Direct / indirect addressing

Addressing methods are mostly tied to variable types, not areas, so the following procedures apply to both DB and Tag variables.

Direct addressing

Direct addressing in Simatic is typically symbolic addressing, meaning in the simplest case we correspond two variables of the same type to each other:

```
fromReal : Real;
fromInt : Int;
toReal : Real;
toInt : Int;
…

#toInt := #fromInt;
#toReal := #fromReal;
```

If the types do not match, conversion will help us:

```
#toInt := REAL_TO_INT(IN := #fromReal);
```

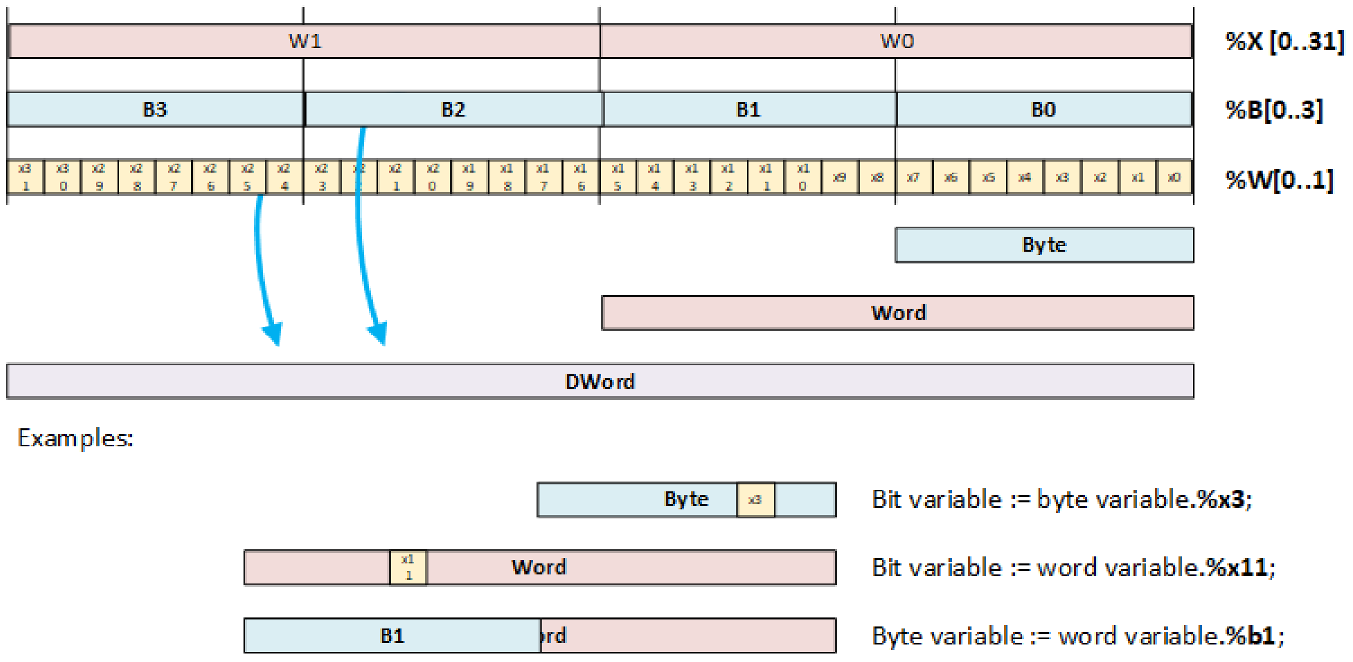
It is crucial to understand that conversion can lead to data loss. In the example above, the REAL type can store much larger numbers and fractional parts, while the INT only handles smaller integers and rounds off fractions. When converting between variables with different ranges, all values outside the smaller range should be considered. In this case, rather than using an INT, a variable with a broader range should be selected (example DINT, LINT).

Direct addressing is also applicable to STRUCTURE and ARRAY types, as long as both sides have identical structures.

Another approach is direct addressing, which involves referring to sub-elements of a variable. Although this method applies to a limited range of variables, it is a simple form of assignment. While it isn't as straightforward as the S7-Classic AT command that many programmers prefer, it is at least available:

Slice addressing

Slice addressing involves dividing a memory region, such as a byte or word, into smaller segments like booleans. With S7-1200 and S7-1500, you can target specific parts within declared variables (**only by byte, word, dword**) and access segments of 1, 8, 16, or 32 bits.


W1
W0
%X [0..31]
B3
B2
B1
B0
%B[0..3]
%W[0..1]
Byte
Word
DWord
Examples:
Byte
Bit variable := byte variable.%x3;
Word
Bit variable := word variable.%x11;
B1
Byte variable := word variable.%b1;

The following example is a SPLIT function that splits a WORD Input variable into bits:

```
// FC Input : inWord (Word)
// FC output: 16 variable bit0..bit15 (Bool)
// splitting
#bit0 := #inWord.%X0;
#bit1 := #inWord.%X1;
#bit2 := #inWord.%X2;
#bit3 := #inWord.%X3;
#bit4 := #inWord.%X4;
#bit5 := #inWord.%X5;
#bit6 := #inWord.%X6;
#bit7 := #inWord.%X7;
#bit8 := #inWord.%X8;
#bit9 := #inWord.%X9;
#bitA := #inWord.%X10;
#bitB := #inWord.%X11;
#bitC := #inWord.%X12;
#bitD := #inWord.%X13;
#bitE := #inWord.%X14;
#bitF := #inWord.%X15;
```