Finding your Way Around
Smart Phone / Tablet Users See
Alternative Navigation
Below

Boolean Logic and Digital Circuits

Modern digital computers are built from digital logic circuits whose basic building blocks are logic gates, each of which is designed t logical function. Information is held in data "words", representing data or instructions, made up from strings of individual "bits" with b These values are analogous to Boolean logic propositions and the statements or conclusions derived from them which were define Boolean algebra is the tool used to design combinations of gates to implement more complex functions such as mathematical opera and data storage.

Boolean Algebra

Boolean algebra is based upon a two-valued, or binary scheme. The two values may be expressed in many ways, such as true or f "off". It is this property which was recognised and developed by Claude Shannon in 1937 which makes it so useful for implementing means of electronic circuits. For example, logic 1 and logic 0 might be implemented as two different voltage levels in a circuit, or by by the presence or absence of a current in the circuit.

Notation

The engineering application of Boole's logic uses a simplified version of the original notation as follows,

- **A logical OR** is equivalent to Boolean addition and is represented by a plus **+** sign as in **A+B ≡A OR B**.
  It can represent *parallel* switch contacts.
- **A logical AND** is equivalent to Boolean multiplication which is represented by a dot **.** sign as in **A.B ≡ A AND B**.
  It can represent *series* switch contacts.
  Note that it has become conventional to drop the dot **.** sign (AND symbol) so that **A.B** is written as **AB**.
- **A logical NOT** is equivalent to Boolean complementation or negation and is represented by an overbar ‾ above the releva **NOT A.**
  It can represent *normally closed* switch contacts.
- **The Exclusive OR** or **logical XOR** was not an original Boolean operator and has its own special symbol **⊕** as in **A⊕B ≡ $\overline{A}$** **A or B** is true but **NOT** both.

Boolean Laws

Duality

Note that every law has two expressions, (a) and (b) which are duals of each other. Duality means

- Changing every OR (+) operation of the expression to an AND (**.**) and every AND (**.**) to an OR (+).
- Changing all the 0 elements to 1's and vice-versa.

Commutative Law

(a) $A + B = B + A$
(b) $A\,B = B\,A$

Associate Law

(a) $(A + B) + C = A + (B + C)$
(b) $(A\,B)\,C = A\,(B\,C)$

Distributive Law

(a) $A\,(B + C) = A\,B + A\,C$
(b) $A + (B\,C) = (A + B)\,(A + C)$

Identity Laws

(a) $A + A = A$
(b) $A\,A = A$

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

Redundance Laws

(a) $A + A\,B = A$
(b) $A\,(A + B) = A$

(a) $0 + A = A$
(b) $0\,A = 0$

(a) $1 + A = 1$
(b) $1\,A = A$

(a) $A+\overline{A} = 1$
(b) $A\,\overline{A} = 0$

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

Involution Law
(a) $\overline{\overline{A}} = A$

De Morgan's Theorem
(a) $\overline{A+B} = \overline{A}\,\overline{B}$ (Breaking the Overbar changes the OR to an AND)
(b) $\overline{AB} = \overline{A} + \overline{B}$ (Breaking the Overbar changes the AND to an OR)
Note: $\overline{A}\,\overline{B}$ is not the same as $\overline{AB}$

In addition to the above Boolean algebra, digital logic gates have been developed to represent Exclusive OR and Exclusive NOR e: original range of Boolean laws. See exclusive OR expressions below.

Logic Gates

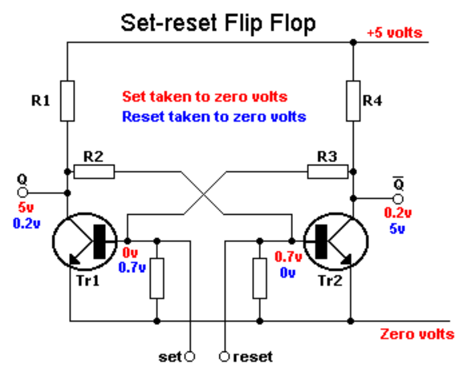
Logic gates may have two or more inputs and, except in some special cases, they have a single output. The status of the input and only be in one of the two binary conditions, either low (0) or high (1), represented by two different voltage levels, typically 0 volts for 5 volts for logic 1, depending on the semiconductor technology used. Logic gates also require a power supply.

The Transistor as a Switch

Electronic gates are generally constructed from transistor circuits which depend for their operation on the use of the transistor as a amplifier for which it was originally invented. With no voltage on the base, there is no current through the transistor which is thus sw (collector) voltage will be high. When a "high" voltage is applied to the base the transistor is switched on and output (collector) volta more on the page about semiconductors.

An early version of a bi-stable switching circuit was the 1919 Eccles and Jordan flip-flop based on valves (vacuum tubes). The later transistor version was one of the first electronic circuits to be implemented as an Integrated Circuit by Robert Noyce in 1959.
Flip-flops rely on the concept of feedback in which the output of a circuit is fed back into the input such that when the input is high, the output is low and vice versa.

See below an example of transistor switches used in the electronic circuit used to implement a three input NOR gate

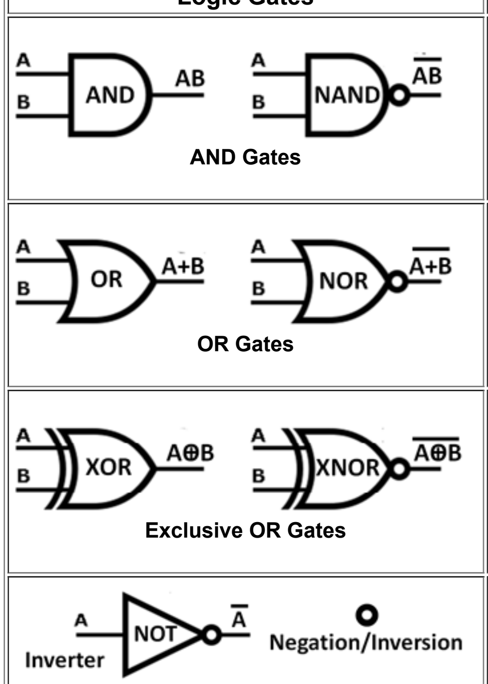


Logic Gates and Truth Tables

Logic circuit truth tables show the status of the output terminal or terminals of logic gates and logic circuits for all the possible input combinations**.** The gate or circuit's input states are shown in the left columns of the table while the corresponding output states are shown in the right columns.

The tables opposite show the range of common logic gates with their corresponding truth tables.

Logic Gates / **Truth Tables**



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

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

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

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

XOR and XNOR Gates

The **Exclusive OR** (XOR) gate with inputs **A** and **B** implements the logical expression $A\oplus B = \overline{A}\,B + A\,\overline{B}$

- When both the inputs are different, then output becomes high or logic 1.
- When both the inputs are same, then output becomes low or logic 0.

[Electricity from Fossil Fuels]
[Electrochemical Energy Generation]
[Electromagnetic Radiation - Radio Waves]
[Electropaedia Technology Search Engine]
[Energy Efficiency]
[Energy from Coal?] (Humour)
[Energy Resources]
[Energy Conversion and Heat Engines]
[Engineering Harmony] (Humour)
[Enigma's Cryptic Secrets]
[Feedback Form]
[Flow Batteries]
[Free Report Request]
[Frequently Asked Questions - FAQ]
[Fuel Cell Comparison Chart]
[Fuel Cells]
[Gas Turbine Power Generators]
[Generators]
[Geothermal Power Generation]
[Glossary]
[Going Solar - Grid Scale (PV)]
[Graphene]
[Grätzel Cell]
[Grid Scale Energy Storage Systems ESS]
[Hall of Fame]
[High Power Batteries]
[High Temperature Batteries]
[History of Technology]
[Historical Themes and Characters]
[Home Page]
[Homebrew Battery]
[How to Specify Batteries]
[Hybrid Power Generation Plants]
[Hydroelectric Power Generation]
[Hydrogen Power]
[Instructions for Using Batteries]
[Lead Acid Batteries]
[Leclanché Cells]
[Legal Statement]
[Liquid Metal Batteries]
[Lithium Battery Shipping Regulations]
[Lithium Cell Failures]
[Lithium Primary Batteries]

The **Exclusive NOR** (XNOR) gate with inputs **A** and **B** implements the logical expression $\overline{A \oplus B} = \overline{\bar{A}B + A\bar{B}} = AB + \bar{A}\,\bar{B}$

- When both the inputs are same, then output becomes high or logic 1.
- When both the inputs are different, then output becomes low or logic 0.

Exclusive OR gates are commonly used for comparisons and parity checks.

The Vernam Cipher

The Vernam cipher is a special application of XOR logic. Also called **Modulo 2 Addition,** it is similar to a digital adder except that the carry digits are ignored. An important cryptography tool, its special property is that a plaintext message string can be enciphered by XORing it with a random symbol scrambler string or key of the same length to create truly unbreakable ciphertext. The ciphertext can however be deciphered directly by XORing it with the original scrambler key.

Vernam used the five bit [Baudot code] to represent each character with the original notation of **+** and **-** to represent the logic states, rather than the more familiar 1 and **0** used today. In the example opposite:

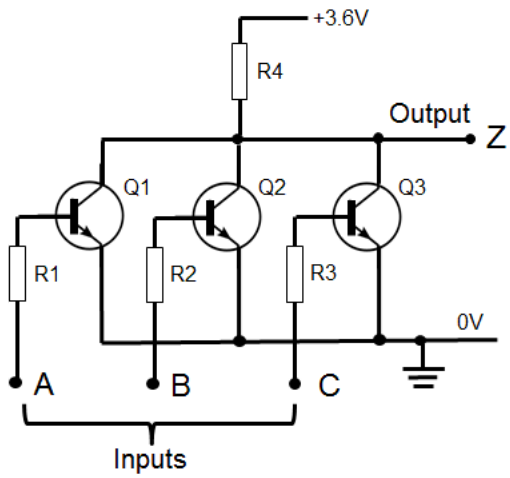
- The plaintext letter A is enciphered by adding a random scrambler letter, for example letter B. This generates the Baudot code for ciphertext G.
- By adding the same scrambler letter B to the ciphertext G the original plaintext letter is regenerated.

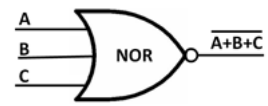
Truth Table			
Inputs		Output / Input	Output
Plaintext	Scrambler	Ciphertext	Plaintext
A	B	$A \oplus B$ = G	$G \oplus B$=A
+	+	-	+
+	-	+	+
-	-	-	-
-	+	+	-
-	+	+	-

Three Input NOR Gate

The diagram opposite is an example of a three input NOR gate showing the electronic circuit from which the gate is constructed together with the circuit symbol for the gate and the truth table associated with the gate.

Applying a logic 1 to any of the input terminals A, B or C cause a current to flow through the load resistor R4 which in turn causes the voltage on the output terminal Z to fall to logic level 0.
Only when all the input terminals are set to logic 0 will the current through the load resistor be cut off and the voltage on the output terminal will rise to logic level 1.
Thus there can only be a logic 1 output if neither, A nor B nor C are set to logic 1.





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

Truth Table			
A	B	C	$\overline{A+B+C}$
0	0	0	1
0	1	0	0
0	0	1	0
0	1	1	0
1	0	0	0
1	1	0	0
1	0	1	0
1	1	1	0

Digital Logic Circuits

Boolean logic is used to design complex digital circuits to perform a wide variety of logical functions. There is however often more tl implement a logic circuit by using alternative types of gates. Some examples follow.

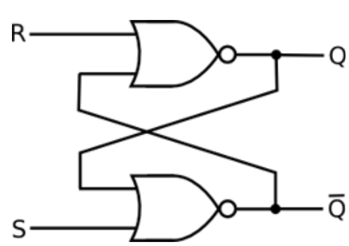
Set-Reset Flip-Flops and Latches

The Set-Reset flip-flop constructed from two cross connected, two input, NOR gates is one of the fundamental digital logic circuits. It is a bi-stable circuit which can store a single data bit in the form of a binary zero or a binary one and is used as a memory device or a latch.

Applying a logic1 to the Set terminal **S** stores a 1 and sets the output terminal **Q** to logic 1.
Applying a logic 1 to the Reset terminal **R** clears the memory, storing a 0 instead and sets **Q** to logic 0.
$\bar{Q}$ is the inverse or complement of **Q**

Flip-flops or latches can also be constructed from NAND gates with similar cross connections.

Set-Reset Flip-Flop with NOR Gates



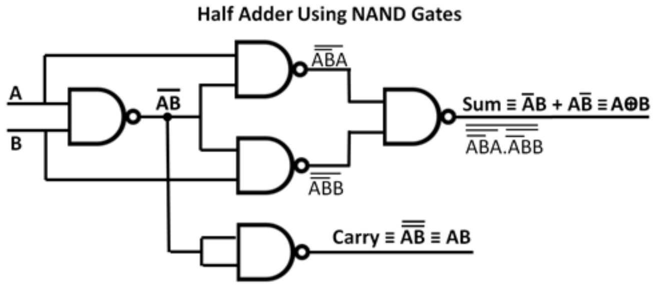
Truth Table				
Set/Reset		Output		
S	R	Q	$\bar{Q}$	Result
0	0			No change
0	1	0	1	Latch Reset
1	0	1	0	Latch Set
1	1			Not allowed

Registers are common storage devices providing temporary storage of multi-bit data words such as 4, 8 or 16 bit words. They are of flip-flops each storing a single bit of information so that *n* flip-flops are used to store an *n bit* word.

Adders

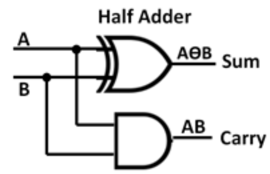
Lithium Secondary Batteries
Low Power Batteries
Magneto-Hydro-Dynamic (MHD) Electricity Generation
Motor Controls
MSDS - Typical Material Safety Data Sheet (PDF)
Nickel Cadmium (NiCad) Batteries
Nickel Hydrogen Batteries
Nickel Iron (NiFe) Batteries
Nickel Metal Hydride (NiMH) Batteries
Nickel Zinc Batteries
Nuclear Power - The Practice
Nuclear Power - The Theory
Other Galvanic Cell Chemistries
Page Quality Rating
Particle Physics - The Standard Model
Piston Engined Power Generators
Primary Batteries
Privacy Promise
Recycling
Redox Batteries
Reference Books
Reserve Batteries
Rocket Science
Satellite Technologies
Secondary Batteries
Semiconductor Primer
Shocking Batteries
Silver Oxide / Silver Zinc Batteries
Site Map More Pages Here
Site Search Engine
Small Scale Electricity Generation
Software Configurable Battery
Solar Batteries
Solar Power Generation
Special Purpose Motors
Sponsors
State of Charge (SOC) Determination
State of Health (SOH) Determination
Steam Turbine Power Generators
Stirling Engine Power Generator
Supercapacitors
Technical Library - White Papers
Thermal Batteries

The circuit opposite is a single bit half adder built from five NAND gates.
Half adders have only two inputs for the two bits to be added and can not accept a carry bit from a previous stage. They do however have two outputs, one for the Sum and the other for the Carry output from the two bit addition.

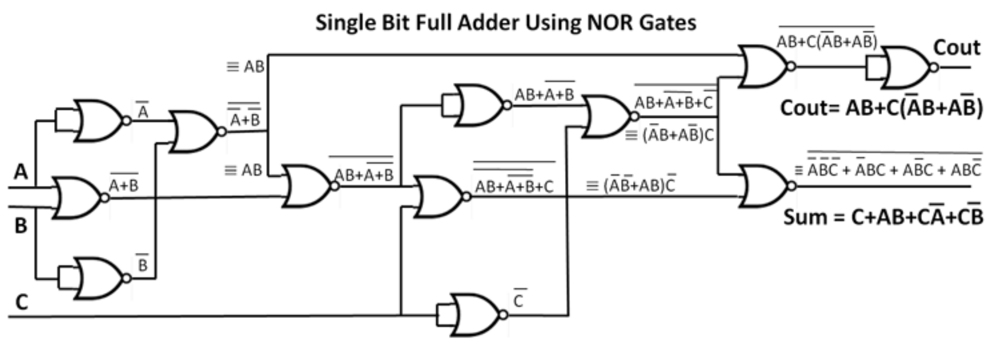


Truth Table			
A	B	Sum	Carry
0	0	0	0
1	0	1	0
0	1	1	0
1	1	0	1

The half adder opposite is another example of how logic functions can be implemented in different ways. In this case the adder circuit can be simplified by using only two gates, an AND gate and an XOR gate to perform the same half adder function as the circuit above.



Full adders are designed to accept a carry bit from a previous stage and hence have three inputs. The circuit below is a an exampl
constructed entirely from two input NOR gates. In this case it is essentially two, two-input, half adders in series with the input carry
adder and being added to the sum of the two input bits from the first adder, in the second adder.
Note that it takes 12 such gates simply to add two single bits plus any input carry bit from a previous addition stage and to provide t
representing the sum of the bits and any associated carry bit. A logic circuit designed to add two eight bit words will require eight tin



Truth Table				
Inputs			Outpu	
A	B	C_{in}	Sum	C
0	0	0	0	
0	0	1	1	
0	1	0	1	
0	1	1	0	
1	0	0	1	
1	0	1	0	
1	1	0	0	
1	1	1	1	

It may seem strange to use so many gates when the circuit could easily be implemented with fewer, more complex gates, but circui
above were used in the **Apollo Guidance Computer** which took the US astronauts to the Moon in 1969. All of its digital circuits we
NOR gates. This was because they needed highly reliable semiconductor components and at the time (1966) when the computer d
integrated circuit technology was still in its infancy and NASA wanted to limit the number of different components used to those whi
record. NOR gates were chosen because they were one of the very few options that met this requirement and because NOR gates
versatile than other available gates for building more complex functions.

Binary Arithmetic

A binary adder can be adapted to perform other arithmetic operations such a subtract, multiply and divide as well as other more cor
functions, avoiding the need for multiple specialist processors, by making use of the following principles of binary arithmetic.

- A 1's complement of a number is the same as the number with all its 1's changed to 0's and all its 0's changed to 1's. T
the adder to deal with negative numbers and subtraction.
- A 2's complement is the same as a 1's complement with the addition of an extra 1 to the least significant bit.
- The sign of a positive or negative binary number is changed by taking its 2's complement.
- A left shift of all the bits in a binary number by 1 position is the same as multiplying the number by 2 (binary 10).
- A right shift of all the bits by 1 position is the same as dividing the number by 2 (binary 10).
- A number can be raised by the *n* th power of 2 by adding *n* zeros to the end of the number.
- Multiplying a multi-bit number by 1 results in the same number. Multiplying it by 0 results in 0 and can be ignored. This
multiplication very simple.
- Dividing a multi-bit number 1 returns the same number.
- Dividing any number 0 is not permitted.
- Multiplication of two multi-bit numbers involves repeating the "left shift and multiply" operations *n* times in a loop, wher
bits in the multiplier.
- Division of two multi-bit numbers involves repeating the "right shift and divide" operations *m-n* times in a loop, where *m*
bits in the dividend (the number being divided) and *n* is the number of bits in the divisor.
- Subroutines carry out a series of instructions to perform arithmetic operations.

The following are three of the most common examples.

- **Subtraction**
  To subtract binary number B (the subtrahend) from binary number A (the minuend).
  - Add the 2's complement of B to A
  - If there is a final carry bit, discard it and the result is positive.
  - If B greater than A there will be no carry bit and the result will be the 2's complement of the sum and is negative.
  - (There's a slightly different way of making the subtraction with the 1's complemet of B instead of the 2's complem

- **Multiplication**
  This involves *n* steps where *n* is the number of bits in the multiplier.
  The first Step
  - Start with the least significant bit (LSB) of the multiplier.
  - If this multiplier bit is 1, add the multiplicand (the number being multiplied) to the product (the result of the multipl of the product will be zero)
  - If this multiplier bit is 0, ignore and move to the next step.

  In each subsequent step:
  - The multiplicand is shifted one place to the left.
  - The next bit of the multiplier is examined.
  - If this bit is 1, the shifted multiplicand is added to the current value of the product.
  - If this bit is 0, ignore and move to the next step.
  - Repeat the step *n* times in a loop (until the multiplicand has been multiplied by every bit of the multiplier).

- **Division**
  This operation involves *m-n* steps where *m* is the number of bits in the dividend (the number being divided) and *n* is the nu divisor.
  Note: Checks must be included and the operation set up to avoid the potential problem of dividing by 0.
  - Start by aligning the most significant bit (MSB) of the divisor directly below the MSB of the dividend.
  - Compare the *n* bits of the divisor the corresponding *n* bits of the dividend directly above.
  - If the divisor is larger than the dividend, set 0 as the first bit of the quotient (the result of the division)
  - If the divisor is smaller than the dividend, subtract the divisor from the dividend to get new dividend/remainder an quotient to 1.
  - Move to the next step

  In each subsequent step:
  - Shift the divisor one place to the right and compare it with next *n* bits of the dividend
  - If the divisor is larger than the corresponding *n* bits of the dividend, set the next bit of the quotient to 0
  - If the divisor is smaller than the dividend, subtract the divisor from the dividend to get new dividend/remainder an quotient to 1.
  - Repeat the step *m-n* times in a loop (until the LSBs of the divisor and the dividend line up)
  - The result of the subtraction in the last step is the remaider of the division.

The subroutines include special check points to detect and insert extra instructions to deal with carry and borrow bits, overflows, ne remainders, divide by zero whether the divisor is greater than the dividend.
Subroutines similar to those above, in combination Boolean logic circuits, are used to enhance the capability of the computer's Arith to enable it to carry out many more complex functions.

See more about Computer Architecture

Floating Point Arithmetic

The computer's calculating unit has just two hardware tools in its tool bag, "*shift*" and "*add*" to carry out all its mathematical operatic function arithmetic operations, outlined in the foregoing section, were all carried out with these two tools and all concerned operatio However in practical applications, even the most simple mathematical calculations involve operating with decimal numbers.

Besides this, the computer may also be required to handle transcendental functions such as logarithms, exponential and trigonome not be represented by simple algebraic expressions and can thus not be processed by the computer's simple *shift* and *add* capabili functions can however be expanded and expressed as an infinite series of simpler algebraic expressions each of which can be pro basic operations but summing an infinite series is not practical. Some form of mathematical approximation is required. Fortunately, of the series usually, but not always, converge quickly down to very small values such that subsequent terms can safely be ignored affecting the accuracy of the result. Thus the transcendental function can be considered as a polynomial consisting of just the first f the series. See examples (below)

In addition to the problem of precision, there's also the problem of scale. It is difficult to manage both very large and very small num registers and communications buses of practical size. Some examples:

- The mass of an electron is about 0.000000000000000000000000000000091093836 kg. In scientific notation, this is written
- The Earth's mass is about 5972400000000000000000000 kg. In scientific notation, this is written 5.9724 x 10^{24} kg.
- The speed of light is about 300,000,000 m/s (3.0 x 10^8 m/s)
- Newton"s gravitational constant is about 0.0000000000667 Nm^2kg^{-2} (6.67 X 10^{-11} Nm^2kg^{-2})

These last two numbers may even appear in the same equation.
The size problem is even more difficult when the numbers are represented in their much longer binary form.
The use of conventional scientific notation solves part of this problem but introduces the need to manipulate the position of the deci

These problems of accuracy, precision and scale are solved by the use of floating point arithmetic which uses the more convenient calculation methods are still limited to the computer's basic *shift* and *add* capability. These basic operations can however be augme subroutines and extra logic and registers to store temporary or intermediate values. The following are some examples of floating pc First some definitions:

Number Definitions

- **Mantissa**
  Also called the **Significand,** contains *all* the number's digits. Negative significands represent negative numbers. (Beware may cause confusion because it can also refer to the fractional part of the common logarithm.)

- **Base**
  The reference number on which the exponent is based.
- **Exponent**
  It is the power to which the base is raised.
  It indicates where the decimal (or binary) point is placed relative to the beginning of the significand.
- **Radix**
  The binary equivalent to the decimal point in decimal numbers.

Example

In scientific notation the decimal number 345.6 can be represented by 3.456 X 10^2 where 3456 is the mantissa or significand, 10 is exponent.

Floating Point Number Benefits

- They can represent numbers of widely different magnitudes (Range is limited by the length of the exponent)
- They provide the same relative accuracy at all magnitudes (Accuracy is limited by the length of the significand)
- In calculations involving numbers with both very large and very small magnitudes they enable the accuracy of both to be p

Floating Point (FP) Operations

Floating point operations can be implemented by hardware (circuitry) or by software (program code). Software is however much slc two or three orders of magnitude.

Bit Shifting

Shifting the mantissa left by 1 bit decreases the exponent by 1 and moves the radix point right by one place.
Shifting the mantissa right by 1 bit increases the exponent by 1 and moved the radix left by one place.

Basic FP Algebraic Functions

- **FP Addition**
  - Align the decimal points by increasing or decreasing one of the exponents so that both exponents are the same. mantissa for the sum.
  - Change the corresponding mantissa accordingly.
  - Add the new mantissas to get a new mantissa for the sum.
- **FP Subtraction**
  - Align decimal points and change the mantissa as above
  - Subtract using 2's complement
- **FP Multiplication**
  - Multiply the mantissas to get the new mantissa
  - Add the exponents to get the new exponent
- **FP Division**
  - Divide the mantissas to get the new mantissa
  - Subtract the exponents to get the new exponent

Transcendental and Other Functions

It is possible, but not necessarily practical, to store all the required values of transcendental functions in look-up tables stored in the While this would be convenient, in many applications however, it would require impractically large memories. Instead the necessary calculated as required.
Estimating the value of a transcendental function involves setting up a loop to calculate the value of each significant term in the ser the values in a separate accumulator (taking the sign into account). The more the terms in the series, the better will be the accuracy time will be correspondingly longer due to the increased number of calculations.
It is often possible to represent the transcendental function using alternative mathematical approximation methods. As in the trade-accuracy above, the alternative methods also involve similar trade-offs. The Taylor series is a common mathematical expansion, th used to approximate the value of transcendental functions. Some typical examples using the Taylor series follow.
See more about the [Taylor series](#).
See also the [CORDIC](#) expansion (below).

- **Trigonometric Functions**
  - The following series are the Taylor expansions for the sine and cosine where the variable X is measured in radia

$$\sin(X) \approx X - \frac{X^3}{3!} + \frac{X^5}{5!} - \frac{X^7}{7!} + \ldots + (-1)^i \frac{X^{(2i+1)}}{(2i+1)!} + \ldots$$

$$\cos(X) \approx 1 - \frac{X^2}{2!} + \frac{X^4}{4!} - \frac{X^6}{6!} + \ldots + (-1)^i \frac{X^{(2i)}}{(2i)!} + \ldots$$

Note that these two series involve both positive an negative terms and the denominators involve increasing facto the terms will converge very quickly to small magnitudes.
Note also that the computer does not usually calculate the numerical values of the factorials which are instead re tables in the memory.

An alternative to the Taylor series for estimating the value of the sine function is the [Hastings approximation](#) whic faster and only slightly less accurate.
The function is divided into a small number of intervals and, in each of these, a straight line approximation is use straight line segments have denominators which are powers of 2, and so the calculation does not need floating p

division operations. This was the algorithm used for trigonometric calculations in the Apollo 11 Guidance Comput astronauts to the Moon.

- **Logarithms**
  - The following is the Taylor expansion for the natural logarithm (**ln**).

$$\textbf{ln}(Z) \approx \frac{(Z\text{-}1)^1}{1} - \frac{(Z\text{-}1)^2}{2} + \frac{(Z\text{-}1)^3}{3} - \frac{(Z\text{-}1)^4}{4} + \ldots + (\text{-}1)^i \frac{(Z\text{-}1)^i}{(i)} + \ldots$$

This is valid for any real number Z that satisfies $0 < Z < 2$.

- **Exponential**
  - This is the Taylor expansion for the exponential ($\mathbf{e}^Z$)

$$\exp(Z) \approx 1 + Z + \frac{Z^2}{2!} + \frac{Z^3}{3!} + \frac{Z^4}{4!} + \frac{Z^5}{5!} + \ldots + \frac{Z^i}{(i)!} + \ldots$$

Note that all the terms in this series are positive and the numerators increase more quickly than the denominator not converge but expands.

- **Square Root**

  The square root is not a transcendental function. Numerous ways for calculating square roots have been developed since was proposed by the Greek mathematician Hero of Alexandria in the first century A.D.

  - **Iterative Method**

    The simple method based on "guess, check and refine" has been used for many years. It works as follows where root √x

    1. Guess an answer *"a"* between 0 and x.
    2. Calculate a^2
    3. Find the error E in the answer $E = x\text{-}a^2$
    4. If E is a positive number, a is too small.
       If E is a negative number, a is too big.
    5. If the magnitude of E is sufficiently small, a = √x
    6. If the error E is too big modify *a,* return to step 2 and repeat.
       The magnitude of E will progressively decrease until the desired accuracy is reached.

    This method was designed for use with decimal numbers. It is not so convenient with binary numbers and can le number of loops to converge on an acceptable answer. Extra steps in this simple program can improve this.

  - **Direct Method**

    The square root can be calculated directly by using the properties of natural or base 10 logarithms but the logarit transcendental functions and the values must first be extracted from a suitable approximation routine such as the The method depends on the following properties of the natural logarithm (**ln**):

    - $\textbf{ln}\ X^n = n\ \textbf{ln}\ X$
      and
    - $\mathbf{e}^{\textbf{ln}\ X} = X$

    The answer is given by: $\sqrt{x} = e^{(\textbf{ln}\ X)/2}$ using the natural logarithm (**ln**)

    Alternatively, using the base 10 logarithm (**log**), the answer is given by: $\sqrt{x} = 10\ e^{(\textbf{log}\ X)/2}$
    (Dividing the exponent by 2 generates a square root, the same as with logarithmic tables).
    Pocket calculators typically use this method of calculating square roots.

CORDIC Approximation Algorithms

The CORDIC algorithm is an iterative method for calculating transcendental functions using only binary *shift* and *add* operations.

Using the example of calculating the value of a tangent, X/Y = tan Θ, expressed in the form $\tan(2^{-n})$
A vector from the origin (zero) is rotated in a series of "*n*" small steps δΘ so that the sum of all the steps is equal to Θ. By accumula changes in the values of its orthogonal coordinates δX and δY at each step, the values of X and Y and hence the tangent can be ca Described by J.E. Volder in 1959, the CORDIC algorithm was used in the first scientific hand-held calculator (the HP-35) and variat concept have since been developed and refined for approximating a wide variety of transcendental functions.