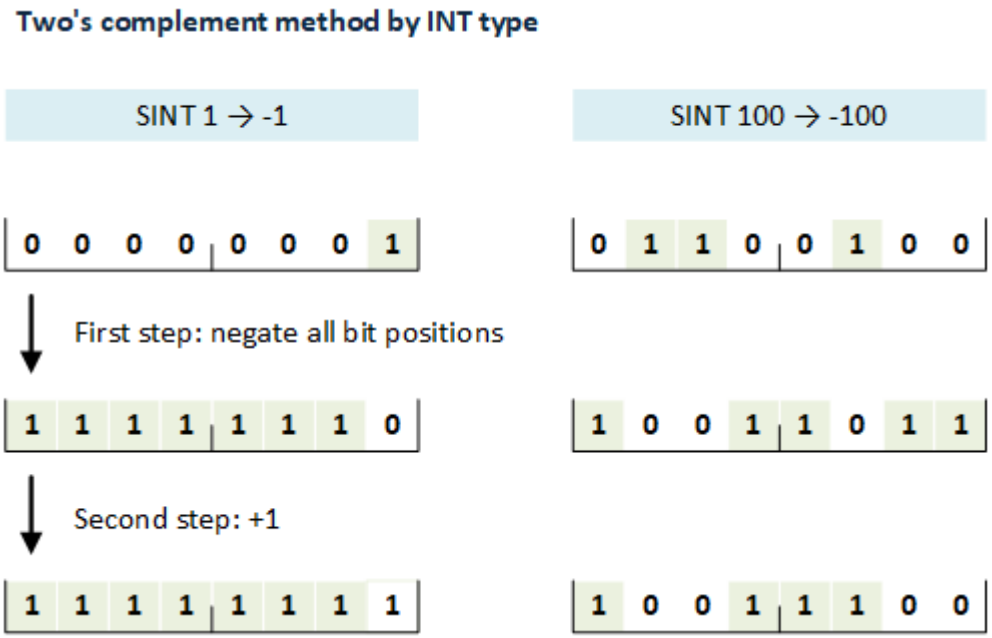
INT type variables

In the case of the INT, which is the integer type, the definition becomes slightly more complex in terms of formal constraints because of the introduction of the sign bit. This means that the highest-order bit of the variable's position, the first bit on the left, represents the sign: if it is “1”, the variable indicates a negative number, whereas if it is “0”, it indicates a positive one.

Really, just for completeness, in the case of negative numbers, the program uses the so-called *“two's complement”* representation. That is, it first negates all the bits of the numerical value, i.e., it converts 0 to 1 and vice versa, and then adds 1 to the resulting value. This conversion means that the negative value cannot be read directly from the bit combination unless the conversion is performed again in the opposite direction:



As a result, Simatic only uses binary and hexadecimal notations for positive numbers, meaning that negative hexadecimal or binary values will not show the actual numerical value but instead the value based on the bit pattern. For example, A, which equals ten, will still be 16#A, but -A, which equals -10, will be displayed in WORD format as FFF6. This misunderstanding is resolved by the rule that hexadecimal and binary signals cannot have negative values in Simatic:

```
1
2  #intVariable := 128;
3
4  #intVariable := -128;
5
6  #intVariable := 16#AA;
7
8  #intVariable := 16#-AA;
9
10 #intVariable := 2#00011111;
11
12 #intVariable := 2#-00011111;
```

In the above example, I tried to assign a value to an INT variable. It is clear that the compiler accepted the negative value when specified in decimal, but not when specified in hexadecimal or binary. Let's look at the contents of the INT variable in several forms:

DEC	SINT HEX (8-bit)	SINT BIN (8-bit)	INT HEX (16-bit)	INT BIN (16-bit)
12	16#7F	2#0111_1111	16#007F	2#0000_0000_0111_1111
1	16#01	2#0000_0001	16#0001	2#0000_0000_0000_0001
-1	16#FF	2#1111_1111	16#FFFF	2#1111_1111_1111_1111
-85	16#AB	2#1010_1011	16#FFAB	2#1111_1111_1010_1011
-128	16#80	2#1000_0000	16#FF80	2#1111_1111_1000_0000

The INT type is optimized for decimal handling; it can also be used in hexadecimal and binary forms, but in these cases, you need to pay close attention to the type's special characteristics.

Compared to byte and word type variables, this means that the maximum value of these variables is almost halved when dealing with decimal numbers. However, roughly the same magnitude can be used in the negative direction. For example, a one-byte-long SINT type will operate within the range -128 to 127, unlike the *"plain"* BYTE range of 0 to 255.

The letter "**S**" in the SINT definition stands for the word *"short"*, as the INT type is the default integer (16 bits), while SINT is short, with half the bit length—8 bits. The letter "**D**" represents the word *"double,"* with its 32 bits.

Type	Name	Bit	Minimum	Maximum	Value range HEX *	Value range DEC
SINT	short integer	8	$-(2^7)$	2^7-1	0 .. 7F	-128 .. 127
INT	integer	16	$-(2^{15})$	$2^{15}-1$	0 .. 7FFF	-32.768 .. 32.767
DINT	double integer	32	$-(2^{31})$	$2^{31}-1$	0 .. 7FFF_FFFF	-2.147.483.648 .. +2.147.483.647
LINT	double long integer	64	$-(2^{63})$	$2^{63}-1$	0 .. 7FFF_FFFF_FFFF_FFFF	-9.223.372.036.854.775.808 .. +9.223.372.036.854.775.807

* *Negative number ranges are not supported in hexadecimal and binary formats.*

UINT type variables

The unsigned UINT type (the letter **U** stands for unsigned) removes the hassle of dealing with negative values from the world of the INT type. It corresponds to basic types like BYTE, WORD, etc., in terms of value range, but with INT it indicates that we want to treat the contents of the variables as numeric values.

Type	Name	Bit	Minimum	Maximum	Value range HEX *	Value range DEC

Type	Name	Bit	Minimum	Maximum	Value range HEX *	Value range DEC
USINT	unsigned short integer	8	0	2^8	0 .. FF	0 .. 255
UINT	unsigned integer	16	0	2^{16}	0 .. FFFF	0 .. 65.535
UDINT	Unsigned double integer	32	0	2^{32}	0 .. FFFF_FFFF	0 .. 4.294.967.295
ULINT	Unsigned long integer	64	0	2^{64}	0 .. FFFF_FFFF_FFFF_FFFF	0 .. 18.446.744.073.709.551.615

* *Negative number ranges are not supported in hexadecimal and binary formats.*

More information: TIA Datatypes: S7 data types summary table