Matthew Ernst

ENGT279

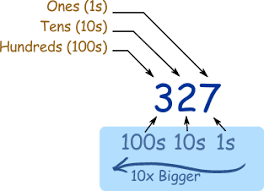
Dale DeRoche

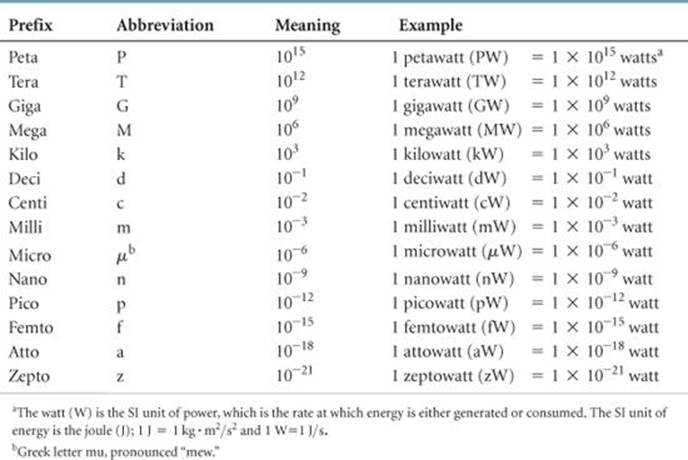
4.2.2018

Binary Numbers and Applications

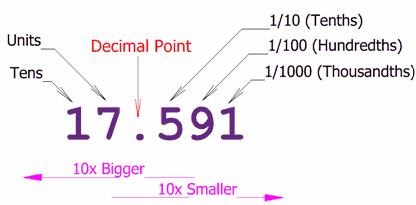
What is binary code? Many people think of 1s and 0s when the word binary comes to mind, but there’s so much more to binary than just 1s and 0s. First let’s start by talking about the decimal system and how binary relates to it. We will start with the decimal system because we are so familiar with it.

The decimal system is composed of 10 numerals or symbols. The symbols are 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9. Using this basic system we are able to express any quantity. It is the most widely used number system in the world. It is also a positional-value system in which the value of a digit depends on its position. Consider the example in the picture above, the number 327. 3 represents 3 hundreds, 2 represents 2 tens, and 7 represents 7 *units.* The 3 carries the most weight of the three digits and is therefore referred to as the most significant digit (MSD). The 7 carries the least weight and is the least significant digit (LSD). And then we are introduced to decimals in numbers. For example if we had a number like, say, 46.29, then that number would be 4 tens plus 6 units plus 2 tenths plus 9 hundredths. In summary, the decimal point is used to separate the integer and fractional parts of a number. More rigorously, the various positions relative to a decimal point carry weights that can be expressed in powers of 10. The table below illustrates different powers of 10 and the quantity they represent.





An example of this is shown below to further demonstrate the purpose of a decimal in a number. The number 17.591 has 1 ten, 7 units, 5 tenths, 9 hundredths, and 1 thousandth.

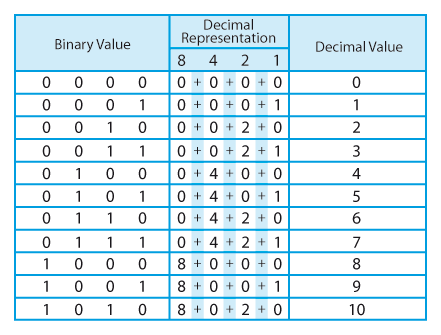


The decimal number system is pretty nifty for convenient applications, but it isn’t very useful in digital systems. It would be very difficult to design an electronic system that operates on 10 voltage levels. Instead, it is easier to design one that operates on only 2 levels, which are 1 and 0. This is where binary numbering comes in. The binary numbering system is a base-2 numbering system that is widely used in digital applications. Each digit in the binary system is represented as a power of 2. To convert from binary to decimal, you need to take the sum of the product of each digit and its positional value. For example, we will convert the number 100101 from binary to decimal. There are six total digits in this number, so we will start with:

(1 x 2^5) + (0 x 2^4) + (0 x 2^3) + (1 x 2^2) + (0 x 2^1) + (1 x 2^0) = 32 + 4 + 1 = 37.

If you are faced with a binary number with a decimal, then any number to the right of that decimal will be expressed with negative values. For example:

11001.011 = (1 x 2^4) + (1 x 2^3) + (0 x 2^2) + (0 x 2^1) + (1 x 2^0) + (0 x 2^-1) + (1 x 2^-2) + (1 x 2^-3) = 16 + 8 + 1 + 0.25 + 0.125 = 25.375. The table below show examples 1 through 10.



So now that we’ve talked about how to convert binary to decimal, you may wonder if there is a say to do it the other way around, decimal to binary. To do this, you would need to take a decimal number and repeatedly divide it by 2. When you are dividing, the binary number will write itself with the number of remainders in each dividend. To further illustrate this, let me use an example:

We have the number 47. We divide it by 2 and get 23 with 1 remainder. Then we divide 23 by 2 and get 11 with 1 remainder. Next we divide 11 by 2 and get 5 with 1 remainder. Then we divide 5 by 2 and get 2 with 1 remainder. Then we divide 2 by 2 and get 1 with no remainder. Finally, we divide 1 by 2 and get 0 with 1 remainder. It is important to keep in mind that converting from decimal to binary will result in the 1s and 0s lining up from right to left, instead of the usual left to right. So the final number for 47 is 101111. We’ll do one more example, this time with the number 138 with step-by-step division to make it clearer.

Dividing 138 by 2 gives us 69 with no remainder. = 0

Dividing 69 by 2 gives us 34 with 1 remainder. = 1

Dividing 34 by 2 gives us 17 with no remainder. = 0

Dividing 17 by 2 gives us 8 with 1 remainder. = 1

Dividing 8 by 2 gives us 4 with no remainder. = 0

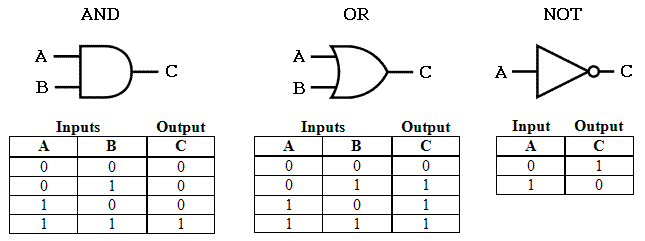
Dividing 4 by 2 gives us 2 with no remainder. = 0

Dividing 2 by 2 gives us 1 with no remainder. = 0

Dividing 1 by 2 gives us 0 with 1 remainder. = 1

Now read all the 1s and 0s backwards to give you the final number in binary. This gives us a binary number of 10001010.

Now that we have a good understanding of the binary number system, let’s talk about some real-life applications. Before we get into real life applications with binary, we need to have a decent idea of Boolean logic. Boolean logic is an algebraic system of logic that was first invented by George Boole in the 1930’s when he defined the algebraic system of logic. Thanks to Boolean algebra binary numbers have had several applications from storing millions of bits of data to recording HD movies and high-end audio. All computer languages and coding are based on the two digit binary number system. Taking data and depicting it with restrained bits of information makes up the digital encoding process. A binary line for every pixel in every image you see on your computer screen is used to encode these images. But it does not stop there. Binary codes are huge in computer programming and electrical engineering applications. In electrical and computer engineering, particularly Boolean algebra and logic, binary digits 1 and 0 are used to represent “true” and “false” (respectively) inputs and outputs. This leads us to truth tables and logic gates:



First and foremost, a truth table is a means for describing how a logic circuit’s output depends on the logic levels present at the circuit’s inputs. Truth tables count their input values in binary to make it easier for designers and programmers to decide exactly which inputs will affect the output. Like I said before, 1 means true, and 0 means false. For the Boolean logic gate AND, it gives that both inputs have to be true for the output to be true. So inputs A and B must be true for output C to be true. For Boolean logic gate OR, it gives that at least one input has to be true for the output to be true. So inputs A or B or both must be true for output C to be true. Boolean logic gate NOT simply inverts the input and makes the output the opposite, as shown above. There are even more Boolean logic gates like NAND, NOR, XOR, and XNOR, but all of these logic gates are obtained through a combination of the first basic three. Here is an example of using AND and OR gates in a logic circuit:



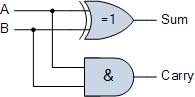
Figure 1- Lab 4 Schematic

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Simulated | | Test | |
|  | Open | Closed | Open | Closed |
| S1 | 0V | 5V | .0081V | 5.0523V |
| S2 | 0V | 5V | .0081V | 5.0523V |
| S3 | 0V | 5V | .0081V | 5.0523V |

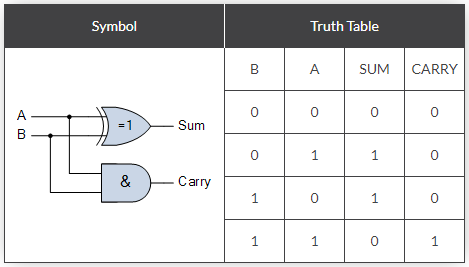
Table 1 (Simulation vs Test)

Above is a schematic of a circuit and a table of a lab using an AND gate and an OR gate. The purpose of the lab was to build and test the circuit using 3 10k ohm resistors to measure voltage levels. This is just one example of many that uses logic gates which were brought here by the binary number system. What are some other common applications of binary numbers? Let’s start by looking at a binary adder.

If you are an electrical or computer engineer, chances are you will be exposed to a binary adder. For those who are unfamiliar, a binary adder is a combinational logic circuit that can be constructed with just a few basic logic gates which allows it to add two or more binary numbers. A basic binary adder can be made with a standard AND and OR gate that will let us add two single bit binary numbers, A and B. Below is a diagram of a basic binary half adder:



The Sum output of the half adder represents the sums least significant bit. The Carry represents the carry output, which is the sums most significant bit. The circuit above is assigned with the function of only adding two binary bits. Here is a truth table of a half adder:



From the truth table shown above we see the sum output is the result of the XOR gate and the Carry-out is the result of the AND gate.

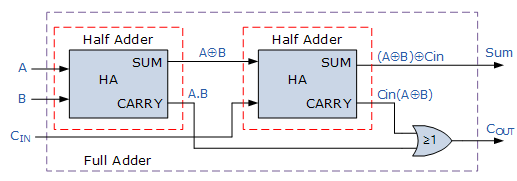
For the SUM:

SUM = A XOR B = A ⊕ B

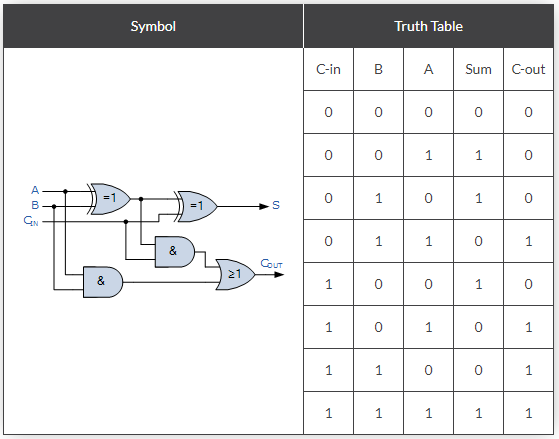
For the CARRY bit:

CARRY = A AND B = A.B

Half adders are not the only binary adders. There are also full binary adders which, unlike half adders, have three inputs. Like a half adder, it inputs A and B plus an additional Carry-in input to receive the carry to the Carry-out output.



Here is a diagram of a full binary adder. This is basically two half adders in conjunction with each other, but the truth table is different. The truth table includes an additional column to record the sum output, and the carry-in input with the carry-out output.



Above is a picture of a full binary adder truth table.

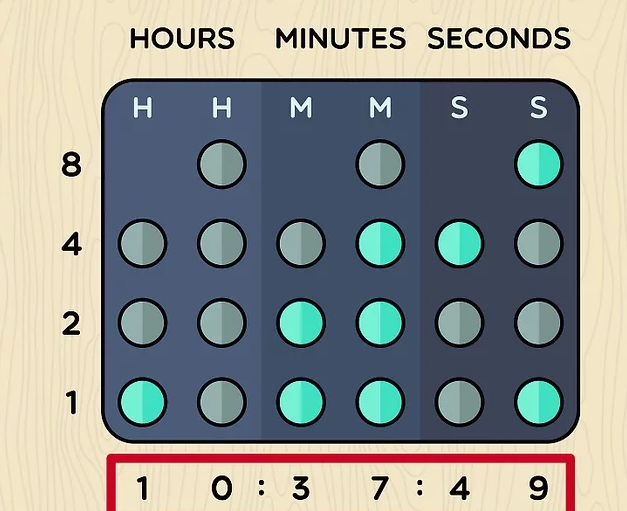
For the SUM(s) bit:

SUM = (A XOR B) XOR Cin = (A ⊕ B) ⊕ Cin

For the CARRY-OUT bit:

CARRY-OUT = A AND B OR Cin(A XOR B) = A.B + Cin(A ⊕ B)

Another application of binary numbers is the binary clock. A digital binary clock consists of LEDs arranged for hours, minutes, and seconds. These three components are controlled by separately by timing circuits.

 It is important to remember that columns are vertical and rows are horizontal. There are six columns and four rows. From bottom to top, the rows represent: 2^0 (1), 2^1 (2), 2^2 (4), and 2^3 (8). In the example above, you would read the time by adding up the binary numbers per column and then reading the final result. The time in the example above is 10:37:49.

If you’re a computer/electrical engineer, you will definitely see binary numbers in your career. 1s and 0s are the lifeblood of computer and digital information. They are used to code almost anything and will be for years to come.

Sources

https://www.allaboutcircuits.com/textbook/digital/chpt-16/binary-adder/

<https://www.electronics-tutorials.ws/combination/comb_7.html>

https://www.ukessays.com/essays/sciences/applications-of-binary-numbers.php

<https://www.wikihow.com/Read-a-Binary-Clock>