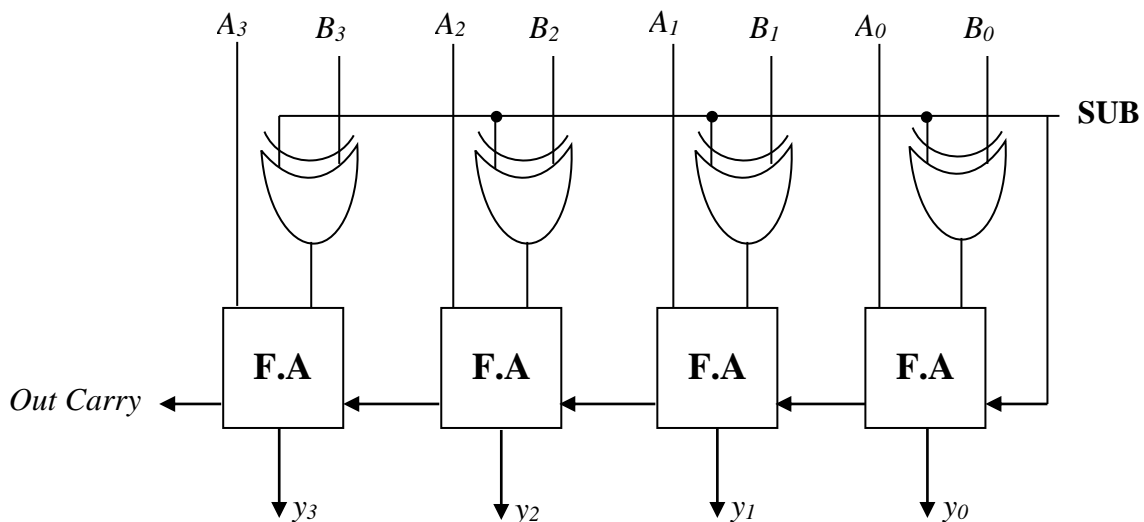


2's Complement Subtractors:



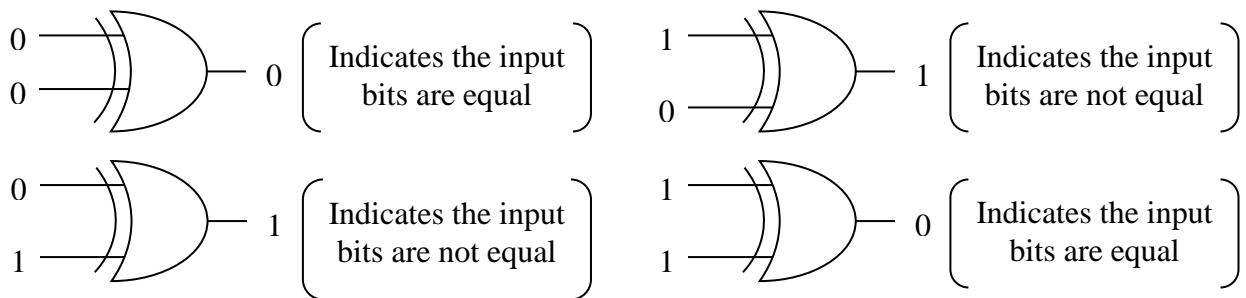
2's Complement Adders/ Subtractors:

The following figure shows an adder/subtractor based on the 2's complement. When SUB is **Low**, the *B* bits pass through the controlled inverter to the full-adder. Therefore; the full adder produces the sum of *A*&*B*. When SUB is **High**, the Bits are inverted before reaching the full-adder. Also, a High SUB adds a 1 to the first full-adders. This addition of 1 forms the 2's complement of *B*. therefore; the output of the full-adders is the difference of *A*&*B*.

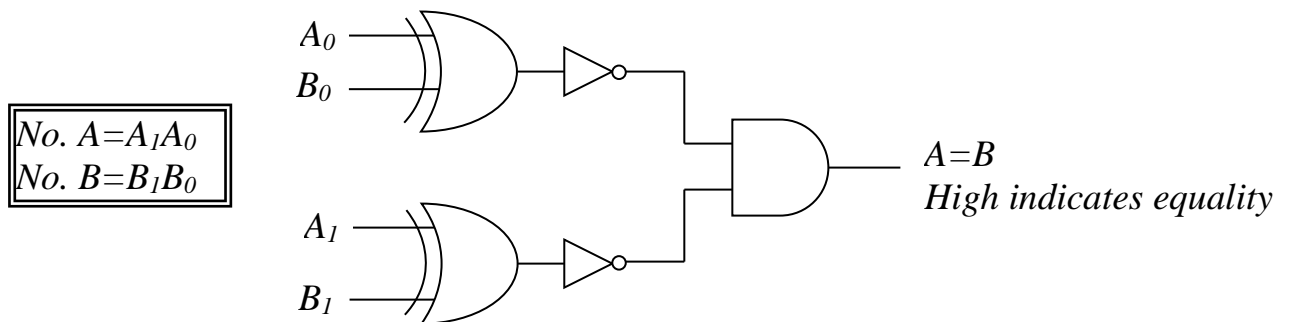


Comparators:

The basic function of a comparator is to compare the magnitudes of two quantities to determine the relationship of those quantities. In its simplest form, a comparator circuit determines whether two numbers are equal. The exclusive-OR gate can be used as a basic comparator because its output is a 1 if two input bits are not equal and a 0 if the input bits are equal.



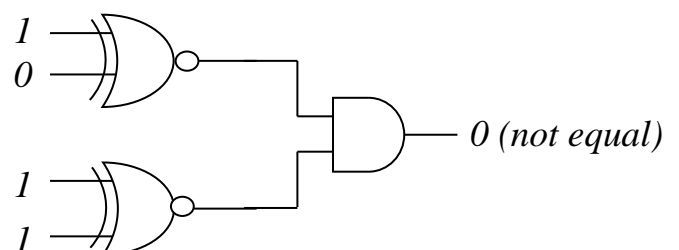
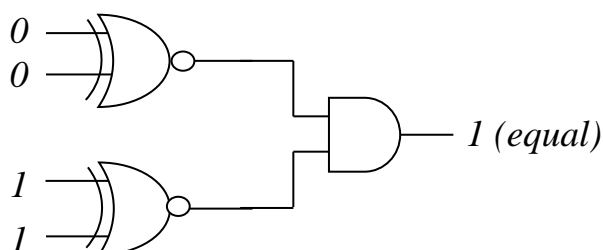
In order to compare binary numbers containing two bits each, an additional exclusive-OR gate is necessary.



Example: Apply each of the following sets of binary numbers to the comparator inputs and determine the output by following the logic levels through the circuits.

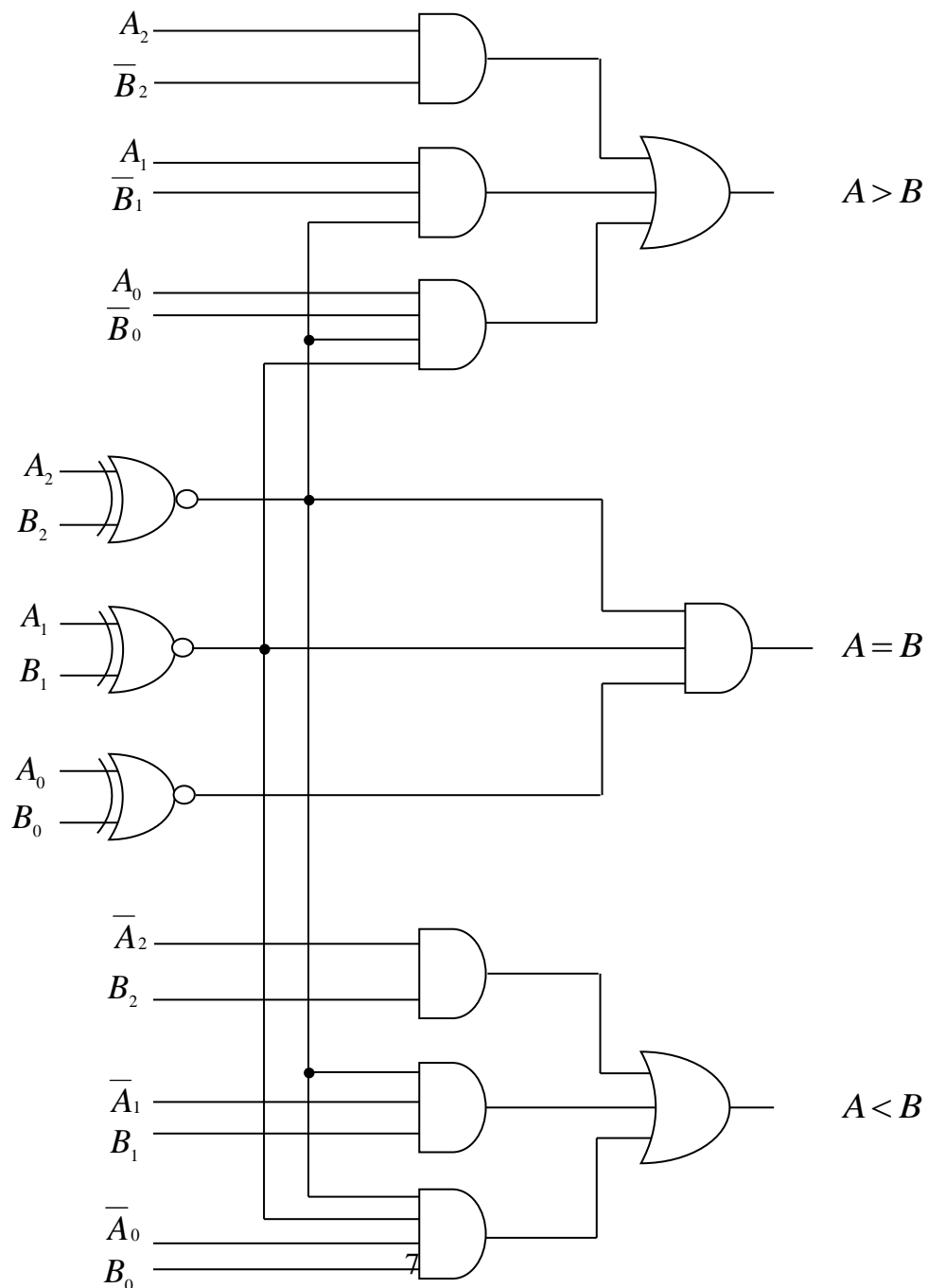
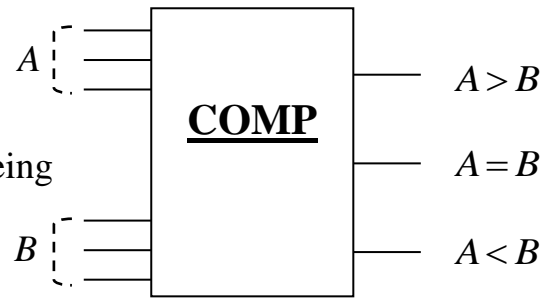
(a) 10 and 10

(b) 11 and 10

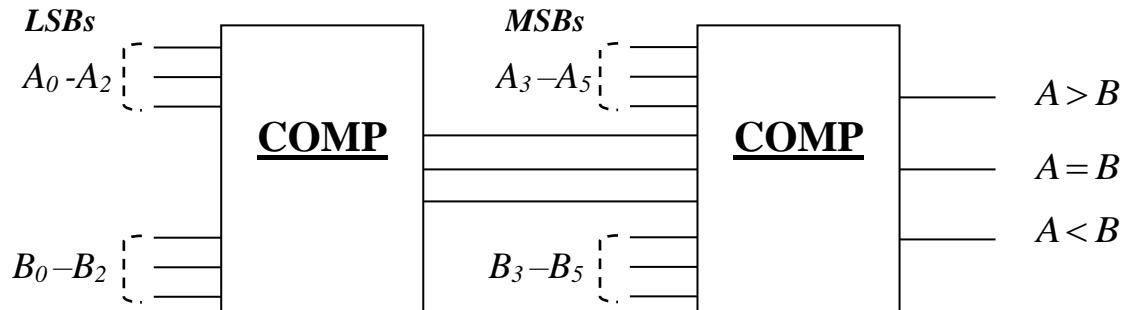


Integrated Circuit Comparators:

In addition to the equality output, many integrated circuit comparators provide additional outputs that indicate which of the two numbers being compared is larger. That is, there is an output that indicates when number A is greater than number B ($A > B$) and an output that indicates when number A is less than B ($A < B$), as shown in the logic symbol for a 3-bits comparator in above figure:

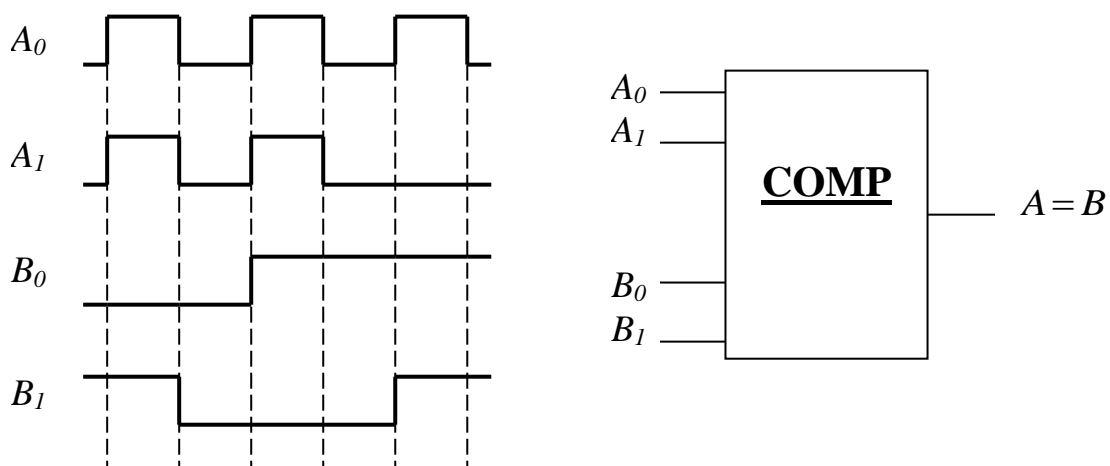


Example: Use 3-bit integrated comparators to compare the magnitude of two 6-bit binary numbers. Show the comparators with proper connections.



Review Questions:

27. Design a full-subtractor circuit.
28. Design a subtractor to subtract two numbers of 4-bit using F.A.
29. Draw circuit for equality comparison of two 4-bit numbers.
30. Draw a general circuit for a 4-bit comparator. Analyze the circuit for $A=1010$ and $B=1100$.
31. The waveforms in figure below are applied to the comparator as shown. Determine the output ($A=B$) waveform.



BCD Adder:

The addition of BCD number is summarized in the following steps:

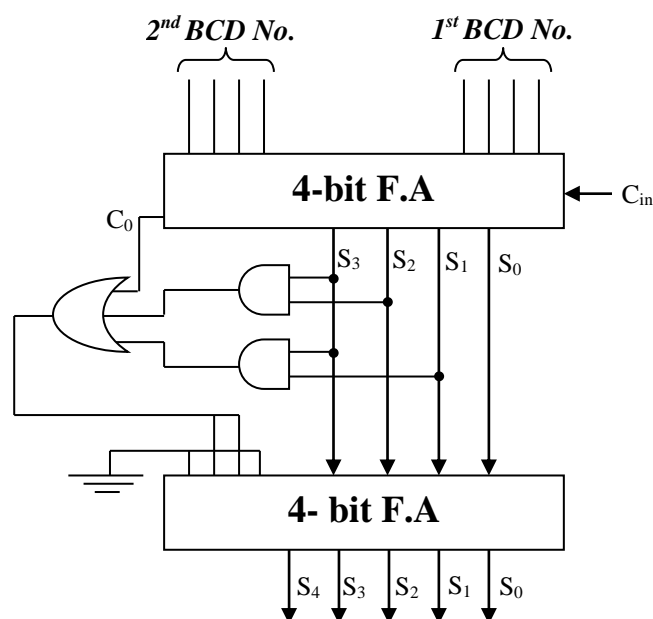
1. Add the BCD code groups for each decimal digit position, use ordinary binary addition.
2. For those positions where the sum is 9 or less, the sum is in proper BCD form and no correction is needed.
3. When the sum of two digits is greater than 9, a correction of 0110 (6) should be added to that sum to produce the proper BCD results. This will produce a carry to be added to the next decimal position.

A **BCD** adder cct. must be able to do the following:-

1. Add two 4-bit BCD code groups, using straight binary addition.
2. Determine if the sum of this addition is greater than 1001 (9), if it is, add 0110 (6) to this sum and generate a carry to the next decimal position.

1000	8
+ 0101	5
1101	Binary equivalent of 13
+ 0110	Add 6 to return to BCD
0001 0011	BCD equivalent of 13

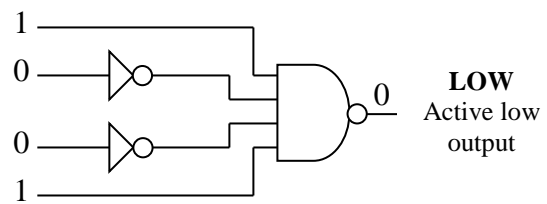
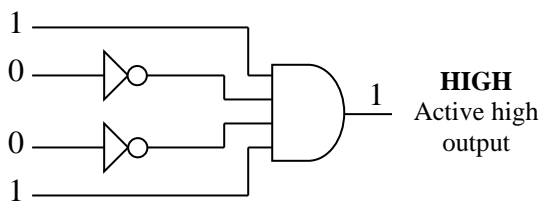
1001	9
+ 0011	3
1100	Binary equivalent of 12
+ 0110	Add 6 to return to BCD
0001 0010	BCD equivalent of 12



Decoders:

The basic function of a decoder is to detect the presence of a specified combination of bits (code) on its inputs and to indicate the presence of that code by a specified output level. In its general form, a decoder has n lines to handle n bits and from one to 2^n output lines to indicate the presence of one or more n -bit combination.

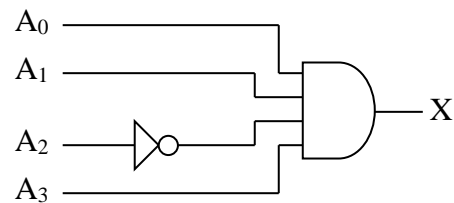
Example: A decoder for binary number 1001.



Example: Determine the logic required to decode the binary number 1011 by producing a **HIGH** level on the output.

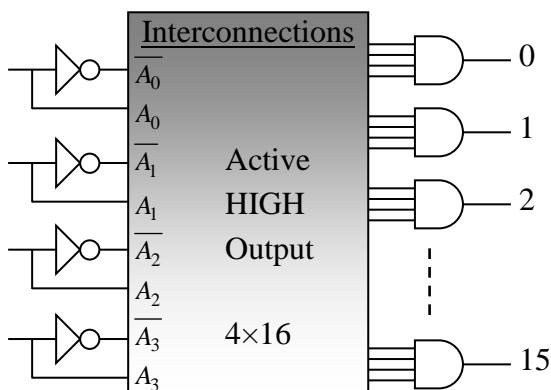
The decoding function can be formed by complementing only the variables that appear as 0 in the binary number, as follows:

$$X = A_3 \bar{A}_2 A_1 A_0 \quad (1011)$$



Four bit binary Decoders:

In order to decode all possible combinations of four bits, sixteen decoding gates are required ($2^4=16$). This type of decoder is commonly called a 4-line-to-16-line decoder because there are four inputs and sixteen outputs or a 1-of-16 decoder because for any given code on the inputs, one of the sixteen outputs is activated.



Pebble is added for each AND gate to produce active low output

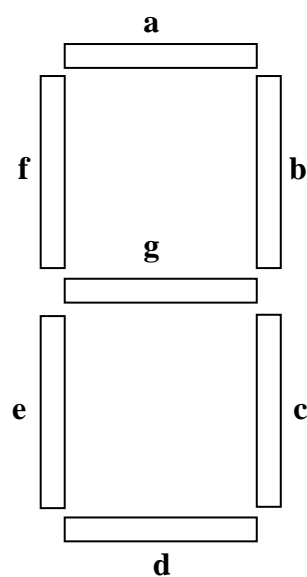


Decimal	Binary Input				Output															
	A ₃	A ₂	A ₁	A ₀	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	1	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
4	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
5	0	1	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
6	0	1	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
7	0	1	1	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
8	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
9	1	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
10	1	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
11	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
12	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
13	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
14	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
15	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

BCD to Seven Segment Decoder:

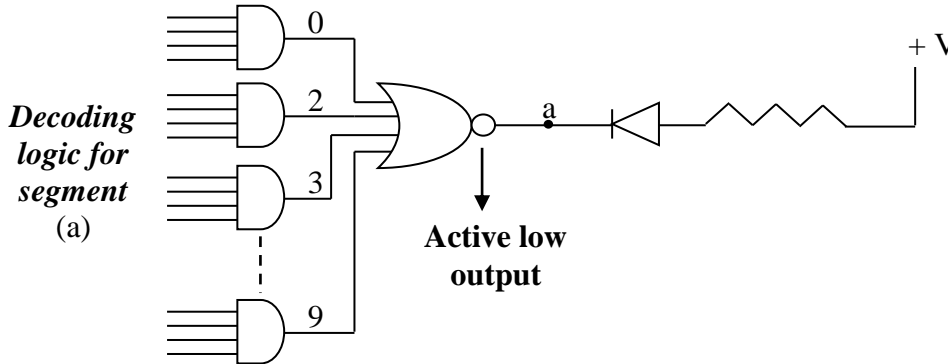
This decoder accepts BCD code on its inputs and provides outputs to energize 7-segment display device.

Digit	Segment illuminated
0	a,b,c,d,e,f
1	e,f
2	a,b,g,e,d
3	a,b,c,d,g
4	b,c,f,g
5	a,c,d,f,g
6	c,d,e,f,g
7	a,b,c
8	a,b,c,d,e,f,g
9	a,b,c,f,g



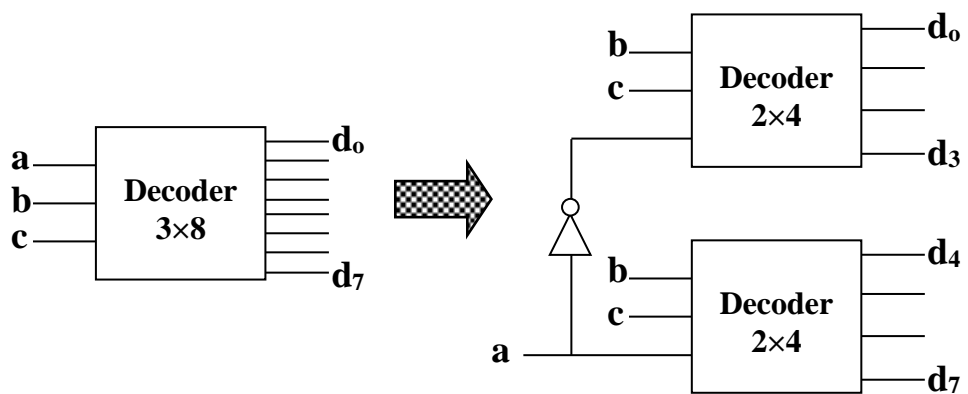
a appears in 0,2,3,5,7,8,9, the Boolean expression for LED **a** is:

$$a = \overline{D}\overline{C}\overline{B}\overline{A} + \overline{D}\overline{C}\overline{B}A + \overline{D}\overline{C}B\overline{A} + \overline{D}\overline{C}BA + \overline{D}C\overline{B}\overline{A} + \overline{D}C\overline{B}A + \overline{D}CBA$$

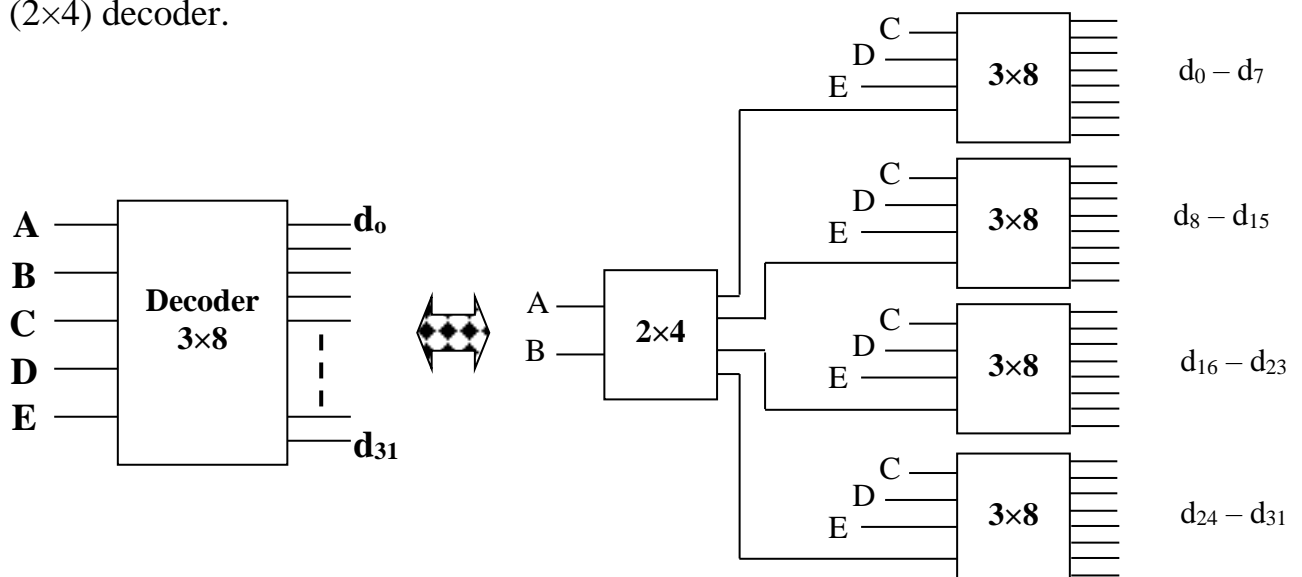


Example: Design (3×8) decoder using (2×4) decoders and NOT gate .

a	b	c	o/p
0	0	0	d ₀
0	0	1	d ₁
0	1	0	d ₂
0	1	1	d ₃
1	0	0	d ₄
1	0	1	d ₅
1	1	0	d ₆
1	1	1	d ₇



Example: Design a 5-input to 32 decoder using 4 of (3×8) decoder and one of (2×4) decoder.

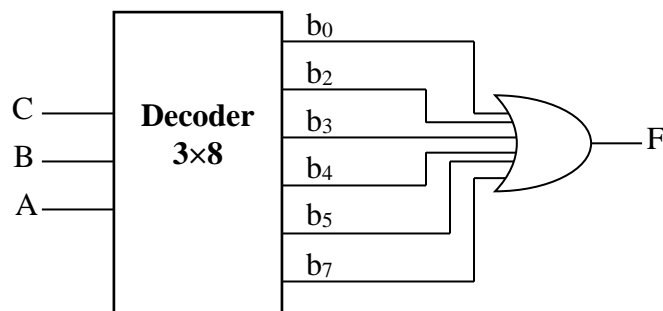


Example: Design the following Boolean function with a (3×8) decoder and OR gate:
gate: $F = \overline{A}B + AC + BC + \overline{B}C$

$$F = \overline{A}BC + \overline{A}B\overline{C} + ABC + A\overline{B}C + A\overline{B}\overline{C} + \overline{A}BC + \overline{A}B\overline{C} + \overline{A}B\overline{C}$$

$$= \overline{A}BC + \overline{A}B\overline{C} + ABC + A\overline{B}C + A\overline{B}\overline{C} + \overline{A}B\overline{C}$$

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

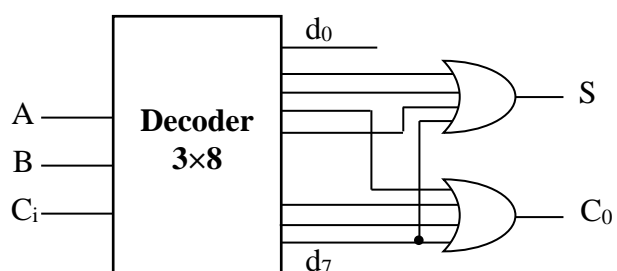


$$f(A, B, C) = \sum_m (0, 2, 3, 4, 5, 7)$$

$$f(A, B, C) = \Pi_m (1, 6)$$

Example: Design a full adder cct. using a decoder of (3×8) and two OR gates.

C _i	A	B	S	C _o
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



$$S = d_1 + d_2 + d_4 + d_7$$

$$C_o = d_3 + d_5 + d_6 + d_7$$

Encoder:

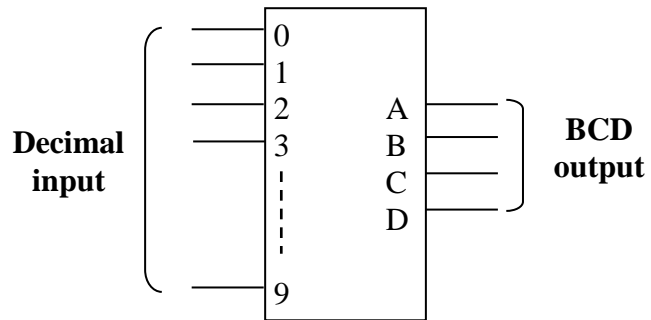
An encoder is a combinational logic circuit that essentially performs a “reverse” decoder function. An encoder accepts an active level on one of its inputs representing a digit such as a decimal or octal digit, and converts it to a coded output, such as BCD or binary. Encoders can also be desired to encode various

symbols and alphabetic characters. The process of converting from familiar symbols or numbers to a coded format is called encoding.

Decimal to BCD Encoder:

This type of encoder has ten inputs – one for each decimal digit – and four outputs corresponding to the BCD code as shown in figure below. This is a basic 10-line-to-4-line encoder.

Decimal	D	C	B	A
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1



The expressions for output in term of decimal are:

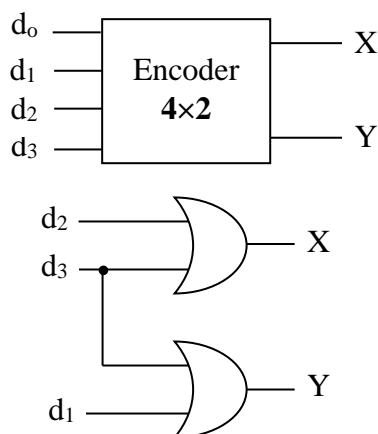
$$D = 8 + 9$$

$$C = 4 + 5 + 6 + 7$$

$$B = 2 + 3 + 6 + 7$$

$$A = 1 + 3 + 5 + 7 + 9$$

Example: (4×2) encoder: where 4= No. of input, 2= No. of output.



d ₀	d ₁	d ₂	d ₃	X	Y
1	0	0	0	0	0
0	1	0	0	0	1
0	0	1	0	1	0
0	0	0	1	1	1

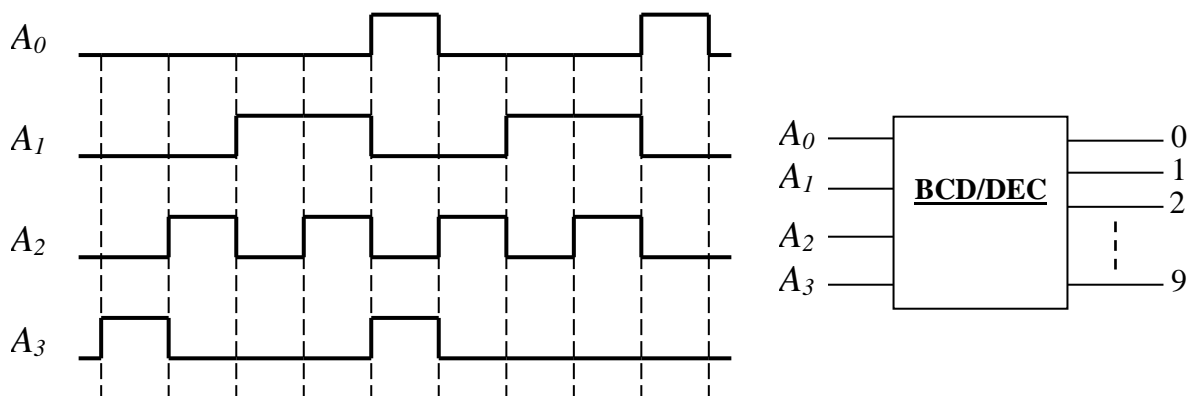
$$X = d_2 + d_3$$

$$Y = d_1 + d_3$$

Review Questions:

32. Design a logic circuit to convert octal number to binary using (8×3) encoder.

33. BCD numbers are applied sequentially to the BCD-to-decimal decoder in figure below. Draw a timing diagram, showing each output in the proper relationship with others and with the inputs.



34. Convert decoder to H.A.

35. Convert decoder to H.S.

36. Convert decoder to comparator.

37. Convert decoder to: **AND, OR, NAND, NOR, NOT, EX-OR, EX-NOR** gates.