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 everyday. Examining some of its characteristics will help us to better understand of other systems.

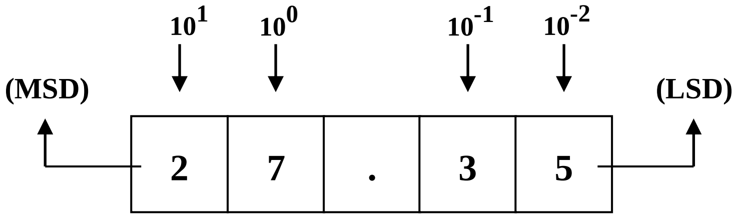
Decimal System:

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:

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 of represent any quantity 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)

Quantity	*Binary No.*	*Decimal No.*
None	0	0
.	1	1
..	01	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:

1 1 0 1 1 **(Binary)**

$2^4 + 2^3 + 2^2 + 2^1 + 2^0 = 16+8+2+1$

$= 27_{(10)}$ (Decimal)

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

(Binary)

1 0 1 1 0 1 0 1

$2^7 + 2^6 + 2^5 + 2^4 + 2^3 + 2^2 + 2^1 + 2^0 = 181_{(10)}$ (Decimal)

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:

etc. ←

	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	

→ etc.

↑

• Binary Point

$$\begin{array}{cccccc} \mathbf{0} & \mathbf{.} & \mathbf{1} & \mathbf{1} & \mathbf{0} & \mathbf{1} \\ \underbrace{2^{0}}_{0} & . & \underbrace{2^{-1} \quad 2^{-2} \quad 2^{-3} \quad 2^{-4}}_{\to\ 0.5+0.25+0.0625=0.8125} \end{array}$$

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

Example: convert binary 110.001 to a decimal number

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

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.

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

As another example,

$$25 = 16+8+1 = \mathbf{16 + 8 + 0 + 0 + 1}$$

$$\mathbf{1 \quad 1 \quad 0 \quad 0 \quad 1} \ \textbf{(Binary)}$$

Amore 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 reminders after each division until a quotient of 0 is obtained. Note that the binary result is obtained by writing the first reminder as the LSB and the last reminder as the MSB. Let us convert decimal 25 to its binary equivalent using the repeated division method.

$\frac{25}{2}=12$ + Remainder 1 ***LSB***

$\frac{12}{2}=6$ + Remainder 0

$\frac{6}{2}=3$ + Remainder 0

$\frac{3}{2}=1$ + Remainder 1

$\frac{1}{2}=0$ + Remainder 1 ***MSB***

$$25_{(10)}=(1\ 1\ 0\ 0\ 1)_2$$

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

0.625×2=1.25 = 0.25 with a carry of 1
0.25×2=0.5 with a carry of 0
0.5×2=1.0 = 0 with a carry of 1

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.

0.6×2=1.2 = 0.2 with a carry of 1
0.2×2=0.4 with a carry of 0
0.4×2=0.8 with a carry of 0
0.8×2=1.6 = 0.6 with a carry of 1
0.6×2=1.2 = 0.2 with a carry of 1

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.

and

$$\frac{21}{2} = 10 + \text{Remainder } 1 \longrightarrow \textbf{\textit{LSB}}$$

$$\frac{10}{2} = 5 + \text{Remainder } 0$$

$$\frac{5}{2} = 2 + \text{Remainder } 1$$

$$\frac{2}{2} = 1 + \text{Remainder } 0$$

$$\frac{1}{2} = 0 + \text{Remainder } 1 \longrightarrow \textbf{\textit{MSB}}$$

hence: $21.6_{(10)} = (1\ 0\ 1\ 0\ 1.1\ 0\ 0\ 1\ 1)_2$

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:

etc. ←		$\mathbf{8^3}$	$\mathbf{8^2}$	$\mathbf{8^1}$	$\mathbf{8^0}$	$\mathbf{8^{-1}}$	$\mathbf{8^{-2}}$	$\mathbf{8^{-3}}$		→ etc.

↑
• Octal Point

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:

$372_{(8)} = 3\times8^2 + 7\times8^1 + 2\times8^0$

$= 3\times64 + 7\times8 + 2\times1 = 250_{(10)}$

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

$24.6_{(8)} = 2\times8^1 + 4\times8^0 + 6\times8^{-1}$

$= 2\times8 + 4\times1 + 6\times0.125 = 20.75_{(10)}$

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:

$\frac{266}{8} = 33$ + Remainder 2

$\frac{33}{8} = 4$ + Remainder 1

$\frac{4}{8} = 0$ + Remainder 4

$$266_{(10)} = 412_{(8)}$$

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.

$0.23 \times 8 = 1.48 = 0.84$ with a carry of 1

$0.84 \times 8 = 6.72 = 0.72$ with a carry of 6

$0.72 \times 8 = 5.76 = 0.76$ with a carry of 5

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

Octal-to-Binary Conversion:

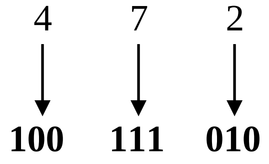
The primary advantage of the octal number system is the ease with which conversion can be made between binary and octal numbers. The conversion from octal to binary is performed by converting each octal digit to its 3-bit binary equivalent. The eight possible digits are converted as indicated in Table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

Table (2)

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

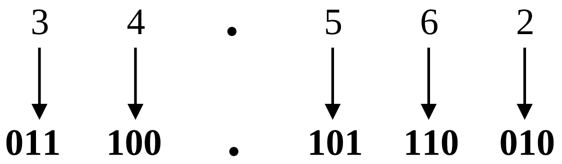
For example, $472_{(8)}$ is converted to its binary equivalent as follows:-

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



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

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

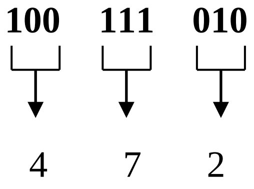


Binary-to-Octal Conversion:

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.

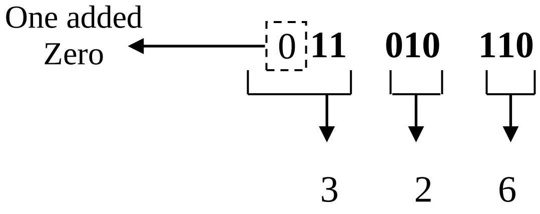
$100111010_{(2)} = 472_{(8)}$



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

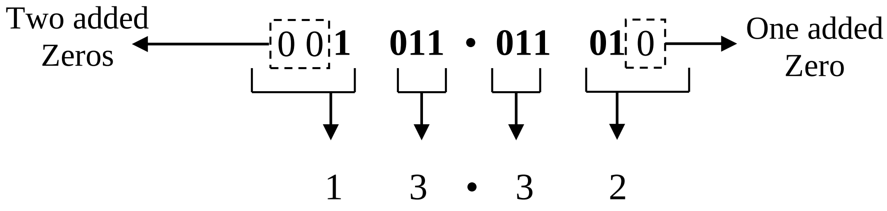
Sometimes the binary number will not have even groups of 3bits. For this case, extra 0's can be added to the left of the MSB of the binary number to fill out the last group as shown below:

$11010110_{(2)} = 326_{(8)}$



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

$1011.01101_{(2)} = 13.32_{(8)}$



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:

$$356_{(16)} = 3\times16^2 + 5\times16^1 + 6\times16^0$$
$$= 768 + 80 + 6 = 854_{(10)}$$

$$2AF_{(16)} = 2\times16^2 + 10\times16^1 + 15\times16^0$$
$$= 512 + 160 + 15 = 687_{(10)}$$

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.

$$\frac{423}{16} = 26 + \text{Remainder } 7$$
$$\frac{26}{16} = 1 \quad + \text{Remainder } 10$$
$$\frac{1}{16} = 0 \quad + \text{Remainder } 1$$

$$423_{(10)} = 1A7_{(16)}$$

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

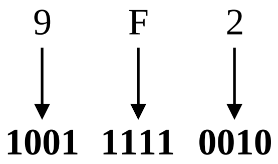
$$\frac{214}{16} = 13 + \text{Remainder } 6$$
$$\frac{13}{16} = 0 \quad + \text{Remainder } 13$$

$$214_{(10)} = D6_{(16)}$$

Hex-to-Binary Conversion:

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

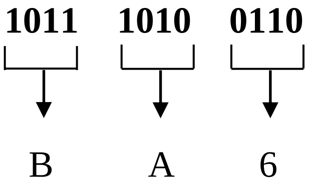
$9F2_{(16)} = 100111110010_{(2)}$



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:

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



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:

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

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:

when 1 is borrowed, a 0 is left .
when 1 is borrowed, making 10 instead of 0.

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

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:

1 0 1 1 0 1	Binary No.	0 1 1 0 1 0	Binary No.
↓↓↓↓↓↓		↓↓↓↓↓↓	
0 1 0 0 1 0	1's complement	1 0 0 1 0 1	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 complement = (1's complement) +1

Example: find 2's complement of 10110010.

```
  1 0 1 1 0 0 1 0     binary number.
  ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓
  0 1 0 0 1 1 0 1     1's complement
+               1     adding 1
-----------------
  0 1 0 0 1 1 1 0     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.

```
  11010
+ 01111    1's complement of 10000
-------
 101001
+     1
-------
  01010
```

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

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.

```
  10000
+ 00101    1's complement of 11010
-------
  10101
  ↓↓↓↓↓  1's complement
  01010
```

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

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 $\longleftarrow$

$$\begin{array}{rl} 11010 & \\ \underline{+\ 10000} & \text{2's complement of } 10000 \\ \mathbf{1}01010 & \end{array}$$

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

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 in needed. $\longleftarrow$

$$\begin{array}{rl} 10000 & \\ \underline{+\ 00110} & \text{2's complement of } 11010 \\ 10110 & \\ \Downarrow & \textit{2's complement} \\ 01010 & \end{array}$$

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

Multiplication of Binary Numbers:

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

the multiplication rules for binary numbers are:

0 × 0 = 0

0 × 1 = 0

1 × 0 = 0

1 × 1 = 1

$$\begin{array}{rcl} 8 & \longrightarrow & \textbf{\textit{multiplicand}} \\ \underline{\times\ 3} & \longrightarrow & \textbf{\textit{multiplier}} \\ 24 & \longrightarrow & \textbf{\textit{product}} \end{array}$$

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

$$\begin{array}{rl} 100 & (4) \\ \underline{\times\ 010} & (2) \\ 000 & \\ 1000\ \ + & \\ \underline{00000\ \ +} & \\ 01000 & (8) \end{array}$$

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

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:

$$\boxed{9 \div 3 = 3} \qquad \boxed{10 \div 4 = 2.5} \qquad \boxed{5 \div 2 = 2.5}$$

$$\begin{array}{r|l} & \underline{0011} \\ \underline{11} & 1001 \\ & \underline{011} \\ & 0011 \\ & \underline{0011} \\ & 0000 \end{array} \qquad \begin{array}{r|l} & \underline{0010.1} \\ 1\underline{00} & 1010 \\ & \underline{100} \\ & 00100 \\ & \underline{100} \\ & 000 \end{array} \qquad \begin{array}{r|l} & \underline{010.1} \\ \underline{10} & 101 \\ & \underline{10} \\ & 0010 \\ & \underline{10} \\ & 00 \end{array}$$

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.

$$\begin{array}{r} 58 \\ \underline{+\ 24} \\ 7C \end{array} \qquad \begin{array}{r} 3AF \\ \underline{+\ 23C} \\ 5EB \end{array} \qquad \begin{array}{r} \boxed{1} \longrightarrow \text{Carry} \\ 58 \\ \underline{+\ 4B} \\ A3 \end{array}$$

Examples: add the following hexadecimal numbers.

(a) $23_{(16)}+16_{(16)}$ (b) $58_{(16)}+22_{(16)}$ (c) $DF_{(16)}+AC_{(16)}$

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

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

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

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

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

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

$$\begin{array}{r} DF \\ +\ AC \\ \hline 18B \end{array}$$

right column: $F_{(16)}+C_{(16)} = 15_{(10)}+12_{(10)} = 27_{(10)}$
$= 27_{(10)}-16_{(10)} = 11_{(10)} = B_{(16)}$ with a carry of 1

left column: $D_{(16)}+A_{(16)}+1_{(16)}= 13_{(10)}+10_{(10)} +1_{(10)}= 24_{(10)}$
$= 24_{(10)}-16_{(10)} = 8_{(10)} = 8_{(16)}$ with a carry of 1

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:

$$\begin{array}{r} D3A \\ -\quad F4 \\ \hline C46 \end{array}$$

right column: $A_{(16)}- 4_{(16)} = 10_{(10)}- 4_{(10)} = 6_{(10)} = 6_{(16)}$
middle column: $3_{(16)} - F_{(16)}= 3_{(10)} - 15_{(10)}$ (need borrow)
$= 19_{(10)} - 15_{(10)}= 4_{(10)} =4_{(16)}$
left column: $D_{(16)} - 1_{(16)} = C_{(16)}$

$$\begin{array}{r} 84 \\ -\ 2A \\ \hline 5A \end{array}$$

right column : $4_{(16)}- A_{(16)} = 4_{(10)} - 10_{(10)}$ (need borrow)
$= 20_{(10)} - 10_{(10)}= 10_{(10)} =A_{(16)}$
left column : $8_{(16)} - 2_{(16)} - 1_{(16)}= 5_{(16)}$

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:

$$\begin{array}{cccl} 7 & 3 & A & \longrightarrow \text{hex number} \\ 0111 & 0011 & 1010 & \longrightarrow \text{convert to binary} \\ 1000 & 1100 & 0101 & \longrightarrow \text{1's complement representation} \\ 1000 & 1100 & 0110 & \longrightarrow \text{2's complement representation} \\ 8 & C & 6 & \longrightarrow \text{conversion back to hex} \end{array}$$

- 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:

$$\begin{array}{cccl} F & F & F & \\ -7 & -3 & -A & \longrightarrow \text{Subtract each digit from F} \\ \hline 8 & C & 5 & \\ & & +1 & \longrightarrow \text{adding 1} \\ \hline 8 & C & 6 & \longrightarrow \text{hex. Equivalent of 2's comp.} \end{array}$$

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.

$$\boxed{\textit{Discarded carry}} \longleftarrow \begin{array}{r} 592 \\ \underline{+\ \text{C5B}} \\ 1\ 1\text{ED} \end{array}$$

Example: subtract the following hexadecimal numbers:

(a) 84 – 2A (b) C3 – 0B

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

$$\boxed{\textit{Drop carry}} \longleftarrow \begin{array}{r} 84 \\ \underline{+\ \text{D6}} \\ 1\ 5\text{A} \end{array} \qquad \textbf{the difference is } \mathbf{5A_{(16)}}$$

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

$$\boxed{\textit{Drop carry}} \longleftarrow \begin{array}{r} \text{C3} \\ \underline{+\ \ \text{F5}} \\ 1\ \text{B8} \end{array} \qquad \textbf{the difference is } \mathbf{B8_{(16)}}$$

Multiplication of Hexadecimal Numbers:

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

$$\begin{array}{r} 3\text{A} \\ \underline{\times\ \ \ \text{F}} \\ 366 \end{array}$$

right column : $A_{(16)} \times F_{(16)} = 10_{(10)} \times 15_{(10)} = 150_{(10)} = 96_{(16)}$

$= 6$ with carry 9

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:

1. perform the following binary additions:

   (a) 1101+1010 (b) 10111+01101

2. perform the following binary subtractions:

   (a) 11101- 0100 (b) 1001- 0111

3. perform the indicated binary operation:

   (a) 110 ×111 (b) 1100 ÷ 011

4. determine the 1's complement of each binary number:

   (a) 11010 (b) 001101

5. determine the 2's complement of each binary number:

   (a) 10111 (b) 010001

6. subtract the hexadecimal numbers:

   (a) $75_{(16)} - 21_{(16)}$ (b) $94_{(16)} - 5C_{(16)}$

7. add the hexadecimal numbers directly:

   (a) $18_{(16)} + 34_{(16)}$ (b) $3F_{(16)} + 2A_{(16)}$

8. multiply the following pairs of binary numbers:

   (a) 101.101 × 110.010 ***Ans.:100011.00101***

   (b) 0.1101 × 0.1011 ***Ans.:0.10001111***

9. perform the following divisions:

   (a) 10110.1101 ÷ 1.1 ***Ans.:1111.0011***

   (b) 111111 ÷ 1001 ***Ans.:111***

Digital Codes

1- Binary Coded Decimal (BCD):

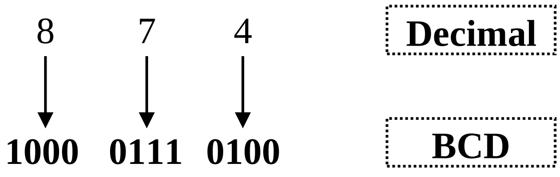
When numbers, letters, or words are represented by a special group of symbols, this is called encoding, and the group of symbols is called a code. Probably one of the familiar codes is the Morse code, where series of dots and dashes represent letters of the alphabet. We have seen that decimal numbers 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 though 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.

$137_{(10)} = 10001001_{(2)}$ (Binary)
$137_{(10)} = 000100110111$ (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

$$\begin{array}{cc} 3 & 5 \\ \downarrow & \downarrow \\ \mathbf{0011} & \mathbf{0101} \end{array} \qquad \begin{array}{cc} 9 & 8 \\ \downarrow & \downarrow \\ \mathbf{1001} & \mathbf{1000} \end{array} \qquad \begin{array}{ccc} 1 & 7 & 0 \\ \downarrow & \downarrow & \downarrow \\ \mathbf{0001} & \mathbf{0111} & \mathbf{0000} \end{array} \qquad \begin{array}{cccc} 2 & 4 & 6 & 9 \\ \downarrow & \downarrow & \downarrow & \downarrow \\ \mathbf{0010} & \mathbf{0100} & \mathbf{0110} & \mathbf{1001} \end{array}$$

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

(a) 10000110 (b) 1001010001110000

$$\begin{array}{cc} \multicolumn{2}{c}{10000110} \\ \downarrow & \downarrow \\ \mathbf{8} & \mathbf{6} \end{array} \qquad\qquad \begin{array}{cccc} \multicolumn{4}{c}{1001010001110000} \\ \downarrow & \downarrow & \downarrow & \downarrow \\ \mathbf{9} & \mathbf{4} & \mathbf{7} & \mathbf{0} \end{array}$$

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.

10110 (Binary)
↓
1 (Gray)

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

1 + 0 110 (Binary)
↓
1 1 (Gray)

Step 3: add the next adjacent pair:

1 **0 + 1** 10 (Binary)
↓
1 1 1 (Gray)

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

1 0 **1 + 1** 0 (Binary)
↓
1 1 1 0 (Gray)

Step 5: add the last adjacent pair:

1 0 1 **1 + 0** (Binary)
↓
1 1 1 0 1 (Gray)

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.**

<u>Example:</u> convert the Gray code number 11011 to binary.

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

$$\begin{array}{lll} 11011 & \qquad & \text{(Gray)} \\ \downarrow & & \\ 1 & & \text{(Binary)} \end{array}$$

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

$$\begin{array}{cclll} 1 & \mathbf{1} & 011 & \qquad & \text{(Gray)} \\ & + \downarrow & & & \\ \mathbf{1} & 0 & & & \text{(Binary)} \end{array}$$

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

$$\begin{array}{cccll} 1 & 1 & \mathbf{0}\ 11 & \qquad & \text{(Gray)} \\ & & + \downarrow & & \\ 1 & \mathbf{0} & 0 & & \text{(Binary)} \end{array}$$

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

$$\begin{array}{cccccl} 1 & 1 & 0 & \mathbf{1} & 1 & \text{(Gray)} \\ & & & + \downarrow & & \\ 1 & 0 & \mathbf{0} & 1 & & \text{(Binary)} \end{array}$$

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

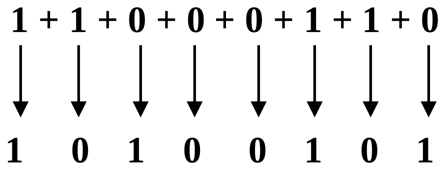
$$\begin{array}{cccccl} 1 & 1 & 0 & 1 & \mathbf{1} & \text{(Gray)} \\ & & & & + \downarrow & \\ 1 & 0 & 0 & \mathbf{1} & 0 & \text{(Binary)} \end{array}$$

Hence the final binary number is 10010

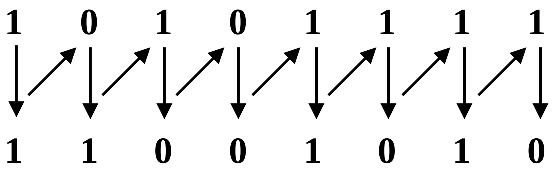
Example: (a) Convert the binary number 11000110 to Gray-code.

(b) Convert the Gray-code 10101111 to binary.

(a) Binary to Gray code:-



(b) Gray code to Binary:-



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:

$$\boxed{\begin{array}{r} 2 \\ \underline{+\,3} \\ 5 \end{array} \longrightarrow \mathbf{(0101)}} \qquad \boxed{\begin{array}{r} 9 \\ \underline{+\,3} \\ 12 \end{array} \longrightarrow \mathbf{(1100)}}$$

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

<u>Example:</u> 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.

$$\begin{array}{cc} \mathbf{1} & \mathbf{3} \\ \mathbf{+\,3} & \mathbf{+\,3} \\ \hline \mathbf{4} & \mathbf{6} \\ \downarrow & \downarrow \\ \mathbf{0100} & \mathbf{0110} \end{array} \;\; \textbf{(Excess-3)} \qquad\qquad \begin{array}{ccc} \mathbf{4} & \mathbf{3} & \mathbf{0} \\ \mathbf{+\,3} & \mathbf{+\,3} & \mathbf{+3} \\ \hline \mathbf{7} & \mathbf{6} & \mathbf{3} \\ \downarrow & \downarrow & \downarrow \\ \mathbf{0111} & \mathbf{0110} & \mathbf{0011} \end{array} \;\; \textbf{(Excess-3)}$$

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

<u>Example:</u> 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

Character	*ASCII*	*Hexadecimal*
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 represents 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?

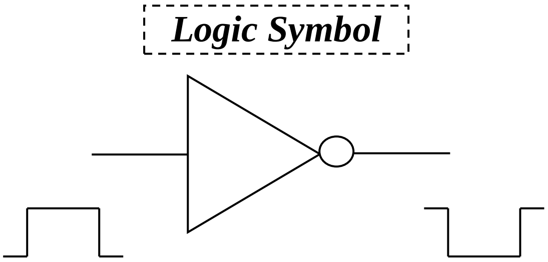
1- Inverter (NOT Gate):

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 ( )	Output ( )
0	1
1	0

Logic Symbol



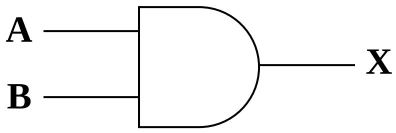
2- AND Gate:

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.

AND Gate Truth Table

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

Logic Symbol



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

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: $N=2^4=16$

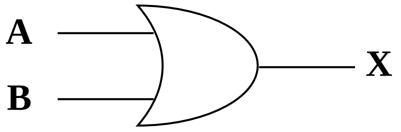
3- OR Gate:

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 know as logical addition.

OR Gate Truth Table

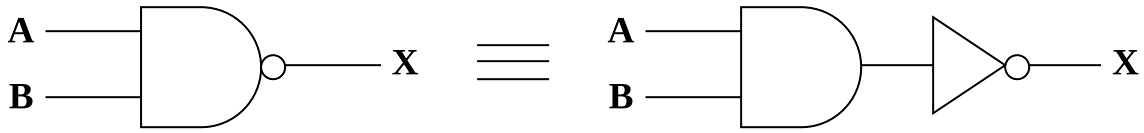
Inputs		Output
A	**B**	**X**
0	0	0
0	1	1
1	0	1
1	1	1

Logic Symbol



4- NAND Gate:

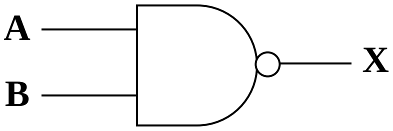
The NAND gate is a popular logic element because it can be used as a universal gate; that is; NAND gate can be used to perform the AND, OR, and Inverter operations, or any combination of these operations. The term NAND is a contraction of NOT-**AND** and implies an AND function with a complemented (Inverted)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



NAND Gate Truth Table

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

Logic Symbol



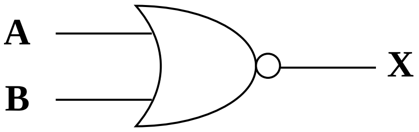
5- NOR Gate:

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.

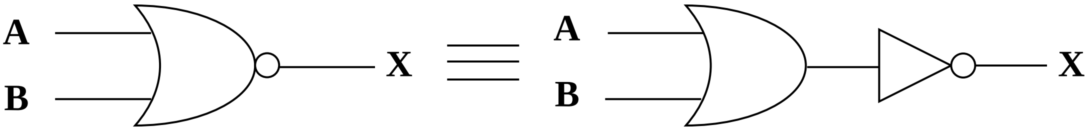
NOR Gate Truth Table

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.



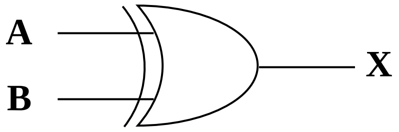
6- Exclusive-OR Gate (XOR):

The Exclusive-OR is actually formed by a combination of other gates.

XOR Gate Truth Table

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

Logic Symbol



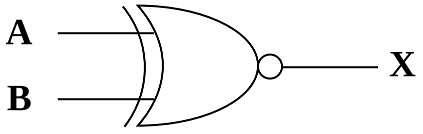
7- Exclusive-NOR Gate (XNOR):

The Exclusive-NOR is actually formed by a combination of other gates.

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.

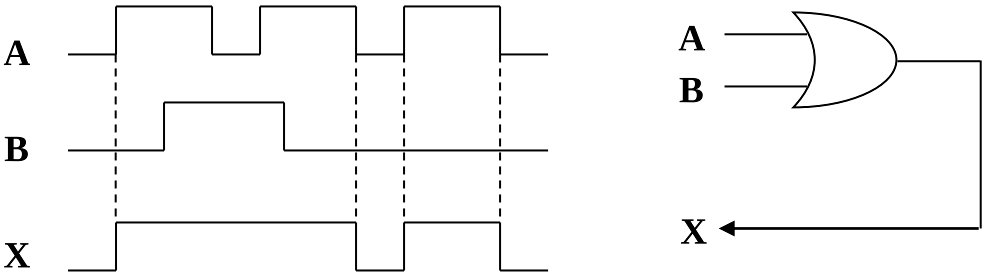
(b) Determine the total number of possible input combinations for a 5-input AND gate.

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

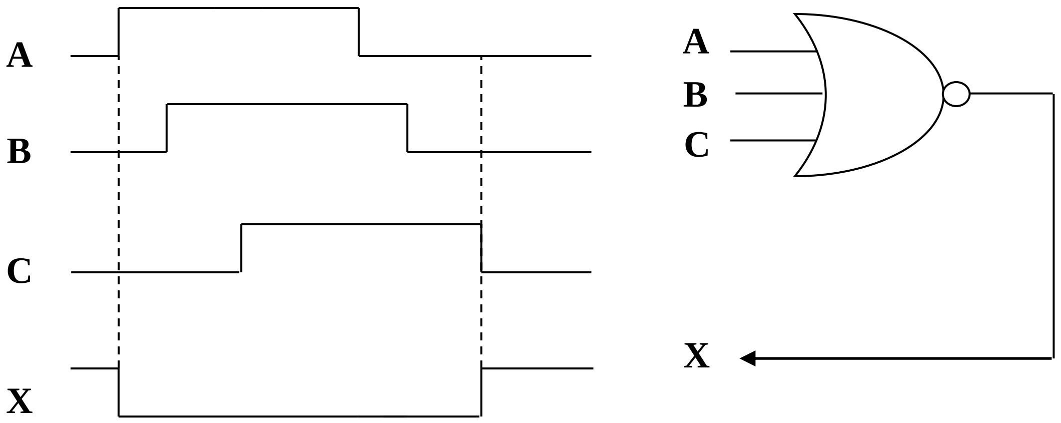
For branch (b), N=2^5 =32. There are 32 possible combinations of input bits for a 5-input AND gate.

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



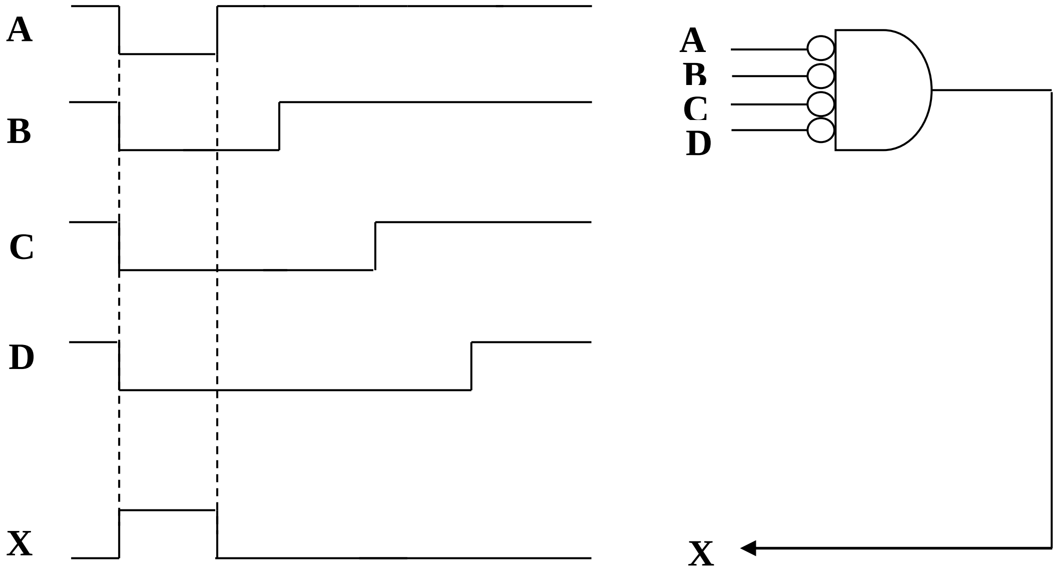
When either or both inputs are **HIGH**, the output is **HIGH** as shown by the output waveform X in the timing diagram.

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.



Boolean algebra:

Boolean algebra is the mathematics of digital systems. It is important that you understand is 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

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

NOR Gate : if inputs are *A* and *B*, the output is

XOR Gate : if inputs are *A* and *B*, the output is $A \oplus B$

XNOR Gate: if inputs are *A* and *B*, the output is $A \odot B$

Laws of Boolean algebra:

1. *Commutative law:*
2. *Associative law:*
3. *Distributive law:*

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.**	$A. \quad =0$
3.	$A.0=0$	**9.**	$=A$
4.	$A.1=A$	**10.**	$A+AB=A$
5.	$A+A=A$	**11.**	$A+ \quad B=A+B$
6.	$A+ \quad =1$	**12.**	$(A+B)(A+C)=A+BC$

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

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

A	B	AB	$A+AB$
0	0	0	0
0	1	0	0
1	0	0	1
1	1	1	1

Equal

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

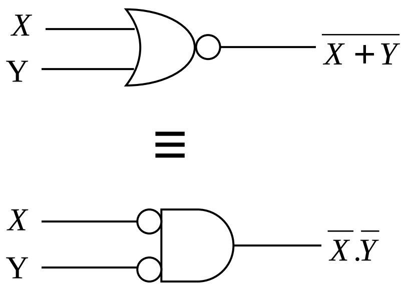
Equal

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:

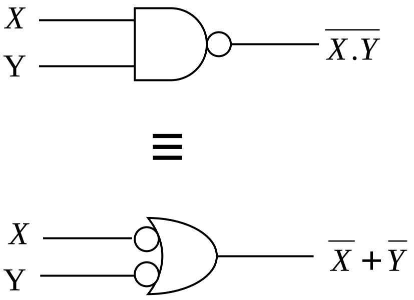
1- $\overline{X+Y} = \overline{X}.\overline{Y}$

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



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

X	Y	$\overline{X.Y}$	$\overline{X} + \overline{Y}$
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)** $\overline{A\overline{B}+\overline{C}D+EF}$

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

$$= \overline{A}.\overline{B}.\overline{C} + \overline{D}$$

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

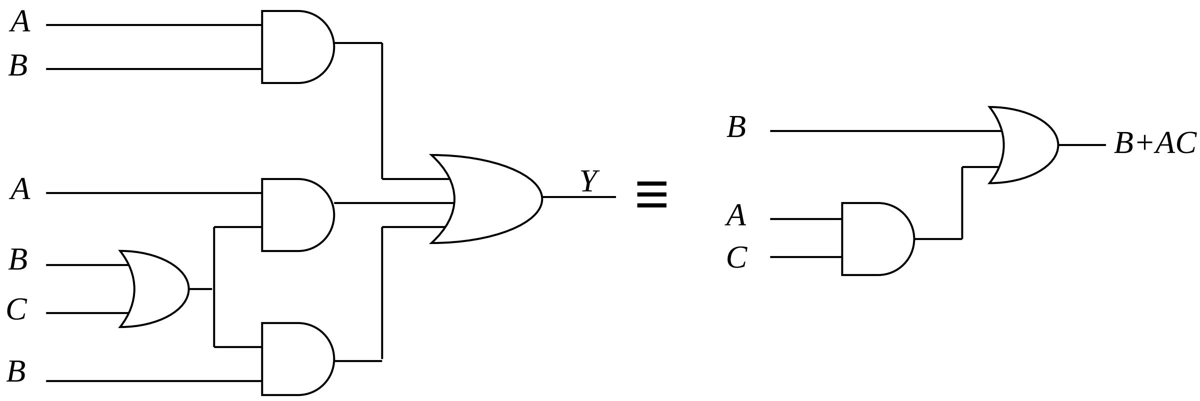
$$= (\overline{A}+B)(C+\overline{D})(\overline{E}+\overline{F})$$

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: $Y = AB + A(B + C) + B(B + C)$.

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



Example: Simplify the expression:

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

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

$$BC.1 + A\bar{B}(C + \bar{C}) + \bar{A}\bar{B}\bar{C}$$

$$BC + A\bar{B}.1 + \bar{A}\bar{B}\bar{C} = BC + \bar{B}(A + \bar{A}\bar{C}) = BC + \bar{B}(A + \bar{C}) = BC + A\bar{B} + \bar{B}\bar{C}$$

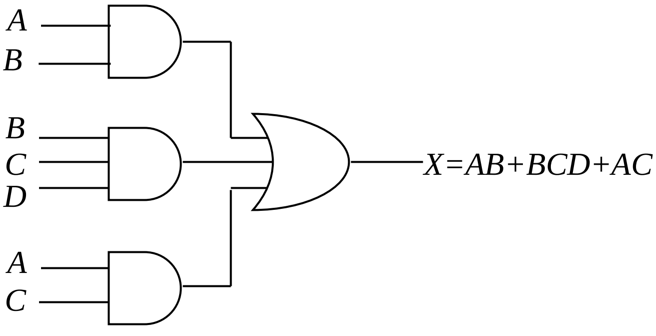
The Sum-of-Product (SOP) form:

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

$$AB + ABC$$

$$ABC + CDE + \bar{B}C\bar{D}$$

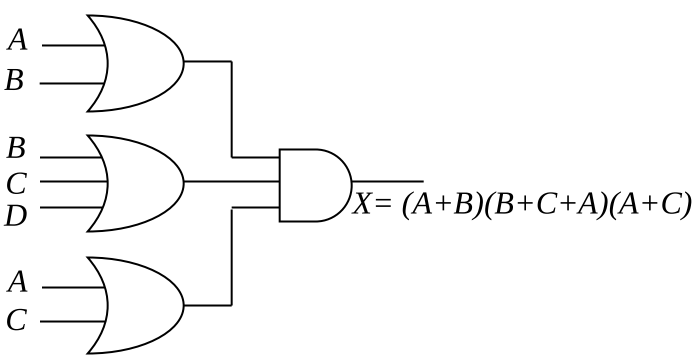
$$\bar{A}B + \bar{A}B\bar{C} + AC$$



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:

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



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

(a) $AB+B(CD+EF)$ (b) $\overline{(\overline{A+B})+C}$

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

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

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

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

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

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

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

$hence: A\overline{B}C+\overline{A}\overline{B}+AB\overline{C}D=A\overline{B}CD+A\overline{B}C\overline{D}+\overline{A}\overline{B}CD+\overline{A}\overline{B}C\overline{D}+\overline{A}\overline{B}\overline{C}\overline{D}+AB\overline{C}D$

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

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

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

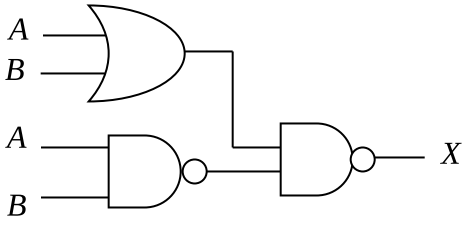
$*(\overline{B}+C+\overline{D})\ \ =\overline{B}+C+\overline{D}+A\overline{A}=(A+\overline{B}+C+\overline{D})(\overline{A}+\overline{B}+C+\overline{D})$

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

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

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

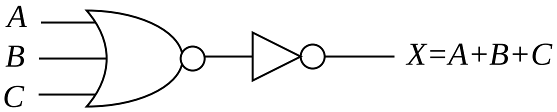
$$X = \overline{(A+B).(\overline{A.B})}$$
$$= \overline{(A+B)} + (\overline{\overline{A.B}})$$
$$= \overline{A}\overline{B} + AB \equiv XNOR\ Gate$$



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 $Y = A\overline{B}D + A\overline{B}\overline{D}$.

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

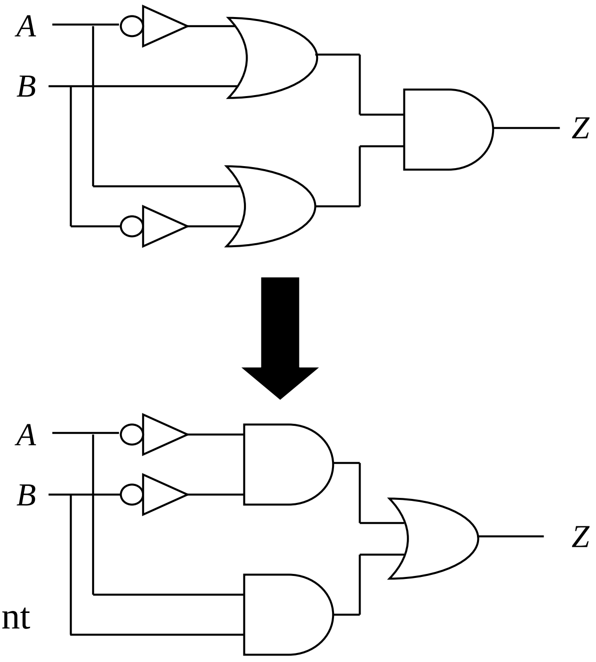
Example: Simplify the expression $X = ACD + \overline{A}BCD$.

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

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

The expression for output Z is:

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



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

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

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 Figrue below:

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

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

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

A	B	C	X	
0	0	0	1	$\overline{A}\,\overline{B}\,\overline{C}$
0	0	1	1	$\overline{A}\,\overline{B}C$
0	1	0	1	$\overline{A}B\overline{C}$
0	1	1	0	
1	0	0	0	
1	0	1	0	
1	1	0	1	$AB\overline{C}$
1	1	1	0	

$$\boxed{\begin{aligned} X &= \overline{A}\,\overline{B}\,\overline{C} + \overline{A}\,\overline{B}C \\ &+ \overline{A}B\overline{C} + AB\overline{C} \end{aligned}}$$

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

A	B	C	D	X	
0	0	0	0	0	
0	0	0	1	1	$\overline{A}\,\overline{B}\,\overline{C}D$
0	0	1	0	0	
0	0	1	1	0	
0	1	0	0	0	
0	1	0	1	1	$\overline{A}B\overline{C}D$
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	$AB\overline{C}D$
1	1	1	0	0	
1	1	1	1	1	$ABCD$

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

$$\boxed{\begin{aligned} X &= \overline{A}\,\overline{B}\,\overline{C}D + \overline{A}B\overline{C}D \\ &\quad + AB\overline{C}D + ABCD \end{aligned}}$$

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:

$$\bar{A}\bar{B}C + \bar{A}B\bar{C} + AB\bar{C} + ABC$$

$\bar{A}\bar{B}C \longrightarrow$ **001**

$\bar{A}B\bar{C} \longrightarrow$ **010**

$AB\bar{C} \longrightarrow$ **110**

$ABC \longrightarrow$ **111**

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

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

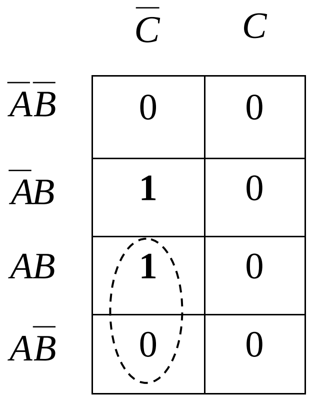
$$\bar{A} + A\bar{B} + AB\bar{C}$$

$\bar{A}$	$\longrightarrow \bar{A}\bar{B}\bar{C}$	**000**
	$\longrightarrow \bar{A}\bar{B}C$	**001**
	$\longrightarrow \bar{A}B\bar{C}$	**010**
	$\longrightarrow \bar{A}BC$	**011**
$A\bar{B}$	$\longrightarrow A\bar{B}\bar{C}$	**100**
	$\longrightarrow A\bar{B}C$	**101**
$AB\bar{C}$	$\longrightarrow AB\bar{C}$	**110**

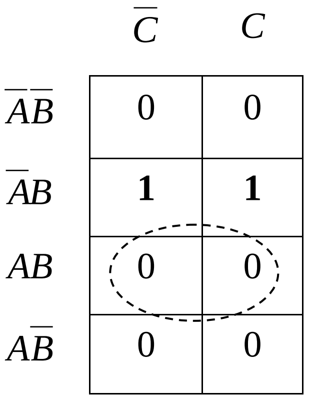
	$\bar{C}$	C
$\bar{A}\bar{B}$	**1**	**1**
$\bar{A}B$	**1**	
AB	**1**	**1**
$A\bar{B}$	**1**	**1**

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:-



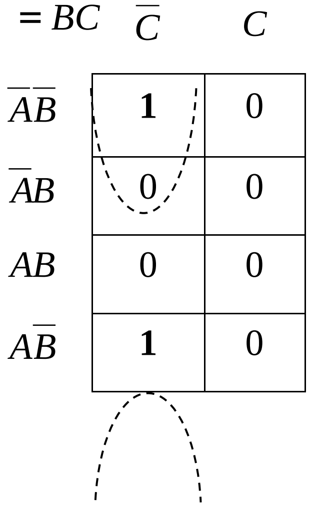
	$\overline{C}$	C
$\overline{A}\,\overline{B}$	0	0
$\overline{A}B$	**1**	0
AB	**1**	0
$A\overline{B}$	0	0



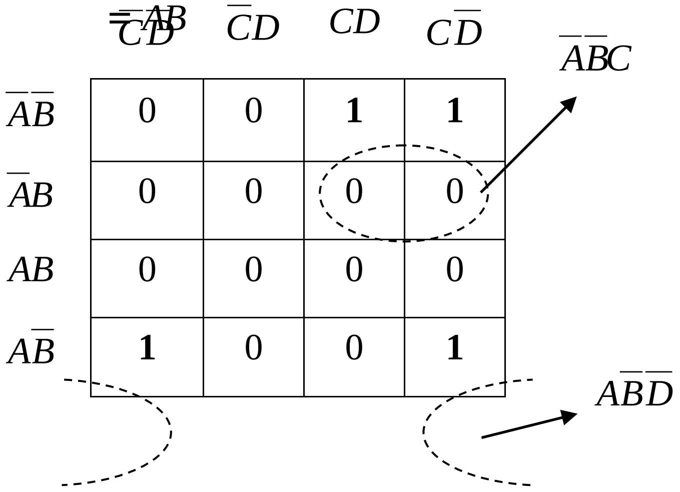
	$\overline{C}$	C
$\overline{A}\,\overline{B}$	0	0
$\overline{A}B$	**1**	**1**
AB	0	0
$A\overline{B}$	0	0

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

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



	$\overline{C}$	C
$\overline{A}\,\overline{B}$	**1**	0
$\overline{A}B$	0	0
AB	0	0
$A\overline{B}$	**1**	0

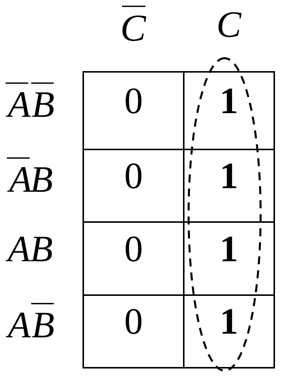


	$\overline{C}\,\overline{D}$	$\overline{C}D$	CD	$C\overline{D}$
$\overline{A}\,\overline{B}$	0	0	**1**	**1**
$\overline{A}B$	0	0	0	0
AB	0	0	0	0
$A\overline{B}$	**1**	0	0	**1**

$$X = \overline{A}\,\overline{B}\,\overline{C} + A\overline{B}\,\overline{C}$$
$$= \overline{B}\,\overline{C}$$

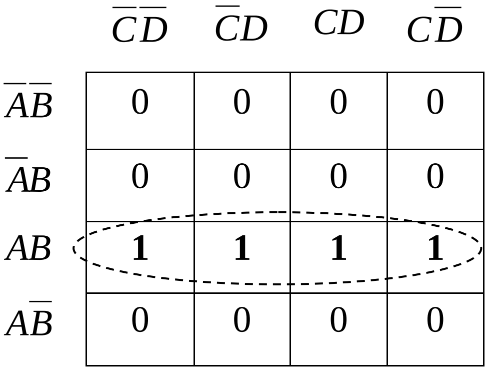
$$X = \overline{A}\,\overline{B}CD + \overline{A}\,\overline{B}C\overline{D} + A\overline{B}\,\overline{C}D + A\overline{B}C\overline{D}$$
$$= \overline{A}\,\overline{B}C + A\overline{B}\,\overline{D}$$

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:



	$\overline{C}$	C
$\overline{A}\,\overline{B}$	0	**1**
$\overline{A}B$	0	**1**
AB	0	**1**
$A\overline{B}$	0	**1**

$$X = C$$



	$\overline{C}\,\overline{D}$	$\overline{C}D$	CD	$C\overline{D}$
$\overline{A}\,\overline{B}$	0	0	0	0
$\overline{A}B$	0	0	0	0
AB	**1**	**1**	**1**	**1**
$A\overline{B}$	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	**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**	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: $X = B.(A+\bar B).(B+C)$

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**

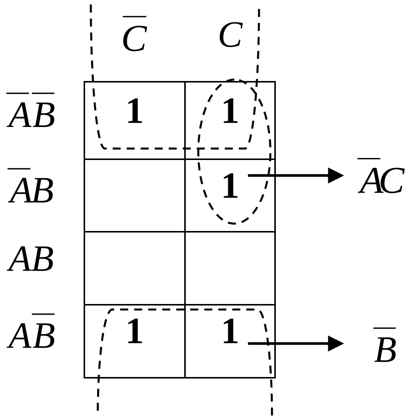
	$\bar C$	C
$\bar A\bar B$		
$\bar A B$		
AB	**1**	**1**
$A\bar B$		

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

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

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

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



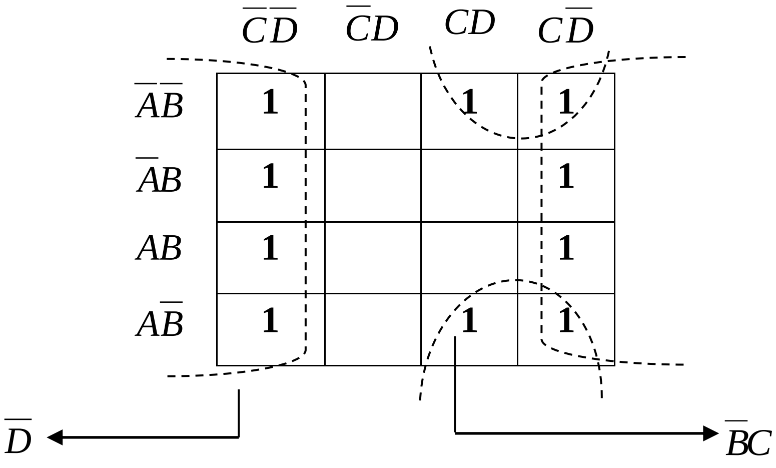
	$\overline{C}$	C
$\overline{A}\,\overline{B}$	**1**	**1**
$\overline{A}B$		**1**
AB		
$A\overline{B}$	**1**	**1**

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

$$\overline{B}\,\overline{C}\,\overline{D} + \overline{A}B\overline{C}\,\overline{D} + AB\overline{C}\,\overline{D} + \overline{A}\,\overline{B}CD + A\overline{B}CD + \overline{A}\,\overline{B}C\overline{D} + \overline{A}BC\overline{D} + ABC\overline{D} + A\overline{B}C\overline{D}.$$

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

$X = \overline{D} + \overline{B}C$



	$\overline{C}\,\overline{D}$	$\overline{C}D$	CD	$C\overline{D}$
$\overline{A}\,\overline{B}$	**1**		**1**	**1**
$\overline{A}B$	**1**			**1**
AB	**1**			**1**
$A\overline{B}$	**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

Don't Care

	$\overline{C}$	C
$\overline{A}\overline{B}$	0	0
$\overline{A}B$	0	x
AB	**1**	**1**
$A\overline{B}$	x	**1**

	$\overline{C}$	C
$\overline{A}\overline{B}$	0	0
$\overline{A}B$	0	0
AB	**1**	**1**
$A\overline{B}$	**1**	**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).

$\boxed{Z = A}$

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.

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

1100 1011 0010 1111 0011

AB \ CD	00 $\overline{C}\overline{D}$	01 $\overline{C}D$	11 CD	10 $C\overline{D}$
00 $\overline{A}\overline{B}$			**0**	**0**
01 $\overline{A}B$				
11 AB	**0**		**0**	
10 $A\overline{B}$			**0**	

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

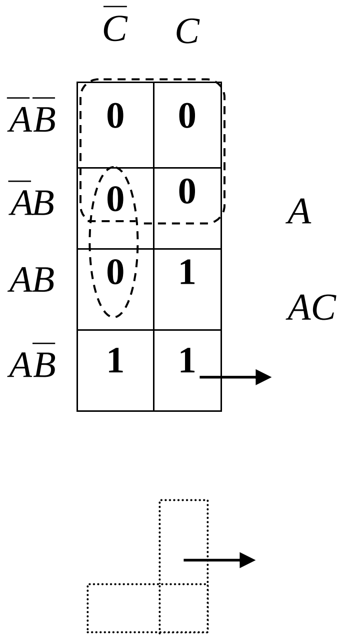
$\overline{A}+B+\overline{C}+\overline{D}$

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

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

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

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 .



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

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

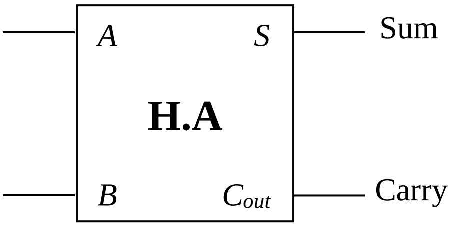
Basic Adders:

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.

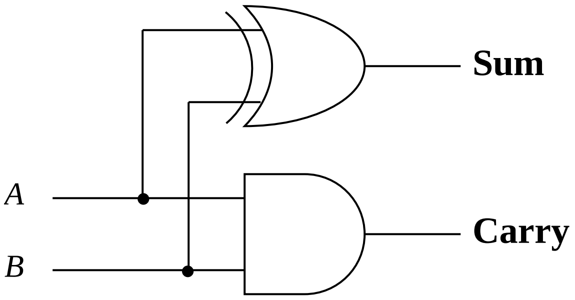
1- *The Half-Adder (H.A):*

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

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.

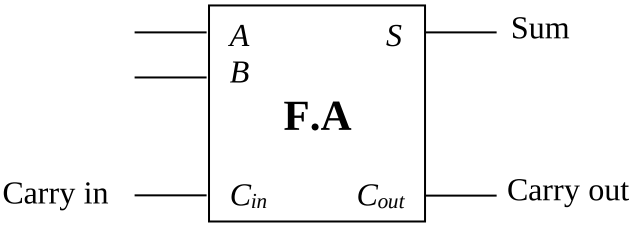


A	B	C_{out}	S
0	**0**	**0**	**0**
0	**1**	**0**	**1**
1	**0**	**0**	**1**
1	**1**	**1**	**0**



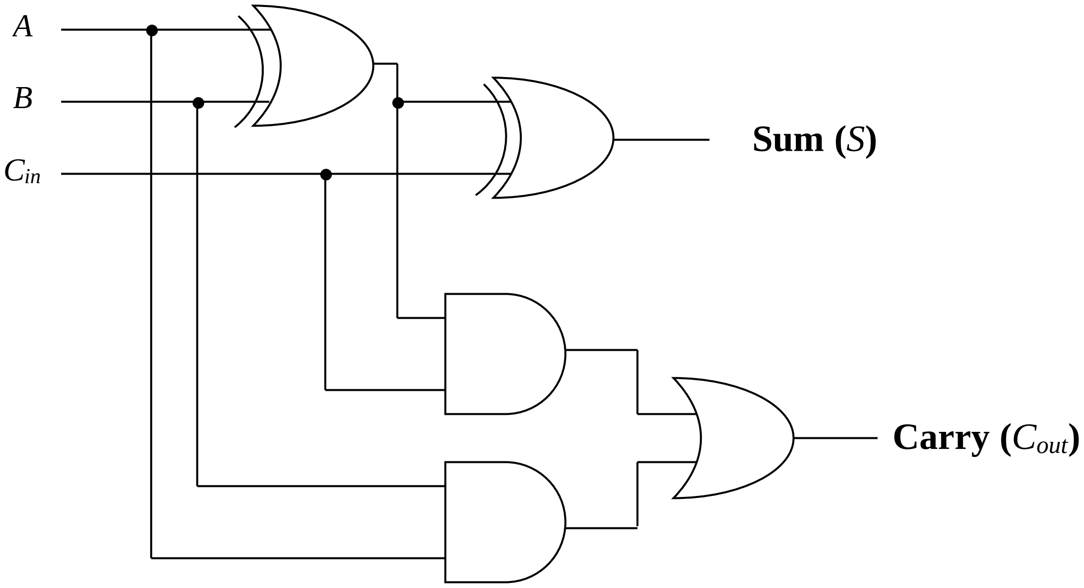
2- *The Full-Adder(F.A):*

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.**

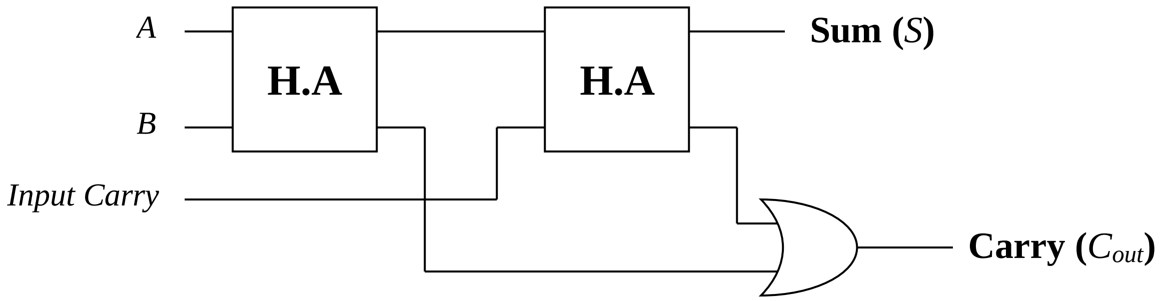


A	B	C_{in}	C_{out}	S
0	**0**	**0**	**0**	**0**
0	**0**	**1**	**0**	**1**
0	**1**	**0**	**0**	**1**
0	**1**	**1**	**1**	**0**
1	**0**	**0**	**0**	**1**
1	**0**	**1**	**1**	**0**
1	**1**	**0**	**1**	**0**
1	**1**	**1**	**1**	**1**

From the truth table:-



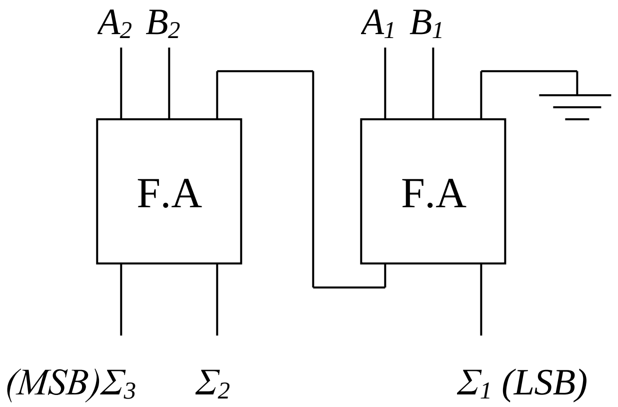
Example: Arrange two half-adders to form full-adder.



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.

$$\begin{array}{r} A_2A_1 \\ +\ B_2B_1 \\ \hline \Sigma_3\ \Sigma_2\ \Sigma_1 \end{array}$$



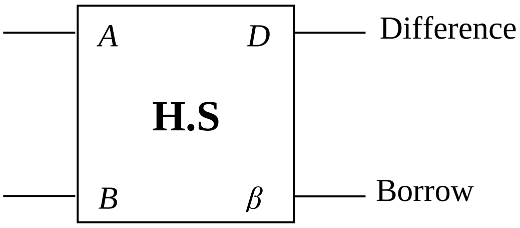
Half- and Full- Subtractors:

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

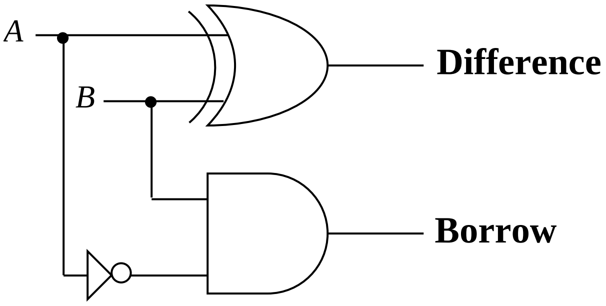
1- *The Half-Subtractor (H.S):*

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

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.



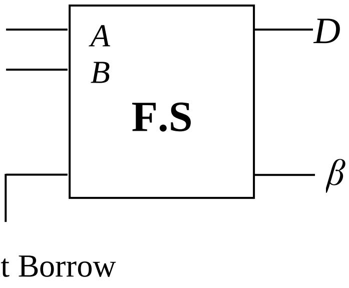
A	*B*	*Borrow ( )*	*Difference (D)*
0	**0**	**0**	**0**
0	**1**	**1**	**1**
1	**0**	**0**	**1**
1	**1**	**0**	**0**

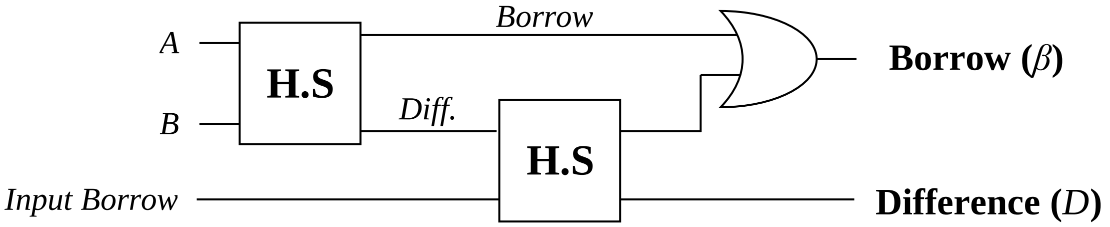


2- *The Full- Subtractor (F.S):*

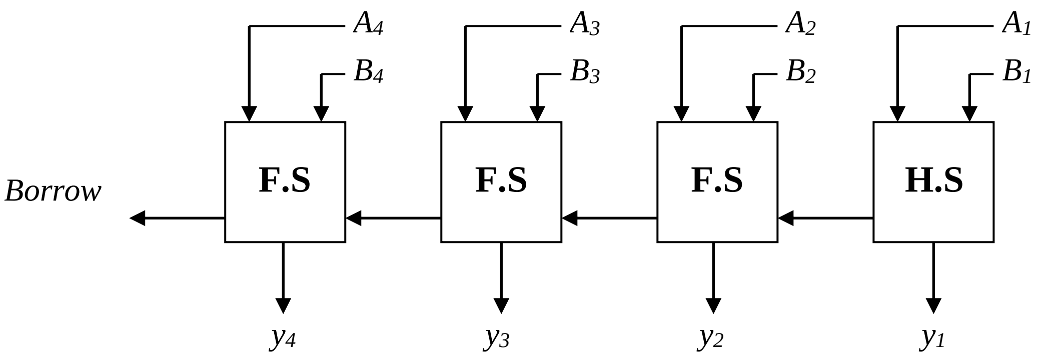
The half-subtractor handles only two bits at a time and can be used for the least significant column of a subtraction problem. To take care of a higher-order column, we need a full-subtractor.

The full-subtractor uses two half-subtractors And an OR gate as shown in figure below:



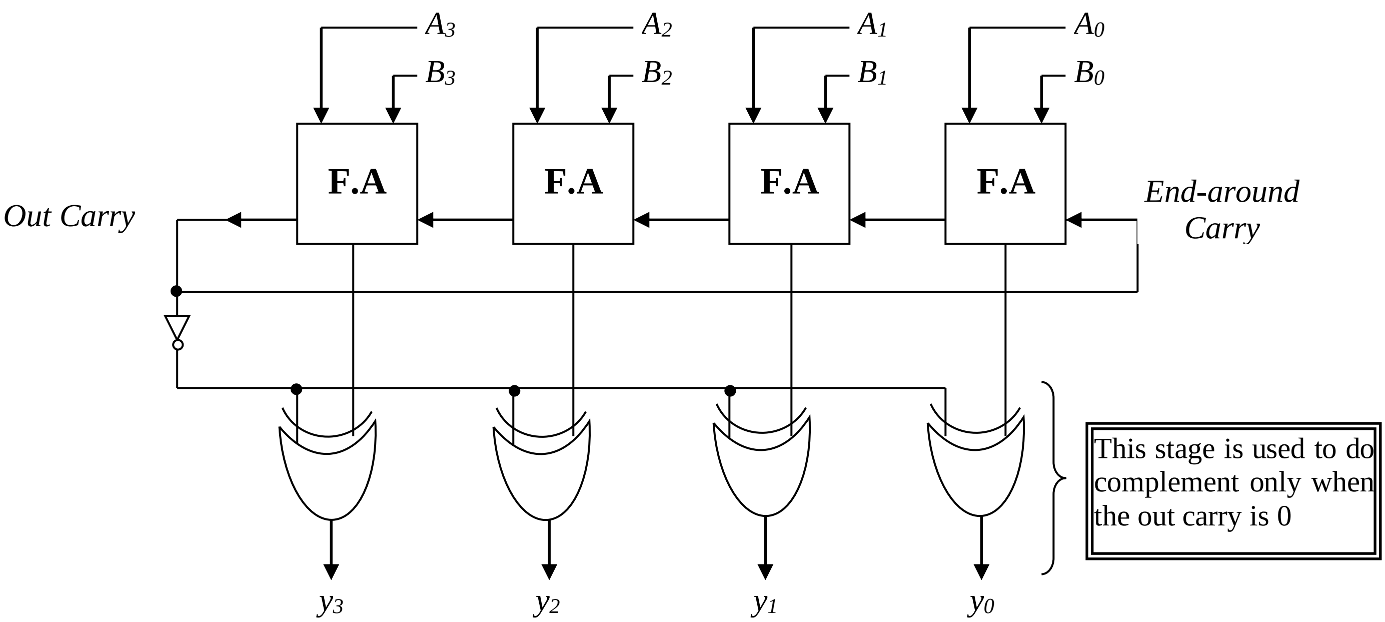


Half- and Full- subtractors are analogous to half- and full- adders; by cascading half- and full- subtractors as shown in figure below, we have a system that directly subtract .



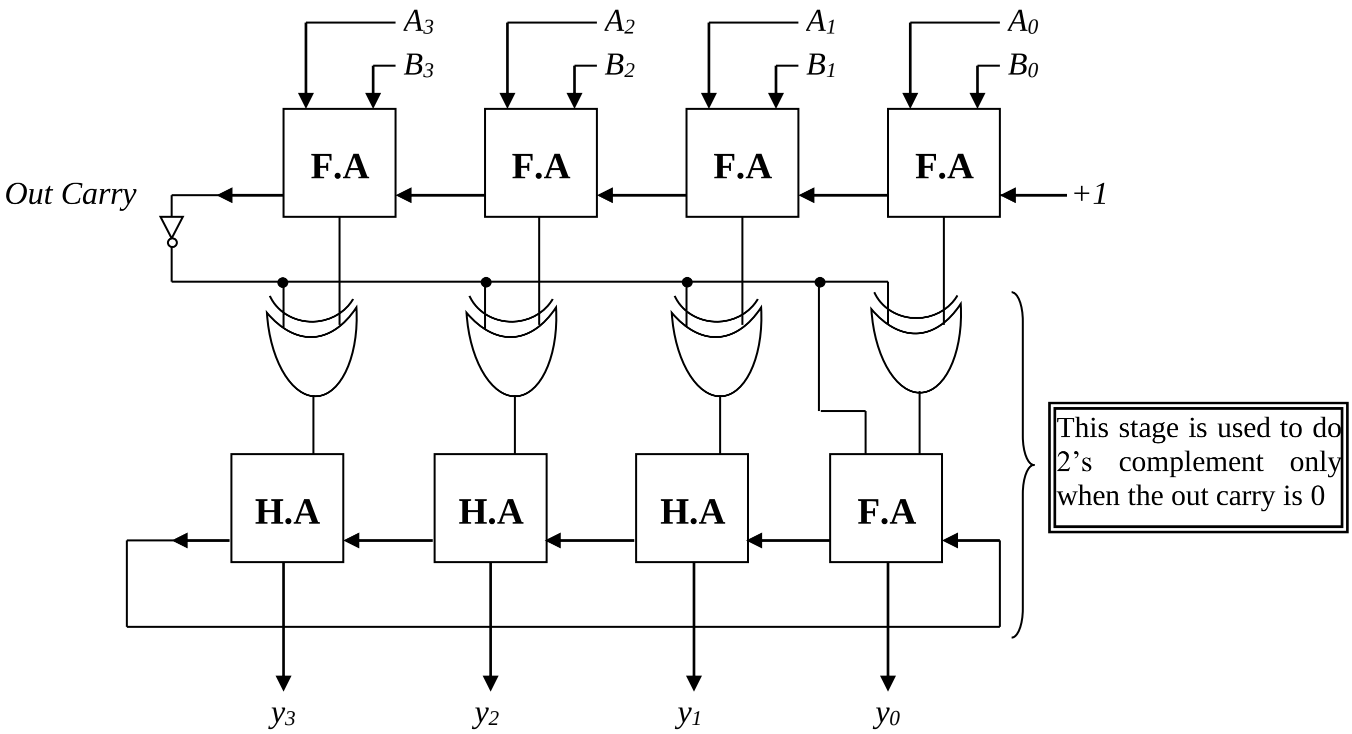
1's Complement Subtractors:

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 , the 1's complement of $B_3B_2B_1B_0$. The full-adders add A and B and the end-around carry.



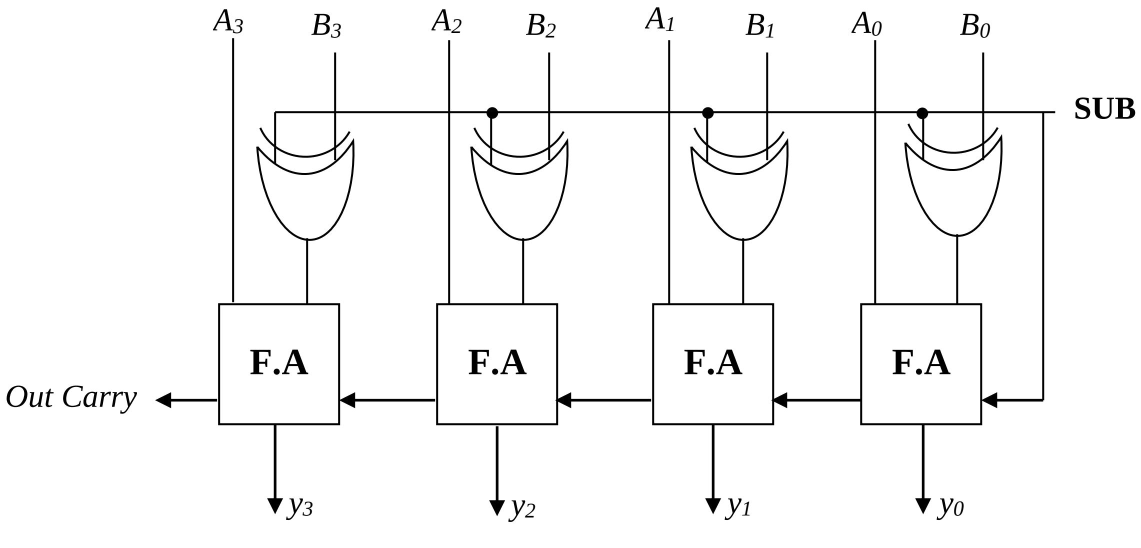
This stage is used to do complement only when the out carry is 0

2’s Complement Subtractors:


A3
B3
A2
B2
A1
B1
A0
B0
F.A
F.A
F.A
F.A
Out Carry
+1
H.A
H.A
H.A
F.A
This stage is used to do 2’s complement only when the out carry is 0
y3
y2
y1
y0

2’s Complement Adders/ Subtractors:

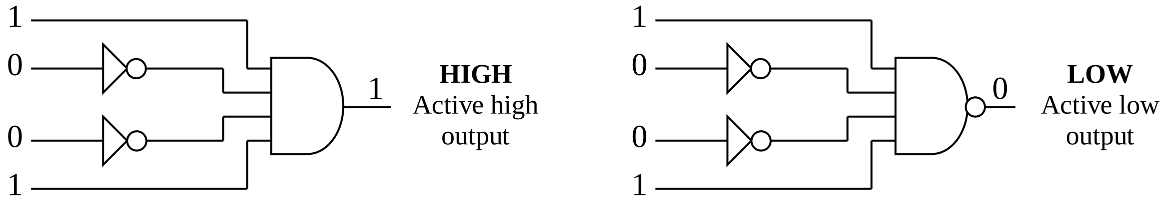
The following figure shows an adder/subtractor based on the 2’s complement. When SUB is **Low**, the *B* bits pass through the controlled inverter to the full-adder. Therefore; the full adder produces the sum of *A*&*B*. When SUB is **High**, the Bits are inverted before reaching the full-adder. Also, 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*.



Decoders:

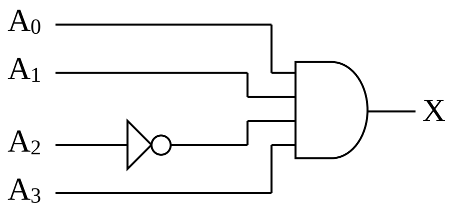
The basic function of a decoder is to detect the presence of a specified combination of bits (code) on its inputs and to indicate the presence of that code by a specified output level. In its general form, a decoder has *n* lines to handle *n* bits and from one to 2^n output lines to indicate the presence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

Example: A decoder for binary number 1001.



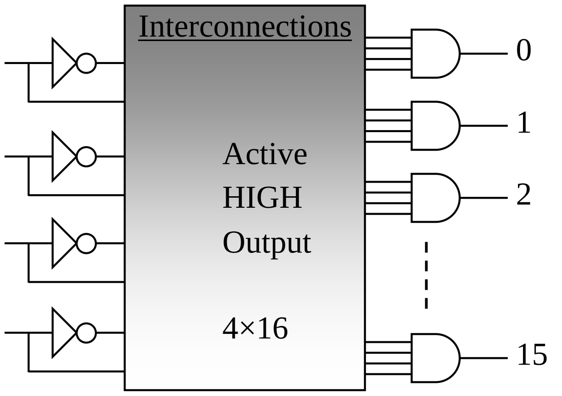
Example: Determine the logic required to decode the binary number 1011 by producing a **HIGH** level on the output.

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



Four bit binary Decoders:

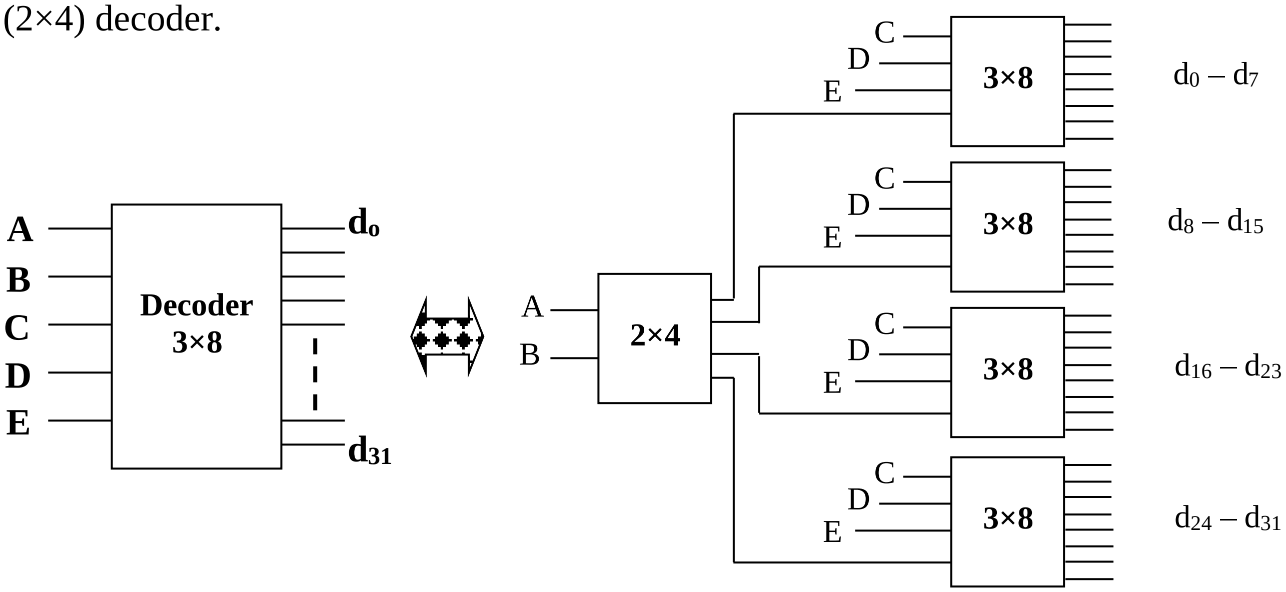
In order to decode all possible combinations of four bits, sixteen decoding gates are required (2^4=16). This type of decoder is commonly called a 4-line-to-16-line decoder because there are four inputs and sixteen outputs or a 1-of-16 decoder because for any given code on the inputs, one of the sixteen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



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

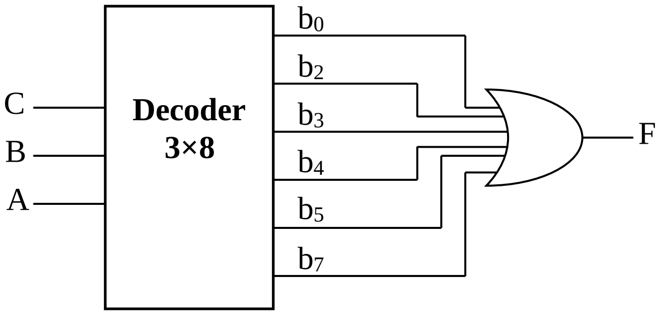
Decimal	***Binary Input***				***Output***															
	A_3	A_2	A_1	A_0	***0***	***1***	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***	***10***	***11***	***12***	***13***	***14***	***15***
0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	1	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

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



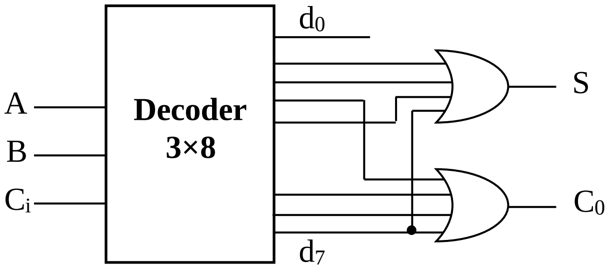
Example: Design the following Boolean function with a (3×8) decoder and OR gate:

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



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



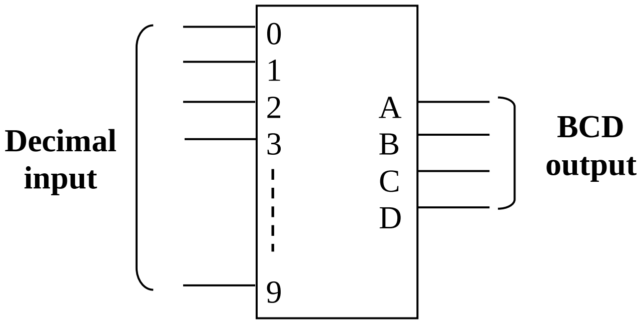
Encoder:

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:

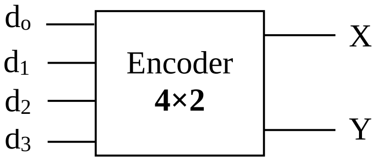
This type of encoder has ten inputs – one for each decimal digit – and four outputs corresponding to the BCD code as shown in figure below. This is a basic 10-line-to-4-line encoder.

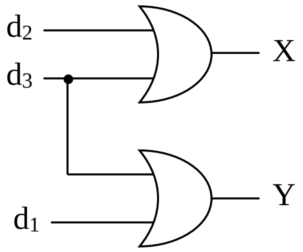
Decimal	D	C	B	A
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1



The expressions for output in term of decimal are:

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

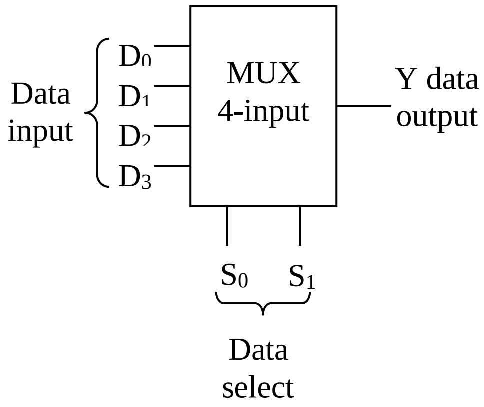




d_0	d_1	d_2	d_3	X	Y
1	0	0	0	0	0
0	1	0	0	0	1
0	0	1	0	1	0
0	0	0	1	1	1

Multiplexers (Data Selectors):

A multiplexer (MUX) is a device that allows digital information from several sources to be routed onto a single line for transmission over that line to a common destination. The basic multiplexer has several data-input lines and a single output line. It is also has data-select inputs, which permit digital data on any one of the inputs to be switched to the output line. Multiplexers are also known as data selectors.



S_0	S_1	Input selected
0	0	D_0
0	1	D_1
1	0	D_2
1	1	D_3

Now let’s look at the logic circuitry required to perform this multiplexing operation. The data output is equal to the state of the selected data input. Therefore, a logical expression for the output in terms of data input and the selected input may be derived.

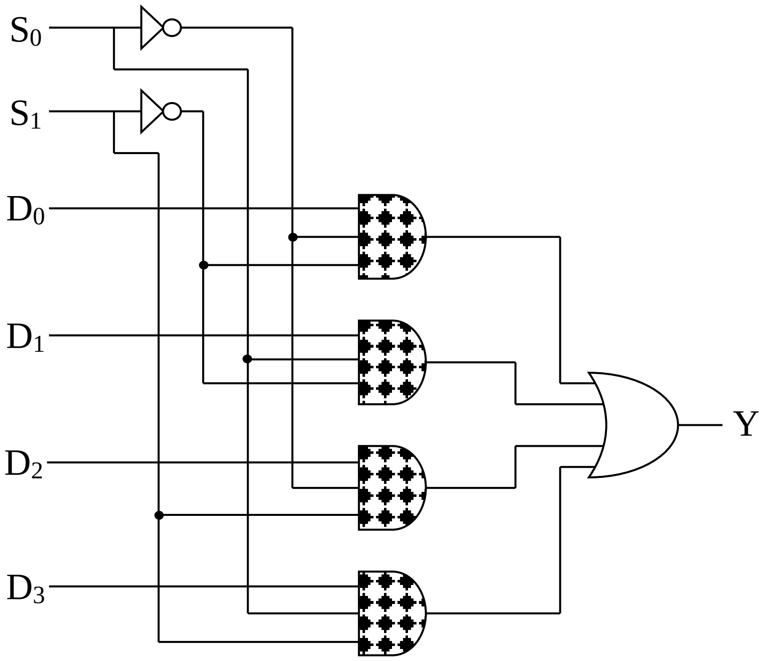
(*Output equal D_0 only when S_1=0 and S_0=0*)

(*Output equal D_1 only when S_1=0 and S_0=1*)

(*Output equal D_2 only when S_1=1 and S_0=0*)

(*Output equal D_3 only when S_1=1 and S_0=1*)

When these terms are **OR**ed, the total expression for the data output is:

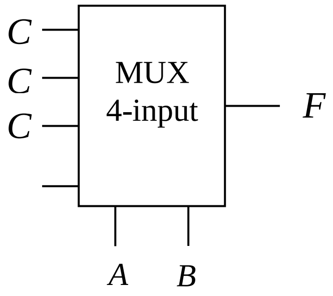


Example: Design the following boolean function using (4×1) MUX. . (A, B are used as a selector switches).

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

F=C

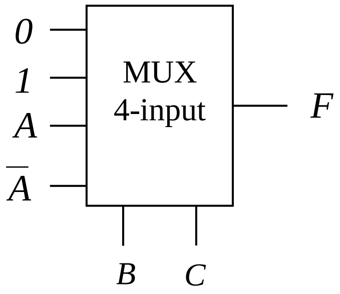
F=



	I_0	I_1	I_2	I_3
	0	2	4	**6**
C	**1**	**3**	**5**	7
	C	*C*	*C*	

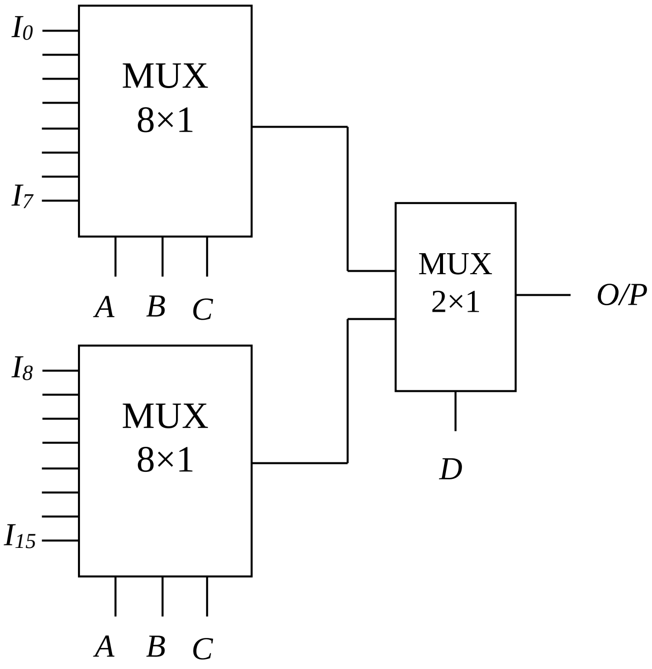
AB=00 results in *F*=*C*
AB=01 results in *F*=*C*
AB=10 results in *F*=*C*
AB=11 results in *F*=

Example: Design the following boolean function using (4×1) MUX. $F(A,B,C)=\sum_m(1,3,5,6)$. (B, C are used as a selector switches).



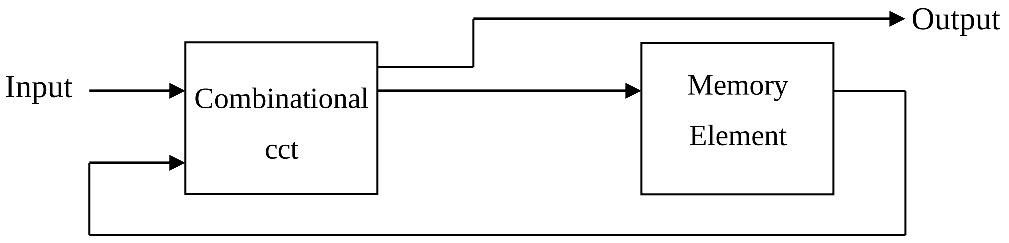
	I_0	I_1	I_2	I_3
$\overline{A}$	0	**1**	2	**3**
A	4	**5**	**6**	7
	0	1	A	$\overline{A}$

Example: Design a (16×1) MUX using 2 of (8×1) MUX and one of (2×1) MUX.



Logic Circuits:

The digital cct. considered thus for have been combinational, i.e., the outputs at any instant of time are entirely dependant upon the inputs presents at that time. Although every digital system likely to have combinational circuits, most systems encountered in practice also includes memory elements, which require the system be described in terms of sequential logic.

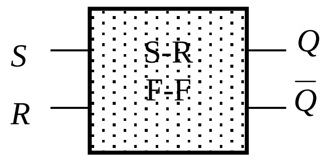


The sequential logic cct. use combinational gates and F-F cct. in their design.

Latches:

The latches are a type of bi-stable storage device that in normally places in a category separate from that of flip-flops. Latches are basically similar to flip-flops because they are bi-stable devices that can reside in either of two states by virtue of feed back arrangement, in which the outputs are connected back to the opposite inputs. The main difference between latches and flip-flops is method used for changing their state.

S-R Latch (Basic F/F)



	$\overline{S}\,\overline{R}$	$\overline{S}R$	$S\overline{R}$	SR
$\overline{Q}$	0	0	x	1
Q	1	0	x	1

S	**R**	**Q_n**	**Q_{n+1}**
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	X
1	1	1	X

If S and R =0
Q_n=Q_{n+1} (no change)

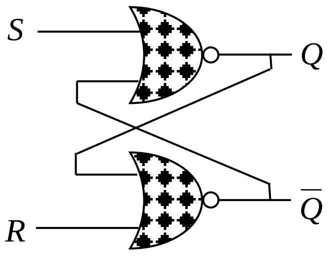
S=0, R=1
Q_{n+1} (Reset)

S=0, R=0
Q_{n+1} (Set)

S=1, R=1
Q_{n+1} (Undefiend)

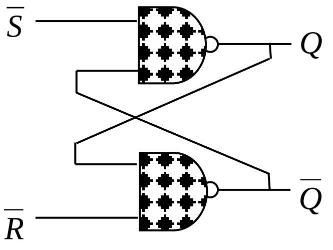
$$Q_{n+1} = S + \overline{R}Q_n$$
$$Q_n = Q_{n+1} = S + \overline{R}Q_n$$
$$\overline{Q_n} = \overline{Q_{n+1}} = \overline{S + \overline{R}Q_n}$$

Using (NOR) gates:



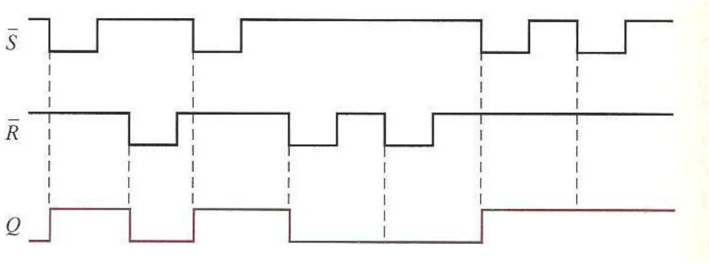
$$Q_{n+1} = S + \overline{R}Q_n$$
$$= \overline{\overline{\overline{\overline{S}} + \overline{\overline{\overline{R}Q_n}}}}$$
$$= \overline{\overline{S + \overline{\overline{Q_n} + R}}}$$

Using (NAND) gates:



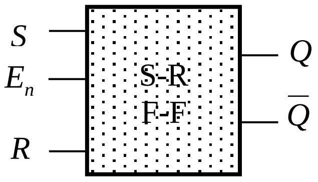
$$Q_{n+1} = S + \overline{R}Q_n$$
$$= \overline{\overline{S + \overline{R}Q_n}}$$
$$= \overline{\overline{S}\,.\,\overline{\overline{R}Q_n}}$$

Example: $\overline{S}, \overline{R}$ waveforms in figure below applied to input of S-R latch, determine the waveform of *Q*. assume that *Q* is initially low.

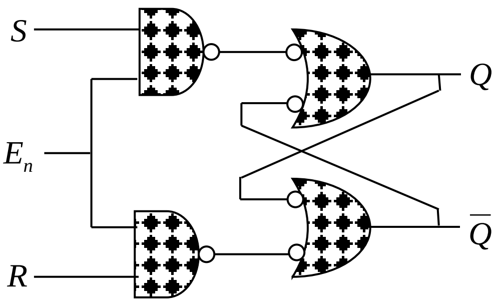


Gated S-R Latch:

A gated latch requires an enable inputs (E_n). The logic diagram and logic symbol for gated S-R latch are shown in following figure. The *S* and *R* inputs control the state to which the latch will go when **high** level is applied to E_n input. The latch will not change until the E_n input is **high**, but as long as it remains **high**, the output is determined by the state of *S* and *R* inputs.

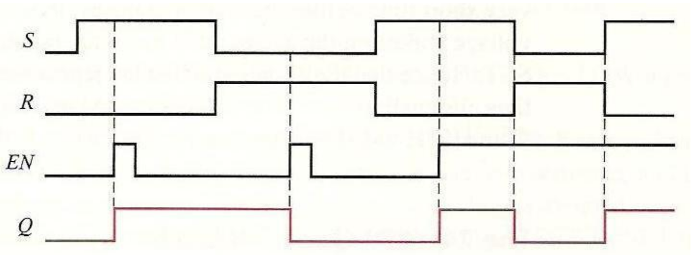


Logic Symbol



Logic Diagram

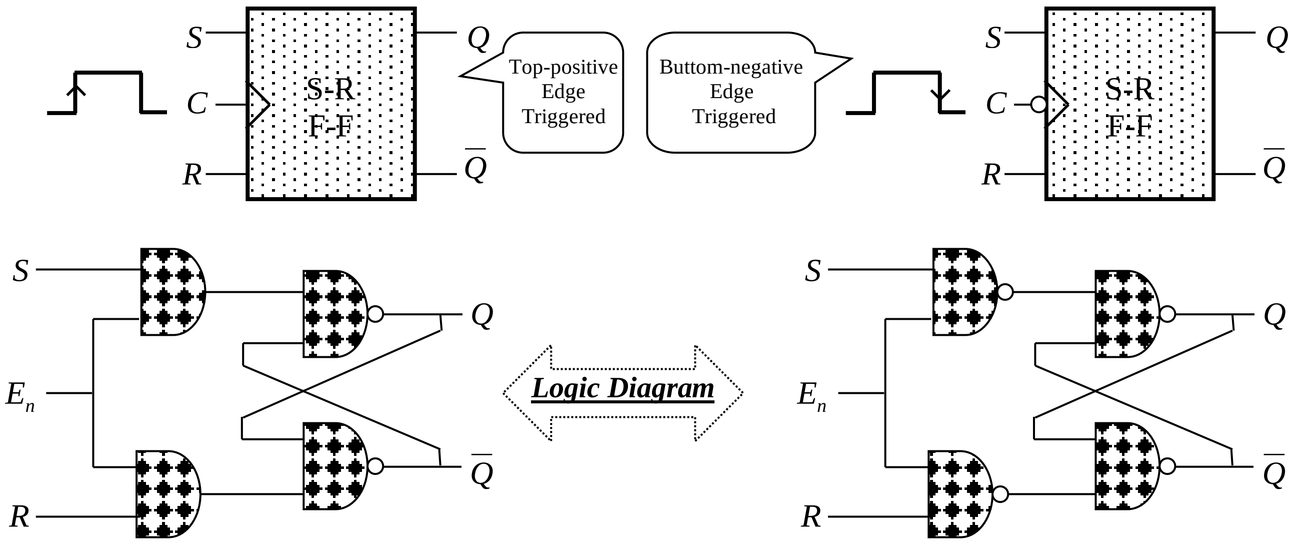
Example: Determine the *Q* waveform if input shown in following figure are applied to a gated S-R latch that is initially RESET.



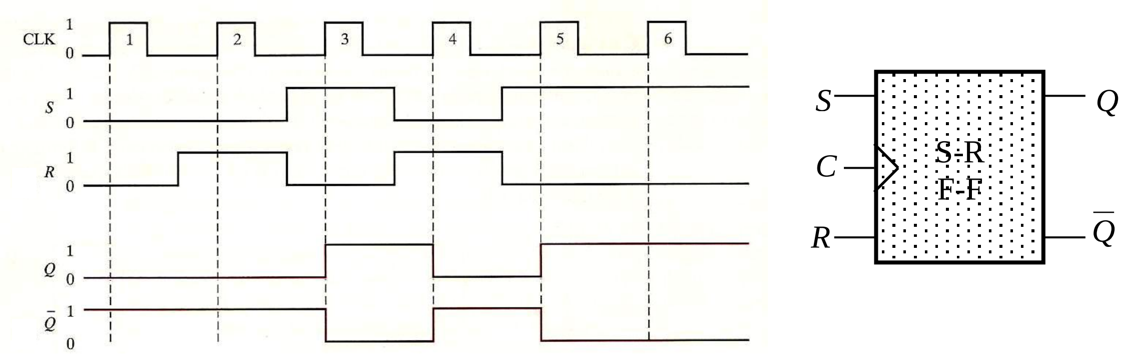
Edge-triggered Flip-Flops:

Flip-Flops are synchronous bi-stable devices. In this case the term synchronous means that the output changes state only at specified point on a triggering input called **Clock** (*designated* <u>*C*</u> *as a control input*); that is, changes in output occur in synchronization with clock.

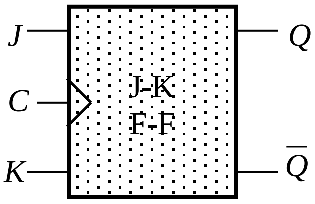
1. *S-R Edge triggered Flip-Flop:*



Example: Determine Q and $\overline{Q}$ output waveform of Flip-Flops in the figure below. Assume that the positive edge triggered F/F is initially RESET.

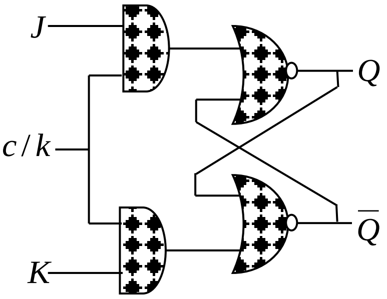


2. *J-K Flip-Flop:*



Symbol Diagram

|J	K	Q_n	Q_{n+1}	
0	0	0	0	(no change)
0	0	1	1	
0	1	0	0	(Reset)
0	1	1	0	
1	0	0	1	(Set)
1	0	1	1	
1	1	0	1	(Toggle)
1	1	1	0	

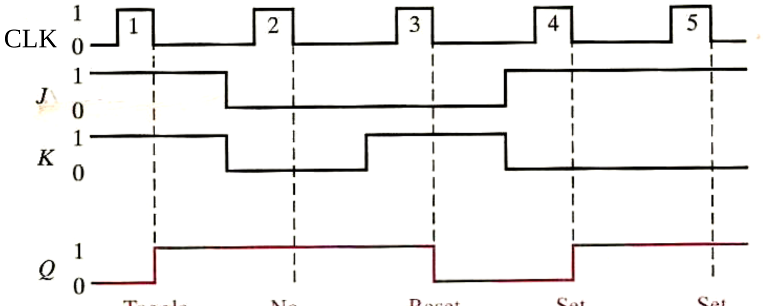


Logic Diagram

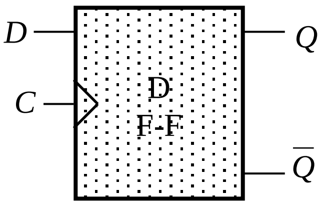
	$\overline{J}\overline{K}$	$\overline{J}K$	$J\overline{K}$	JK
$\overline{Q}$			1	1
Q	1			1

$$\boxed{Q_{n+1} = J\overline{Q}_n + \overline{K}Q_n}$$

Example: Find the output Q for the following input waveforms, assuming negative edge triggered J-K flip-flop.

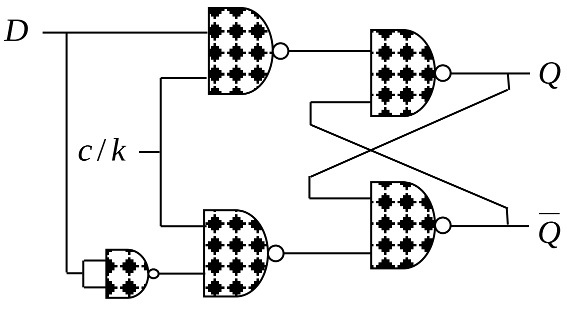


3. *Delay Flip-Flop (D- Flip/Flop):*



Symbol Diagram

D	Q_n	Q_{n+1}
0	0	0
0	1	0
1	0	1
1	1	1

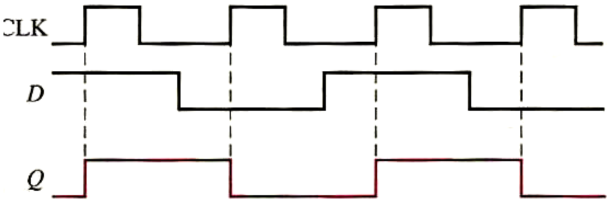


Logic Diagram

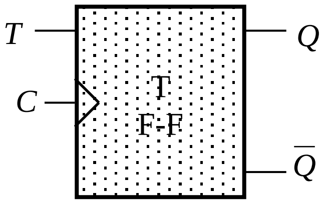
	$\overline{D}$	D
$\overline{Q}$		1
Q		1

$$\boxed{Q_{n+1} = D}$$

Example: Find the output Q for the following input waveforms, assuming positive edge triggered D- flip/flop.



4. *|Toggle Flip-Flop (T- Flip/Flop):*

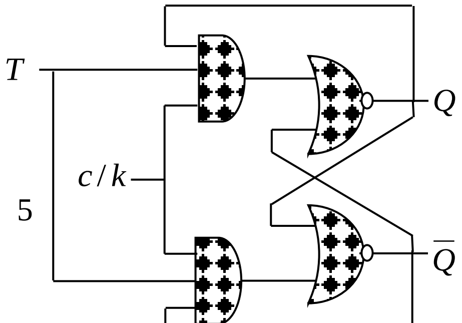


Symbol Diagram

T	Q_n	Q_{n+1}
0	0	0
0	1	1
1	0	1
1	1	0

(no change)

(Toggle Q_n)

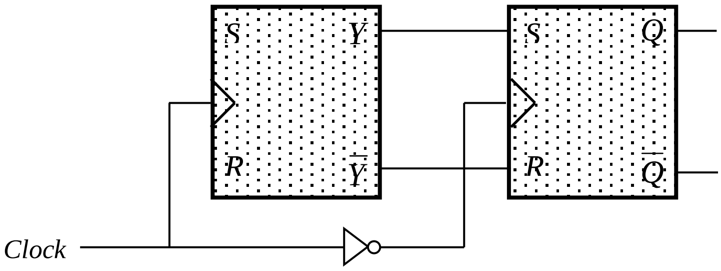


Master-Slave Flip-Flops:

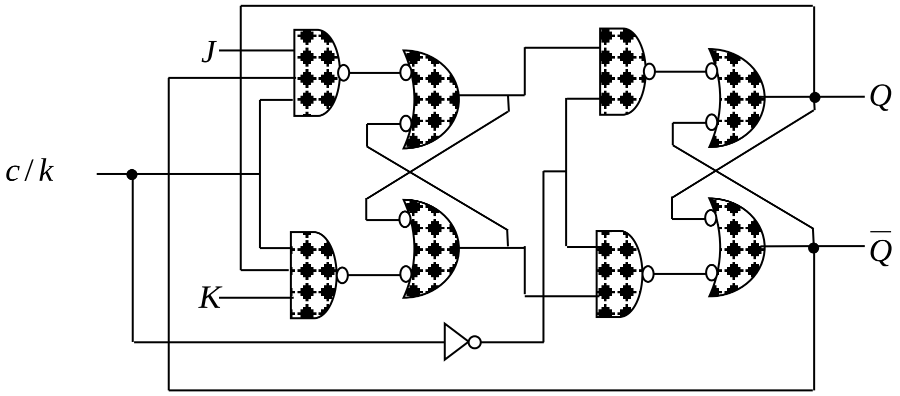
Another class of F/Fs is the master-slave. Although this type of F/Fs has largely been replaced by the edge-triggered devices, a limited selection is still available from Ic manufacturer. There are two basic types of Master-Slave F/F:-

- Pulse triggered: does not allow data changed while clock active.
- Data-look out: has no restriction.

In both types data are entered into F/Fs on the leading edge of the clock pulse, but the output does not reflect the input state until trailing edge.



- Flip Flop without pebble changes output on leading edge of positive going clock pulse.
- Flip Flop with pebble changes output on trailing edge of positive going clock pulse.

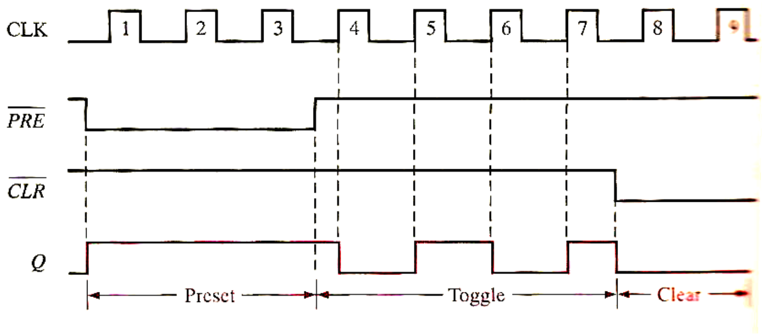


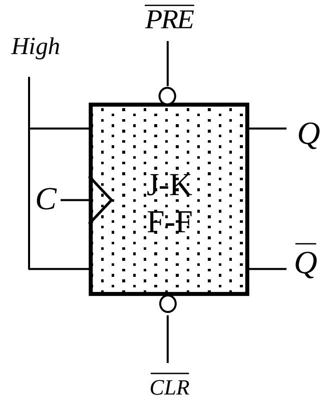
Asynchronous inputs:

The flip-flops discussed before S-R, D, J-K, T inputs are called synchronous inputs because data on these inputs are transferred to the flip-flop output only on triggering edge of the clock pulse. That is, the data are transferred synchronously with the clock. Most integrated circuits flip-flops also have asynchronous inputs. These inputs are affecting the state of the flip-flop independent of the clock. They are normally labeled Preset (**PRE**) and Clear (**CLR**) or direct set (**SD**) and direct reset (**RD**) by some manufacturers.

- An active level on the preset input will *SET* the flip-flop.
- An active level on the clear input will *RESET* the flip-flop.

Example:





Conversion between flip-flops:

Q_n	Q_{n+1}	D	T	S	R	J	K
0	0	0	0	0	X	0	X
0	1	1	1	1	0	1	X
1	0	0	1	0	1	X	1
1	1	1	0	X	0	X	0

Example: Design a D- flip/flop using S-R flip/flop?

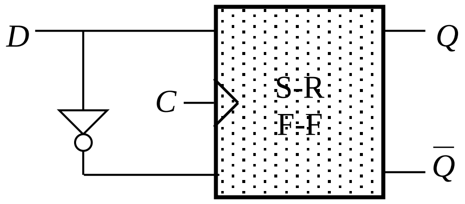
D	Q_n	Q_{n+1}	S	R
0	0	0	0	X
0	1	0	0	1
1	0	1	1	0
1	1	1	X	0

	$\overline{D}$	D
$\overline{Q}_n$	0	1
Q_n	0	x

$$\boxed{S = D}$$

	$\overline{D}$	D
$\overline{Q}_n$	x	0
Q_n	1	0

$$\boxed{R = \overline{D}}$$



Example: Design a T- flip/flop using J-K flip/flop?

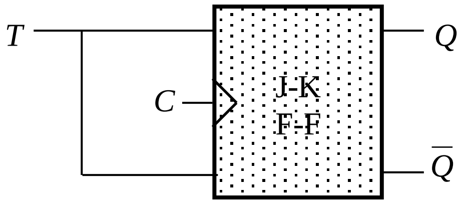
T	***Q_n***	***Q_{n+1}***	***J***	***K***
0	0	0	0	X
0	1	1	X	0
1	0	1	1	X
1	1	0	X	1

	$\overline{T}$	T
$\overline{Q_n}$	0	1
Q_n	x	x

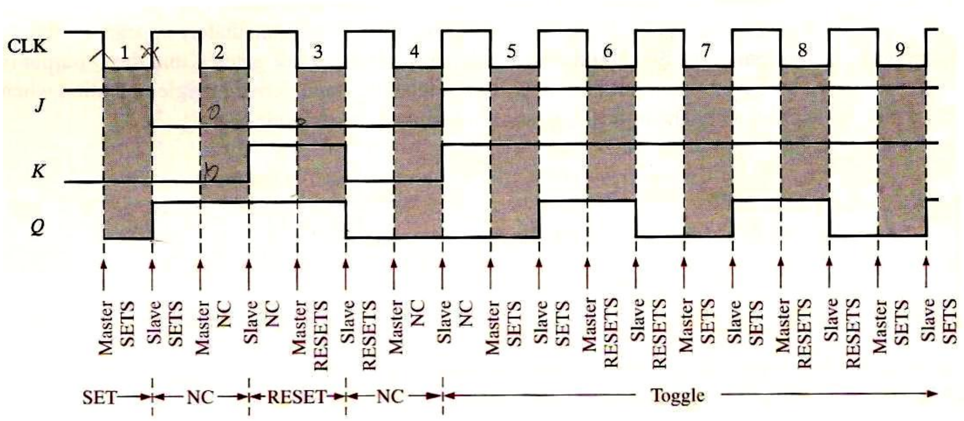
$\boxed{J = T}$

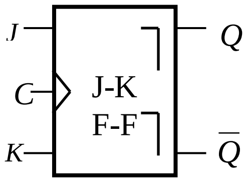
	$\overline{T}$	T
$\overline{Q_n}$	x	x
Q_n	0	1

$\boxed{K = T}$



Example: Find the output *Q* for the following input waveforms using the circuit below?

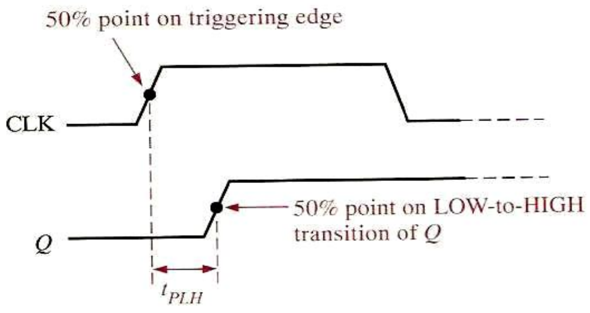




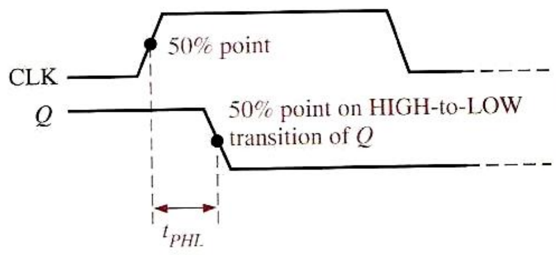
Propagation Delay Times:

A propagation delay is an interval of time required after an inputs signal has been applied for the resulting output change to occur. Several categories of propagation delay are important in the operation of F/F.

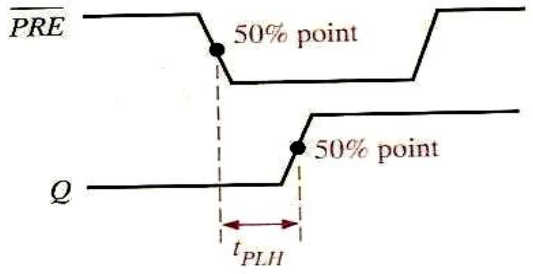
1. Propagation t_{PLH} as measured from the triggering edge of the clock pulse to the Low-to-High transition of output.


50% point on triggering edge
CLK
Q
50% point on LOW-to-HIGH transition of Q
t_{PLH}

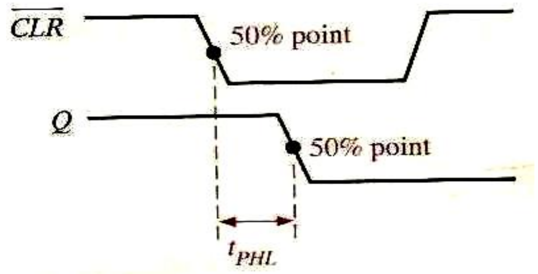
2. Propagation delay t_{PHL} as measured from the triggering edge of the clock pulse to the High-to-Low transition of output.



3. Propagation delay t_{PLH} as measured from the preset input to the Low-to-High transition of output.

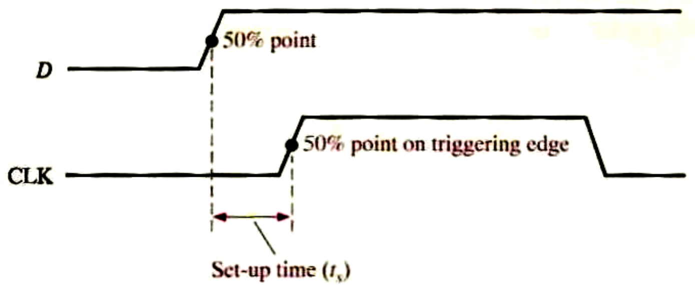


4. Propagation delay t_{PHL} as measured from the clear input to the High-to-Low transition of the output.



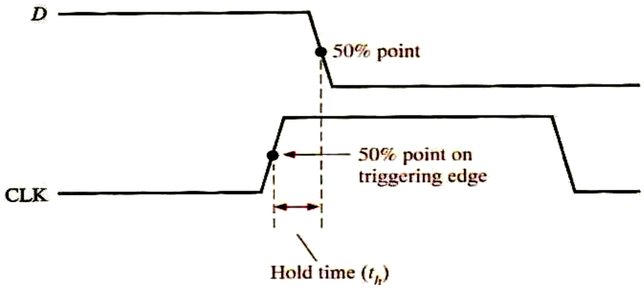
Set up Time:

The set up time (t_s) is the minimum interval required for the logic levels to be maintained constantly on the inputs (J & K) or (S & R) or D prior to the triggering edge of the clock pulse in order the levels to be reliably clocked into the F/F. This is illustrated in the figure below.



Hold Time:

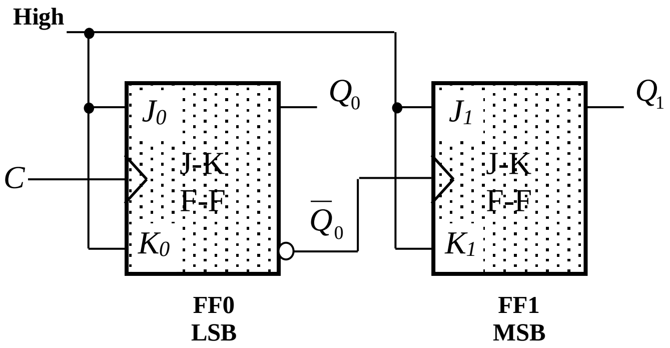
The hold time (t_h) is the minimum interval required for the logic levels to remain on the input after the triggering edge of the clock pulse in order the levels to be reliably clocked into the F/F. This is illustrated in the figure below for D flip-flop.



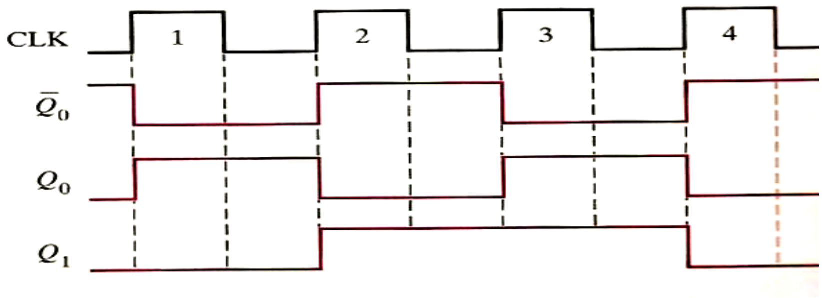
Asynchronous Counter:

The term asynchronous refers to events that do not have a fixed time relationship with each other and, generally, do not occur at the same time. An asynchronous counter is one in which the flip-flop within the counter do not change states at exactly the same time because they do not have a common clock pulse.

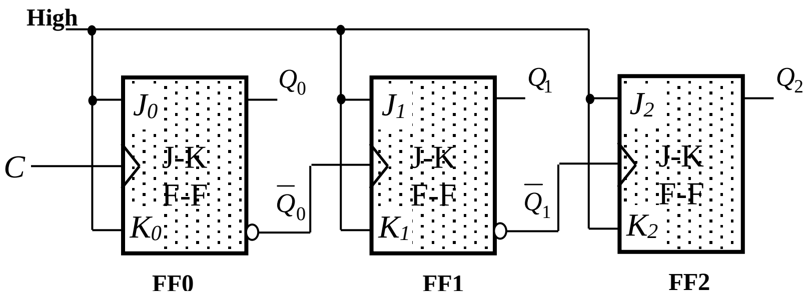
- ***A 2-bits Asynchronous Binary Counter:***



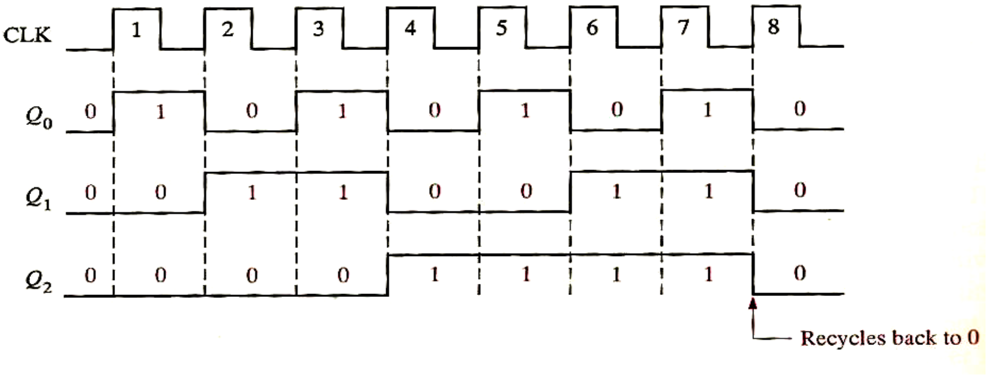
Clock Pulse	Q_1	Q_0
Initially	0	0
1	0	1
2	1	0
3	1	1
4	0	0



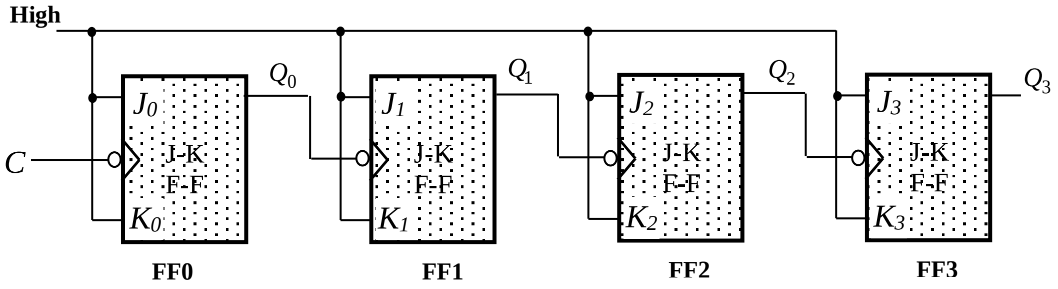
- ***A 3-bits Asynchronous Binary Counter:***

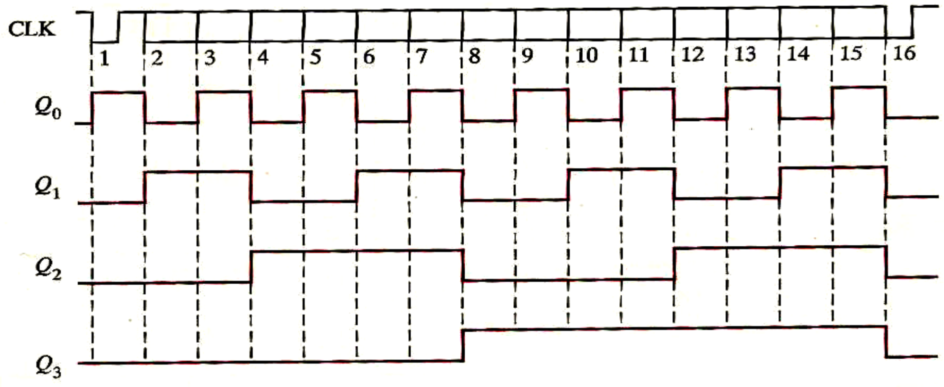


Clock Pulse	Q_2	Q_1	Q_0
Initially	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1
8	0	0	0

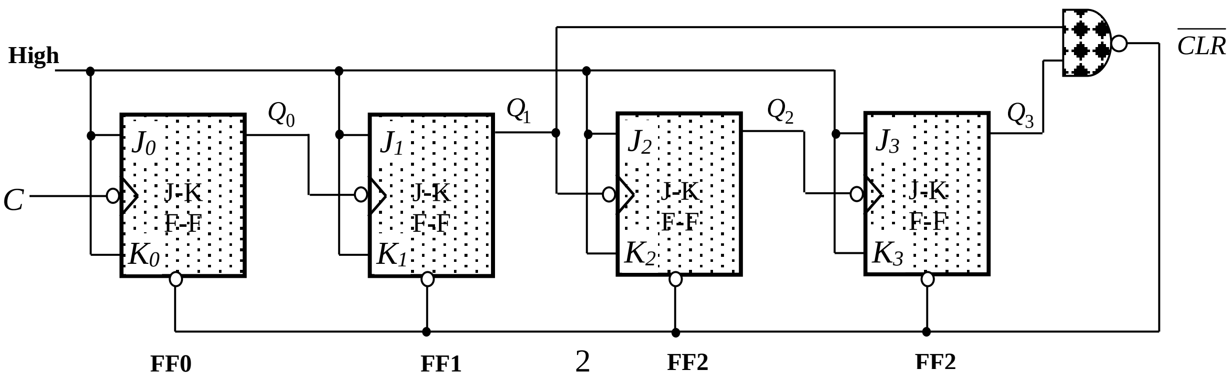


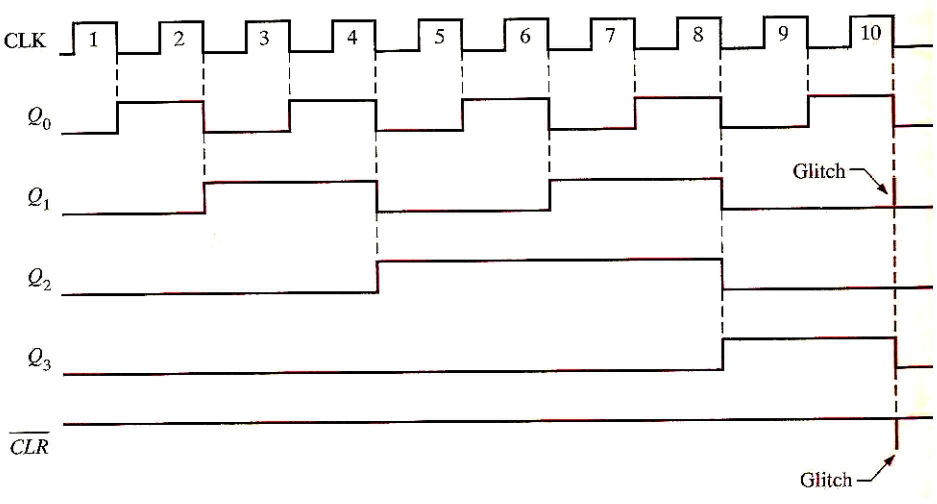
Example: A 4-bit asynchronous counter as shown in figure below. Each F/F is a negative edge triggered. Draw the timing diagram.



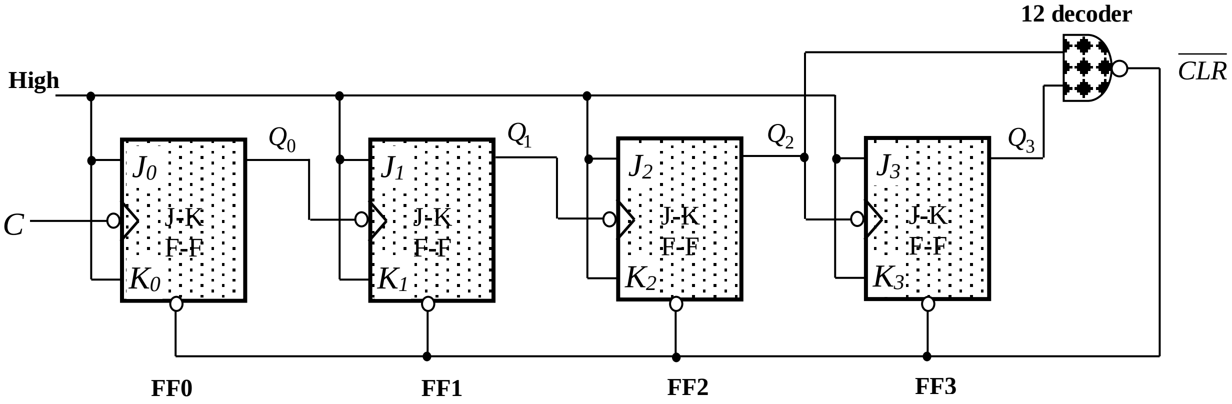


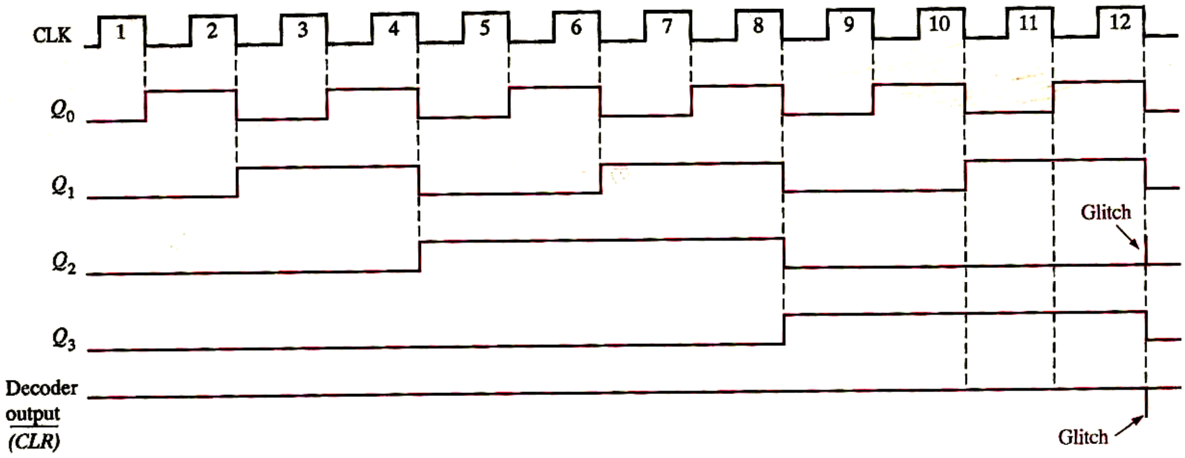
- ***Asynchronous Decade Counter:***





Example: Show how an asynchronous counter can be implemented having a modulus of twelve with a straight binary sequence from 0000 through 1011.




CLK
1
2
3
4
5
6
7
8
9
10
11
12
Q_0
Q_1
Q_2
Q_3
Glitch
Decoder output $(\overline{CLR})$
Glitch