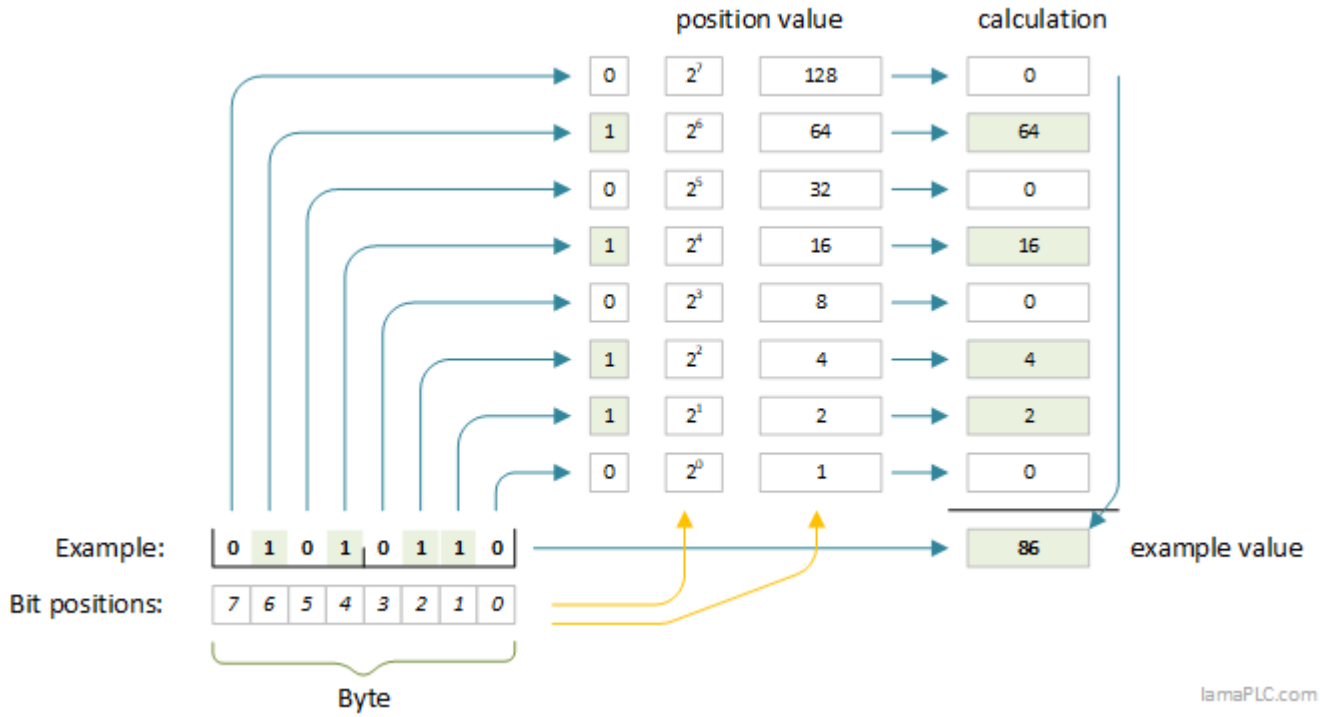
Bit & Byte

The **Bit** is the simplest form; it's a signal that can be true or false, with its official English equivalents being *"TRUE"* or *"FALSE,"* or even simply 0 or 1. There is no 2 anymore because two is represented by 10 according to the rules of the binary number system, which in this case is not ten but one zero. To clearly distinguish this, we write numbers in the decimal number system *"just like that,"* for example, 10. If this is a number in the binary number system, then we denote it as **2#**10.

The decimal number system stems from the fact that we have ten fingers and, historically, used them to perform all our calculations. If we had, say, three fingers on each hand, meaning six in total, then we would be using the six-number system now. Computing is based on the above yes-or-no logic, i.e., the binary number system, which is why we often use the hexadecimal number system. I'll talk about that later. Let's first look at the binary number system through a byte to see how it works.

A byte is a variable type consisting of 8 bits. The value stored in it must be somewhere between 0 and 255, depending on the bit positions. The example below may help you understand this a little:



Let's take the bit sequence *"01010110"* as an example, which fills the above byte. The bits of a byte are always numbered from right to left; position 0 is always on the right. Each position corresponds to a given power of the binary number system; position 4 corresponds to 24 = 16. If there is a 0 in this position in the example, then it does not *"count"*; if there is a one, then its value "counts", and as can be seen in the rows marked in green, the sum of the *"counting"* rows gives the current decimal value of the byte, 86. That is, **2#**01010110 = 86.

Therefore, the byte reaches its maximum value when all bits are set to 1. It can be calculated that **2#**11111111 = 255. The byte data type holds values between 0 and 255.

In computing, we use the base-10 number system, as well as the binary and base-16 number systems. The values described in it are called hexadecimal numbers and are denoted by the prefix "**16#**" or sometimes "**hex#**". Sometimes the hexadecimal number system is simply the hash, like

this: “#ABCD”. The hexadecimal number system changes order of magnitude at 16, meaning that a position can contain a value between 0 and 15. This can be very confusing in the base 10 number system, so the two-digit positions are denoted by letters:

10 = 16#A
11 = 16#B
12 = 16#C
13 = 16#D
14 = 16#E
15 = 16#F

If a byte reaches its maximum value, meaning every bit is set to “1”, then: **2#**11111111 = 255 = **16#**FF

If we calculate: F, i.e., “15” * 16 + “15” = 255

In some ways, this can make our lives easier, because if we see a value of “**16#**FF” somewhere, or a longer series of these, for example “**16#**FFFF_FFFF”, then we can suspect that we have reached the maximum value of one of the variable types. I would also like to mention the 8-bit, i.e., octal number system, it sometimes still occurs here and there, for example, in the case of numerical symbols, but only rarely, we don’t really use it.

DEAD_BEEF

Just as *“FF”* is likely to represent the maximum of a given variable type, dead beef is a test value designation, a play on letters. The letters of the hexadecimal number system are a, b, c, d, e, f. #dead_beef contains all of them except c, so it is helpful for testing. The Windows calculator, switched to programmer mode, is very helpful for hex-dec-bin conversions. From this, it turns out that the value of **16#dead_beef** is:

2026/01/06 15:01

More information: TIA Datatypes: S7 data types summary table