[Comments]

210-Hexadecimal and Bitwise, an explanation

29-Jul-20

What is Hexadecimal and why do I need it?

Hexadecimal, or hex for short is base 16 notation.

Data is transmitted in bytes and hex gives a very good representation of a byte. Each character represents 4 bits, so each 2 characters in hex are a byte.

Because it's assumed you're looking at bytes when you're using hex, you will usually see hex numbers with an even number of digits. In a single byte the first digit is the upper nybble or most significant 4 bits (b7, b6, b5, b4), and the second digit is the lower nybble or least significant 4 bits (b3, b2, b1, b0).

The value of each bit is 2 to b# power. That is. b0 = 2 to the 0 power = 1, b1 = 2 to the 1 power = 2, b2 = 2 squared = 4, b3 = 2 cubed = 8, etc.

In Axcess we denote hex numbers with a $. For example decimal 2 = $02; decimal 255 = $FF.

So if I'm looking at bits, what do these bitwise operators do?

Bitwise AND (in Axcess BAND or &) :

Use BAND to check if bits are set, or clear bits.

Check bits example:

Checksum = Checksum BAND $FF.

Since $FF has all bits set, the low byte remains unchanged, but any higher values get stripped off. This ensures that checksum's value is limited to a number $FF or less.

Clear bits example:

LED_STATUS = LED_STATUS BAND $F7

This will clear b3, which has a value of 8. This works because $FF - $08 = $F7.

Bitwise OR (in Axcess BOR or |) :

Use BOR to set bits:

NEW_LED_STATE = NEW_LED_STATE BOR $08

This will set b3. If b3 is already set it will leave it unchanged.

Bitwise Exclusive OR (in Axcess BXOR or ^) :

Use BXOR to toggle a bit:

NEW_LED_STATE = NEW_LED_STATE BXOR $08

If b3 is off , the "or" part of BXOR will turn it on. If it is on, the "exclusive" part will turn it off.

You'll find that you'll use BXOR in bitwise operations where you would use NOT in logical statements.

Bitwise NOT (in Axcess BNOT or ~) :

Use BNOT to toggle all the bits.

Also known as the 1's complement of a number.

Example:

BNOT $08 = $FFF7

You could modify the BAND clear bits example and do this:

LED_STATUS = LED_STATUS BAND BNOT $08

This is a little easier to see that you're clearing b3.

Last of all, a few suggestions to help you keep your sanity now that we're have all the new data types in NetLinx:

Use leading 0's to help you remember the size of the data type you're dealing with.

For Example: b0 as a CHAR is $01

b0 as an INTEGER is $0001

b0 as a LONG is $00000001

b5 as a CHAR is $20

b5 as an INTEGER is $0020

b5 as a LONG is $00000020

Don't use Signed data types for bitwise manipulation. These types use the MSB (Most Significant Bit) as a "sign" bit. If it's set the number is negative, if it's not set the number is positive. You probably don't need the extra confusion.

Don't use Floating Point data types for bitwise manipulation. These types not only use the MSB as a sign bit, they use the next 8 (for a FLOAT) or 11 (for a DOUBLE) bits for the exponent.

You REALLY don't need the extra confusion.