

Basic Hardware-Table of Contents

Topic	Page
Table of Contents	1
Working With the Binary Number System	2
The Binary Number System	2
Bits and Bytes	3
Converting Binary to Decimal	3
Converting Decimal to Binary	4
Binary Addition	4-5
Binary Subtraction	5
Introduction to Logic Gates	6
Basic Logic Gates	7
The AND Gate	7
The OR Gate	7-8
The NOT Gate	8
Other Logic Gates	8
The NAND Gate	9
The NOR Gate	10
The XOR Gate	11
Inside Gates	12
The NOT Gate	12
The AND Gate	12-13
The OR Gate	13
Binary Adders	13
The Binary Half-Adder	14
The Binary Full-Adder	15
Parallel Binary Adder	16
Boolean Algebra	17
Logic Gates and their Boolean Expressions	18
Making a Truth Table with an Equation	19

Working with the Binary Number System

We are all familiar with the decimal system. We utilize ten digits in different combinations to represent numbers. These numerals, of course, are 0,1,2,3,4,5,6,7,8 and 9. The computer, on the other hand, uses the binary system. This system is much simpler than our familiar decimal system. Instead of utilizing ten digits, it only uses 0 and 1.

The four basic arithmetic operations (addition, subtraction, multiplication and division) can be reduced to two - addition and subtraction. Multiplication is really repeated addition, while division is really repeated subtraction. This fact makes it possible to handle these operations in binary form.

The Binary Number System

In order to convert a decimal number into a binary number, we have to keep a few things in mind. For example, in the decimal system, the number 151 really means this:

Hundreds	Tens	Units
1	5	1

Thus, the number 150 can also be represented as $(1 * 10^2) + (5 * 10^1) + (1 * 10^0)$. The 1 is in the hundreds place. The five is in the tens place, and the zero is in the units place. Each position in the decimal system stands for a value multiplied by a different power of ten. This is why the decimal system is sometimes referred to as the **base ten** system. The numbers 10^2 , 10^1 and 10^0 represent the place values of the digits- the numbers by which the digits are multiplied. Here is a table to show you what this means, using the number 150 as an example:

	Hundreds	Tens	Units
Digit	1	5	1
Meaning	$1 * 10^2$	$5 * 10^1$	$1 * 10^0$
Value	100	50	1

Similarly, the binary system can be represented in exactly the same way, except, the place values are the powers of two. Thus, the binary system is sometimes called the **base two** system. Since binary can only have the 1 and 0 digits, we will use the number 110101 as an example:

Digit	1	1	0	1	0	1
Meaning	$1 * 2^5$	$1 * 2^4$	$0 * 2^3$	$1 * 2^2$	$0 * 2^1$	$1 * 2^0$

Value	32	16	0	4	0	1
-------	----	----	---	---	---	---

Bits and Bytes

Although a bit is the smallest form of data that a computer, there are names for larger amounts of bits. For example, 8 bits is considered to be 1 byte. With this information in mind, here is a table which shows you the different data size names used by the computer, and how many bits each is:

Name	Abbrivation	Number of Bytes (as a power)	Number of Bytes
Byte	-	2^0	1
KiloByte	KB	2^{10}	1,024
MegaByte	MB	2^{20}	1,048,576
GigaByte	GB	2^{30}	1,073,741,824
TeraByte	TB	2^{40}	1,099,511,627,776
PetaByte	PB	2^{50}	1,125,899,906,842,624
ExaByte	EB	2^{60}	1,152,921,504,606,846,976
ZettaByte	ZB	2^{70}	1,180,591,620,717,411,303,424
YottaByte	YB	2^{80}	1,208,925,819,614,629,174,706,176

As you can see, each step goes up by 2^{10} . Note, you only need to be acquainted with Byte up to GigaByte, because the other sizes are not used yet (computers have not advanced enough to be able to use them), although there are a few hard drives that have a storage of a TeraByte or more.

Converting Binary to Decimal

Converting a binary number into a decimal number is simple. To figure out what this number stands for, we have to add up numbers at each place value of the binary number. Let us use the binary number 110101 as an example:

1 * 32 =	32
1 * 16 =	16
0 * 8 =	0
1 * 4 =	4
0 * 2 =	0
1 * 1 =	1
Total	53

Answer - 110101 in binary is 53 in decimal.

Converting Decimal to Binary

Converting a decimal number into binary is also very simple. It involves a continuous division of the decimal number and keeping track of the remainder. Let us use the decimal number 220 as an example:

Division	Remainder
$220 / 2 = 110$	0- Least Significant Bit (LSB)
$110 / 2 = 55$	0
$55 / 2 = 27$	1
$27 / 2 = 13$	1
$13 / 2 = 6$	1
$6 / 2 = 3$	0
$3 / 2 = 1$	1
$1 / 2 = 0$	1 - Most Significant Bit (MSB)

Answer - 220 in decimal is 11011100 in binary.

The Most Significant Bit is exactly what the name infers, the most significant. This means that it has the largest value out of all the bits. Since the MSB is the last on the table, we have to place the last bit from the table first, and work our way up. Therefore the decimal number 220 is 11011100 in decimal, not 00111011.

Binary Addition

Now that we know how to convert binary to decimal and vice versa, it is time to learn how to add two binary numbers together. Binary addition is done the same way as decimal addition, which is using the carry method. Here is an example of using the carry method to add up two decimal numbers:

$$\begin{array}{r}
 1 \\
 3248 \\
 + 1426 \\
 \hline
 4674
 \end{array}$$

As you can see, when 8 and 6 were added, 4 was recorded as the units digit for the answer, and a 1 was carried over to the tens column. Here is an example of how binary numbers are added.

For the binary addition example, the binary numbers 1001 and 0101 will be added:

$$\begin{array}{r}
 1 \\
 1001 \\
 + 0101 \\
 \hline
 1110
 \end{array}$$

We first add the two least significant bits from the two numbers to be added, 1 and 1. Recall that binary can only have 0's and 1's. $1 + 1$ would be recorded as 0 in the 2^0 column, and a 1 would have to be carried to the next 2^1 column. We then add $0 + 0 +$ the carry of 1, and get an answer of 1 in the 2^1 column. Next we add the 0 and the 1, and get an answer of 1 in the 2^2 . Finally, we add up the 1 and 0 and get an answer of 1 in the 2^3 . The final answer is then revealed to be 1110. As you can see, binary addition is very easy for people who know how to add larger decimal numbers together.

Binary Subtraction

There are a few ways to subtract binary numbers, but to avoid any confusion, we will use the borrow method already familiar to you. Let's recall decimal subtraction with an example:

$$\begin{array}{r}
 21 \\
 3248 \\
 - 1426 \\
 \hline
 1822
 \end{array}$$

When we subtract the numbers in the units column, as well as the tens column, there is no problem. When we get to the 100's column, notice how the number on the top is smaller than the number on the bottom. To continue with the subtraction, we borrow from the 1000's column, and place it in the 100's column. Now, we get $12 - 4$ which is 8. Finally, the 3 is changed to the two in the 1000's column because of the borrow, and $2 - 1 = 1$, leaving us with the final answer of 1822.

As an example of binary subtraction, we will subtract 0101 from 1011:

$$\begin{array}{r}
 011 \\
 4001 \\
 - 0101 \\
 \hline
 0100
 \end{array}$$

In the 2^0 and 2^1 column, subtraction occurs the same way as it would in decimal subtraction. When we get to the 2^3 column, we have to borrow. The 0 in the 2^3 column is changed to two 1's because you need to borrow from 2^4 column, which is

worth twice as much as the 2^3 column. Because you borrowed a 1 from the 2^4 column you change it to a 0. The rest of the subtraction is simple, and done the same way it would be done in normal decimal subtraction.

Introduction to Logic Gates

In the English language, the term "Gate" is "a structure that can be swung, drawn, or lowered to block an entrance or a passageway". An example is a door on a fence, which can be opened to let people in, and closed to keep people out. In computers, logic gates have a similar application. A logic gate has one or more inputs (either 1 or 0), and one output. Logic gates are designed to allow the passage of binary signals through the gate for certain combinations of inputs. These gates might seem simple, primitive, and useless. Although one logic gate alone is almost useless, a combination of the gates working together can serve many purposes. Today's computers, which do very complex work, are made up of millions of these simple logic gates. There are three basic logic gates. They are the AND, OR and NOT gates. These gates are used in different combinations to make more complex gates such as the XOR. As you will find out in a moment, the three basic logic gates are very easy to understand. But before we learn about logic gates, there are some terms that we should know about:

1. **Truth Table:** A truth table is a table of combinations. It shows all the input possibilities and the outputs which result from these possibilities. For example, here is a truth table showing what will be the products when the number 2 is multiplied by the numbers from 1-5.

Numbers	Product
2 * 1	2
2 * 2	4
2 * 3	6
2 * 4	8
2 * 5	10

This is called a truth table because it is a table which has facts about the product of two and numbers from 1-5. A truth table for logic gates will be used to show you all the possible inputs for the gates, and the resulting outputs for each combination of inputs.

2. **Gate Symbols:** Gate symbols are used in diagrams for planning before computer circuitry is built. The gates do not really look like the symbols, the symbols are used to make the diagrams simpler and more comprehensive.

3. **More Things to Know About Binary:** From the previous lessons, we know that 1's and 0's are the two possibilities in binary. In the computer world, you will frequently hear "HIGH" used instead of 1, and "LOW" being used instead of 0.

4. Inputs and Outputs: The inputs and outputs for logic gates are represented by letters. Commonly, the output is represented by the letter Q or the letter Y. We will be using the letter Y for the output. The inputs are almost always represented by the first letters of the alphabet. A and B are used the most often, while C, D and other letters are used when the diagrams are more complex.

Basic Logic Gates

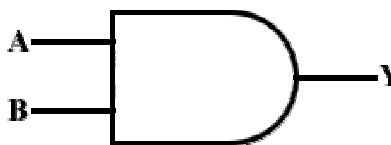
This will help you to understand how the three basic gates function. You will learn that their names resemble the type of operation that they were made to do. With this said, there are three basic gates. They are the AND Gate, OR Gate and the NOT Gate..

The AND Gate

The AND gate gives a HIGH output only when all of the inputs are HIGH. If any one of the inputs are LOW, the output is also LOW. This information about the gate can be represented in a truth table. Here is the AND gate truth table:

INPUT		OUTPUT
A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

The AND gate can have more than two inputs. A binary HIGH will only be produced when all inputs are HIGH. As well as all other gates, this gate has a symbol used to represent it in a diagram:



The OR Gate

The OR gate is a very simple gate. It delivers a HIGH output when any of the inputs are HIGH. The only way that an OR gate would produce a LOW output is if all of the inputs were LOW. With this said, here is the truth table and the symbol for the 2-input OR gate:



INPUT		OUTPUT
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

Just like the AND gate, the OR gate can have more than two inputs. The OR gate would work it's logic the same way, only one inputs would have to be HIGH in order for the output to be HIGH.

The NOT Gate

The simplest of the three basic gates is the NOT gate. It is commonly called an inverter. The purpose of the NOT gate is to take the input, change it to its binary opposite, and output the inverted input. Unlike the AND or the OR Gates, the NOT gate can only have one input. Here is the truth table and symbol for the NOT gate:



INPUT	OUTPUT
A	B
0	1
1	0

Other Logic Gates

All of the other gates are created using a combination of two or more of the AND, OR and/or NOT gates.

The NAND Gate

The NAND gate is just like the basic AND gate. The only difference is that the NAND gate inverts the final output. Here is the truth table for the NAND GATE:

INPUT		OUTPUT
A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

Although this gate seems fairly simple, it is more complex than any of the basic gates. As you can see by the truth table, the outputs are the exact opposite of the AND gate. This is achieved by using a combination of two basic gates, the AND and the NOT. Since it uses a combination of two basic gates, it is not considered a basic gate. Here is how the components of the NAND gate are arranged:



As you can see, the inputs go into an AND gate. The output of the AND gate is then inverted by the OR gate to achieve the result of the whole NAND gate. The NAND gate is really an AND gate with an inverted output.

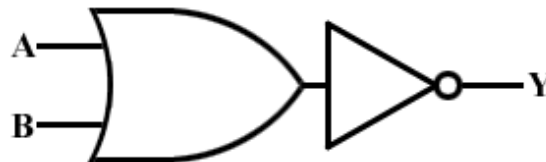
Instead of having to use the combination of the two gates in the above diagram as a symbol for the NAND gate, a simplified symbol is used:



Note that the NOT gate has been replaced with a small circle. This small circle is commonly used to replace a whole NAND gate when designing logic circuits.

The NOR Gate

The NOR gate is the exact opposite of the OR gate. The output is HIGH only when all of the inputs are LOW. When any of the inputs are HIGH, the output is always HIGH. The NOR gate is also constructed using two basic gates. This time the OR and NOT gate are employed to create the NOR gate. Here is a diagram of the NOR gate along with the truth table:



INPUT		OUTPUT
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

As you can see, the NOR gate is the same as the NAND gate except it uses an OR gate in place of the AND gate. To make diagrams simpler, the same thing is done with the NOR gate as was done with the NAND gate:

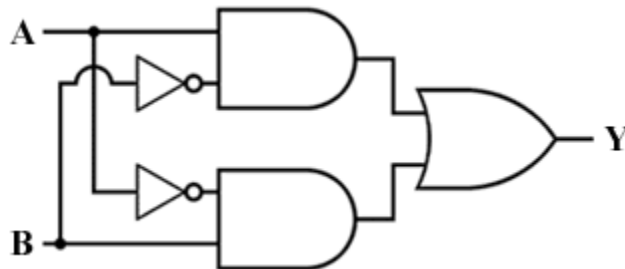


The XOR Gate

The EXCLUSIVE OR (XOR) gate is very commonly used in arithmetic circuits. The XOR gate compares the inputs. If both inputs (there can only be two inputs for this gate) are the same, the output is LOW. In contrast, if the inputs differ, then the output is HIGH. Here is the truth table for the XOR gate:

INPUT		OUTPUT
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

The XOR gate is most complex of all gates; it requires two AND gates, two NOT gates, as well as an OR gate. Here is a diagram which shows how the five gates are arranged to create the XOR gate:



The A and B inputs are both split into two to allow them to be the inputs for two gates at the same time. The two inputs that were just created from the split are inverted. There are two AND gates. The two original inputs go to the different AND gates. The inverted input of A becomes the second input of the AND gate in which B is the other input. Likewise, the inverted input of B becomes the second input of the AND gate in which A is the other input. The outputs of the AND gates become the inputs of the OR gate. If either of the inputs is HIGH, then the output of the whole XOR gate is HIGH. Otherwise, the output of the XOR gate is LOW.

Just like the two other gates in this section, the XOR gate is simplified in diagrams. Here is the symbol used to represent the XOR gate:

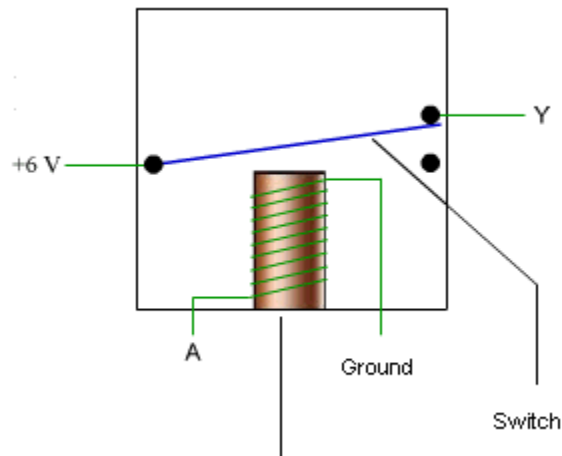


Inside Gates

This section will take us inside basic gates like the AND, OR, and NOT to see how they work mechanically.

The NOT Gate

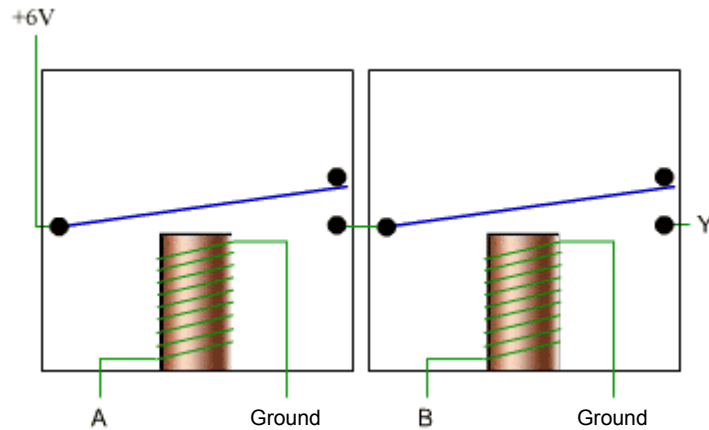
Before we understand how the AND and OR gates work we must first understand how the most basic NOT gate works.



Electromagnet - A conductive metal with a coil wrapped around it. When power passes through the coil, the metal magnetizes and attracts the switch.

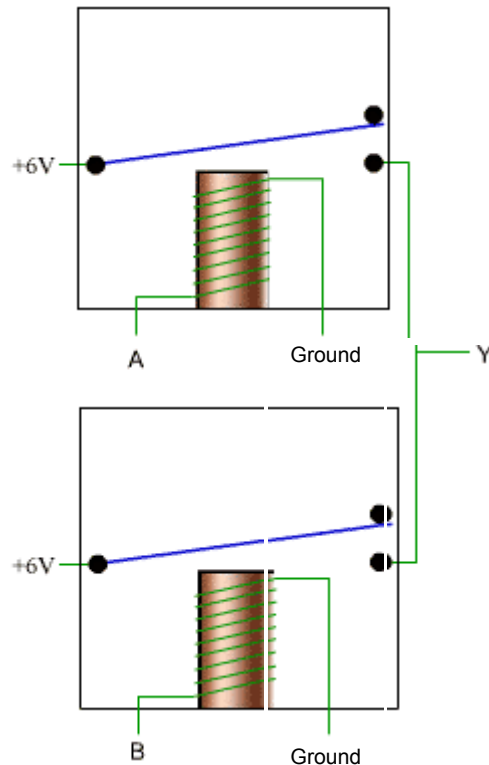
When A is at 0 Volts, the switch lets the current through, and the output at Y is 1. When A is at + 6 Volts, which gives an input of 1, the electromagnet starts, pulling the switch and breaking the flow of the current, giving an output of 0.

The AND Gate



When both A and B input are LOW, neither of the electromagnets are powered, and the output (Y) is LOW. When B is HIGH, its switch is set to allow the current through. But the current must pass through the first switch first before it goes through the second. Since it cannot Y stays LOW. When A is HIGH, its switch is set to allow the current through. But the current must pass through the second switch after it goes through the first. Since it cannot Y stays LOW. When A and B are HIGH, both switches let current through, leaving the output at Y as HIGH.

The OR Gate



When both inputs (A and B) are LOW, the electromagnets stay off and the switch does not allow the current to flow to the output, giving Y a binary 0 output. If one of the inputs is LOW and one is high, one electromagnet goes on, causing one switch to

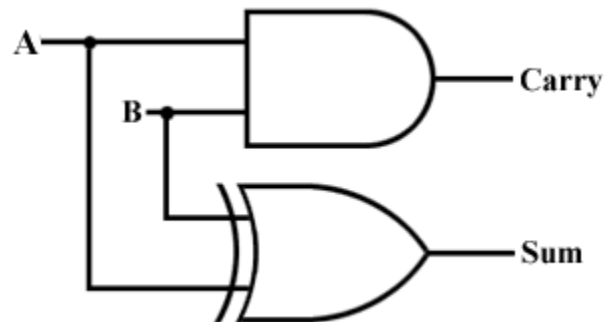
open. One is all that is needed for Y to have a binary 1 output. Finally, when both inputs are HIGH, Y is still HIGH because there are two sources of power and at least one is required for a HIGH output.

Binary Adders

By now, you have learned how to add binary numbers. You have also learned how the 6 common logic gates operate. Now it is time to learn how the logic gates are used to add binary numbers.

The Binary Half-Adder

You may be surprised to see how easy it is to create a logic circuit that adds binary numbers. The binary half-adder is perfect for adding 1-bit binary numbers. It is made out of one AND gate and one XOR gate. Here is how it is arranged in a diagram:



A and B are the two inputs. In this case they are the two bits that are to be added. The XOR gate is used to figure out the sum, while the AND gate is used to figure out the carry. Let us start with the XOR gate. A and B are inputted, and if the two bits differ, the output of the XOR gate is 1, which means that the sum is 1. Otherwise, the output of the XOR gate is 0. If you recall addition of bits, $0 + 0 = 0$, $0 + 1 = 1$, while $1 + 1 = 0$ with a carry of 1. This is where the AND gate comes in. If both A and B are 1, then the output for the AND gate, which represents the carry, is 1. Otherwise the carry is 0. This can be summarized in a truth table for the half-adder:

Input		Output	
A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

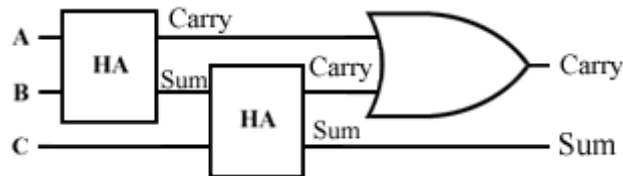
The Half-Adder has its own special symbol in diagrams in order to make them simpler. This is what it looks like:



Now that we know how the computer adds two digits at a time, it is time to learn how it adds two bits and a carry at the same time.

The Binary Full-Adder

The binary full-adder is very useful because it can add three digits at a time. If you recall from our binary addition section, there are sometimes carries that need to be added along with the two other digits. This requires the addition of three bits at a time, which is exactly what the full-adder is used for. Here is the logic diagram for the more complex full-adder, which employs two HA's (half-adders) as well as an OR gate:



As you can see, there are now three inputs: A, B and C. A and B are usually used for the two bits which would be normally added, while C is usually used as a carry from a lower place value. As you can see, inputs A and B go into a half-adder. The sum of the half-adder is sent as one of the inputs (along with input C) to the second half-adder. The sum of the second half-adder is the sum of all three inputs. If there was a carry from either or both of the half-adders, the Carry output for the full adder is HIGH. There are 8 possible combinations of inputs for the full-adder, all recorded on the full-adder truth table:

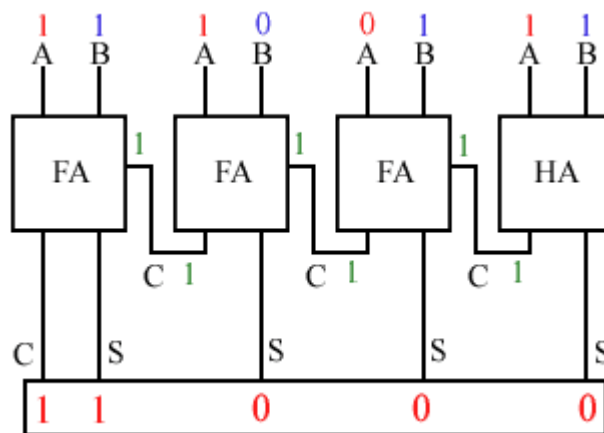
Input			Output	
A	B	C	Sum	Carry
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0

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

To simplify this diagram, a symbol is used for the full-adder. The symbol is identical to the half-adder symbol, except it bears the letters FA instead of the HA on it.

Parallel Binary Adder

The use of one half-adder or one full-adder alone are great for adding up two binary numbers with a length of one bit each, but what happens when the computer needs to add up two binary numbers with a longer length? Well, there are several ways of doing this. The fastest way by far is to use the Parallel Binary Adder. The parallel binary adder uses one half-adder, along with one or more full adders. The number of total adders needed depends on the length of the largest of the two binary numbers that are to be added. For example, if we were to add up the binary numbers 1011 and 1, we would need four adders in total, because the length of the larger number is four. Keeping this in mind, here is a demonstration of how a four-bit parallel binary adder works, using 1101 and 1011 as the two numbers to add:



Just like when we add without the computer, in the parallel binary adder, the computer adds from right to left. Here is a step by step list, showing you what happens in the parallel Binary Adder:

1. In the only half-adder, inputs of 1 and 1 give us 0 with a carry of 1.
2. In the first full-adder (going from right to left), the inputs of 1 and 0 plus the carry of 1 from the half-adder give us a 0 with a carry of 1.
3. In the second full adder, the inputs of 0 and 1 plus the carry of 1 from the previous full-adder give us a 0 with a carry of 1.
4. In the third and final full adder, the inputs of 1 and 1 plus the carry of 1 from the previous full-adder give us a 1 with a carry of 1.
5. Since there are no more numbers to add up, and there is still a carry of 1, the carry becomes the most significant bit.
6. The sum of 1101 and 1011 is 11000.

Boolean Algebra

As most people are well aware, mathematical expressions are used to make computations and other math problems simpler and smaller. Boolean Algebra is used to do the same thing, except it does it with logic circuits instead of other mathematical problems.

The first thing that needs to be known for Boolean algebra is the meaning of the various signs. There are three of these, all deriving from the three basic gates. They are AND, OR and NOT.

The AND operation is also known as conjunction. It gives the product of two binary bits. Using A and B as inputs, it would be written as AB or sometimes $A \cdot B$. This of course means that A is multiplied by B, which is exactly how an AND gate functions.

The OR operation is also known as disjunction. This operation gives the sum of two binary bits. Again, using A and B as inputs, it would be written as $A + B$, which is how an OR gate functions. NOTE, the "+" does necessarily mean the same thing that it does in normal mathematics we are all used to. In Boolean algebra, it stands for OR. For example, $1 + 1$ would equal to 1, not 10, the binary equivalent of two. If either of the inputs is 1, the output is 1. The NOT operation is also known as negation. This operation gives the opposite of a single term. For example, the negation of A is written as \bar{A} . While the negation of AB would be \overline{AB} . We can relate the AND, OR and NOT operations to the corresponding gates:

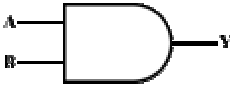

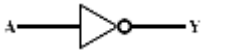
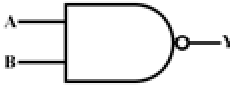


AND Truth Table	OR Truth Table	NOT Truth Table
$A \cdot B = Y$	$A + B = Y$	$\bar{A} = Y$
$0 \cdot 0 = 0$	$0 + 0 = 0$	0 negated = 1
$0 \cdot 1 = 0$	$0 + 1 = 1$	
$1 \cdot 0 = 0$	$1 + 0 = 1$	1 negated = 0

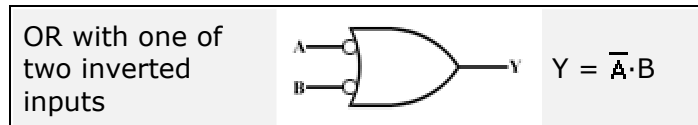
$1 \cdot 1 = 1$

$1 + 1 = 1$

Logic Gates and their Boolean Expressions

All of the logic gates have a Boolean expression. To show you how Boolean algebra works further, here are some examples of gates and their Boolean expressions:

Logic Gate Name	Symbol	Boolean Expression
AND		$Y = A \cdot B$
OR		$Y = A + B$
NOT		$Y = \bar{A}$
NAND		$Y = \overline{AB}$
NOR		$Y = \bar{A} + \bar{B}$
AND with one of two inverted inputs		$Y = \bar{A} + B$



Making a Truth Table with an Equation

When given an equation, it is easy to create a truth table for the equation. Here is an example of how it is done, with the expression $Y = \bar{A} \cdot B$. We plug in all the possible values for A and B, and record the values of Y:

1. A = 0, B = 0 $Y = \bar{A} \cdot B$ $Y = 1 \cdot 0$ Y = 0	2. A = 0, B = 1 $Y = \bar{A} \cdot B$ $Y = 1 \cdot 1$ Y = 1
3. A = 1, B = 0 $Y = \bar{A} \cdot B$ $Y = 0 \cdot 0$ Y = 0	4. A = 1, B = 1 $Y = \bar{A} \cdot B$ $Y = 0 \cdot 1$ Y = 0

Once this is done, We organize it in a truth table:

Input		Output
A	B	Y
0	0	0
0	1	1
1	0	0
1	1	0

We hope that you now have a greater understanding of the binary number system and the Boolean Logic Gates.