BITSTRING OPERATORS

AND

The operator is used for bitwise AND operation of bit operands.

Permitted data types: BOOL, BYTE, WORD, DWORD, LWORD.

A	B	A AND B
0	0	0
0	1	0
1	0	0
1	1	1

```
PROGRAM Bitstring_Operators
VAR
   wVarAnd: WORD;
END_VAR

wVarAnd := 2#1001_0011 AND 2#1000_1010;
(* Result in wVarAnd: 2#1000_0010 *)
```

AND_THEN

The operator is an extension of the IEC 61131-3 standard, used exclusively in Structured Text (ST) for AND operation with shortcircuit evaluation on BOOL and BIT operands.

When all operands yield TRUE, the result of the operands also yield TRUE; otherwise FALSE.

```
PROGRAM Bitstring_Operators
VAR
   pxSensor : POINTER TO BOOL;    // Pointer to a sensor
   xAlarm : BOOL;                 // Variable to activate the alarm
END_VAR

(* pxSensor := ADR(xSomethingValue); Is NULL if not initialized *)
IF (pxSensor <> 0 AND_THEN pxSensor^) THEN
   (* Additional logic can be implemented below *)
   xAlarm := TRUE;
END_IF
```

Check if the pointer `pSensor` is not null. If it is null, the rest of the condition is not evaluated. The `pxSensor^` is only evaluated if `pxSensor <>0` is `TRUE`, avoiding dereference errors.

Using `AND_THEN` prevents the program from accessing `psSensor^` when `pxSensor` is 0, avoiding runtime errors, making it safer and more efficient than using `AND`.

OR_ELSE

The operator is an extension of the IEC 61131-3 standard, used exclusively in Structured Text (ST) for OR operation with shortcircuit evaluation on BOOL and BIT operands.

When at least one of the operands yields TRUE, the result of the operation also yields TRUE; otherwise FALSE.

```
FUNCTION_BLOCK FB_OrElse
VAR
   iCounter : INT;
END_VAR
```

```
(* Method of the Function Block FB_OrElse *)
METHOD TestMethod : BOOL
iCounter := iCounter + 1;
TestMethod := TRUE; (* Set the method's return value to TRUE *)
```

```
PROGRAM Bitstring_Operators
VAR
    fbSampleOrElse : FB_OrElse; //Instance of the FB_OrElse function block
    xResult :  BOOL;
    xVar :  BOOL;
END_VAR

xResult := xVar OR_ELSE fbSampleOrElse.TestMethod();
```

1. If `xVar` is `TRUE`, the method does not execute, and the counter iCounter does not increment.
2. If `xVar` is `FALSE`, the method executes, and the counter iCounter increments by 1.

With OR_ELSE, if any operand is TRUE, the rest of the expressions are not evaluated, unlike the regular OR operator.

OR

The IEC operator is used for bitwise OR operation of bit operands.

Permitted data types: BOOL, BYTE, WORD, DWORD, LWORD.

A	B	A OR B
0	0	0
0	1	1
1	0	1
1	1	1

```
PROGRAM Bitstring_Operators
VAR
   wVarOr: WORD;
END_VAR


wVarOr:= 2#1001_0011 OR 2#1000_1010;
(* Result in wVarOr: 2#1001_1011 *)
```

XOR

The IEC operator is used for bitwise XOR operation of bit operands.

Permitted data types: BOOL, BYTE, WORD, DWORD, LWORD.

A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

Note the following behavior of the XOR POU in extended form (more than two inputs): compares the inputs in pairs and then the corresponding results (according to the standard, but not necessarily according to expectations).

```
PROGRAM Bitstring_Operators
VAR
   wVarXor: WORD;
END_VAR

wVarXor := 2#1001_0011 XOR 2#1000_1010;
(* Result in wVarXor: 2#0001_1001 *)
```

NOT

The IEC operator is used for bitwise NOT of bit operands.

Permitted data types: BOOL, BYTE, WORD, DWORD, LWORD.

A	NOT A
0	1
1	0

```
PROGRAM Bitstring_Operators
VAR
    wVarNot: WORD;
END_VAR


wVarNot := NOT 2#1001_0011;
(* Result in wVarNot: 2#0110_1100 *)
```