Republic of Iraq

Ministry of Higher Education

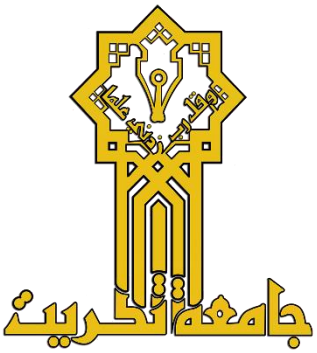
and Scientific Research

Tikrit University

College of Petroleum Processes Engineering

Department of Petroleum

Control Systems Engineering



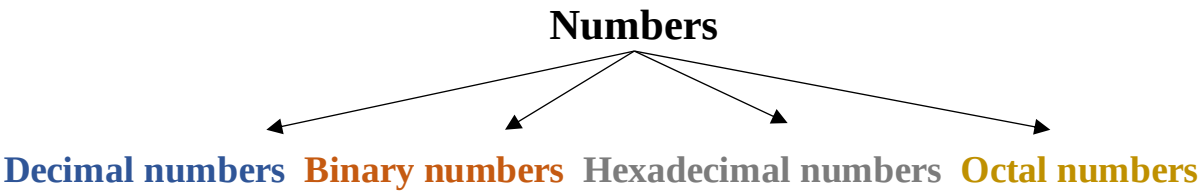
Digital Techniques

Prepared

By:

MSc. Mohammed Khalis

2024 A.D **1446 A.H**



Decimal numbers

Introduction

A decimal number is a way of representing numbers using a base-10 system. The decimal system is the standard system for denoting integer and non-integer numbers. It is also referred to as the base-10 numeral system because it is based on 10 different digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9.

Explanation

Each digit in a decimal number has a place value. The place value of each digit depends on its position relative to the decimal point. Positions to the left of the decimal point represent whole numbers, and each place value is 10 times the place value to its right. Positions to the right of the decimal point represent fractional parts of a whole, and each place value is 1/10th the place value to its left.

For example, in the number 123.45:

- The digit 1 is in the hundreds place and represents 100.
- The digit 2 is in the tens place and represents 20.
- The digit 3 is in the ones place and represents 3.
- The digit 4 is in the tenths place and represents 0.4.
- The digit 5 is in the hundredths place and represents 0.05.

So, 123.45 can be expressed as:

123.45 = 100 + 20 + 3 + 0.4 + 0.0

Examples

1. Whole Numbers:

456 : Here, 4 is in the hundreds place, 5 is in the tens place, and 6 is in the ones place.

$$456 = 400 + 50 + 6$$

2. Fractional Numbers:

78.9 : Here, 7 is in the tens place, 8 is in the ones place, and 9 is in the tenths place.

$$78.9 = 70 + 8 + 0.9$$

3. Mixed Numbers:

- 3.141: Here, 3 is in the ones place, 1 is in the tenths place, 4 is in the hundredths place, and 1 is in the thousandths place.

$$3.141 = 3 + 0.1 + 0.04 + 0.001$$

Binary Numbers

Introduction

Binary numbers are the foundation of digital technology and computing. The binary number system, also known as base-2, uses only two digits: 0 and 1. Each digit in a binary number is called a bit. The simplicity of the binary system makes it ideal for use in computers and digital systems, which rely on two states (on and off) to process data.

Explanation

Each position in a binary number represents a power of 2, starting from 2^0 on the right. The value of the binary number is the sum of these powers of 2 for the positions where there is a 1.

$$\ldots . 2^5\ 2^4\ 2^3\ 2^2\ 2^1\ 2^0 .\ 2^{-1}\ 2^{-2}\ 2^{-3}\ 2^{-4}\ 2^{-5} . \ldots$$

For example, in the binary number 1011:

- The rightmost bit (1) is in the 2^0 place and represents 1.
- The next bit to the left (1) is in the 2^1 place and represents 2.
- The next bit (0) is in the 2^2 place and represents 0 (because it's 0).
- The leftmost bit (1) is in the 2^3 place and represents 8.

So, 1011 in binary can be expressed as:

$1011_2 = 1 * 2^3 + 0 * 2^2 + 1 * 2^1 + 1 * 2^0$

$1011_2 = 8 + 0 + 2 + 1 = 11_{10}$

Examples

1. Binary to Decimal Conversion:

   - Binary 1101:

   $1101_2 = 1 * 2^3 + 1 * 2^2 + 0 * 2^1 + 1 * 2^0$

   $1101_2 = 8 + 4 + 0 + 1 = 13_{10}$

2. Decimal to Binary Conversion:

   - Decimal 9:

To convert 9 to binary, find the highest power of 2 less than or equal to 9.

The highest power is $2^3 = 8$.

9 - 8 = 1

The binary equivalent of 9 is: 1001_2

H.W

1. Convert the binary number 1010 to decimal.
2. Convert the decimal number 14 to binary.

Decimal to Binary Conversion

Introduction

Converting decimal numbers to binary is a fundamental skill in digital technology and computer science. The binary system uses only two digits, 0 and 1, and it is based on powers of 2. This system is used internally by almost all modern computers and digital systems because it aligns well with the on/off state of electronic circuits.

Explanation

To convert a decimal number to binary, you repeatedly divide the number by 2 and record the remainder. The binary number is formed by reading the remainders from bottom to top (from the last division to the first).

Steps for Conversion:

1. Divide the decimal number by 2.
2. Record the remainder (0 or 1).
3. Update the decimal number to the quotient obtained from the division.
4. Repeat steps 1-3 until the quotient is 0.
5. The binary number is the sequence of remainders read in reverse (from the last remainder to the first).

Example

Let's convert the decimal number 23 to binary.

1. Divide 23 by 2, quotient = 11, remainder = 1.
2. Divide 11 by 2, quotient = 5, remainder = 1.
3. Divide 5 by 2, quotient = 2, remainder = 1.
4. Divide 2 by 2, quotient = 1, remainder = 0.

5. Divide 1 by 2, quotient = 0, remainder = 1.

Now, read the remainders from bottom to top: 10111.

So, 23 in decimal is 10111 in binary: ($23_{10} = 10111_2$)

Example

Convert the decimal number 45 to binary.

1. Divide 45 by 2, quotient = 22, remainder = 1.

2. Divide 22 by 2, quotient = 11, remainder = 0.

3. Divide 11 by 2, quotient = 5, remainder = 1.

4. Divide 5 by 2, quotient = 2, remainder = 1.

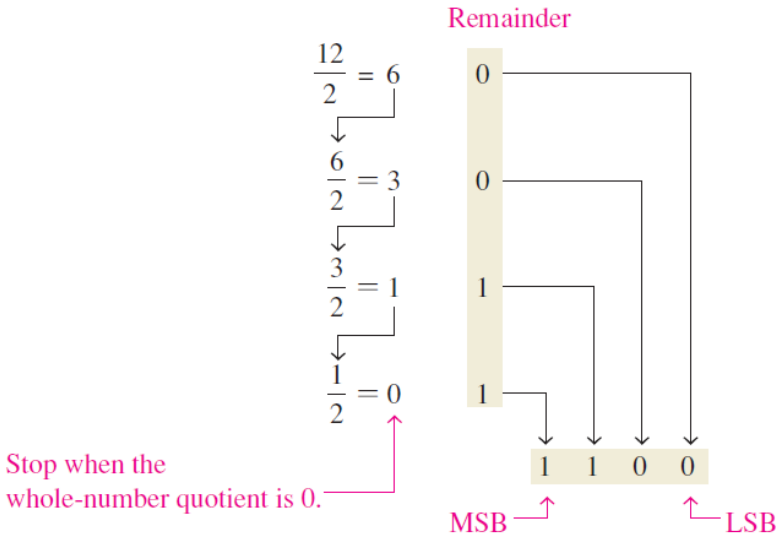
5. Divide 2 by 2, quotient = 1, remainder = 0.

6. Divide 1 by 2, quotient = 0, remainder = 1.

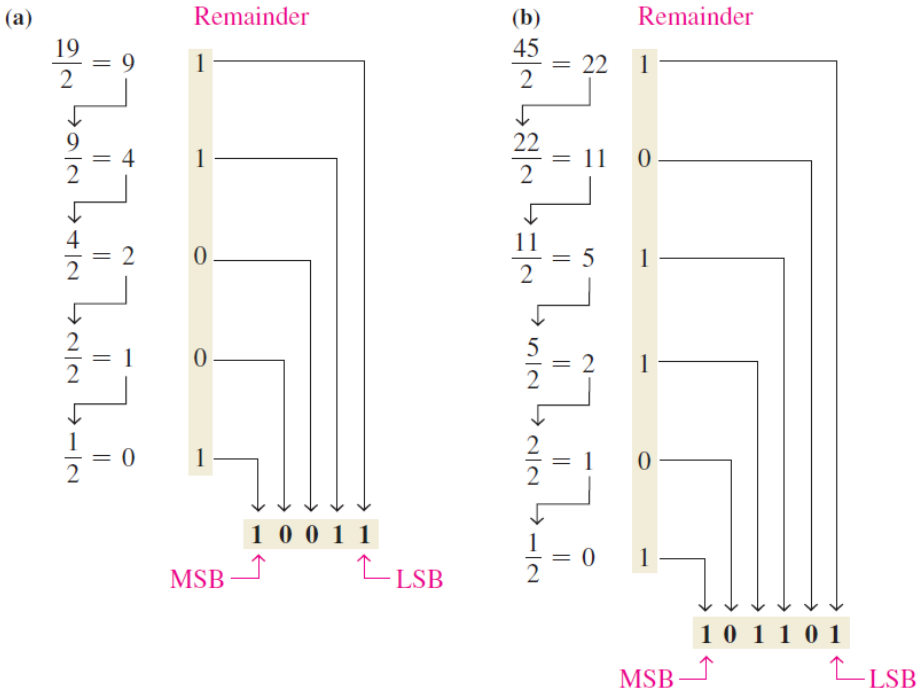
Reading the remainders from bottom to top: 101101.

So, 45 in decimal is 101101 in binary: ($45_{10} = 101101_2$)

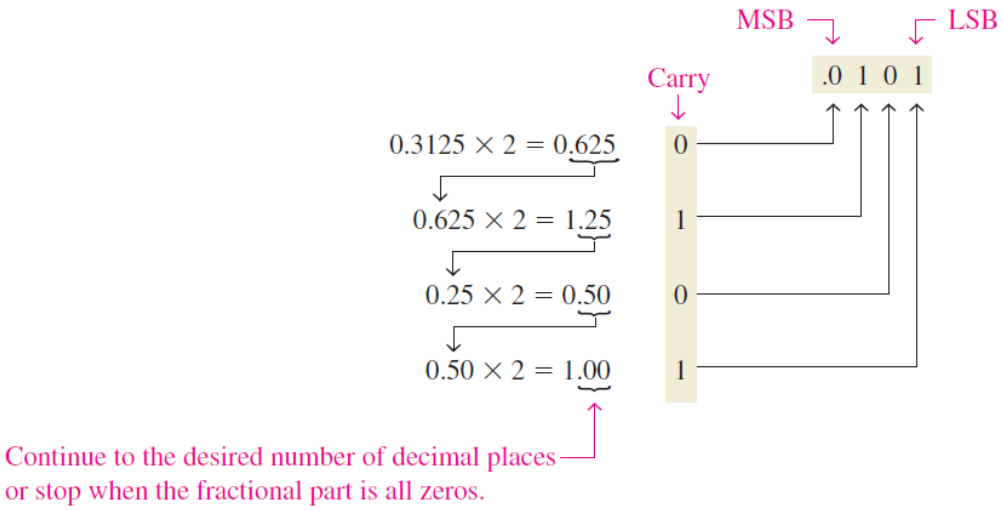
Example: convert 12_{10} to a binary number using the division method



Example: convert (a) 19_{10} and (b) 45_{10} to a binary number using the division method



Example: convert 0.3125_{10} to a binary number



Example: Find the result of (a) 110 / 11 and (b) 110 / 10

$$\text{(a)}\quad \begin{array}{r} \mathbf{10} \\ 11\overline{)110} \\ \underline{11} \\ 000 \end{array} \qquad \begin{array}{r} 2 \\ 3\overline{)6} \\ \underline{6} \\ 0 \end{array}$$

$$\text{(b)}\quad \begin{array}{r} \mathbf{11} \\ 10\overline{)110} \\ \underline{10} \\ 10 \\ \underline{10} \\ 00 \end{array} \qquad \begin{array}{r} 3 \\ 2\overline{)6} \\ \underline{6} \\ 0 \end{array}$$

Summary

Converting decimal numbers to binary involves dividing the number by 2 repeatedly and recording the remainders. The binary representation is obtained by reading the remainders in reverse order. This method helps in understanding how computers process and store data, making it a crucial skill in digital technology and computer science.

H.W

1. Convert the decimal number 10 to binary.
2. Convert the decimal number 37 to binary.
3. Convert the decimal number 100 to binary.

Binary to Decimal Conversion

Introduction

Converting binary numbers to decimal is essential for understanding how computers interpret binary data. The binary number system (base-2) uses only two digits: 0 and 1. Each position in a binary number represents a power of 2, starting from 2^0 on the right.

Explanation

To convert a binary number to a decimal number, you need to sum the products of each binary digit (bit) and its corresponding power of 2.

Steps for Conversion:

1. Write down the binary number.
2. Starting from the right, assign each digit a power of 2, beginning with 2^0.
3. Multiply each binary digit by its corresponding power of 2.

4. Sum all the products to get the decimal number.

Example

Let's convert the binary number 1101 to decimal.

1. Write down the binary number: 1101.

2. Assign powers of 2 to each digit, starting from the right:

   $1 * 2^3, 1 * 2^2, 0 * 2^1, 1 * 2^0$

3. Calculate each term:

   1 * 2^3 = 8

   1 * 2^2 = 4

   0 * 2^1 = 0

   1 * 2^0 = 1

4. Sum the results:

   8 + 4 + 0 + 1 = 13

So, 1101 in binary is 13 in decimal: ($1101_2 = 13_{10}$)

Example

Convert the binary number 10110 to decimal.

1. Write down the binary number: 10110.

2. Assign powers of 2 to each digit:

   $1 * 2^4, 0 * 2^3, 1 * 2^2, 1 * 2^1, 0 * 2^0$

3. Calculate each term:

   $1 * 2^4 = 16$

   $0 * 2^3 = 0$

   $1 * 2^2 = 4$

   $1 * 2^1 = 2$

   $0 * 2^0 = 0$

4. Sum the results:

$16 + 0 + 4 + 2 + 0 = 22$

So, 10110 in binary is 22 in decimal: ($10110_2 = 22_{10}$)

Summary

Converting binary numbers to decimal involves multiplying each binary digit by its corresponding power of 2 and then summing the results. This process helps in understanding how binary data is interpreted and manipulated in digital systems.

H.W

1. Convert the binary number 1001 to decimal.
2. Convert the binary number 11101 to decimal.
3. Convert the binary number 100110 to decimal.

Binary Arithmetic

Introduction

Binary arithmetic is essential in digital electronics and computer systems. It involves performing arithmetic operations like addition, subtraction, multiplication, and division using binary numbers (base-2). Since binary arithmetic is fundamental to how computers operate, understanding these operations is crucial for anyone studying digital technology or computer science.

Binary Addition

Binary addition is similar to decimal addition but follows simpler rules because there are only two digits involved: 0 and 1.

Rules for Binary Addition:

1. 0 + 0 = 0
2. 0 + 1 = 1
3. 1 + 0 = 1
4. 1 + 1 = 10 (which means 0 with a carry of 1)

Example:

Add 1011 and 1101.

```
  1011
+ 1101
------
 11000
```

Steps:

1. Start from the rightmost bit: 1 + 1 = 10 (write 0, carry 1).
2. Next bits: 1 + 0 + 1 (carry) = 10 (write 0, carry 1).
3. Next bits: 0 + 1 + 1 (carry) = 10 (write 0, carry 1).
4. Leftmost bits: 1 + 1 (carry) = 10 (write 0, carry 1).
5. Finally, write down the remaining carry: 1.

Result: 11000.

Binary Subtraction

Binary subtraction uses borrowing, similar to decimal subtraction.

Rules for Binary Subtraction:

1. 0 - 0 = 0
2. 1 - 0 = 1
3. 1 - 1 = 0
4. 0 - 1 = 1 (with a borrow from the next higher bit)

Example:

Subtract 1001 from 1101.

```
  1101
- 1001
------
  0100
```

Steps:

1. Start from the rightmost bit: 1 - 1 = 0.
2. Next bits: 0 - 0 = 0.
3. Next bits: 1 - 0 = 1.
4. Leftmost bits: 1 - 1 = 0.

Result: 0100.

Binary Multiplication

Binary multiplication is similar to decimal multiplication but simpler because it only involves multiplying by 0 or 1.

Rules for Binary Multiplication:

1. 0 * 0 = 0
2. 0 * 1 = 0
3. 1 * 0 = 0
4. 1 * 1 = 1

Example:

Multiply 101 by 11.

```
   101
 x  11
 ------
   101          (101 * 1)
+ 1010          (101 * 1 shifted one position to the left)
 ------
  1111
```

Steps:

1. Multiply 101 by 1: 101.
2. Multiply 101 by 1 (shifted one position to the left): 1010.
3. Add the results: 101 + 1010 = 1111.

Result: 1111.

Binary Division

Binary division is similar to decimal division but uses binary subtraction.

Example:

Divide 1101 by 11.

Steps:

1. Divide the leftmost bits of 1101 by 11: 11 goes into 11 once (1), remainder 0.

2. Bring down the next bit: 0.

3. Divide: 11 goes into 00 zero times (0), remainder 0.

4. Bring down the next bit: 1.

5. Divide: 11 goes into 01 zero times (0), remainder 1.

6. Bring down the next bit: 0.

7. Divide: 11 goes into 10 zero times (0), remainder 10.

8. Subtract 10 - 11: borrow from 0.

9. Repeat until no more bits remain.

Result: 1101 ÷ 11 = 100 with remainder 1.

Summary

Binary arithmetic involves addition, subtraction, multiplication, and division using binary numbers. Understanding these operations is essential for working with digital systems and computers.

1's and 2's Complements of Binary Numbers

Introduction

1's and 2's complements are methods used to represent negative numbers in binary systems. They are essential for performing binary arithmetic, particularly subtraction and representation of signed numbers.

1's Complement

The 1's complement of a binary number is found by inverting all the bits in the number, changing 0s to 1s and 1s to 0s.

Steps to Find 1's Complement:

- Invert each bit of the binary number.

Example:

Find the 1's complement of 101010.

Original: 101010

1's Complement: 010101

2's Complement

The 2's complement of a binary number is found by adding 1 to the 1's complement of the number. This method is widely used in computers to represent signed numbers because it simplifies the design of arithmetic circuits.

Steps to Find 2's Complement:

1. Find the 1's complement of the binary number.
2. Add 1 to the least significant bit (LSB) of the 1's complement.

Example:

Find the 2's complement of 101010.

1. First, find the 1's complement:

Original: 101010

1's Complement: 010101

2. Add 1 to the 1's complement:

```
  010101
+      1
--------
  010110
```

So, the 2's complement of 101010 is 010110.

Summary

1's complement is obtained by inverting all bits of a binary number, while 2's complement is found by adding 1 to the 1's complement. These methods are crucial for representing negative numbers and performing binary arithmetic operations.

H.W

1. Find the 1's complement of 110011.
2. Find the 2's complement of 110011.
3. Find the 1's complement of 1001.
4. Find the 2's complement of 1001.

Hexadecimal Numbers

Introduction

The hexadecimal number system, also known as base-16, uses sixteen distinct symbols: 0-9 to represent values zero to nine, and A-F to represent values ten to fifteen. Hexadecimal is widely used in computing and digital electronics because it provides a more human-friendly way to represent binary-coded values.

Explanation

Each digit in a hexadecimal number represents a power of 16, starting from 16^0 on the right. For example, in the hexadecimal number 2F3:

- 3 is in the 16^0 place and represents 3.

- F is in the 16^1 place and represents $15 \times 16 = 240$.

- 2 is in the 16^2 place and represents $2 \times 256 = 512$.

$$\ldots .16^3\ 16^2 16^1 16^0.\ 16^{-1}\ 16^{-2} 16^{-3} 16^{-4} \ldots .$$

So, 2F3 in hexadecimal can be expressed as:

$2F3_{16} = 2 * 16^2 + F * 16^1 + 3 * 16^0$

$2F3_{16} = 2 * 256 + 15 * 16 + 3$

$2F3_{16} = 512 + 240 + 3 = 755_{10}$

Binary to Hexadecimal Conversion

Introduction

Converting binary numbers to hexadecimal is a straightforward process because both are positional number systems, and one hexadecimal digit represents exactly four binary digits (bits). This makes it easy to group binary digits into sets of four and convert them directly to hexadecimal.

Explanation

To convert a binary number to a hexadecimal number:

1. Group the binary digits into sets of four, starting from the right. Add leading zeros if necessary to complete the last group.

2. Convert each group of four binary digits to its corresponding hexadecimal digit.

Example

Convert the binary number 101110101011 to hexadecimal.

1. Group the binary digits into sets of four, starting from the right:

   1011 , 1010 , 1011

2. Convert each group to its hexadecimal equivalent:

   (1011) (binary) = B (hexadecimal)

   (1010) (binary) = A (hexadecimal)

   (1011) (binary) = B (hexadecimal)

So, 101110101011 in binary is BAB in hexadecimal:

$(101110101011_2 = BAB_{16})$

Example

Convert the binary number 110010111011 to hexadecimal.

1. Group the binary digits into sets of four, starting from the right:

   1100 , 1011 , 1011

2. Convert each group to its hexadecimal equivalent:

   (1100) (binary) = C (hexadecimal)

(1011) (binary) = B (hexadecimal)

(1011) (binary) = B (hexadecimal)

So, 110010111011 in binary is CBB in hexadecimal:

$110010111011_2 = CBB_{16}$

Summary

Converting binary numbers to hexadecimal involves grouping the binary digits into sets of four and then converting each group to its corresponding hexadecimal digit. This method is efficient and aligns well with the binary representation used in digital systems.

H.W

1. Convert the binary number 110110101 to hexadecimal.
2. Convert the binary number 101010101010 to hexadecimal.
3. Convert the binary number 11110000 to hexadecimal.
4. Convert the binary number 100111000011 to hexadecimal.

Hexadecimal to Binary Conversion

Introduction

Converting hexadecimal numbers to binary is straightforward because each hexadecimal digit directly maps to a 4-bit binary sequence. This conversion is useful in computing, where hexadecimal notation is often used to simplify the representation of binary data.

Explanation

To convert a hexadecimal number to binary:

1. Replace each hexadecimal digit with its 4-bit binary equivalent.

2. Concatenate all the binary sequences to get the final binary number.

Hexadecimal to Binary Mapping

Here is a quick reference for converting each hexadecimal digit to binary:

- 0 = 0000
- 1 = 0001
- 2 = 0010
- 3 = 0011
- 4 = 0100
- 5 = 0101
- 6 = 0110
- 7 = 0111
- 8 = 1000
- 9 = 1001
- A = 1010
- B = 1011
- C = 1100
- D = 1101
- E = 1110
- F = 1111

Example

Convert the hexadecimal number 2F3 to binary.

1. Convert each hexadecimal digit to binary:

- 2 = 0010
- F = 1111
- 3 = 0011

2. Concatenate the binary sequences:

$2F3_{16}$= 0010 , 1111 , 0011_2

So, 2F3 in hexadecimal is 001011110011 in binary.

Example

Convert the hexadecimal number A9C to binary.

1. Convert each hexadecimal digit to binary:

- A = 1010
- 9 = 1001
- C = 1100

2. Concatenate the binary sequences:

$A9C_{16}$ = 1010 , 1001 , 1100_2

So, A9C in hexadecimal is 101010011100 in binary.

Summary

Converting hexadecimal numbers to binary involves mapping each hexadecimal digit to its 4-bit binary equivalent and concatenating these binary sequences. This process simplifies working with binary data and is essential for various applications in computing.

H.W

1. Convert the hexadecimal number 7F to binary.
2. Convert the hexadecimal number 1A3 to binary.
3. Convert the hexadecimal number 3D4 to binary.
4. Convert the hexadecimal number B2 to binary.

Hexadecimal to Decimal Conversion

Introduction

Hexadecimal (base-16) numbers are often used in computing to represent binary data in a more compact form. To convert hexadecimal numbers to decimal (base-10), you need to understand the positional value of each digit in the hexadecimal system.

Explanation

Each digit in a hexadecimal number represents a power of 16. The rightmost digit represents 16^0, the next represents 16^1, then 16^2, and so on.

Steps for Conversion:

1. Write down the hexadecimal number.
2. Assign each digit a power of 16, starting from 16^0 on the right.
3. Convert each hexadecimal digit to its decimal equivalent.
4. Multiply each decimal value by its corresponding power of 16.
5. Sum all the results to get the final decimal value.

Example

Convert the hexadecimal number 1A3 to decimal.

1. Write down the hexadecimal number: 1A3.

2. Assign powers of 16:

$1 * 16^2 + A * 16^1 + 3 * 16^0$

3. Convert each hexadecimal digit to decimal:

(1 = 1)

(A = 10)

(3 = 3)

4. Calculate each term:

$1 * 16^2 = 1 * 256 = 256$

$A * 16^1 = 10 * 16 = 160$

$3 * 16^0 = 3 * 1 = 3$

5. Sum the results:

$256 + 160 + 3 = 419$

So, 1A3 in hexadecimal is 419 in decimal: $1A3_{16} = 419_{10}$

Example

Convert the hexadecimal number 4F2 to decimal.

1. Write down the hexadecimal number: 4F2.

2. Assign powers of 16:

$4 * 16^2 + F * 16^1 + 2 * 16^0$

3. Convert each hexadecimal digit to decimal:

(4 = 4)

(F = 15)

(2 = 2)

4. Calculate each term:

$4 * 16^2 = 4 * 256 = 1024$

$F * 16^1 = 15 * 16 = 240$

$2 * 16^0 = 2 * 1 = 2$

5. Sum the results:

$1024 + 240 + 2 = 1266$

So, 4F2 in hexadecimal is 1266 in decimal: $4F2_{16} = 1266_{10}$

Summary

To convert hexadecimal numbers to decimal, each digit in the hexadecimal number is multiplied by its corresponding power of 16, and the results are summed. This process provides a clear understanding of the decimal value of a hexadecimal number.

H.W

1. Convert the hexadecimal number 3E7 to decimal.
2. Convert the hexadecimal number B4 to decimal.
3. Convert the hexadecimal number 7A9 to decimal.
4. Convert the hexadecimal number 2F5 to decimal.

Decimal to Hexadecimal Conversion

Introduction

Converting decimal (base-10) numbers to hexadecimal (base-16) is useful in computing and digital electronics for compactly representing large binary values. The process involves dividing the decimal number by 16 and recording the remainders, which are then converted to hexadecimal digits.

Steps for Conversion

1. **Divide the decimal number by 16** and record the quotient and remainder.

2. **Repeat the division** for the quotient until it is zero, recording each remainder.

3. **Convert each remainder** to its hexadecimal equivalent.

4. **Write the remainders in reverse order** (from the last division to the first) to get the hexadecimal number.

Example

Convert the decimal number 2023 to hexadecimal.

1. **Divide 2023 by 16**:
   - Quotient: 126
   - Remainder: 7
   - Hexadecimal equivalent of remainder: 7

2. **Divide 126 by 16**:
   - Quotient: 7
   - Remainder: 14
   - Hexadecimal equivalent of remainder: E (14 in hexadecimal)

3. **Divide 7 by 16**:
   - Quotient: 0
   - Remainder: 7
   - Hexadecimal equivalent of remainder: 7

4. **Combine the remainders** in reverse order:
   - Hexadecimal number: 7E7

So, 2023 in decimal is 7E7 in hexadecimal: $2023_{10} = 7E7_{16}$

Example

Convert the decimal number 255 to hexadecimal.

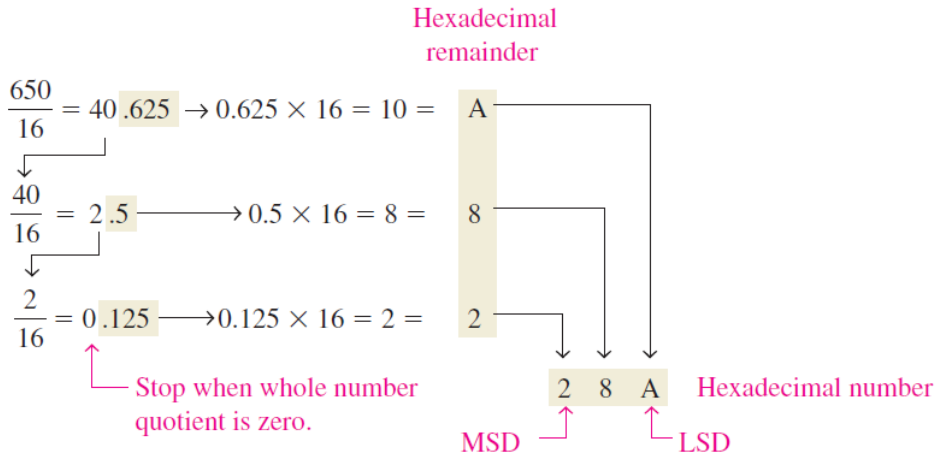
1. **Divide 255 by 16**:
   - Quotient: 15
   - Remainder: 15
   - Hexadecimal equivalent of remainder: F

2. **Divide 15 by 16**:
   - Quotient: 0
   - Remainder: 15
   - Hexadecimal equivalent of remainder: F

3. **Combine the remainders** in reverse order:
   - Hexadecimal number: FF

So, 255 in decimal is FF in hexadecimal: $255_{10} = FF_{16}$

Example: Convert the decimal number 650 to hexadecimal by repeated division by 16.



Summary

To convert decimal numbers to hexadecimal, repeatedly divide the number by 16, record the remainders, and convert them to hexadecimal digits. The hexadecimal number is obtained by arranging the remainders in reverse order.

H.W

1. Convert the decimal number 4095 to hexadecimal.
2. Convert the decimal number 1234 to hexadecimal.
3. Convert the decimal number 845 to hexadecimal.
4. Convert the decimal number 500 to hexadecimal.

H.W

1. Convert the hexadecimal number 4D2 to decimal.
2. Convert the decimal number 523 to hexadecimal.
3. Convert the binary number 101101111 to hexadecimal.
4. Convert the hexadecimal number 9C to binary.

Hexadecimal Arithmetic

Hexadecimal arithmetic involves performing basic arithmetic operations (addition, subtraction, multiplication, and division) using hexadecimal (base-16) numbers. It follows the same principles as decimal arithmetic but operates with base-16 digits (0-9 and A-F).

1. Hexadecimal Addition

Steps for Addition:

1. Align the hexadecimal numbers by their least significant digit.

2. Add each pair of digits, taking into account any carry from the previous digit.

3. Convert the sum to hexadecimal, if necessary, and handle carries.

Example: Add `2F` and `1A`.

1. Align the numbers:

  2F

+ 1A

2. Add from right to left:

F (15_{10}) + A (10_{10}) = 25_{10} , which is `19` in hexadecimal. Write down `9` and carry `1`.

2 (2_{10}) + 1 (1_{10}) + 1 (carry) = 4_{10}, which is `4` in hexadecimal.

So, 2F + 1A = 49.

Example:

Add the following hexadecimal numbers:

(a) $23_{16} + 16_{16}$ (b) $58_{16} + 22_{16}$ (c) $2B_{16} + 84_{16}$ (d) $DF_{16} + AC_{16}$

Solution

(a)
$$\begin{array}{r} 23_{16} \\ +\ 16_{16} \\ \hline \mathbf{39_{16}} \end{array}$$
right column: $3_{16} + 6_{16} = 3_{10} + 6_{10} = 9_{10} = 9_{16}$
left column: $2_{16} + 1_{16} = 2_{10} + 1_{10} = 3_{10} = 3_{16}$

(b)
$$\begin{array}{r} 58_{16} \\ +\ 22_{16} \\ \hline \mathbf{7A_{16}} \end{array}$$
right column: $8_{16} + 2_{16} = 8_{10} + 2_{10} = 10_{10} = A_{16}$
left column: $5_{16} + 2_{16} = 5_{10} + 2_{10} = 7_{10} = 7_{16}$

(c)
$$\begin{array}{r} 2B_{16} \\ +\ 84_{16} \\ \hline \mathbf{AF_{16}} \end{array}$$
right column: $B_{16} + 4_{16} = 11_{10} + 4_{10} = 15_{10} = F_{16}$
left column: $2_{16} + 8_{16} = 2_{10} + 8_{10} = 10_{10} = A_{16}$

(d)
$$\begin{array}{r} DF_{16} \\ +\ AC_{16} \\ \hline \mathbf{18B_{16}} \end{array}$$
right column: $F_{16} + C_{16} = 15_{10} + 12_{10} = 27_{10}$
$27_{10} - 16_{10} = 11_{10} = B_{16}$ with a 1 carry
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 1 carry

2. Hexadecimal Subtraction

Steps for Subtraction:

1. Align the hexadecimal numbers by their least significant digit.

2. Subtract each pair of digits, borrowing from the next higher digit if needed.

3. Convert the result to hexadecimal and handle any necessary borrowing.

Example: Subtract `1A` from `2F`.

1. Align the numbers:

2F

- 1A

2. Subtract from right to left:

F (15_{10}) - A (10_{10}) = 5_{10}, which is `5` in hexadecimal.

2 (2_{10}) - 1 (1_{10}) = 1_{10}, which is `1` in hexadecimal.

So, `2F - 1A = 15`.

Example:

Subtract the following hexadecimal numbers:

(a) $84_{16} - 2A_{16}$ **(b)** $C3_{16} - 0B_{16}$

Solution

(a) $2A_{16} = 00101010$

2's complement of $2A_{16} = 11010110 = D6_{16}$ (using Method 1)

$$\begin{array}{rl} 84_{16} & \\ +\ D6_{16} & \text{Add} \\ \hline \not{1}5A_{16} & \text{Drop carry, as in 2's complement addition} \end{array}$$

The difference is $\mathbf{5A_{16}}$.

(b) $0B_{16} = 00001011$

2's complement of $0B_{16} = 11110101 = F5_{16}$ (using Method 1)

$$\begin{array}{rl} C3_{16} & \\ +\ F5_{16} & \text{Add} \\ \hline \not{1}B8_{16} & \text{Drop carry} \end{array}$$

The difference is $\mathbf{B8_{16}}$.

3. Hexadecimal Multiplication

Steps for Multiplication:

1. Multiply each digit of the first number by each digit of the second number, just like decimal multiplication.

2. Add the partial products, taking care to align them according to their place values.

3. Convert the result to hexadecimal.

Example: Multiply `A` by `3`.

1. Convert hexadecimal to decimal:

   - `A` = 10 in decimal.

   - `3` = 3 in decimal.

2. Multiply in decimal:

   10 * 3 = 30.

3. Convert the result to hexadecimal:

   $30_{10} = 1E_{16}$.

   So, `A × 3 = 1E`.

4. Hexadecimal Division

Steps for Division:

1. Divide the hexadecimal number by another hexadecimal number using long division.

2. Convert the result and remainder to hexadecimal.

Example: Divide `4E` by `3`

1. Convert hexadecimal to decimal:

   - `4E` = 78 in decimal.

   - `3` = 3 in decimal.

2. Divide in decimal:

78 / 3 = 26, remainder `0`.

3. Convert the result to hexadecimal:

`26` in decimal is `1A` in hexadecimal.

So, `4E ÷ 3 = 1A`.

Summary

Hexadecimal arithmetic involves operations similar to decimal arithmetic but with base-16 digits. Understanding the conversion between hexadecimal and decimal is crucial for performing these operations. Practice with various examples will help solidify your understanding of hexadecimal arithmetic.

H.W

1. Add `7B` and `C4`.
2. Subtract `5D` from `9A`.
3. Multiply `B` by `4`.
4. Divide `A6` by `6`.

Subtracting Hexadecimal Numbers Using 2's Complement

To subtract hexadecimal numbers using 2's complement, follow these steps:

1. **Convert the numbers to binary**.
2. **Find the 2's complement** of the subtrahend (the number being subtracted).

3. **Add the minuend (the number from which another number is subtracted)** and the 2's complement of the subtrahend.

4. **Handle any overflow** (carry beyond the most significant bit) if necessary.

5. **Convert the result back to hexadecimal**.

Example: Subtract `5A` from `B4`

1. **Convert to Binary**:

   - `B4` in hexadecimal = `1011 0100` in binary

   - `5A` in hexadecimal = `0101 1010` in binary

2. **Find the 2's Complement** of `5A`:

   - **Invert the bits** of `0101 1010`: `1010 0101`

   - **Add 1**: `1010 0101` + `0000 0001` = `1010 0110`

3. **Add `B4` and the 2's complement of `5A`**:

   - `1011 0100` (B4)

   + `1010 0110` (2's complement of 5A)

   `0101 0010` (Result, binary)

4. **Convert the Result to Hexadecimal**:

   - `0101 0010` = `52` in hexadecimal

Result: `B4 - 5A = 52`

Example: Subtract `3C` from `8F`

1. **Convert to Binary**:

   - `8F` in hexadecimal = `1000 1111` in binary

   - `3C` in hexadecimal = `0011 1100` in binary

2. **Find the 2's Complement** of `3C`:
   - **Invert the bits** of `0011 1100`: `1100 0011`
   - **Add 1**: `1100 0011` + `0000 0001` = `1100 0100`
3. **Add `8F` and the 2's complement of `3C`**:
   - `1000 1111` (8F)
  + `1100 0100` (2's complement of 3C)

    `0101 0011` (Result, binary)
4. **Convert the Result to Hexadecimal**:
   - `0101 0011` = `53` in hexadecimal

Result: `8F - 3C = 53`

Example: Subtract `1D` from `7A`

1. **Convert to Binary**:
   - `7A` in hexadecimal = `0111 1010` in binary
   - `1D` in hexadecimal = `0001 1101` in binary
2. **Find the 2's Complement** of `1D`:
   - **Invert the bits** of `0001 1101`: `1110 0010`
   - **Add 1**: `1110 0010` + `0000 0001` = `1110 0011`
3. **Add `7A` and the 2's complement of `1D`**:
   - `0111 1010` (7A)
  + `1110 0011` (2's complement of 1D)

    `0110 0001` (Result, binary)

4. **Convert the Result to Hexadecimal**:

   - `0110 0001` = `61` in hexadecimal

Result: `7A - 1D = 61`

Summary

To subtract hexadecimal numbers using 2's complement:

1. Convert the numbers to binary.
2. Find the 2's complement of the subtrahend.
3. Add the minuend and the 2's complement of the subtrahend.
4. Convert the result back to hexadecimal.

Octal Numbers

Introduction

Octal (base-8) numbers are another positional number system, similar to decimal (base-10) and hexadecimal (base-16). In the octal system, each digit represents a power of 8. Octal numbers are less commonly used today but are still relevant in computing, particularly in programming and digital systems.

Digits in Octal System

Octal numbers use digits from 0 to 7. Each digit in an octal number represents a power of 8:

$(8^0) = 1$

$(8^1) = 8$

$(8^2) = 64$

$(8^3) = 512$

$$\ldots . 8^2\, 8^1\, 8^0 .\, 8^{-1}\, 8^{-2}\, 8^{-3} \cdots$$

Example

Let's take the octal number `375`.

1. **Write down the octal number**: 375.

2. **Assign powers of 8**:

   $3 * 8^2$

   $7 * 8^1$

   $5 * 8^0$

3. **Convert each digit**:

   3 * 64 = 192

   7 * 8 = 56

   5 * 1 = 5

4. **Add the results**:

   192 + 56 + 5 = 253

So, the decimal equivalent of octal `375` is `253`.

Converting Octal to Decimal

To convert an octal number to decimal:

1. Write down the octal number.

2. Assign powers of 8 to each digit, starting from the right.

3. Multiply each digit by its corresponding power of 8.

4. Sum the results to get the decimal number.

Example

Convert octal `427` to decimal.

1. **Write down the octal number**: 427.

2. **Assign powers of 8**:

   $4 * 8^2$

   $2 * 8^1$

   $7 * 8^0$

3. **Convert each digit**:

   4 * 64 = 256

   2 * 8 = 16

   7 * 1 = 7

4. **Add the results**:

   256 + 16 + 7 = 279

So, the decimal equivalent of octal `427` is `279`.

Converting Decimal to Octal

To convert a decimal number to octal:

1. Divide the decimal number by 8.

2. Record the remainder.

3. Divide the quotient by 8, recording the new remainder.

4. Repeat until the quotient is zero.

5. The octal number is the remainders read from bottom to top.

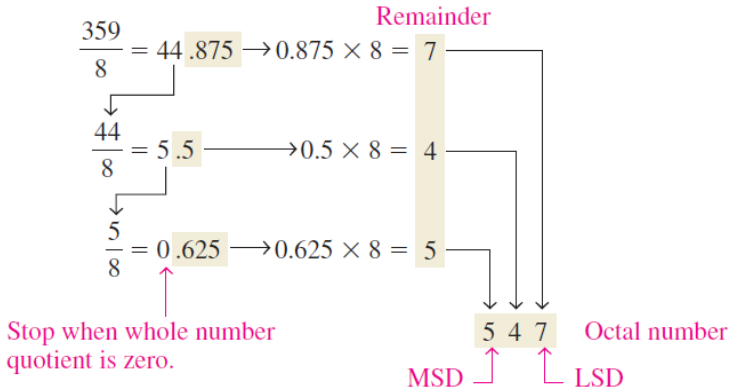
Example

Convert decimal `125` to octal.

1. **Divide 125 by 8**:
   - Quotient: 15
   - Remainder: 5
2. **Divide 15 by 8**:
   - Quotient: 1
   - Remainder: 7
3. **Divide 1 by 8**:
   - Quotient: 0
   - Remainder: 1
4. **Combine the remainders** in reverse order:
   - Octal number: 175

So, the octal equivalent of decimal `125` is `175`.

Example: Convert the decimal number 359 to the octal number



Summary

Understanding octal numbers involves converting between octal and decimal systems. The key is to be comfortable with the base-8 representation and to practice converting between octal and decimal to reinforce the concepts.

H.W

1. Convert the octal number `536` to decimal.

2. Convert the decimal number `345` to octal.

3. Convert the octal number `752` to decimal.

4. Convert the decimal number `64` to octal.

Octal to Binary Conversion

Converting octal (base-8) numbers to binary (base-2) is straightforward because each octal digit corresponds to exactly three binary digits. Here's how you can do the conversion:

1. **Write down the octal number.**

2. **Convert each octal digit to its 3-bit binary equivalent.**

3. **Combine the binary groups.**

Conversion Table

Here’s a quick reference table for converting octal digits to binary:

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

Example: Convert Octal `157` to Binary

1. **Write down the octal number**: `157`.
2. **Convert each digit**:
   - `1` in octal = `001` in binary
   - `5` in octal = `101` in binary
   - `7` in octal = `111` in binary
3. **Combine the binary groups**:
   - `1` = `001`
   - `5` = `101`
   - `7` = `111`

   Binary Equivalent: `001 101 111`

   Result: `157` in octal = `001101111` in binary.

Example: Convert Octal `374` to Binary

1. **Write down the octal number**: `374`.
2. **Convert each digit**:
   - `3` in octal = `011` in binary
   - `7` in octal = `111` in binary
   - `4` in octal = `100` in binary
3. **Combine the binary groups**:
   - `3` = `011`
   - `7` = `111`
   - `4` = `100`

  Binary Equivalent: `011 111 100`

   Result: `374` in octal = `011111100` in binary.

Example:

Convert each of the following octal numbers to binary:

(a) 13_8 (b) 25_8 (c) 140_8 (d) 7526_8

Solution

(a) 1 3 → 001011 (b) 2 5 → 010101 (c) 1 4 0 → 001100000 (d) 7 5 2 6 → 111101010110

Summary

To convert an octal number to binary:

1. Convert each octal digit to its 3-bit binary equivalent using the conversion table.

2. Combine all the binary groups together.

Binary to Octal Conversion

To convert binary (base-2) numbers to octal (base-8), follow these steps:

1. **Group the binary digits into sets of three**, starting from the right. If necessary, pad the leftmost group with zeros to make a complete set of three.

2. **Convert each group of three binary digits** to its octal equivalent using a conversion table.

3. **Combine the octal digits** to form the final octal number.

Example: Convert Binary `110101` to Octal

1. **Group the binary digits into sets of three**:

   - Binary: `110101`

   - Grouping: `110 101`

2. **Convert each group**:
   - `110` in binary = `6` in octal
   - `101` in binary = `5` in octal
3. **Combine the octal digits**:
 Octal Equivalent: `65`

Example: Convert Binary `100110010` to Octal

1. **Group the binary digits into sets of three**:
   - Binary: `100110010`
   - Grouping (from right): `100 110 010`
2. **Convert each group**:
   - `100` in binary = `4` in octal
   - `110` in binary = `6` in octal
   - `010` in binary = `2` in octal
3. **Combine the octal digits**:
   Octal Equivalent: `462`

Example: Convert Binary `101010101` to Octal

1. **Group the binary digits into sets of three**:
   - Binary: `101010101`
   - Grouping (from right): `101 010 101`
2. **Convert each group**:

   - `101` in binary = `5` in octal
   - `010` in binary = `2` in octal
   - `101` in binary = `5` in octal

3. **Combine the octal digits**:

Octal Equivalent: `525`

Example:

Convert each of the following binary numbers to octal:

(a) 110101 (b) 101111001 (c) 100110011010 (d) 11010000100

Solution

(a) $\underbrace{110}_{6}\underbrace{101}_{5} = 65_8$

(b) $\underbrace{101}_{5}\underbrace{111}_{7}\underbrace{001}_{1} = 571_8$

(c) $\underbrace{1001}_{4}\underbrace{1001}_{6}\underbrace{1010}_{3}\underbrace{}_{2} = 4632_8$

(d) $\underbrace{0110}_{3}\underbrace{1000}_{2}\underbrace{0100}_{0}\underbrace{}_{4} = 3204_8$

Summary

To convert a binary number to octal:

1. Group the binary digits into sets of three from the right. Add leading zeros if needed.

2. Convert each set of three binary digits to its octal equivalent using the conversion table.

3. Combine the results to get the final octal number.

Binary-Coded Decimal (BCD)

Introduction

Binary-Coded Decimal (BCD) is a binary encoding scheme for decimal numbers in which each decimal digit is represented by a fixed number of binary digits (usually four). BCD is used in digital systems where it is essential to represent decimal digits precisely, such as in digital watches, calculators, and certain computer systems.

How BCD Works

In BCD, each decimal digit is individually encoded into its 4-bit binary equivalent. For example:

- Decimal `0` is represented as `0000` in BCD.

- Decimal `1` is represented as `0001` in BCD.

- Decimal `9` is represented as `1001` in BCD.

Example:

Convert the decimal number `259` to BCD.

1. **Break down the number into its individual decimal digits**: `2`, `5`, `9`.

2. **Convert each digit to its 4-bit binary equivalent**:

   - Decimal `2` = `0010`

   - Decimal `5` = `0101`

   - Decimal `9` = `1001`

3. **Combine the results**:

   - BCD representation of `259` = `0010 0101 1001`

So, `259` in decimal is `0010 0101 1001` in BCD.

BCD Addition

When adding BCD numbers, you follow the same rules as binary addition but need to correct the result if it exceeds the BCD range (`0000` to `1001` for `0` to `9`).

Example: Add `25` and `37` in BCD.

1. **Convert decimal numbers to BCD**:

   - `25` in BCD: `0010 0101`

   - `37` in BCD: `0011 0111`

2. **Add the BCD numbers**:

   - Align the numbers and add:

      0010 0101

    + 0011 0111

     01101 1100

3. **Adjust the result** if needed:

   - Binary addition gives `0110 11100`.

   - Since `0110` (6 in decimal) is valid but `11100` (28 in decimal) exceeds `1001`, adjust with a correction:

     - Add `0001 0000` (6 in BCD) to correct it, so we get:

      0110 11100

     + 0001 0000

      0111 0000

Result: `0111 0000` (BCD representation of 62).

4. **Convert to decimal to verify**:

   - `0111` = `7`

   - `0000` = `0`

   - Decimal result: `62`.

BCD Subtraction

BCD subtraction involves similar steps to binary subtraction but also corrects the result if necessary.

Example: Subtract `17` from `24` in BCD.

1. **Convert decimal numbers to BCD**:

   - `24` in BCD: `0010 0100`

   - `17` in BCD: `0001 0111`

2. **Subtract the BCD numbers**:

   - Align the numbers and subtract:

     0010 0100

   - 0001 0111

     0000 1111

3. **Convert to decimal to verify**:

   - `0000` = `0`

   - `1111` = `9`

   Result: `7`.

Extended BCD and Packed BCD

- **Packed BCD**: Each byte (8 bits) contains two BCD digits (4 bits each). For example, `259` would be represented as `0010 0101 1001` in three separate bytes.

Example:

Convert each of the following decimal numbers to BCD:

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

Solution

(a) 3 5 → 00110101

(b) 9 8 → 10011000

(c) 1 7 0 → 000101110000

(d) 2 4 6 9 → 0010010001101001

- **Unpacked BCD**: Each byte (8 bits) contains only one BCD digit. For example, `259` would be represented as `0010 0000 0101 0000 1001 0000`.

Example:

Convert each of the following BCD codes to decimal:

(a) 10000110 (b) 001101010001 (c) 1001010001110000

Solution

(a) 10000110 → 8 6

(b) 001101010001 → 3 5 1

(c) 1001010001110000 → 9 4 7 0

Summary

BCD is a method of encoding decimal numbers where each digit is represented by its binary equivalent. It is useful in systems that require precise decimal representation and can be handled through specific arithmetic operations.

H.W

1. Convert the decimal number `482` to BCD.
2. Add the BCD numbers `413` and `289`.
3. Subtract the BCD number `174` from `256`.
4. Convert the BCD number `0100 1001` to decimal.

BCD (Binary-Coded Decimal) Addition

Binary-Coded Decimal (BCD) is a binary-encoded representation of integer values that uses a 4-bit nibble to represent each digit of a decimal number. When performing BCD addition, if the result of adding two BCD digits exceeds 9 (i.e., `1001` in binary), you need to adjust the result to ensure it remains a valid BCD representation.

Steps for BCD Addition

1. **Add the BCD digits** as you would normally add binary numbers.

2. **Check if the result exceeds 9** (i.e., check if it is greater than `1001` in binary).

3. **Correct the result** by adding `6` (binary `0110`) to the result if it exceeds `9`. This correction adjusts the result to fit within the BCD range.

Example: Adding Two BCD Numbers

Let's add `7` and `5` in BCD:

1. **Convert to BCD**:

   - `7` in BCD = `0111`

   - `5` in BCD = `0101`

2. **Add the BCD digits**:

   0111 (7)

+ 0101 (5)

  1100

3. **Check the result**:

   - The result `1100` in binary is `12` in decimal, which is greater than `9`.

4. **Adjust the result**:

- Add `6` (binary `0110`) to the result to correct it.

1100 (Original result)

+ 0110 (Adjustment for BCD)

0010 (Corrected result)

Carry = 1

5. **Include the carry**:

- Since there is a carry, add `0001` to the next higher digit (if there were one).

Final Result: `0001 0010`.

BCD Result: `12`.

Example: Adding Two BCD Numbers with Correction

Add `9` and `8` in BCD:

1. **Convert to BCD**:

- `9` in BCD = `1001`

- `8` in BCD = `1000`

2. **Add the BCD digits**:

1001 (9)

+ 1000 (8)

2001

3. **Check the result**:

- The result `2001` in binary is `17` in decimal, which is greater than `9`.

4. **Adjust the result**:

- Since the result `0001` exceeds `9`, add `6` (binary `0110`) to adjust.

0001 (Lower nibble of the result)

+ 0110 (Adjustment for BCD)

0111 (Corrected result for this nibble)

Carry = 1

5. **Add the carry** to the next higher digit (which was `0` in this case).

- Add `1` to the higher nibble (which results in `0001` with no further adjustment needed).

Final Result: `0001 0111`, which corresponds to `17` in decimal.

Example:

Add the following BCD numbers:

(a) 0011 + 0100 (b) 00100011 + 00010101

(c) 10000110 + 00010011 (d) 010001010000 + 010000010111

Solution

The decimal number additions are shown for comparison.

(a)	0011	3		(b)	0010	0011	23		
	+ 0100	+ 4			+ 0001	0101	+ 15		
	0111	7			**0011**	**1000**	38		
(c)	1000	0110	86	(d)	0100	0101	0000	450	
	+ 0001	0011	+ 13		+ 0100	0001	0111	+ 417	
	1001	**1001**	**99**		**1000**	**0110**	**0111**	**867**	

Note that in each case the sum in any 4-bit column does not exceed 9, and the results are valid BCD numbers.

Example:

Add the following BCD numbers:

(a) 1001 + 0100 **(b)** 1001 + 1001

(c) 00010110 + 00010101 **(d)** 01100111 + 01010011

(a)

1001			9
+ 0100			+4
1101		Invalid BCD number (>9)	13
+ 0110		Add 6	
0001	**0011**	Valid BCD number	
↓	↓		
1	3		

(b)

1001			9
+ 1001			+ 9
1 0010		Invalid because of carry	18
+ 0110		Add 6	
0001	**1000**	Valid BCD number	
↓	↓		
1	8		

(c)

0001	0110		16
+ 0001	0101		+ 15
0010	1011	Right group is invalid (>9), left group is valid.	31
	+ 0110	Add 6 to invalid code. Add carry, 0001, to next group.	
0011	**0001**	Valid BCD number	
↓	↓		
3	1		

(d)

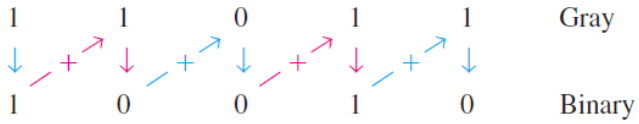
0110	0111			67
+ 0101	0011			+ 53
1011	1010	Both groups are invalid (>9)		120
+ 0110	+ 0110	Add 6 to both groups		
0001	**0010**	**0000**	Valid BCD number	
↓	↓	↓		
1	2	0		

Summary

1. **Perform BCD addition** as with binary numbers.
2. **Check the result** for validity in BCD (i.e., if it exceeds `1001`).
3. **Adjust the result** by adding `6` (binary `0110`) if necessary.
4. **Include any carry** in the final result.

Convert Gray Code to Binary

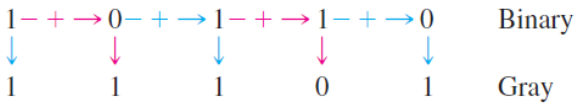
For example, the conversion of the Gray code word 11011 to binary is as follows:



The binary number is 10010.

Convert Binary to Gray Code

For example, the conversion of the binary number 10110 to Gray code is as follows:



The Gray code is 11101.

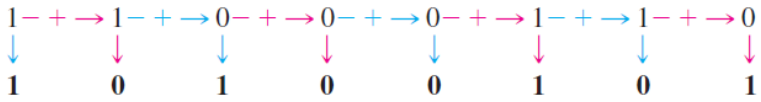
Example:

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

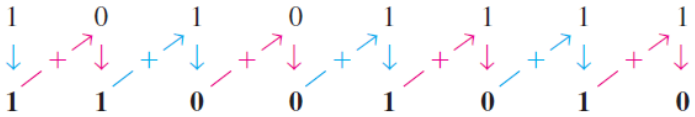
(b) Convert the Gray code 10101111 to binary.

Solution

(a) Binary to Gray code:



(b) Gray code to binary:

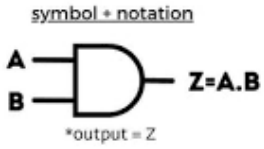


Logic Gates

Logic gates are fundamental building blocks in digital circuits. They perform basic logical functions that are essential for digital circuit design. Each gate has a specific function and truth table that describes its behavior.

Basic Logic Gates:

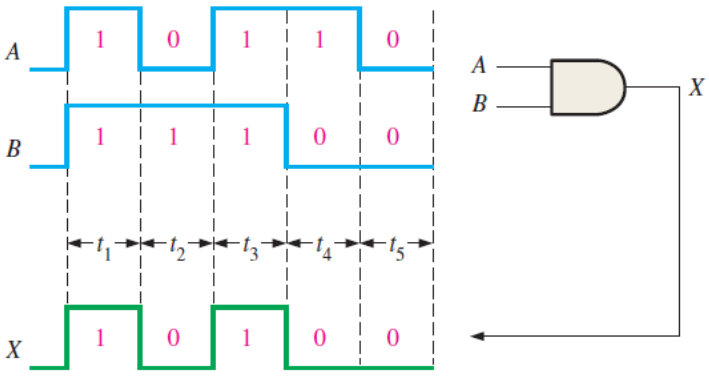
1. **AND Gate**
- Symbol:



truth table

A	B	Z=A.B
0	0	0
0	1	0
1	0	0
1	1	1

- AND Gate Operation with Waveform Inputs:



A diagram of input and output waveforms showing time relationships is called a *timing diagram*.

Example:

If two waveforms, *A* and *B*, are applied to the AND gate inputs as in Figure 3–11, what is the resulting output waveform?

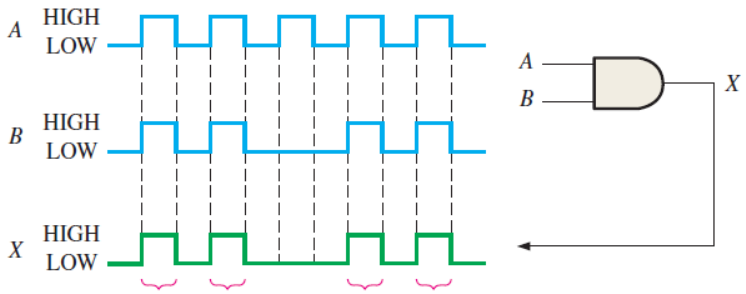


FIGURE 3–11

Example:

For the 3-input AND gate in Figure 3–13, determine the output waveform in relation to the inputs.

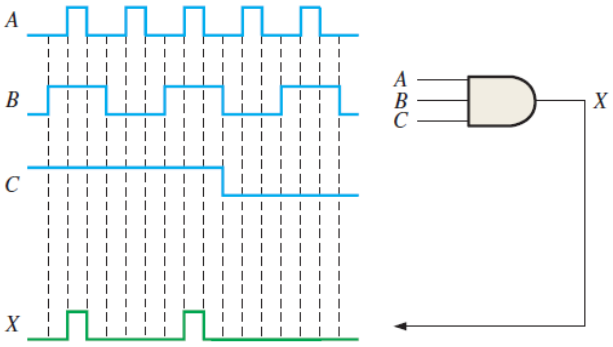


FIGURE 3–13

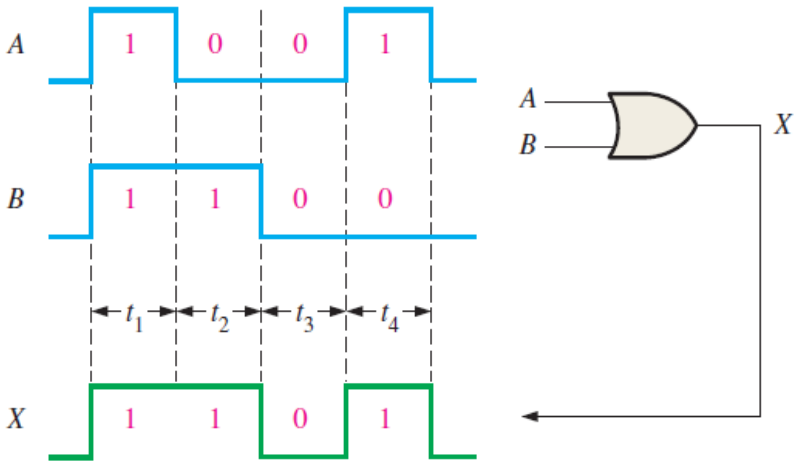
2. **OR Gate**

- Symbol:

2 input OR gate

A	B	A+B
0	0	0
0	1	1
1	0	1
1	1	1

- OR Gate Operation with Waveform Inputs



Example:

If the two input waveforms, *A* and *B*, in Figure 3–21 are applied to the OR gate, what is the resulting output waveform?

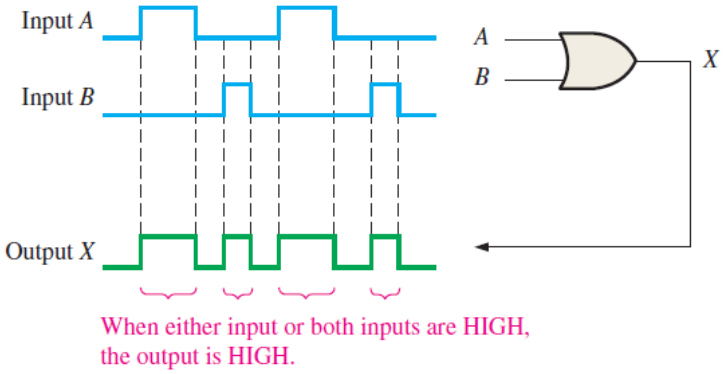


FIGURE 3–21

Example:

For the 3-input OR gate in Figure 3–23, determine the output waveform in proper time relation to the inputs.

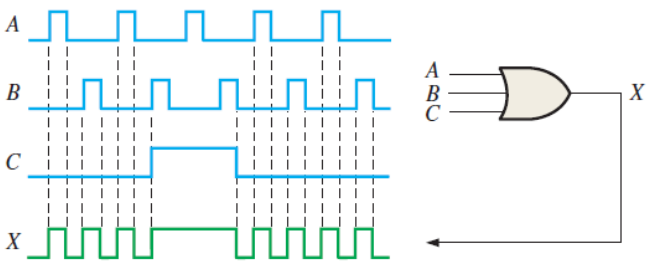
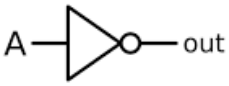


FIGURE 3–23

3. **NOT Gate (Inverter)**

- Symbol:



Input	Output
A	Y
0	1
1	0

Example:

If the two waveforms *A* and *B* shown in Figure 3–28 are applied to the NAND gate inputs, determine the resulting output waveform.

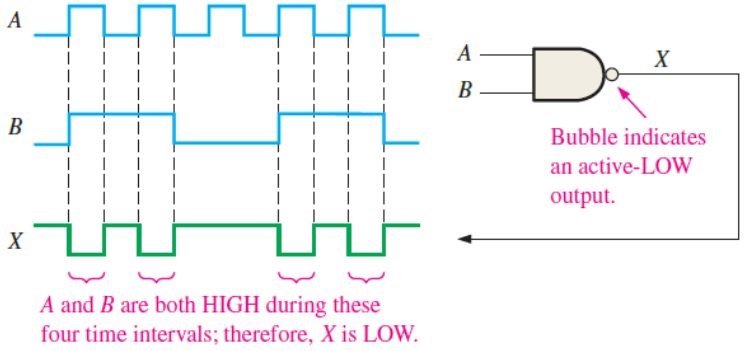
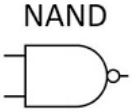


FIGURE 3–28

4. **NAND Gate**

- Symbol:



A	B	Output
0	0	1
1	0	1
0	1	1
1	1	0

Example:

Show the output waveform for the 3-input NAND gate in Figure 3–29 with its proper time relationship to the inputs.

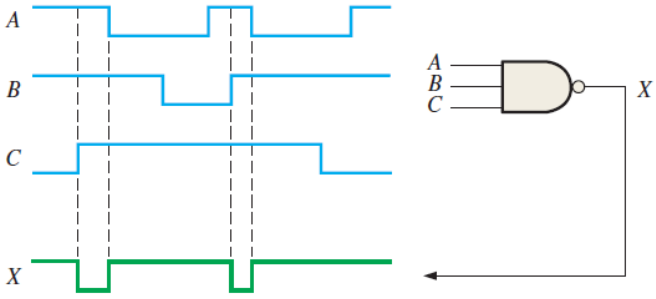
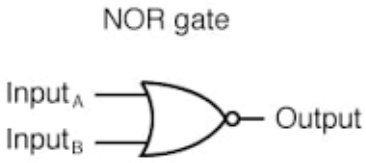


FIGURE 3–29

5. **NOR Gate**

- Symbol:



A	B	Output
0	0	1
0	1	0
1	0	0
1	1	0

Example:

If the two waveforms shown in Figure 3–36 are applied to a NOR gate, what is the resulting output waveform?

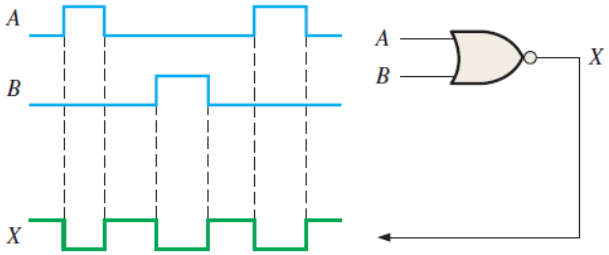


FIGURE 3–36

Example:

Show the output waveform for the 3-input NOR gate in Figure 3–37 with the proper time relation to the inputs.

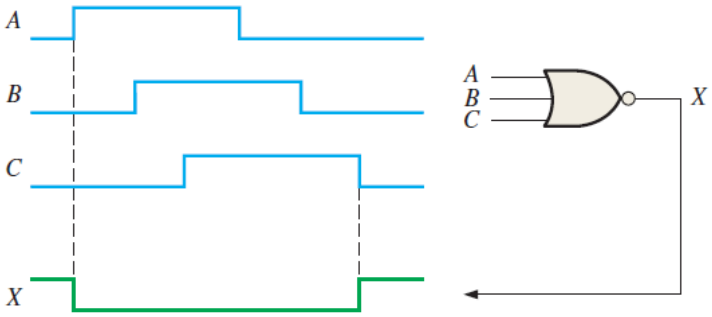
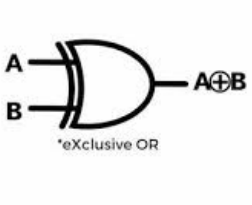


FIGURE 3–37

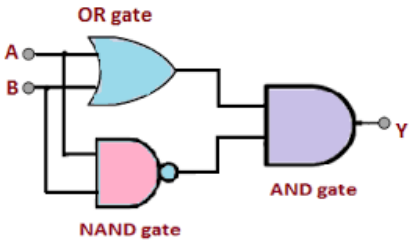
6. **XOR Gate (Exclusive OR)**

- Symbol:

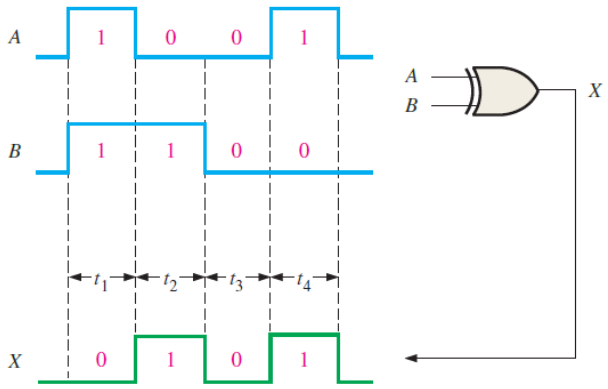


2 input XOR gate

A	B	A⊕B
0	0	0
0	1	1
1	0	1
1	1	0



- Operation with Waveform Inputs:



Example:

Determine the output waveforms for the XOR gate and for the XNOR gate, given the input waveforms, *A* and *B*, in Figure 3–48.

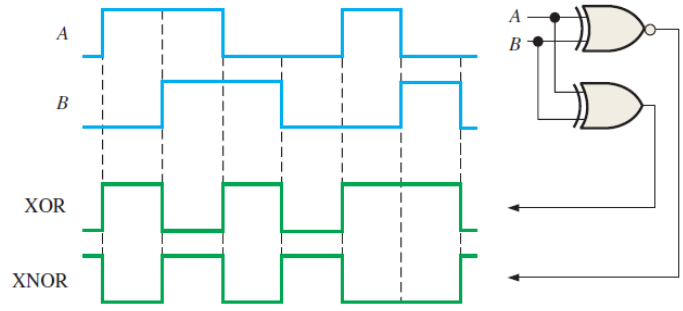
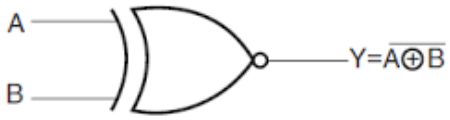


FIGURE 3–48

7. **XNOR Gate (Exclusive NOR)**

- Symbol:



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

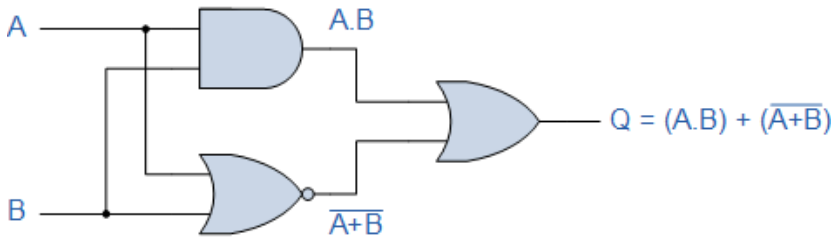
A	B	Y
0	0	1
0	1	0
1	0	0
1	1	1

Summary

- **AND Gate**: True if both inputs are true.
- **OR Gate**: True if at least one input is true.
- **NOT Gate**: Inverts the input.
- **NAND Gate**: True if not both inputs are true.
- **NOR Gate**: True if neither input is true.
- **XOR Gate**: True if exactly one input is true.
- **XNOR Gate**: True if both inputs are the same.

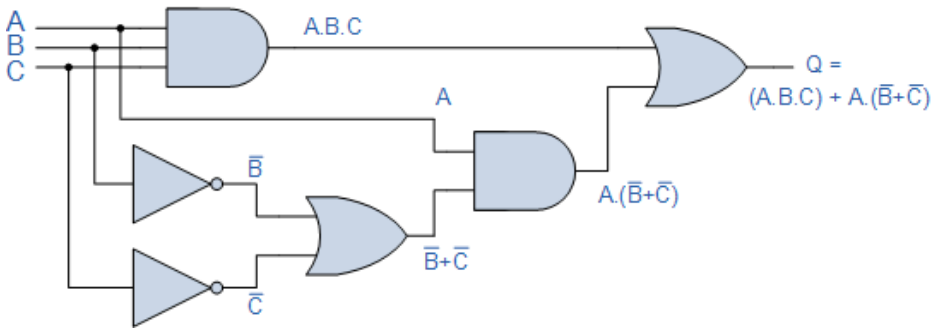
Logic Function	Boolean Notation
AND	A.B
OR	A+B
NOT	$\overline{A}$
NAND	$\overline{A.B}$
NOR	$\overline{A+B}$
EX-OR	$(A.\overline{B}) + (\overline{A}.B)$ or $A \oplus B$
EX-NOR	$(A.B) + (\overline{A}.\overline{B})$ or $\overline{A \oplus B}$

Example:



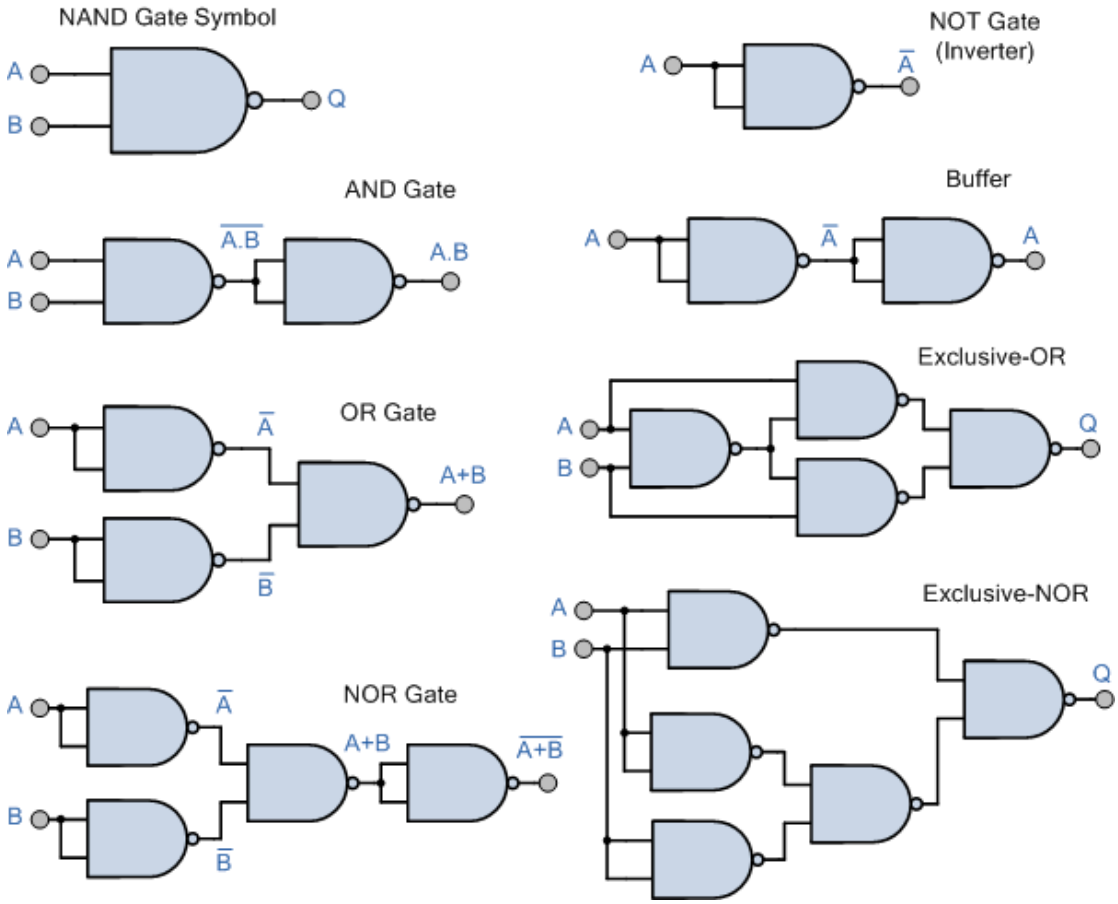
Inputs		Intermediates		Output
B	A	A.B	$\overline{A + B}$	Q
0	0	0	1	1
0	1	0	0	0
1	0	0	0	0
1	1	1	0	1

Example:

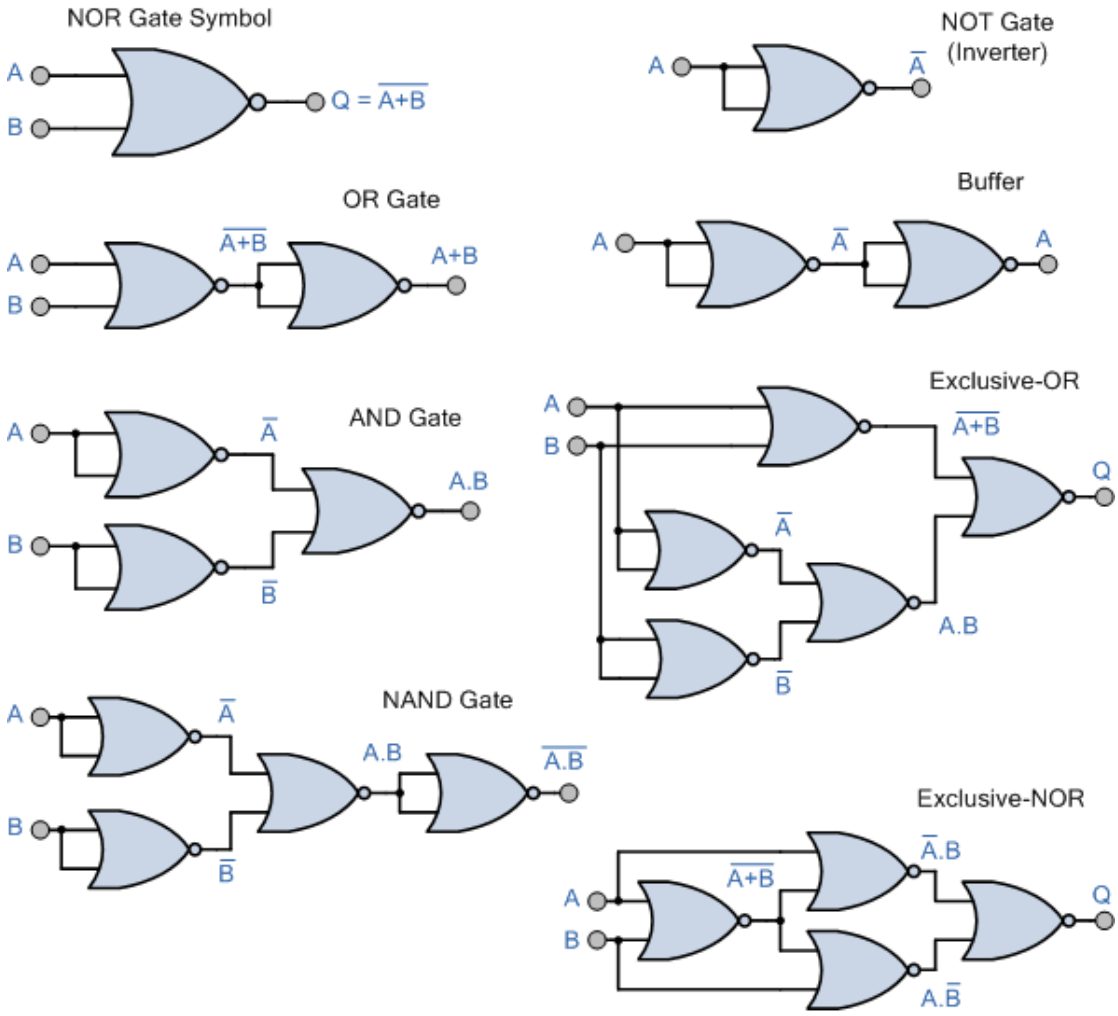


Inputs			Intermediates					Output
C	B	A	A.B.C	$\overline{B}$	$\overline{C}$	$\overline{B}+\overline{C}$	$A.(\overline{B}+\overline{C})$	Q
0	0	0	0	1	1	1	0	0
0	0	1	0	1	1	1	1	1
0	1	0	0	0	1	1	0	0
0	1	1	0	0	1	1	1	1
1	0	0	0	1	0	1	0	0
1	0	1	0	1	0	1	1	1
1	1	0	0	0	0	0	0	0
1	1	1	1	0	0	0	0	1

Implementation of Universal Logic Gates Function Using Only NAND



Implementation of Universal Logic Gates Function Using Only NOR



Boolean Algebra

Boolean algebra is a branch of algebra that deals with boolean values (true and false), typically represented by 1 and 0. It is used to analyze and simplify digital circuits and logical expressions. Boolean algebra has several fundamental operations, laws, and properties.

Laws of Boolean Algebra

1. Identity Laws:
   - $A + 0 = A$
   - $A \cdot 1 = A$
2. Null Laws:
   - $A + 1 = 1$
   - $A \cdot 0 = 0$
3. Idempotent Laws:
   - $A + A = A$
   - $A \cdot A = A$
4. Complement Laws:
   - $A + \overline{A} = 1$
   - $A \cdot \overline{A} = 0$
5. Double Negation Law:
   - $\overline{\overline{A}} = A$
6. Commutative Laws:
   - $A + B = B + A$
   - $A \cdot B = B \cdot A$
7. Associative Laws:
   - $(A + B) + C = A + (B + C)$
   - $(A \cdot B) \cdot C = A \cdot (B \cdot C)$
8. Distributive Laws:
   - $A \cdot (B + C) = (A \cdot B) + (A \cdot C)$
   - $A + (B \cdot C) = (A + B) \cdot (A + C)$
9. Absorption Laws:
   - $A + (A \cdot B) = A$
   - $A \cdot (A + B) = A$
10. De Morgan's Theorems:
    - $\overline{A \cdot B} = \overline{A} + \overline{B}$
    - $\overline{A + B} = \overline{A} \cdot \overline{B}$

Simplification Using Boolean Algebra

Boolean algebra simplification involves reducing complex Boolean expressions to their simplest form. This process is crucial in digital design to minimize the number of logic gates required, which leads to more efficient circuits.

Example 1: Simplify $A + \overline{A}B$

1. Apply the Absorption Law:

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

Example 2: Simplify $\overline{(A + B) \cdot (A + \overline{B})}$

1. Apply De Morgan's Theorem:

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

2. Apply De Morgan's Theorem again to each term:

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

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

3. Combine the results:

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

4. Factor out the common term:

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

5. Apply the Complement Law:

$$\overline{B} + B = 1$$

6. Simplify:

$$\overline{A} \cdot 1 = \overline{A}$$

Result: The simplified expression is $\overline{A}$.

Example:

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

Step 1: Identify the terms to which you can apply DeMorgan's theorems, and think of each term as a single variable. Let $A + B\overline{C} = X$ and $D(\overline{E + \overline{F}}) = Y$.

Step 2: Since $\overline{X + Y} = \overline{X}\,\overline{Y}$,

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

Step 3: Use rule 9 ($\overline{\overline{A}} = A$) to cancel the double bars over the left term (this is not part of DeMorgan's theorem).

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

Step 4: Apply DeMorgan's theorem to the second term.

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

Step 5: Use rule 9 ($\overline{\overline{A}} = A$) to cancel the double bars over the $E + \overline{F}$ part of the term.

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

Example:

Apply DeMorgan's theorems to each of the following expressions:

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

(b) $\overline{ABC + DEF}$

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

Solution

(a) Let $A + B + C = X$ and $D = Y$. The expression $\overline{(A + B + C)D}$ is of the form $\overline{XY} = \overline{X} + \overline{Y}$ and can be rewritten as

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

Next, apply DeMorgan's theorem to the term $\overline{A + B + C}$.

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

(b) Let $ABC = X$ and $DEF = Y$. The expression $\overline{ABC + DEF}$ is of the form $\overline{X + Y} = \overline{X}\,\overline{Y}$ and can be rewritten as

$$\overline{ABC + DEF} = (\overline{ABC})(\overline{DEF})$$

Next, apply DeMorgan's theorem to each of the terms $\overline{ABC}$ and $\overline{DEF}$.

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

(c) Let $A\overline{B} = X$, $\overline{C}D = Y$, and $EF = Z$. The expression $\overline{A\overline{B} + \overline{C}D + EF}$ is of the form $\overline{X + Y + Z} = \overline{X}\,\overline{Y}\,\overline{Z}$ and can be rewritten as

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

Next, apply DeMorgan's theorem to each of the terms $\overline{A\overline{B}}$, $\overline{\overline{C}D}$, and $\overline{EF}$.

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

Example:

Apply DeMorgan's theorems to each expression:

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

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

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

Solution

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

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

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

Summary

- **Boolean algebra** provides a set of rules for manipulating binary variables.
- **Fundamental operations** include AND, OR, and NOT.
- **Laws of Boolean algebra** help simplify expressions to reduce the number of operations.
- **Simplification** involves applying these laws step-by-step to achieve a simpler form of the expression.

Boolean Algebra Simplification Examples

When simplifying complex Boolean expressions, we can use Boolean algebra laws systematically to reduce them to their simplest form. Here are a few detailed examples that demonstrate the step-by-step process of Boolean simplification.

Example 1: Simplify $A \cdot (B + C) + \overline{A} \cdot C$

1. Expand the Expression Using Distributive Law:
   - $A \cdot (B + C) = A \cdot B + A \cdot C$
   - So, the expression becomes:
     $$A \cdot B + A \cdot C + \overline{A} \cdot C$$
2. Combine Like Terms:
   - Notice that $A \cdot C$ and $\overline{A} \cdot C$ can be combined:
     $$A \cdot C + \overline{A} \cdot C = (A + \overline{A}) \cdot C$$

3. **Apply the Complement Law**:
   - $A + \overline{A} = 1$
   - Thus:
   $$(A + \overline{A}) \cdot C = 1 \cdot C = C$$
4. **Substitute Back**:
   - The simplified expression is:
   $$A \cdot B + C$$

Result: The simplified expression is $A \cdot B + C$.

Example 2: Simplify $(A + B) \cdot (\overline{A} + C)$

1. **Expand the Expression Using Distributive Law**:
   - Apply distributive property:
   $$(A + B) \cdot (\overline{A} + C) = A \cdot (\overline{A} + C) + B \cdot (\overline{A} + C)$$
2. **Distribute Each Term**:
   - For $A \cdot (\overline{A} + C)$:
   $$A \cdot \overline{A} + A \cdot C$$
     - $A \cdot \overline{A} = 0$ (Complement Law)
     - Thus:
     $$A \cdot \overline{A} + A \cdot C = 0 + A \cdot C = A \cdot C$$
   - For $B \cdot (\overline{A} + C)$:
   $$B \cdot \overline{A} + B \cdot C$$
3. **Combine Results**:
   - Combining:
   $$A \cdot C + B \cdot \overline{A} + B \cdot C$$
   - Group $A \cdot C$ and $B \cdot C$:
   $$(A + B) \cdot C + B \cdot \overline{A}$$

4. **Combine Terms**:
   - The final simplified expression is:
$$(A + B) \cdot C + B \cdot \overline{A}$$

Result: The simplified expression is $(A + B) \cdot C + B \cdot \overline{A}$.

Example 3: Simplify $A \cdot (B + C) \cdot \overline{B}$

1. **Apply the Distributive Law**:
   - Expand $A \cdot (B + C) \cdot \overline{B}$:
$$A \cdot (\overline{B} \cdot B + \overline{B} \cdot C)$$

2. **Apply the Complement Law**:
   - $\overline{B} \cdot B = 0$:
$$A \cdot (0 + \overline{B} \cdot C) = A \cdot \overline{B} \cdot C$$

Result: The simplified expression is $A \cdot \overline{B} \cdot C$.

Example 4: Simplify $A \cdot \overline{B} + A \cdot B + \overline{A} \cdot C$

1. **Group Terms**:
   - Group A terms:
$$A \cdot (\overline{B} + B) + \overline{A} \cdot C$$

2. **Apply the Complement Law**:
   - $\overline{B} + B = 1$:
$$A \cdot 1 + \overline{A} \cdot C = A + \overline{A} \cdot C$$

3. **Apply the Distributive Law**:
   - The term $\overline{A} \cdot C$ cannot be simplified further with A:
$$A + \overline{A} \cdot C$$

Result: The simplified expression is $A + \overline{A} \cdot C$.

Example: Simplify the following Boolean expression:

Q =	(A + B).(A + C)	
	A.A + A.C + A.B + B.C	– Distributive law
	A + A.C + A.B + B.C	– Idempotent AND law (A.A = A)
	A(1 + C) + A.B + B.C	– Distributive law
	A.1 + A.B + B.C	– Identity OR law (1 + C = 1)
	A(1 + B) + B.C	– Distributive law
	A.1 + B.C	– Identity OR law (1 + B = 1)
Q =	A + (B.C)	– Identity AND law (A.1 = A)

Example: Simplify the following Boolean expression: A.(A + B)

	A.(A+B)	Start
multiply:	A.A + A.B	Distributive Law
but:	A.A = A	Idempotent Law
then:	A + A.B	Reduction
thus:	A.(1 + B)	Annulment Law
equals to:	A	Absorption Law

Example: Simplify the following Boolean expression: (A + B)(A + C)

	(A + B)(A + C)	Start
multiply:	A.A + A.C + A.B + B.C	Distributive law
but:	A.A = A	Idempotent Law
then:	A + A.C + A.B + B.C	Reduction
however:	A + A.C = A	Absorption Law
thus:	A + A.B + B.C	Distributive Law
again:	A + A.B = A	Absorption Law
thus:	A + B.C	Result

Example: Simplify the following Boolean expression: $AB(\overline{B}C + AC)$

	$AB(\overline{B}C+AC)$	Start
multiply	$A.B.\overline{B}.C + A.B.A.C$	Distributive Law
again:	$A.A = A$	Idempotent Law
then:	$A.B.\overline{B}.C + A.B.C$	Reduction
but:	$B.\overline{B} = 0$	Complement Law
so:	$A.0.C + A.B.C$	Reduction
becomes:	$0 + A.B.C$	Reduction
as:	$0 + A.B.C = A.B.C$	Identity Law
thus:	ABC	Result

Example: Simplify the following Boolean expression:

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

$(A+\overline{B}+\overline{C})(A+\overline{B}+C)(A+B+\overline{C})$
$A(A+\overline{B}+C)(A+B+\overline{C}) + \overline{B}(A+\overline{B}+C)(A+B+\overline{C}) + \overline{C}(A+\overline{B}+C)(A+B+\overline{C})$
$(AA+A\overline{B}+AC)(AA+AB+A\overline{C}) + (A\overline{B}+\overline{B}\,\overline{B}+\overline{B}\,\overline{C})(A\overline{B}+B\overline{B}+\overline{B}\,\overline{C}) + (A\overline{C}+\overline{B}\,\overline{C}+C\overline{C})(A\overline{C}+B\overline{C}+\overline{C}\,\overline{C})$
$AA = A$ (Idempotent Law)
$B\overline{B} = C\overline{C} = 0$ (Complement Law)
$(A+A\overline{B}+AC)(A+AB+A\overline{C}) + (A\overline{B}+\overline{B}+\overline{B}\,\overline{C})(A\overline{B}+0+\overline{B}\,\overline{C}) + (A\overline{C}+\overline{B}\,\overline{C}+0)(A\overline{C}+B\overline{C}+\overline{C})$
$A + AB = A$ (Absorption Law)
$A + AC = A$ (Absorption Law)
$A + 0 = A$ (Identity Law)
$(A+AC)(A+AB+A\overline{C}) + (A\overline{B}+\overline{B}+\overline{B}\,\overline{C})(A\overline{B}+\overline{B}\,\overline{C}) + (A\overline{C}+\overline{B}\,\overline{C})(A\overline{C}+B\overline{C}+\overline{C})$
$A(A+AB+A\overline{C}) + (\overline{B}+\overline{B}\,\overline{C})(A\overline{B}+\overline{B}\,\overline{C}) + (A\overline{C}+\overline{B}\,\overline{C})(B\overline{C}+\overline{C})$
$A(A+A\overline{C}) + \overline{B}(A\overline{B}+\overline{B}\,\overline{C}) + \overline{C}(A\overline{C}+\overline{B}\,\overline{C})$
$AA + (A\overline{B}+\overline{B}\,\overline{B}\,\overline{C}) + (A\overline{C}\,\overline{C}+\overline{B}\,\overline{C}\,\overline{C})$

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

Sum of Products (SOP)

The Sum of Products (SOP) is a canonical form used in Boolean algebra to represent Boolean expressions. In SOP form, the expression is written as a sum (OR operation) of several product terms (AND operations). Each product term is a conjunction of literals (variables or their complements).

Example:

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

Converting Non-Standard SOP to Standard SOP

Example:

$f(A, B, C) = AB + BC$

1. **Expand AB:**
   - Missing variable: C
   - Expand AB by adding $C + \overline{C}$:
     $AB = AB(C + \overline{C})$
2. **Distribute:**
   $AB = ABC + AB\overline{C}$
3. **Expand BC:**
   - Missing variable: A
   - Expand BC by adding $A + \overline{A}$:
     $BC = BC(A + \overline{A})$

4. **Distribute**:
   $BC = ABC + B\overline{A}C$
5. **Combine All Terms**:
   $f(A, B, C) = ABC + AB\overline{C} + ABC + B\overline{A}C$
6. **Remove Redundant Terms**:
   $f(A, B, C) = ABC + AB\overline{C} + B\overline{A}C$
   - The term ABC is repeated and can be written once.

Final Standard SOP Form

$f(A, B, C) = AB\overline{C} + ABC + B\overline{A}C$

Example:

Given Expression:
$f(A, B, C, D) = A\overline{B} + AC + B\overline{C}D$

1. **Expand $A\overline{B}$**:
   - Missing variables: C and D
   - Expand $A\overline{B}$:
   $$A\overline{B} = A\overline{B}(C + \overline{C})(D + \overline{D})$$
2. **Distribute**:
   $$A\overline{B} = A\overline{B}C(D + \overline{D}) + A\overline{B}\overline{C}(D + \overline{D})$$
   $$A\overline{B} = A\overline{B}CD + A\overline{B}C\overline{D} + A\overline{B}\overline{C}D + A\overline{B}\overline{C}\overline{D}$$
3. **Expand AC**:
   - Missing variables: B and D
   - Expand AC:
   $$AC = AC(B + \overline{B})(D + \overline{D})$$
4. **Distribute**:
   $$AC = ACB(D + \overline{D}) + AC\overline{B}(D + \overline{D})$$
   $$AC = ACBD + ACB\overline{D} + AC\overline{B}D + AC\overline{B}\overline{D}$$

5. **Expand $B\overline{C}D$:**
   - Missing variable: A
   - Expand $B\overline{C}D$:

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

6. **Distribute:**

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

7. **Combine All Terms:**

$$f(A, B, C, D) = A\overline{B}CD + A\overline{B}C\overline{D} + A\overline{B}\overline{C}D + A\overline{B}\overline{C}\overline{D} + ACBD + ACB\overline{D} + AC$$

8. **Remove Redundant Terms:**
   - Combine like terms if any are present.

Final Standard SOP Form

The expanded and simplified expression, ensuring each product term includes all variables $A, B, C,$ and D, is:

$$f(A, B, C, D) = A\overline{B}CD + A\overline{B}C\overline{D} + A\overline{B}\overline{C}D + A\overline{B}\overline{C}\overline{D} + ACBD + ACB\overline{D} + AC\overline{B}D + AC\overline{B}\overline{D} + AB\overline{C}D + \overline{A}B\overline{C}D$$

Example:

Convert the following Boolean expression into standard SOP form:

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

Solution

The domain of this SOP expression is A, B, C, D. Take one term at a time. The first term, $A\overline{B}C$, is missing variable D or $\overline{D}$, so multiply the first term by $D + \overline{D}$ as follows:

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

In this case, two standard product terms are the result.

The second term, $\overline{A}\,\overline{B}$, is missing variables C or $\overline{C}$ and D or $\overline{D}$, so first multiply the second term by $C + \overline{C}$ as follows:

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

The two resulting terms are missing variable D or $\overline{D}$, so multiply both terms by $D + \overline{D}$ as follows:

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

In this case, four standard product terms are the result.

The third term, $AB\overline{C}D$, is already in standard form. The complete standard SOP form of the original expression is as follows:

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

Product of Sums (POS) in Boolean Algebra

A Product of Sums (POS) expression is a form of Boolean algebra where the expression is a product (AND operation) of multiple sum (OR operation) terms. Each sum term (also known as a maxterm) includes all the variables of the function, either in complemented or uncomplemented form.

Definition

In a POS expression, each sum term represents a maxterm, and the overall expression is the AND of these maxterms.

Example of POS Expression

Non-Standard POS Expression:

$f(A, B, C) = (A + B)(B + C)$

Converting Non-Standard POS to Standard POS

To convert a non-standard POS expression to a standard POS expression, each sum term must include all variables in the function. Here's how to do it step-by-step.

Example 1

Given Non-Standard POS Expression:

$f(A, B, C) = (A + B)(B + C)$

1. **Identify Missing Variables:**
   - Each sum term needs to include all three variables A, B, and C.
2. **Expand Each Term:**
   - Expand each sum term by introducing the missing variables in both complemented and uncomplemented forms.

Step-by-Step Conversion:

1. **Expand $(A + B)$:**
   - Missing variable: C
   - Expand $(A + B)$:

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

2. **Expand $(B + C)$:**
   - Missing variable: A
   - Expand $(B + C)$:

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

3. **Combine All Terms**:
   - Combine the expanded terms using AND:
     $$f(A, B, C) = (A + B + C)(A + B + \overline{C})(B + C + A)(B + C + \overline{A})$$

Final Standard POS Form

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

Example 2

Given Non-Standard POS Expression:
$f(A, B, C, D) = (A + \overline{B})(B + C)(C + \overline{D})$

1. **Identify Missing Variables**:
   - Each sum term needs to include all four variables A, B, C, and D.
2. **Expand Each Term**:
   - Expand each sum term by introducing the missing variables in both complemented and uncomplemented forms.

Step-by-Step Conversion:

1. **Expand** $(A + \overline{B})$:
   - Missing variables: C and D
   - Expand $(A + \overline{B})$:
     $$(A + \overline{B}) = (A + \overline{B} + C + D)(A + \overline{B} + C + \overline{D})(A + \overline{B} + \overline{C} + D)(A + \overline{B} + \overline{C}$$
2. **Expand** $(B + C)$:
   - Missing variables: A and D
   - Expand $(B + C)$:
     $$(B + C) = (B + C + A + D)(B + C + A + \overline{D})(B + C + \overline{A} + D)(B + C + \overline{A}$$
3. **Expand** $(C + \overline{D})$:
   - Missing variables: A and B
   - Expand $(C + \overline{D})$:
     $$(C + \overline{D}) = (C + \overline{D} + A + B)(C + \overline{D} + A + \overline{B})(C + \overline{D} + \overline{A} + B)(C + \overline{D} + \overline{A}$$
4. **Combine All Terms**:
   - Combine the expanded terms using AND:
     $$f(A, B, C, D) = (A + \overline{B} + C + D)(A + \overline{B} + C + \overline{D})(A + \overline{B} + \overline{C} + D)(A + \overline{B}$$

Final Standard POS Form

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

Example:

Convert the following Boolean expression into standard POS form:

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

Solution

The domain of this POS expression is A, B, C, D. Take one term at a time. The first term, $A + \overline{B} + C$, is missing variable D or $\overline{D}$, so add $D\overline{D}$ and apply rule 12 as follows:

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

The second term, $\overline{B} + C + \overline{D}$, is missing variable A or $\overline{A}$, so add $A\overline{A}$ and apply rule 12 as follows:

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

The third term, $A + \overline{B} + \overline{C} + D$, is already in standard form. The standard POS form of the original expression is as follows:

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

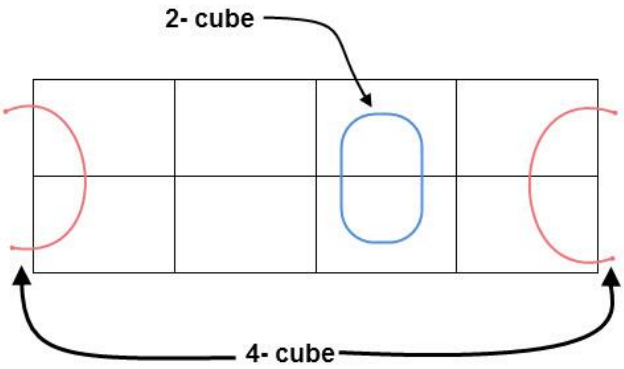
Karnaugh Map (K-Map)

A Karnaugh Map (K-Map) is a visual method used in Boolean algebra to simplify expressions and minimize the number of logical operations. It organizes truth table values into a grid format, making it easier to identify and eliminate redundant terms using grouping techniques.

3 variables:

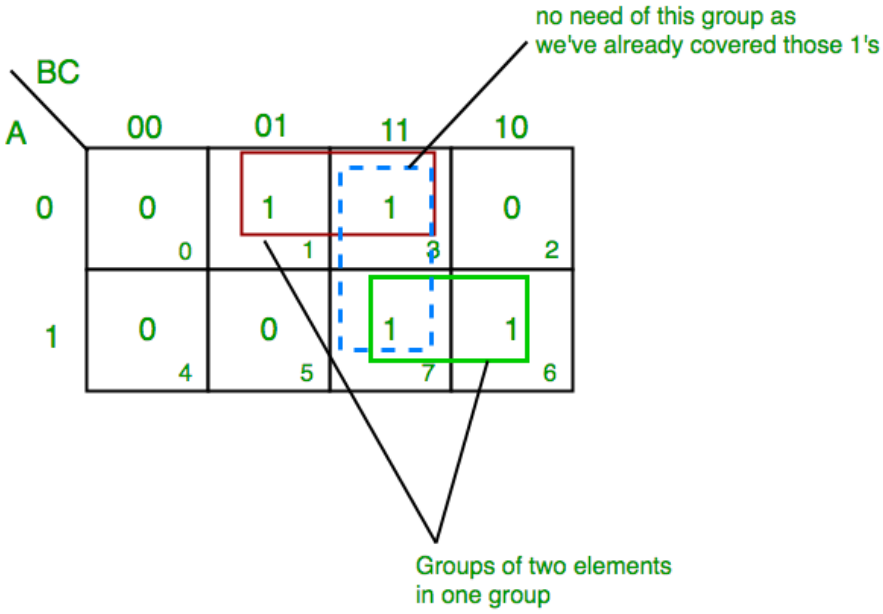
A \ BC	00	01	11	10
0	0 A'B'C'	1 A'B'C	3 A'BC	2 A'BC'
1	4 AB'C'	5 AB'C	7 ABC	6 ABC'

We can take:



Example: Simplify the following expression

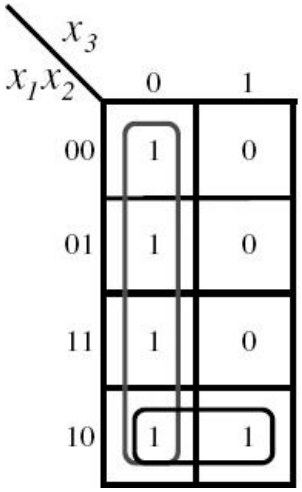
$$\bar{A}\bar{B}C+\bar{A}B\bar{C}+AB\bar{C}+AB\bar{C}$$



Result: $\bar{A}C+AB$

Example: Simplify the following expression

$$F=\overline{X}_1\,\overline{X}_2\,\overline{X}_3+\overline{X}_1\,X_2\,\overline{X}_3+X_1\,X_2\,\overline{X}_3+X_1\,\overline{X}_2\,\overline{X}_3+X_1\,\overline{X}_2\,X_3$$

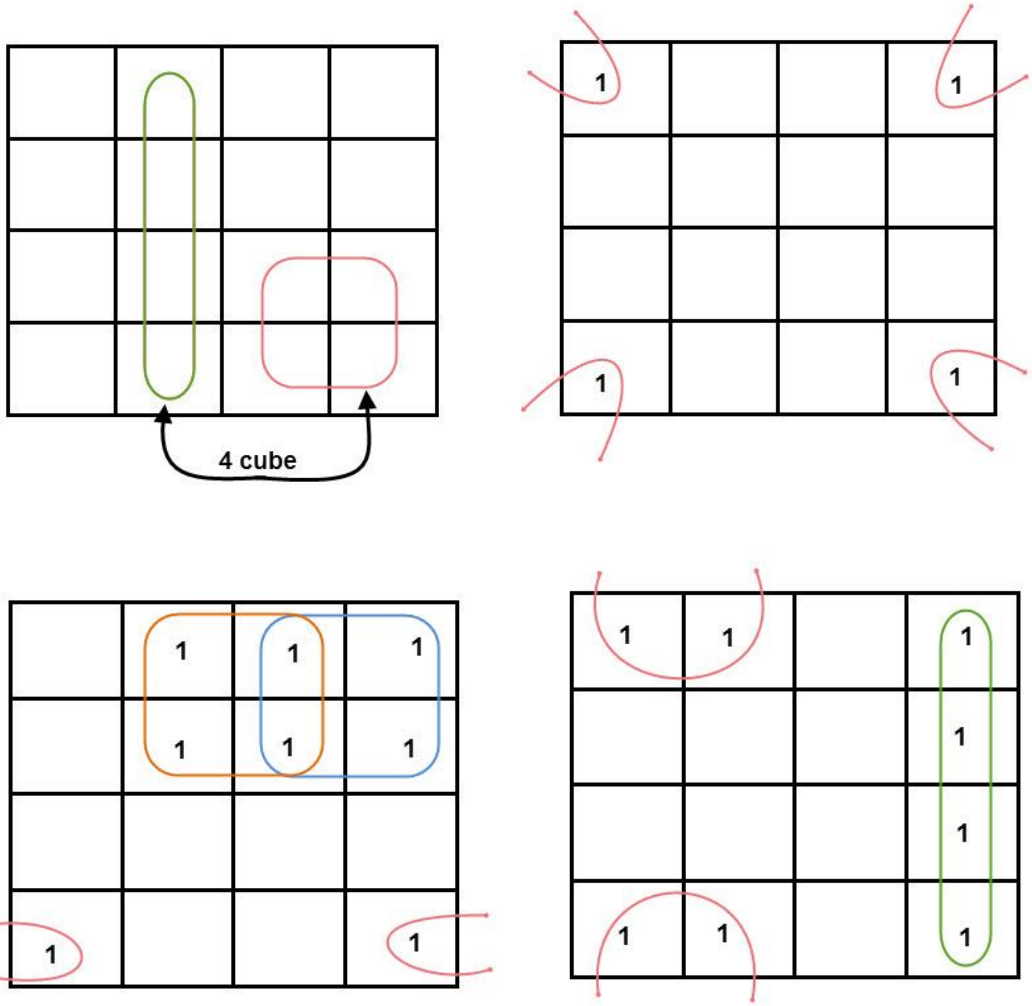


$$f = \overline{x}_3 + x_1\overline{x}_2$$

4 variables:

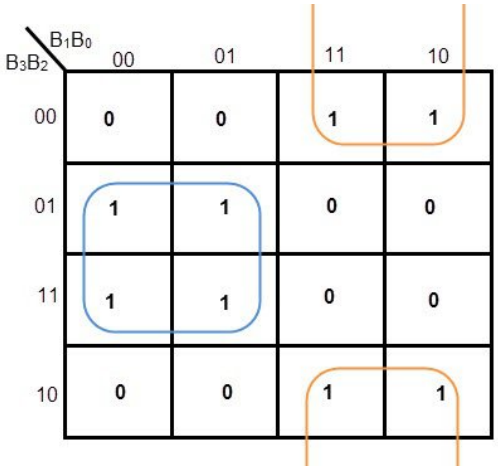
AB \ CD	00	01	11	10
00	0 A' B' C' D'	1 A' B' C' D	3 A' B' C D	2 A' B' C D'
01	4 A' B C' D'	5 A' B C' D	7 A' B C D	6 A' B C D'
11	12 A B C' D'	13 A B C' D	15 A B C D	14 A B C D'
10	8 A B' C' D'	9 A B' C' D	11 A B' C D	10 A B' C D'

We can take:



Example: Simplify the following expression

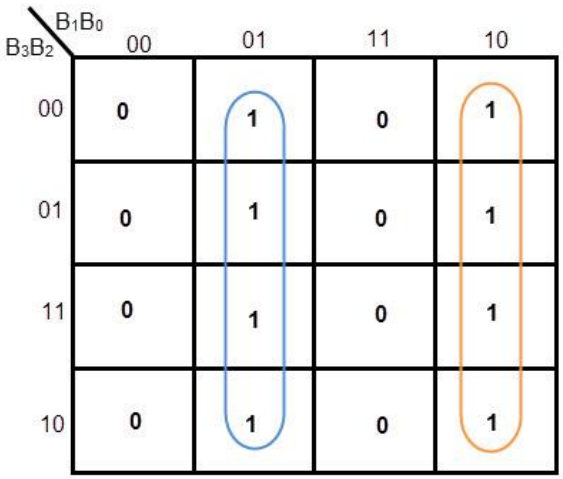
$$F= \overline{B}_0\,\overline{B}_1\,B_2\,\overline{B}_3 + B_0\,\overline{B}_1\,B_2\,\overline{B}_3 + \overline{B}_0\,\overline{B}_1\,B_2\,B_3 + B_0\,\overline{B}_1\,B_2\,B_3 + B_0\,B_1\,\overline{B}_2\,\overline{B}_3 + \overline{B}_0\,B_1\,\overline{B}_2\,\overline{B}_3 + B_0\,B_1\,\overline{B}_2\,B_3 + \overline{B}_0\,B_1\,\overline{B}_2\,B_3$$



Result: $F= \overline{B}_1 B_2 + B_1 \overline{B}_2$

Example: Simplify the following expression

$$F= B_0 \overline{B}_1 \overline{B}_2 \overline{B}_3 + B_0 \overline{B}_1 B_2 \overline{B}_3 + B_0 \overline{B}_1 B_2 B_3 + B_0 \overline{B}_1 \overline{B}_2 B_3 + \overline{B}_0 B_1 \overline{B}_2 \overline{B}_3 + \overline{B}_0 B_1 B_2 \overline{B}_3 + \overline{B}_0 B_1 B_2 B_3 + \overline{B}_0 B_1 \overline{B}_2 B_3$$



Result: $F= B_0 \overline{B}_1 + \overline{B}_0 B_1$

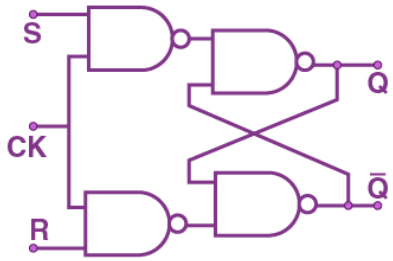
Flip-Flops

Flip-flops are fundamental building blocks in digital electronics, primarily used for storing binary data and implementing sequential logic circuits. They are bistable devices, meaning they have two stable states and can be used to store a single bit of information.

Types of Flip-Flops:

1. SR Flip-Flop (Set-Reset Flip-Flop)
2. D Flip-Flop (Data or Delay Flip-Flop)
3. JK Flip-Flop
4. T Flip-Flop (Toggle Flip-Flop)

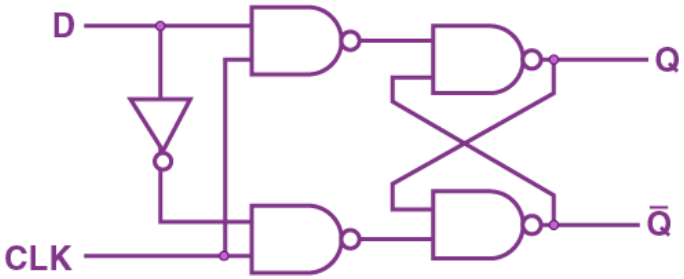
1. SR Flip-Flop



Truth Table

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	-
1	1	1	-

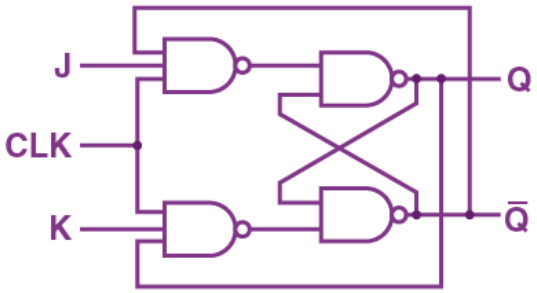
2. D Flip-Flop



Truth Table

Q	D	$Q_{(t+1)}$
0	0	0
0	1	1
1	0	0
1	1	1

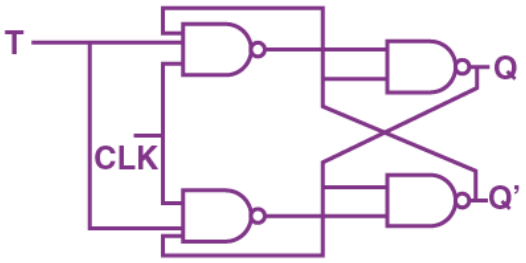
3. JK Flip-Flop



Truth Table

J	K	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	1
1	1	1	0

4. T Flip-Flop



Truth Table

T	Q_N	Q_{N+1}
0	0	0
0	1	1
1	0	1
1	1	0

Applications of Flip-Flops

1. Counters: Used to count occurrences of events.

2. Shift Registers: Used for data storage and transfer.

3. Memory Storage: Used in RAM and other memory devices.

4. Data Synchronization: Helps in synchronizing data between different parts of a system.

Counters

A Counter is a device which stores (and sometimes displays) the number of times a particular event or process has occurred, often in relationship to a clock signal. Counters are used in digital electronics for counting purpose, they can count specific event happening in the circuit. For example, in UP counter a counter increases count for every rising edge of clock. Not only counting, a counter can follow the certain sequence based on our design like any random sequence 0,1,3,2… .They can also be designed with the help of flip flops. They are used as frequency dividers where the frequency of given pulse waveform is divided. Counters are sequential circuit that count the number of pulses can be either in binary code or BCD form. The main properties of a counter are timing , sequencing , and counting. Counter works in two modes :

- Up counter
- Down counter

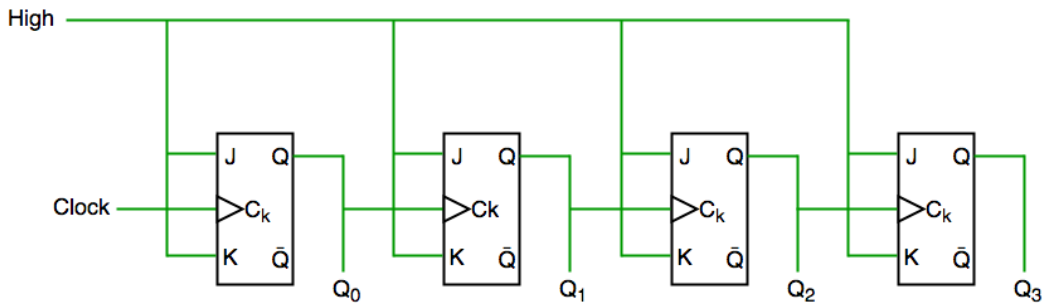
- Counters are broadly divided into two categories

1. Asynchronous counter
2. Synchronous counter

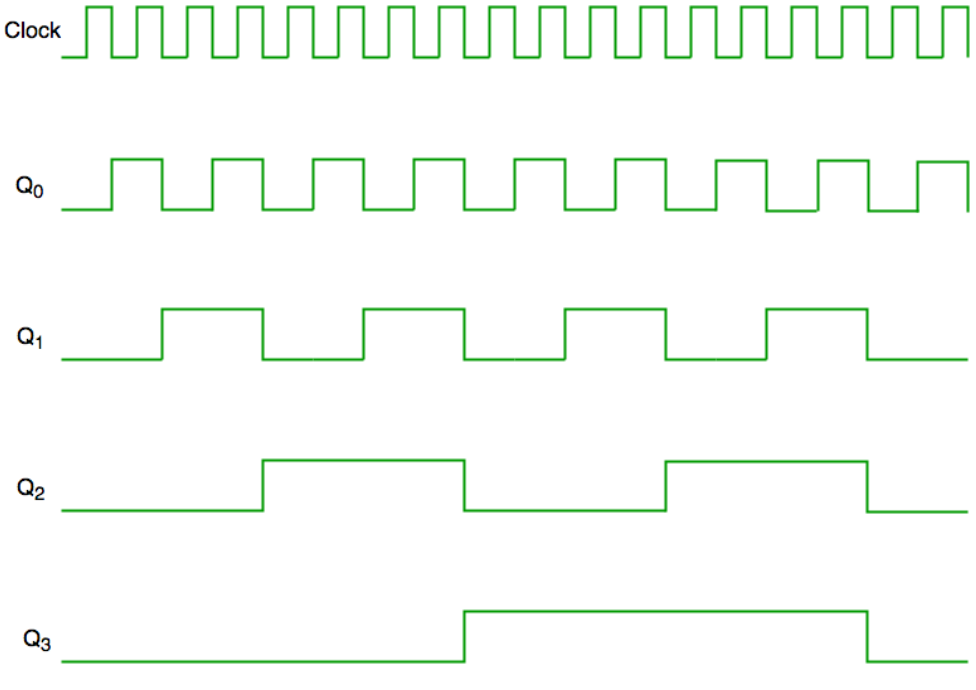
1. Asynchronous Counter

In asynchronous counter we don't use universal clock, only first flip flop is driven by main clock and the clock input of rest of the following flip flop

is driven by output of previous flip flops. We can understand it by following diagram :



(a) Asynchronous counter

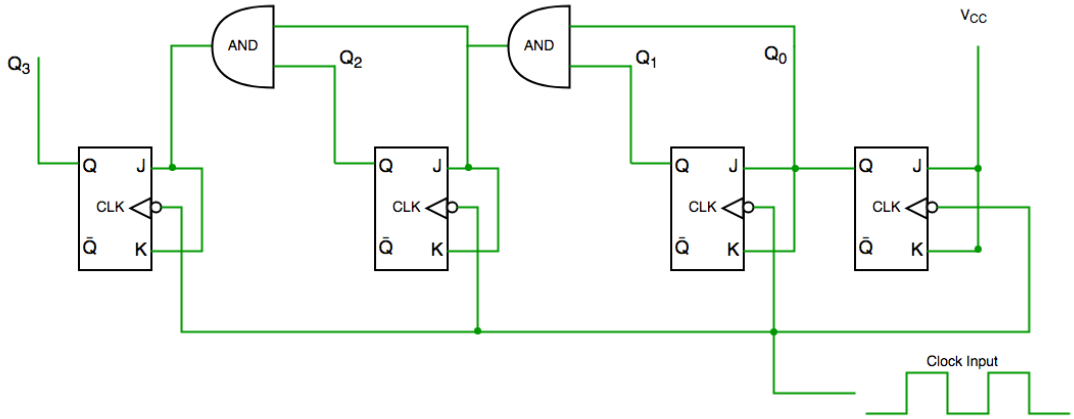


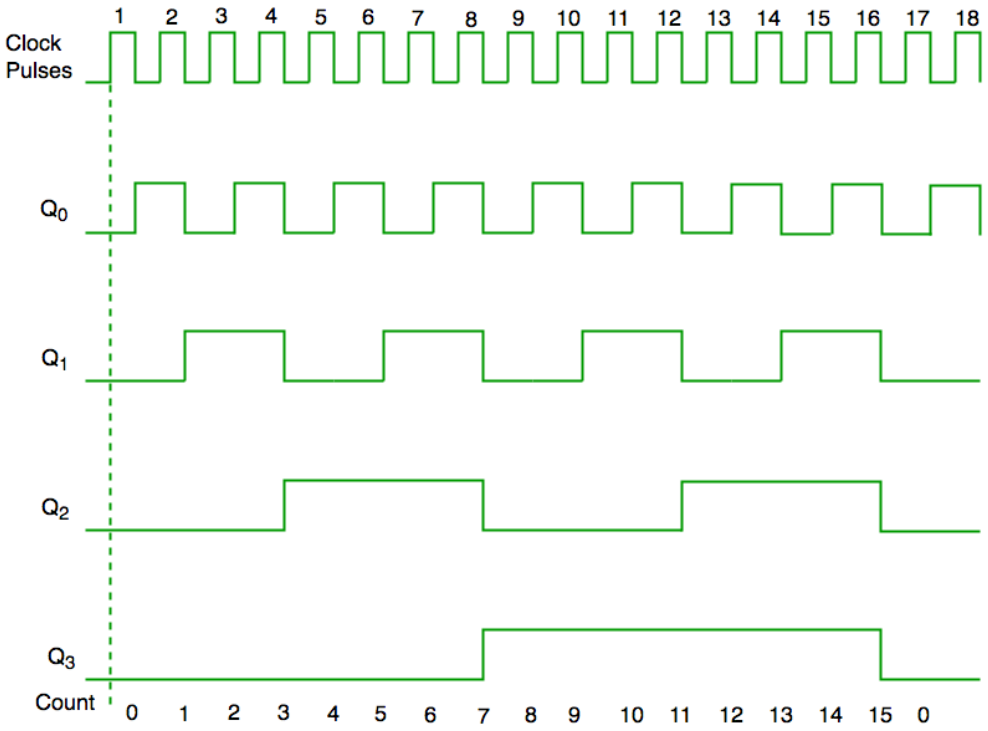
(b) Timing Diagram

It is evident from timing diagram that Q0 is changing as soon as the rising edge of clock pulse is encountered, Q1 is changing when rising edge of Q0 is encountered (because Q0 is like clock pulse for second flip flop) and so on. In this way ripples are generated through Q0,Q1,Q2,Q3 hence it is also called RIPPLE counter and serial counter**.** A ripple counter is a cascaded arrangement of flip flops where the output of one flip flop drives the clock input of the following flip flop

2. **Synchronous Counter**

Unlike the asynchronous counter, synchronous counter has one global clock which drives each flip flop so output changes in parallel. The one advantage of synchronous counter over asynchronous counter is, it can operate on higher frequency than asynchronous counter as it does not have cumulative delay because of same clock is given to each flip flop. It is also called as parallel counter.





From circuit diagram we see that Q0 bit gives response to each falling edge of clock while Q1 is dependent on Q0, Q2 is dependent on Q1 and Q0 , Q3 is dependent on Q2,Q1 and Q0.

Shift Registers

Flip flops can be used to store a single bit of binary data (1 or 0). However, in order to store multiple bits of data, we need multiple flip-flops. N flip flops are to be connected in order to store n bits of data. A Register is a device that is used to store such information. It is a group of flip-flops connected in series used to store multiple bits of data. The information stored within these registers can be transferred with the help of shift registers.

Shift Register is a group of flip flops used to store multiple bits of data. The bits stored in such registers can be made to move within the registers and in/out of the registers by applying clock pulses. An n-bit shift register can be formed by connecting n flip-flops where each flip-flop stores a single bit of data. The registers which will shift the bits to the left are called “Shift left registers”. The registers which will shift the bits to the right are called “Shift right registers”. Shift registers are basically of following types.

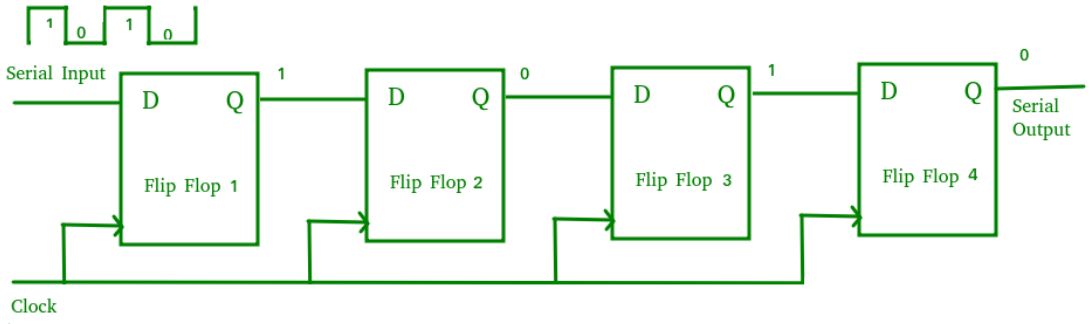
Types of Shift Registers:

- Serial In Serial Out shift register
- Serial In parallel Out shift register
- Parallel In Serial Out shift register
- Parallel In parallel Out shift register
- Bidirectional Shift Register
- Universal Shift Register
- Shift Register Counter

Serial-In Serial-Out Shift Register (SISO)

The shift register, which allows serial input (one bit after the other through a single data line) and produces a serial output is known as a

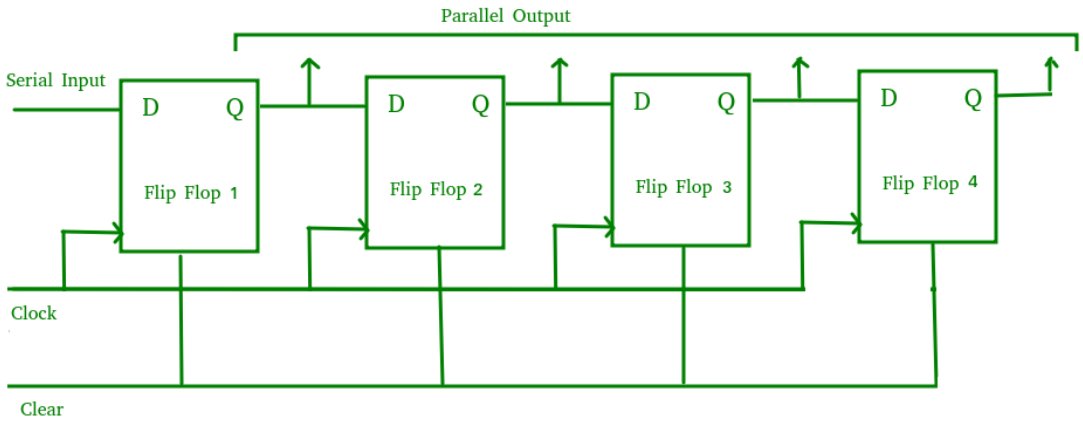
Serial-In Serial-Out shift register. Since there is only one output, the data leaves the shift register one bit at a time in a serial pattern, thus the name Serial-In Serial-Out Shift Register. The logic circuit given below shows a serial-in serial-out shift register. The circuit consists of four D flip-flops which are connected in a serial manner. All these flip-flops are synchronous with each other since the same clock signal is applied to each flip-flop.



The above circuit is an example of a shift right register, taking the serial data input from the left side of the flip flop. The main use of a SISO is to act as a delay element.

Serial-In Parallel-Out Shift Register (SIPO)

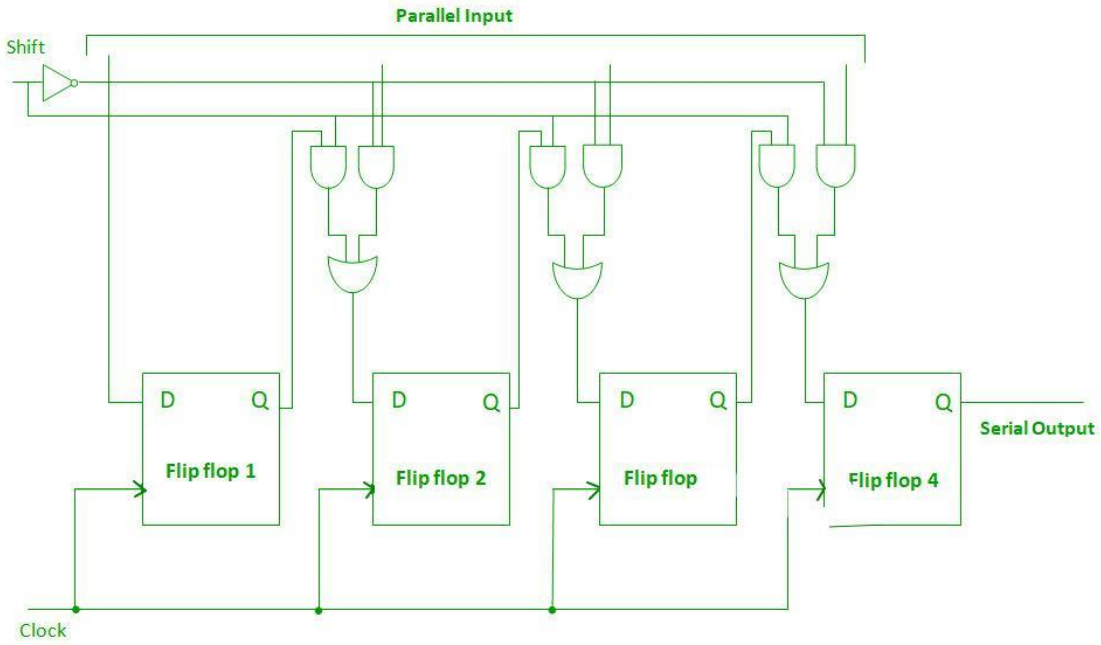
The shift register, which allows serial input (one bit after the other through a single data line) and produces a parallel output is known as the Serial-In Parallel-Out shift register. The logic circuit given below shows a serial-in-parallel-out shift register. The circuit consists of four D flip-flops which are connected. The clear (CLR) signal is connected in addition to the clock signal to all 4 flip flops in order to RESET them. The output of the first flip-flop is connected to the input of the next flip flop and so on. All these flip-flops are synchronous with each other since the same clock signal is applied to each flip-flop.



The above circuit is an example of a shift right register, taking the serial data input from the left side of the flip-flop and producing a parallel output. They are used in communication lines where demultiplexing of a data line into several parallel lines is required because the main use of the SIPO register is to convert serial data into parallel data.

Parallel-In Serial-Out Shift Register (PISO)

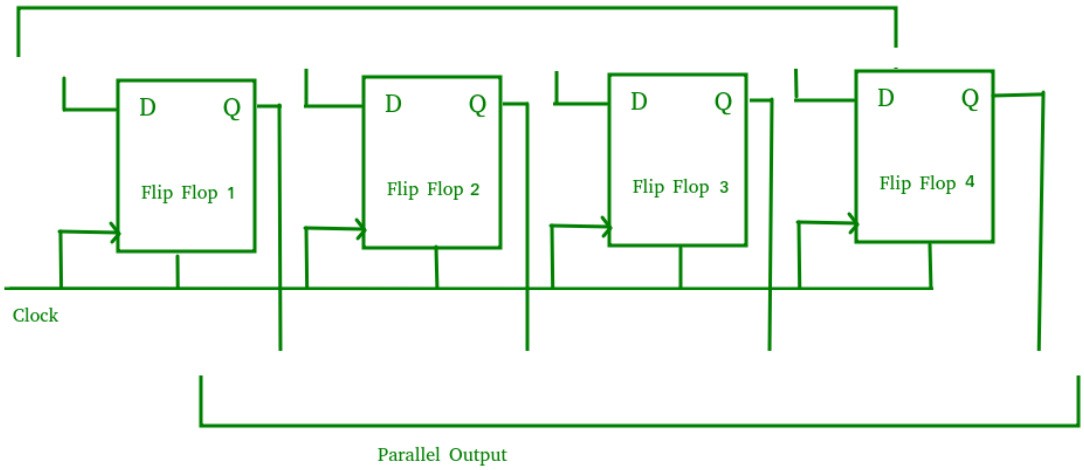
The shift register, which allows parallel input (data is given separately to each flip flop and in a simultaneous manner) and produces a serial output is known as a Parallel-In Serial-Out shift register. The logic circuit given below shows a parallel-in-serial-out shift register. The circuit consists of four D flip-flops which are connected. The clock input is directly connected to all the flip-flops but the input data is connected individually to each flip-flop through a multiplexer at the input of every flip-flop. The output of the previous flip-flop and parallel data input are connected to the input of the MUX and the output of MUX is connected to the next flip-flop. All these flip-flops are synchronous with each other since the same clock signal is applied to each flip-flop.



A Parallel in Serial Out (PISO) shift register is used to convert parallel data to serial data.

Parallel-In Parallel-Out Shift Register (PIPO)

The shift register, which allows parallel input (data is given separately to each flip flop and in a simultaneous manner) and also produces a parallel output is known as Parallel-In parallel-Out shift register. The logic circuit given below shows a parallel-in-parallel-out shift register. The circuit consists of four D flip-flops which are connected. The clear (CLR) signal and clock signals are connected to all 4 flip-flops. In this type of register, there are no interconnections between the individual flip-flops since no serial shifting of the data is required. Data is given as input separately for each flip flop and in the same way, output is also collected individually from each flip flop.



A Parallel in Parallel out (PIPO) shift register is used as a temporary storage device and like SISO Shift register it acts as a delay element.

Summary

- SR Flip-Flop: Simple set-reset latch, but has an invalid state.

- D Flip-Flop: Stores the value of D on the rising edge of the clock.

- JK Flip-Flop: Versatile flip-flop that can perform set, reset, and toggle operations.

- T Flip-Flop: Simplified version of the JK flip-flop, primarily used for toggling.

Reference:

1. Jan Friso Groote • Rolf Morel • Julien Schmaltz • Adam Watkins "Logic Gates, Circuits, Processors, Compilers and Computers".
2. Electronics Tutorials.
3. GeeksforGeeks.
4. Jean-Pierre Deschamps , Elena Valderrama , Lluís Terés "Digital Systems".
5. Thomas L. Floyd "Digital Fundamentals".