Contents 10

10. Digital Operations 10-2

10.1. Task Description: Counting the Transported Parts and Timed Monitoring of the Transportation 10-3
10.1.1. Acquiring, Processing and Outputting Data 10-4
10.1.2. Integer (INT, 16-Bit Integer) Data Type 10-5
10.1.2.1. Double Integer (DINT, 32-Bit Integer) Data Type 10-6
10.1.3. REAL and LREAL (Floating-point Number) Data Type 10-7
10.1.4. Data Types and Display Formats... 10-8

10.2. Task Description: Counting the Transported Parts using Addition in "FB_CountADD"...... 10-9
10.2.1. Basic Mathematical Functions: Addition 10-10
10.2.1.1. CALCULATE Box.. 10-11
10.2.2. Comparison Operations 10-12
10.2.3. Value Assignment of a Variable.. 10-13
10.2.4. Programming All Instructions using Empty Box 10-14
10.2.5. Exercise 1: Counting the Transported Parts using Addition in "FB_CountADD"............. 10-15
10.2.6. Exercise 2: Calling "FB_CountADD".. 10-16

10.3. Task Description: Timed Monitoring of the Transport Sequences and Counting Parts using IEC Functions 10-17
10.3.1. IEC Timer 10-18
10.3.2. IEC Timer TON (ON Delay) Pulse Diagram 10-19
10.3.3. Exercise 3: Programming the Timed Monitoring of the Transports in "FC_Fault"............ 10-20

10.4. IEC Counters: CTU, CTD, CTUD.. 10-21
10.4.1. IEC Counters UP/DOWN: Inputs 10-22
10.4.2. IEC Counters UP/DOWN: Outputs 10-23
10.4.3. Exercise 4: Counting the Transported Parts using an IEC Counter 10-24
10.4.4. Exercise 5: Replacing the "FB_CountADD" Call with the "FB_Count" Call...................... 10-25

10.5. Additional Exercise 6: Counting the Conveyor Faults by Expanding "FC_Fault" 10-26
10.5.1.1. Additional Exercise 7: Timely Lock-out of the Conveyor Motor Jogging 10-27

10.6. Additional Information 10-28
10.6.1. Comparator Operations: IN_RANGE, OUT_RANGE 10-29
10.6.2. Digital Logic Operations.. 10-30
10.6.2.1. Application Example: Digital Edge Evaluation 10-31
10.6.3. SIMATIC-Timer 10-32
10.6.3.1. SIMATIC-Counter 10-33
10.6.4. Value Ranges of Various Number Formats... 10-34

10. Digital Operations

At the end of the chapter the participant will …

… be familiar with the structure of the data types INT, DINT, REAL and LREAL

… be familiar with various presentation types

… be able to use basic mathematical operations

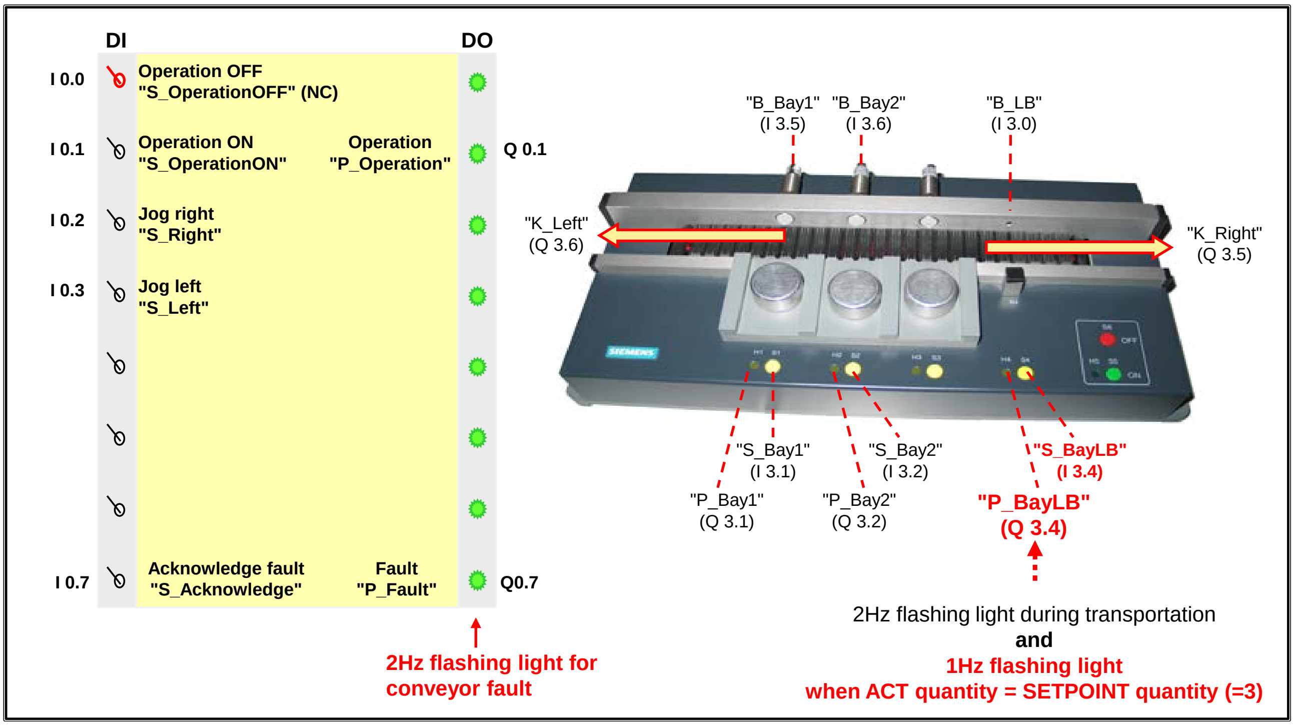
… be familiar with comparison functions

… be familiar with the MOVE function for value assignment

… be familiar with the different counter and timer functions

… be able to use and program counter and timer functions

10.1. Task Description: Counting the Transported Parts and Time Monitoring of the Transportation



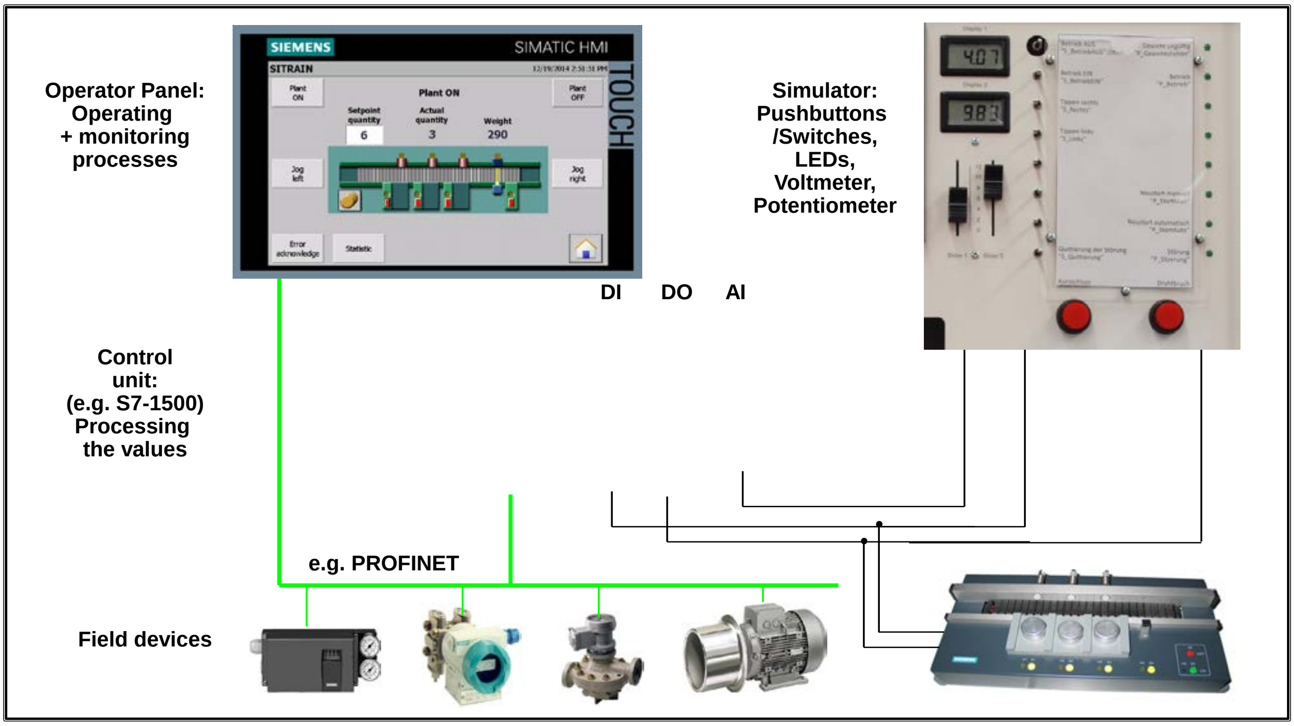
Function Up Till Now

When "P_Operation" (Q0.1) is switched on**,** parts are transported from Bay 1 or Bay 2 through the light barrier. The transport sequence starts as soon as a part is placed on Bay 1 or Bay 2 and the associated bay's pushbutton is pressed and it ends as soon as the part has passed the light barrier.

Task Description

- When "P_Operation" (Q0.1) is switched on, the transported parts are to be counted as soon as they have passed through the "B_LB" (I 3.0) light barrier ("B_LB" 0 → 1). The number of transported parts (ACTUAL quantity) is to be counted.
- The counter can be acknowledged (reset to 0) via the pushbutton "S_BayLB" (I 3.4). When "P_Operation" (Q0.1) is **switched on**, the ACTUAL quantity is also reset to 0.
- The indicator light "P_BayLB" (Q3.4) shows a 1Hz flashing light when the actual quantity has reached the setpoint quantity of 3 (= a new part must not be placed on the conveyor -> lock-out in "FC_Signal") and no further transport sequence can be started (-> lock-out in "FC_Conveyor"), and a 2Hz flashing light during parts transportation.
- Furthermore the time, that is required for the transport of a workpiece from Bay 1 and Bay 2 until it is through the light barrier, is measured.
- If the time it takes to transport exceeds 6 seconds, the conveyor is stopped and this is indicated at the output "P_Fault" (Q0.7) with a 1Hz flashing light.
- Only after this error is acknowledged with "S_Acknowledge" (I 0.7) can a new part be transported.

10.1.1. Acquiring, Processing and Outputting Data



Binary/Digital Processing

True logic control systems are recognizable in the fact that they exclusively process binary data. The performance of today‘s control computer, as well as tasks in the areas of data processing, quality control, among others, has increased the importance of digital data processing using PLCs. Digital process variables can be found in all areas of open-loop control - such as in connected devices for process operating and monitoring or in the control of field devices.

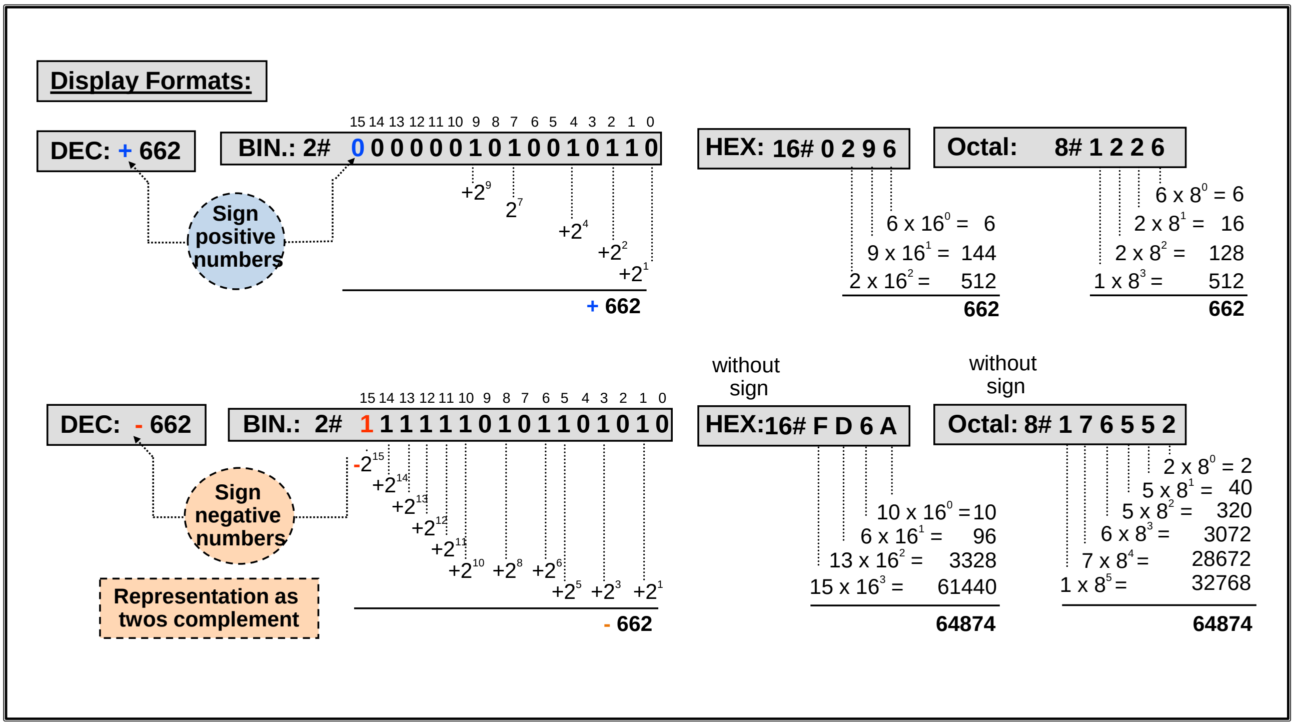
Operating and Monitoring

The goal of process monitoring is to provide the operator with up-to-the-minute information about the working machine or system quickly, concisely and clearly as well as the opportunity to intervene, control and influence the process. Depending on the type of device connected, different number formats for the coding of data are used to transmit data between devices and PLC, as well as for storing and processing data in the PLC.

Field Devices

Today as well, field devices that acquire process data or that control the process are supplied directly with digital variables through field bus systems.

10.1.2. Integer (INT, 16-Bit Integer) Data Type



Integer (16-Bit) Data Type

An *Integer* data type value is a whole number value, that is, a value without a decimal point. SIMATIC S7 stores *Integer* data type values with sign in 16 bit code. This results in a value range from -32768 to +32767. As well, SIMATIC S7 provides arithmetic operations for processing Integer values.

Decimal

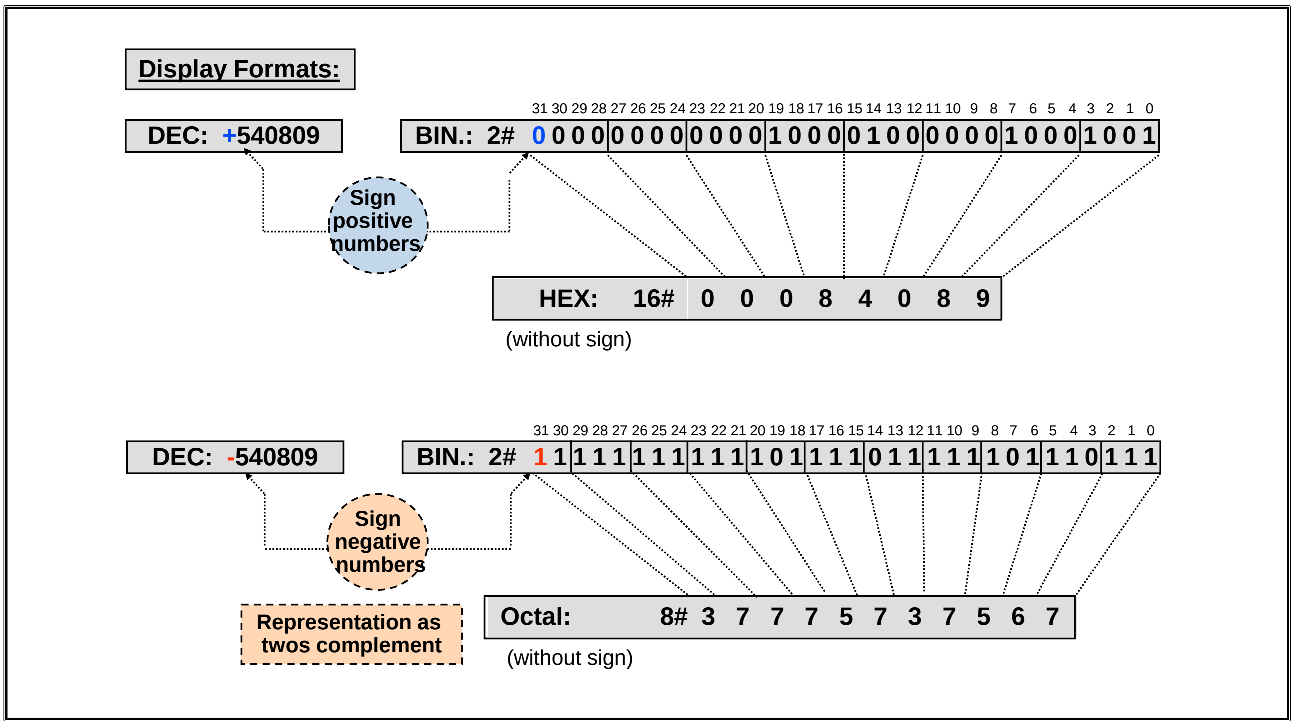
STEP 7 uses the *Decimal* display format, that is, with sign and without explicit format description, to specify the constants of the *Integer* data type. The use of constant Integer values in the *Binary, Hexadecimal, Octal* display formats is possible in principle, but because of the poor legibility, they are more or less not suitable.

Binary

In a digital computer system, all values are stored in a binary-coded form. Only the digits 0 and 1 are available in the binary number system. Base 2 of this number system results from the number of available digits. Accordingly, the value of every bit of a binary number results from a power of Base 2. This is also expressed in the format specification **2#**.... .

Negative values are represented as binary numbers in twos complement. In this representation, the most significant bit (bit no. 15 for the Integer data type) has the value -2^{15}. Since this value is greater than the sum of all residual values, this bit also has the sign information. That is, if this bit = 0, then the value is positive; if the bit is = 1, then the value is negative. The conversion of a binary number into a decimal number is made by adding the values of the bits that have a 1 (see picture).

10.1.2.1. Double Integer (DINT, 32-Bit Integer) Data Type



Double Integer (32-Bit Integer)

SIMATIC S7 stores *Double Integer* data type values with sign as 32 bit code. This results in the value range from -2147483648 to +2147483648.

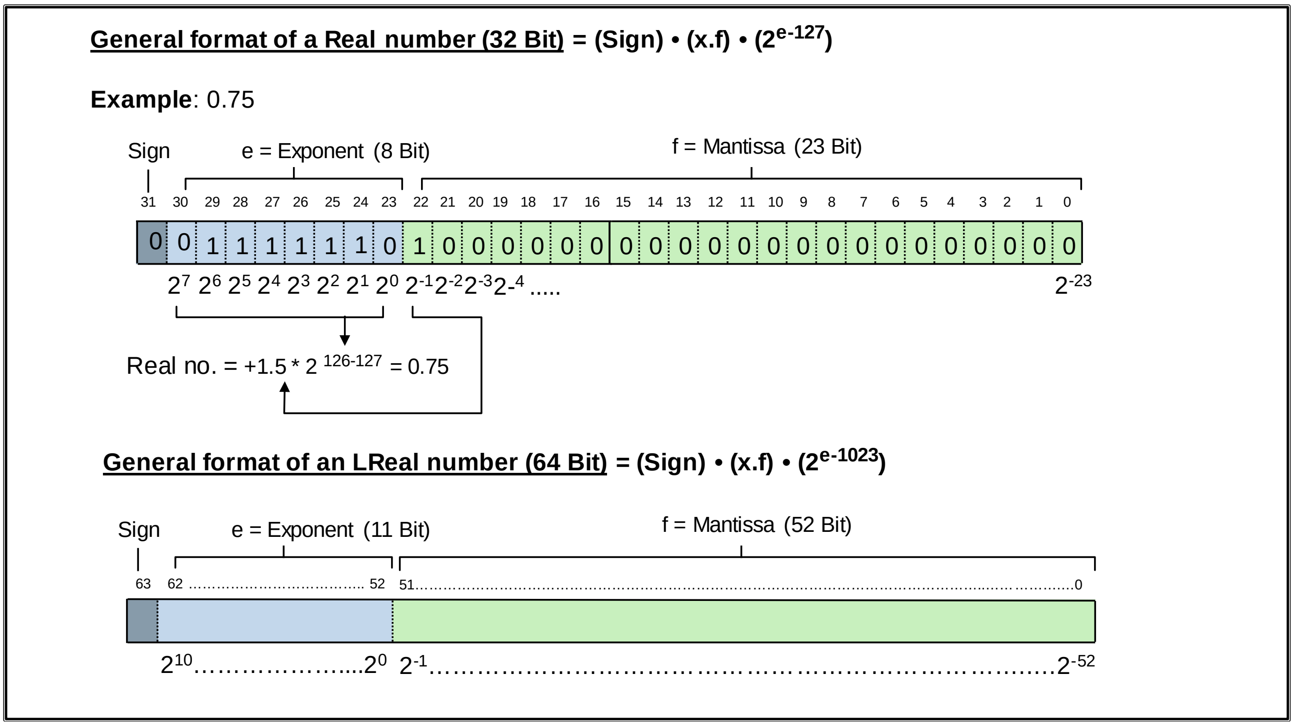
Hexadecimal

The hexadecimal number system provides 16 different digits (0 to 9 and A to F). This results in Base 16 of this numbers system. Accordingly, the value of every bit of a hexadecimal number results from a power of Base 16. Hexadecimal numbers are specified with **16#** for identifying the basic numbering system. The number of specifiable bits is variable from 1 to 16. The digits A to F correspond to the decimal values 10 to 15. The value 15 is the last value that can be binary-coded - without sign - with 4 bits. Out of this correlation, the simple conversion of a binary number into a hexadecimal number and vice versa can be obtained. In this way, four binary bits each can easily make up one digit of a hexadecimal number.

Octal Number

The octal number system provides 8 different digits (0 to 7). This results in Base 8 of this numbers system. Accordingly, the value of every bit of an octal number results from a power of Base 8. Octal numbers are specified with **8#** for identifying the basic numbering system. The value 7 is the value that can be binary-coded - without sign - with 3 bits. In this way, three binary bits each can make up one digit of an octal number.

10.1.3. REAL and LREAL (Floating-point Number) Data Type



General format of a Real number (32 Bit) = (Sign) • (x.f) • (2^{e-127})

Example: 0.75

General format of an LReal number (64 Bit) = (Sign) • (x.f) • (2^{e-1023})

Real/LReal

The previously described INT and DINT data types are used to store whole number values with sign. Accordingly, only operations that supply a whole number value as the result can be performed with these values. In cases where analog process variables such as voltage, current, and temperature etc., have to be processed, it becomes necessary to use *Real* values (real numbers, "decimal numbers"). In order to be able to represent such values, binary digits have to be defined whose value is less than 1 (power of base 2 with negative exponent).

Format

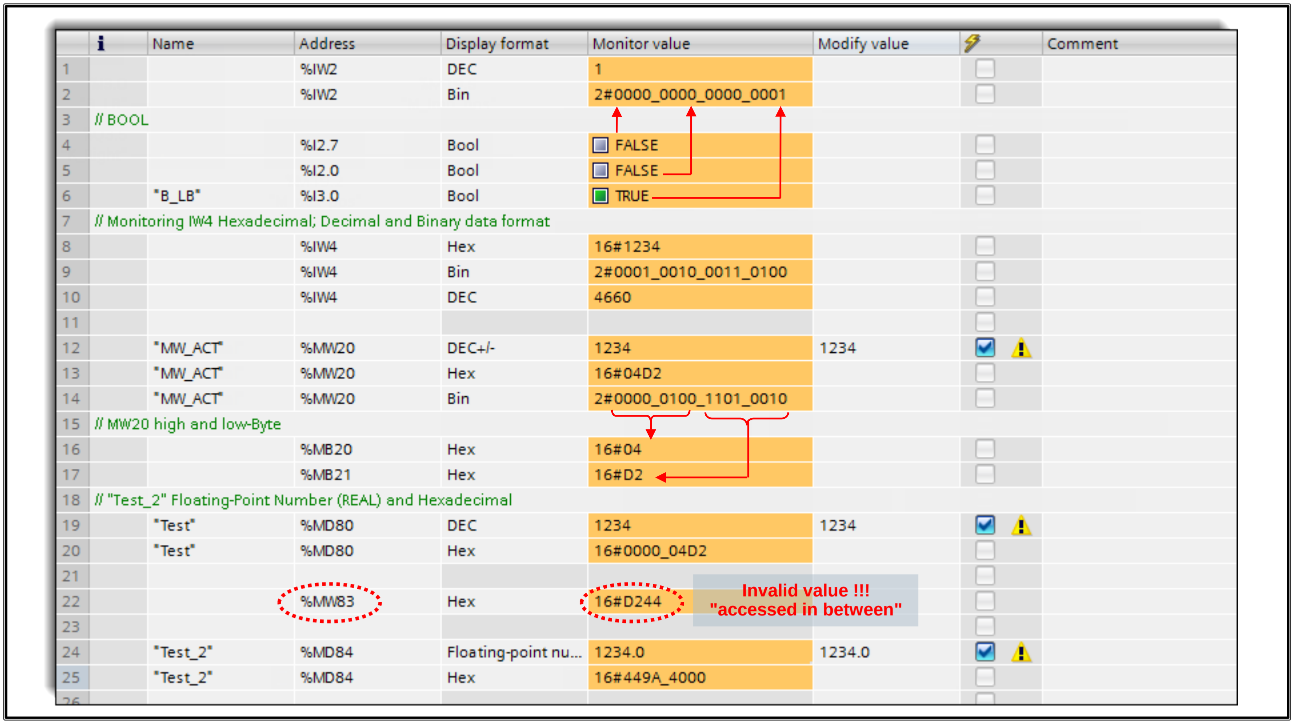
In order to be able to form the greatest possible value range within a defined memory capacity, you must be able to select the decimal point position as required. Early on, IEEE defined a format for floating-point numbers. This format was laid down in IEC 61131 and was included in STEP 7. This format makes it easy to process a variable decimal point position. . In the binary code of a 32 Bit floating-point number, a portion of the binary digits contain the mantissa (23 Bit) and the rest contain the exponent (8 Bit) and the sign bit of the floating-point number. A 64 Bit floating-point number also has the sign bit; however, the exponent is 11 Bit and the mantissa 52 Bit.

After you enter a constant real value (for example: 0.75), the Editor automatically makes a conversion to scientific notation (for example: 7.5000e-001).

Application

Floating-point numbers are used for "analog value processing", among other things. A great advantage of floating-point numbers is in the number of operations possible with such numbers. These include, in addition to the standard operations such as: +, -, * , / also instructions such as sin, cos, exp, ln, etc, that are used mainly in closed-loop control algorithms.

10.1.4. Data Types and Display Formats



	i	Name	Address	Display format	Monitor value	Modify value	⚡	Comment
1			%IW2	DEC	1		□	
2			%IW2	Bin	2#0000_0000_0000_0001		□	
3	// BOOL							
4			%I2.7	Bool	□ FALSE		□	
5			%I2.0	Bool	□ FALSE		□	
6		"B_LB"	%I3.0	Bool	🟩 TRUE		□	
7	// Monitoring IW4 Hexadecimal; Decimal and Binary data format							
8			%IW4	Hex	16#1234		□	
9			%IW4	Bin	2#0001_0010_0011_0100		□	
10			%IW4	DEC	4660		□	
11							□	
12		"MW_ACT"	%MW20	DEC+/-	1234	1234	☑ ⚠	
13		"MW_ACT"	%MW20	Hex	16#04D2		□	
14		"MW_ACT"	%MW20	Bin	2#0000_0100_1101_0010		□	
15	// MW20 high and low-Byte							
16			%MB20	Hex	16#04		□	
17			%MB21	Hex	16#D2		□	
18	// "Test_2" Floating-Point Number (REAL) and Hexadecimal							
19		"Test"	%MD80	DEC	1234	1234	☑ ⚠	
20		"Test"	%MD80	Hex	16#0000_04D2		□	
21							□	
22			%MW83	Hex	16#D244		□	
23							□	
24		"Test_2"	%MD84	Floating-point nu...	1234.0	1234.0	☑ ⚠	
25		"Test_2"	%MD84	Hex	16#449A_4000		□	
26							□	

Invalid value !!!
"accessed in between"

Display Formats

Different display formats can be selected in both the "Monitor / Modify Variables" and the "Monitor (Block)" test functions to display variables or register contents. Basically, every variable can be monitored with several display format options. Depending on the variable's data type, it becomes apparent that monitoring with the appropriate display format makes more sense.

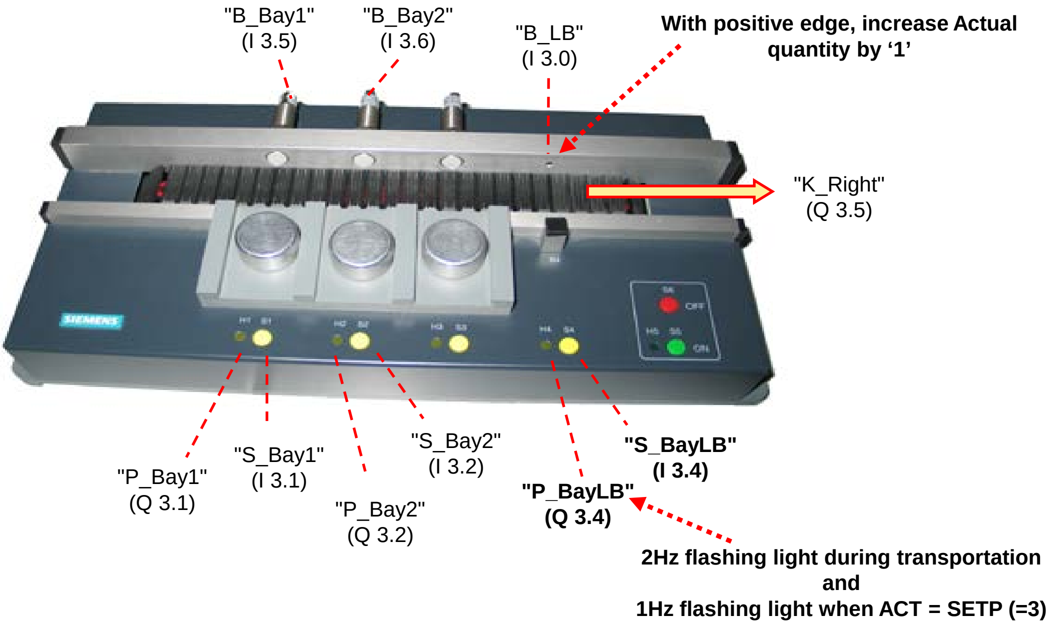
- **BOOL:** Display a single bit (only possible for a variable of the BOOL data type)
- **BIN**: Display the individual bits of a variable (makes sense for variables of the BYTE, WORD, DWORD, LWORD data types)
- **HEX, BCD**: Display the contents of a variable as a hexadecimal number, or, a BCD number (makes sense for variables of the BYTE, WORD, DWORD, LWORD data types)
- **DEC**: Display the contents of a variable as decimal number (not BCD!) **without** sign (makes sense for variables of the USINT, UINT, UDINT, ULINT data types)
- **DEC+/-**: Display the contents of a variable as decimal number (not BCD!) **with** sign (makes sense for variables of the SINT, INT, DINT, LINT data types)
- **FLOATING Point**: Display the contents of a variable as floating-point number (makes sense for variables of the REAL, LREAL data types)
- ***and others…***

Addressing

The SIMATIC S7 memory is universally byte-oriented. Accordingly, memory word MW 20, for example, contains the memory bytes MB 20 (high byte) and MB 21 (low byte, see picture), the memory double-word MD 80, the memory bytes MB 80, 81, 82 and 83.

For absolute accesses to variables (such as, with *MD 82*), you must make sure that the dimension of the access (here *MD*...) as well as the address (always equal to the address of the high byte, here 82) is correct. Through an inadvertent "accessing in between", an invalid value would be loaded (such as, with *MW 83,* see picture). Such errors can be avoided with the symbolic addressing of variables.

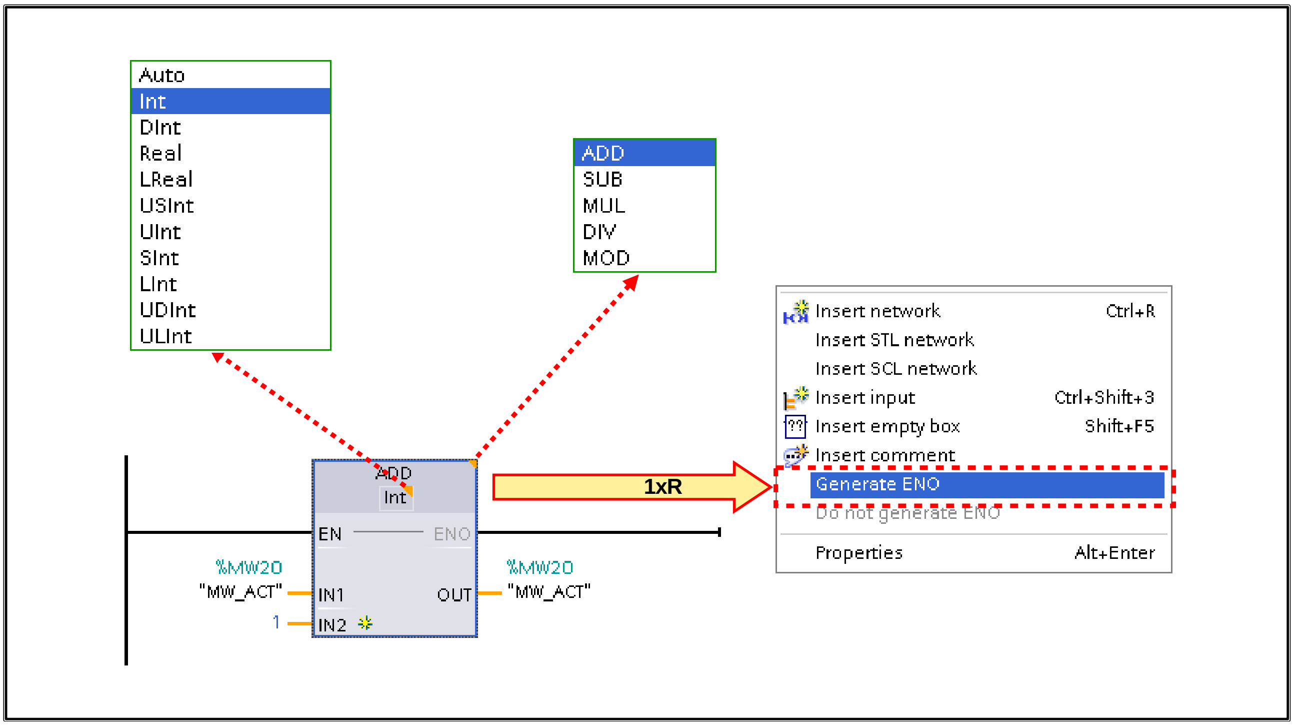
10.2. Task Description: Counting the Transported Parts using Addition in "FB_CountADD"



Task Description

- When "P_Operation" (Q0.1) is switched on, the transported parts are to be counted as soon as they have passed the light barrier "B_LB" (I 3.0) ("B_LB" 0->1).
- When the maximum quantity is reached, it is displayed via the LED "P_Bay_LB" (Q3.4) and further transport is no longer possible.
- As long as no transport is possible, the LEDs at the Bays 1 and 2 are dark and thus also signal that no transport is possible.
- The counter can be acknowledged (reset to 0) at any time via the pushbutton "S_BayLB" (I 3.4). When "P_Operation" (Q0.1) is **switched on**, the ACTUAL quantity is also reset to 0.

10.2.1. Basic Mathematical Functions: Addition



Arithmetic Operations

There is a series of arithmetic operations available for the processing of variables of the arithmetic data types, such as, integer (INT), double integer (DINT) and real (REAL).

Inputs and Outputs of the LAD/FBD Elements:

- **EN (enable)**

The execution of the operation can be determined as follows via the EN input:

- EN is not connected: The operation is always (regardless of the RLO) executed
- Logic operation at EN is fulfilled (RLO = 1): The operation is executed
- Logic operation at EN is not fulfilled (RLO = 0): the operation is not executed

- **IN1 / IN2**

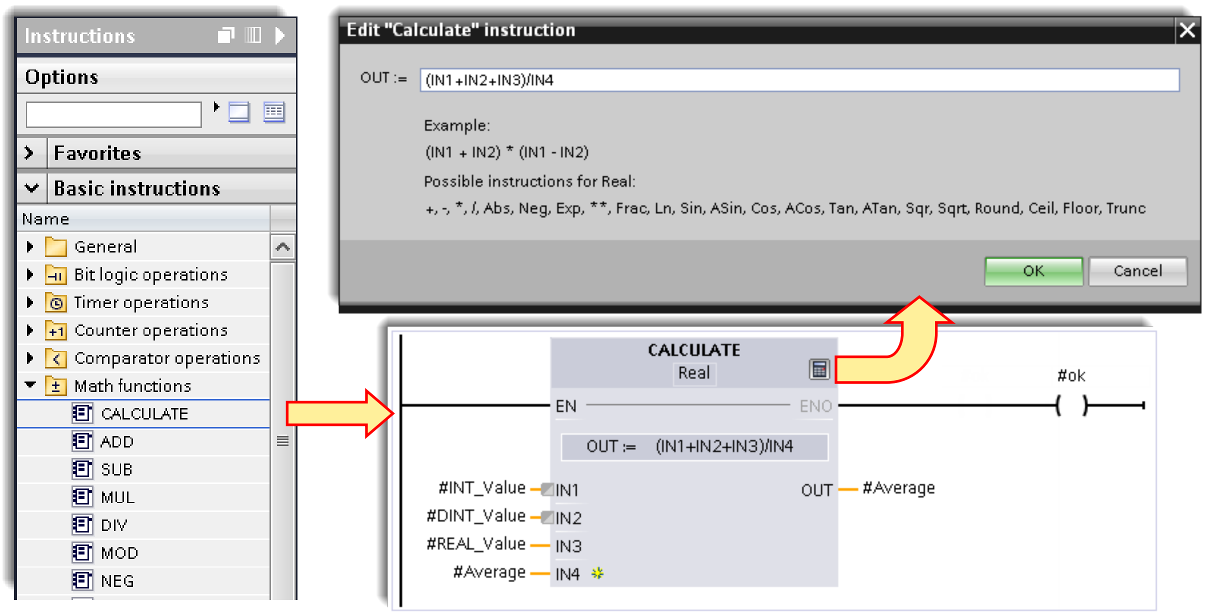
The arithmetic calculation is applied to the values delivered to IN1 and IN2 and the result is output to OUT.

- **ENO (enable output)**
  (You can activate and deactivate the generation of the ENO output via the context menu. Caution: The setting is valid for all further functions that are inserted into the project.)

  If ENO is activated, it can accept the following values:

  - ENO = 0 since the actual parameter at the EN input has the value FALSE
    - OUT is not written (the variable delivered to OUT is not written, that is, it keeps its original value)
  - ENO = 0 because an error has occurred
    - OUT contains an invalid value (the variable delivered to OUT is overwritten with an invalid value)
  - ENO = 1 (instruction was executed without error):
    - OUT contains result (the variable delivered to OUT is overwritten with the result)

10.2.1.1. CALCULATE Box



Edit "Calculate" instruction

OUT := (IN1+IN2+IN3)/IN4

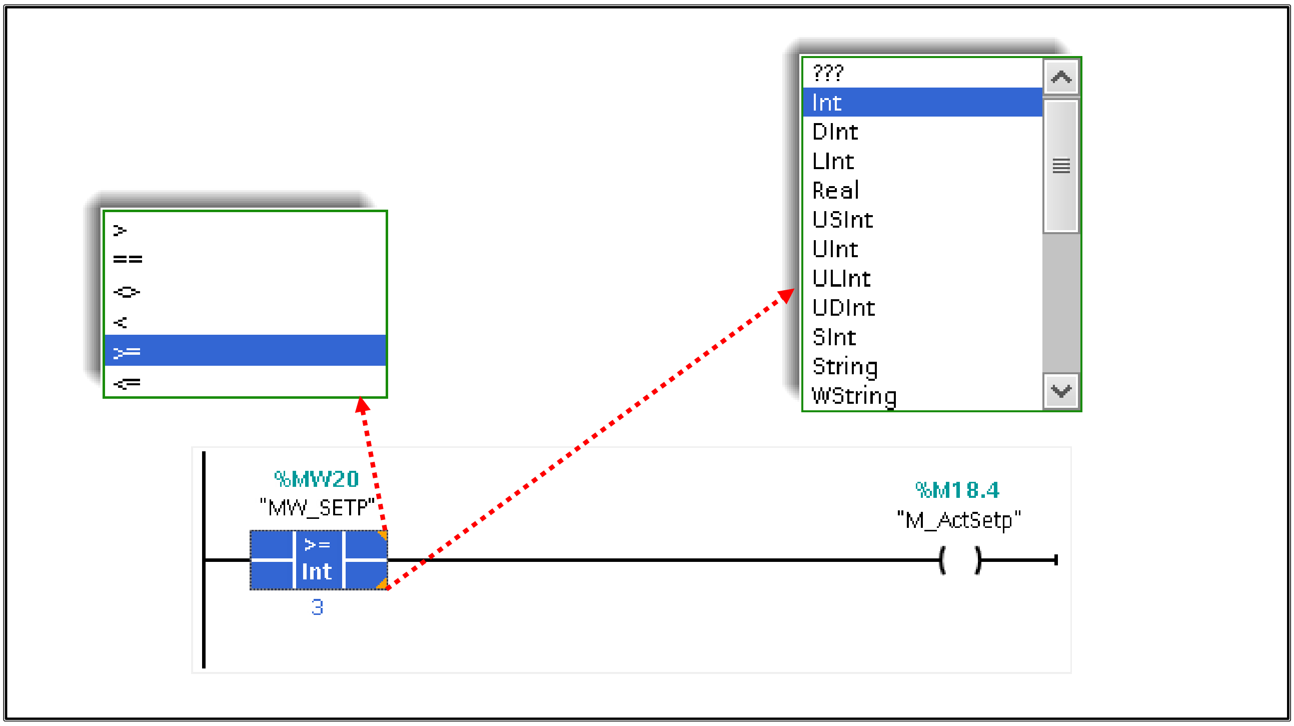
Example:
(IN1 + IN2) * (IN1 - IN2)
Possible instructions for Real:
+, -, *, /, Abs, Neg, Exp, **, Frac, Ln, Sin, ASin, Cos, ACos, Tan, ATan, Sqr, Sqrt, Round, Ceil, Floor, Trunc

Calculate Box

With the Calculate Box, calculations can be combined which contain several different math operations.

The implicit data type conversion is available at the inputs and outputs of the box and a check can be made through the ENO output as to whether errors occurred in the calculation, such as, overflows in the type conversions or the math operations, or, that the result of the calculate box is error-free.

10.2.2. Comparison Operations



Comparison Function

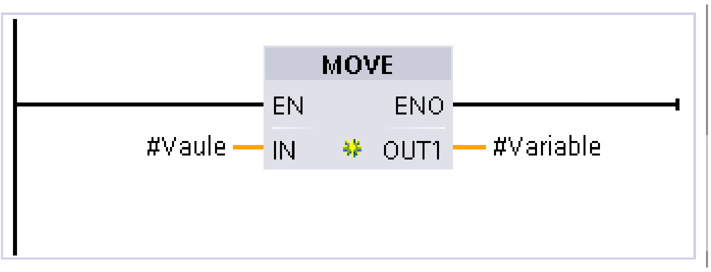
With the comparison instructions, the following pairs of numerical values can be compared:

- all variations of integers
- all variations of floating-point numbers (Real = IEEE floating-point numbers)
- all variations of TIME data types

If the result of the comparison is "true", then the output of the operation is "1", otherwise it is "0". The input IN1 is compared with IN2 according to the selected type of comparison:

•	**==**	IN1	is equal to	IN2
•	**<>**	IN1	is not equal to	IN2
•	**>**	IN1	is greater than	IN2
•	**<**	IN1	is less than	IN2
•	**>=**	IN1	is greater than or equal to	IN2
•	**<=**	IN1	is less than or equal to	IN2.

10.2.3. Value Assignment of a Variable



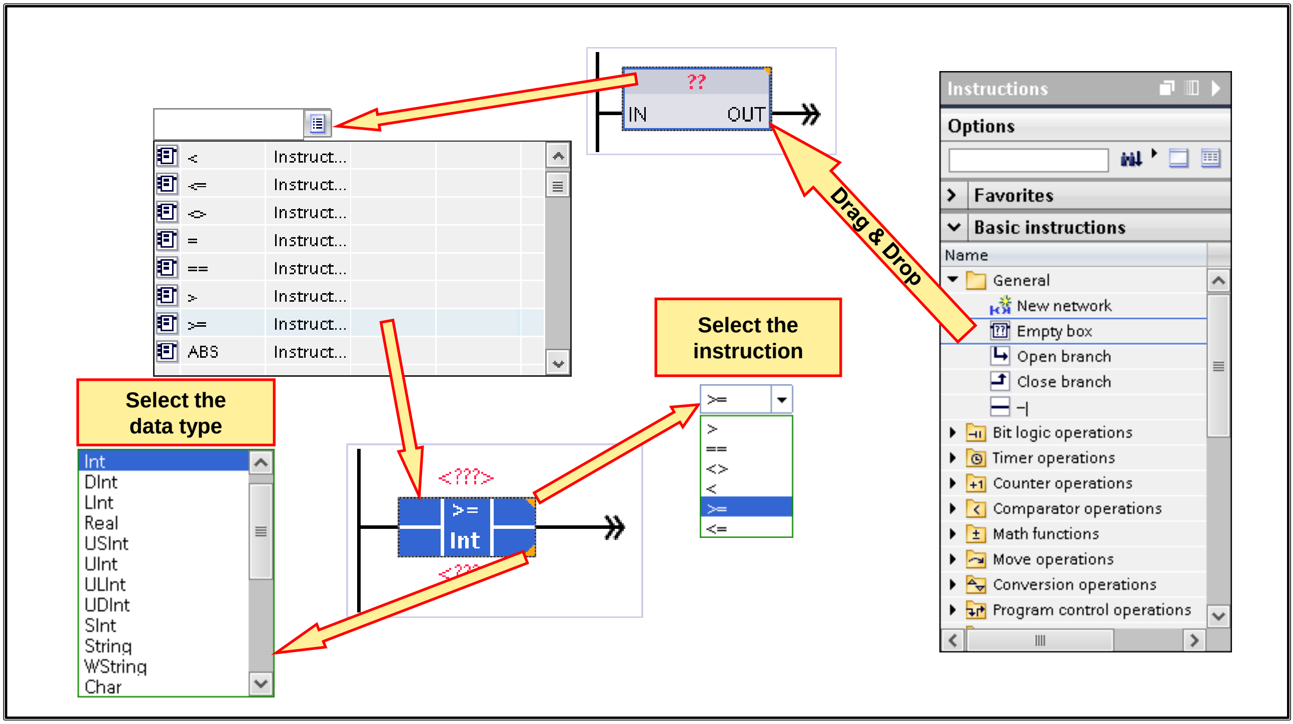
Parameter	Data type	Memory area	Description
EN	BOOL	I, Q, M, D, L	Enable input
ENO	BOOL	I, Q, M, D, L	Enable output
IN	All elementary data types, DTL, STRUCT, ARRAY	I, Q, M, D, L or constant	Source value
OUT1	All elementary data types, DTL, STRUCT, ARRAY	I, Q, M, D, L	Destination address

MOVE

You use the "MOVE" instruction to transfer the content of the operand at the IN input to the operand at the OUT1 output.

The operation is only executed if the signal status at the enable input EN is "1" or is not assigned. In this case and with error-free transfer, the ENO output also has signal status "1".

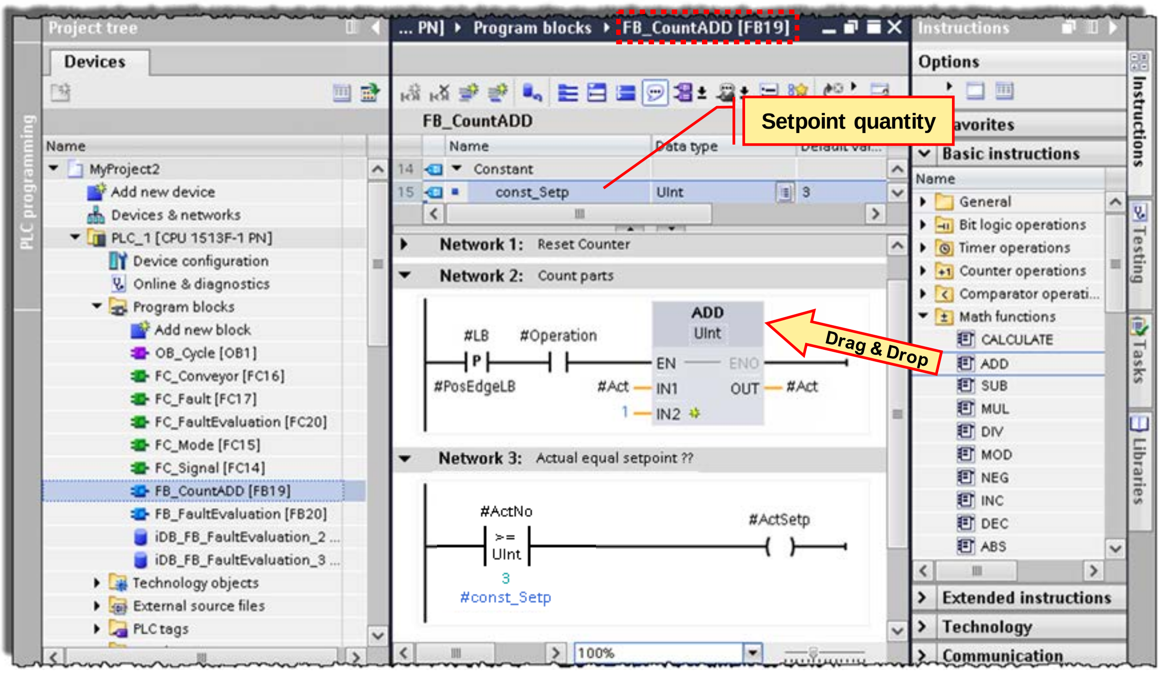
10.2.4. Programming Instructions using Empty Box



Programming an Instruction using "Empty Box"

An instruction can also be programmed in the so-called "Empty box". The empty box is first pulled from the "Instructions" task card using drag & drop, or it is pulled from the “Favorites” onto the network. Then at the empty box, you select which instruction is to be used with which data type.

10.2.5. Exercise 1: Counting the Transported Parts using Addition in "FB_CountADD"



Task

- When "P_Operation" (Q0.1) is switched on, the transported parts are to be counted as soon as they have passed the light barrier "B_LB" (I 3.0) ("B_LB" 0->1). The number of transported parts (ACTUAL quantity) is to be recorded with a counter and stored in the static variable #ACT.
- When the setpoint quantity (constant value 3) is reached, then no new transport sequence can take place which is also visible on the LEDs "P_Bay1" and "P_Bay2".
- The counter can be acknowledged (reset to 0) at any time via the pushbutton "S_BayLB" (I 3.4). When the operation is switched on, that is, with a positive edge at "P_Operation" (Q0.1), the ACTUAL quantity is also reset to 0.

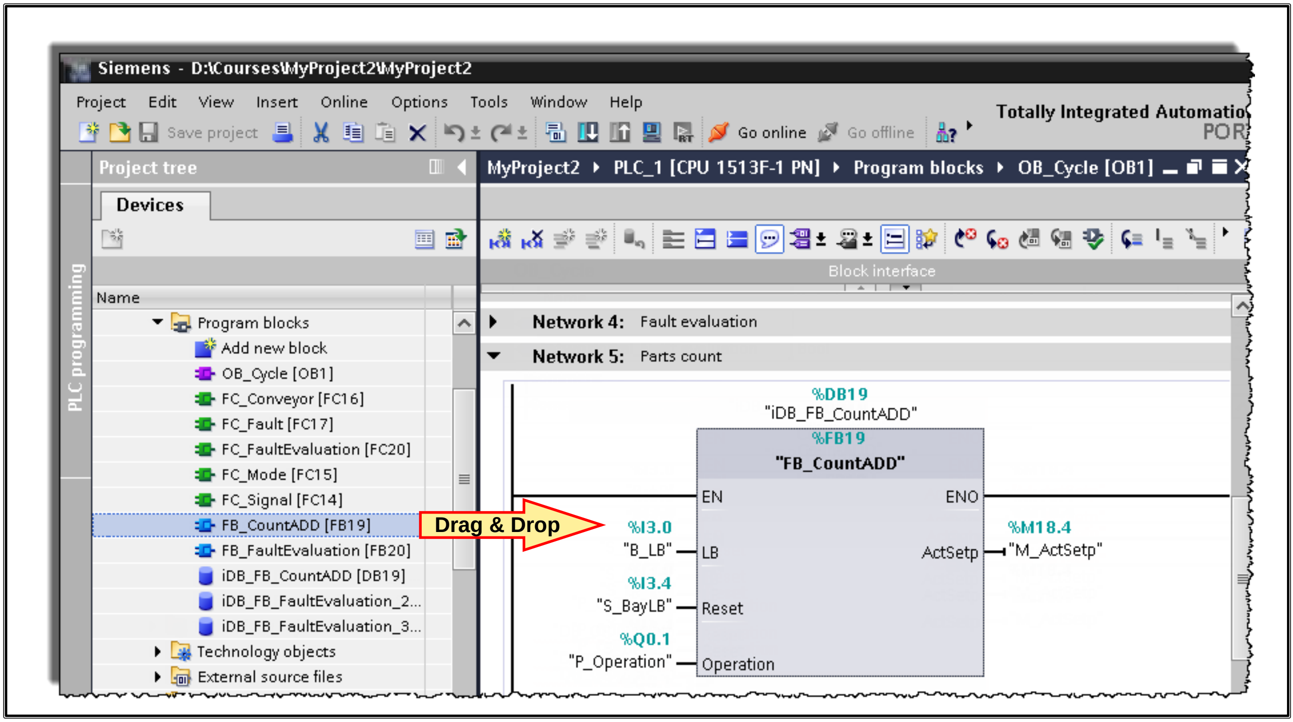
What to Do:

1. Create the new function block "FB_CountADD".
2. Declare the necessary parameters and variables:

Input:	- LB	BOOL,	Light barrier signal
	- Reset	BOOL,	Reset the actual no. of parts to 0
	- Operation	BOOL,	Operation mode
Output:	- ActSetp	BOOL,	the actual no. of parts has reached the setpoint value
Static:	- PosEdgeLB	BOOL,	auxiliary bit for the edge evaluation of the Light barrier signal
	- PosEdgeOp	BOOL,	auxiliary bit for the edge evaluation of the Operation mode signal
	- Act	UINT,	actual no. of transported parts
Constant:	- const_Setp	UINT,	setpoint no. of parts to be transported

3. Program the new "FB_CountADD" block with the appropriate arithmetic operations and don’t use any global variables (tags) to do so. For the edge evaluation of the light barrier, use the static variable #PosEdgeLB and save the current quantity of the transported parts in the static variable #ACT.

10.2.6. Exercise 2: Calling "FB_CountADD"



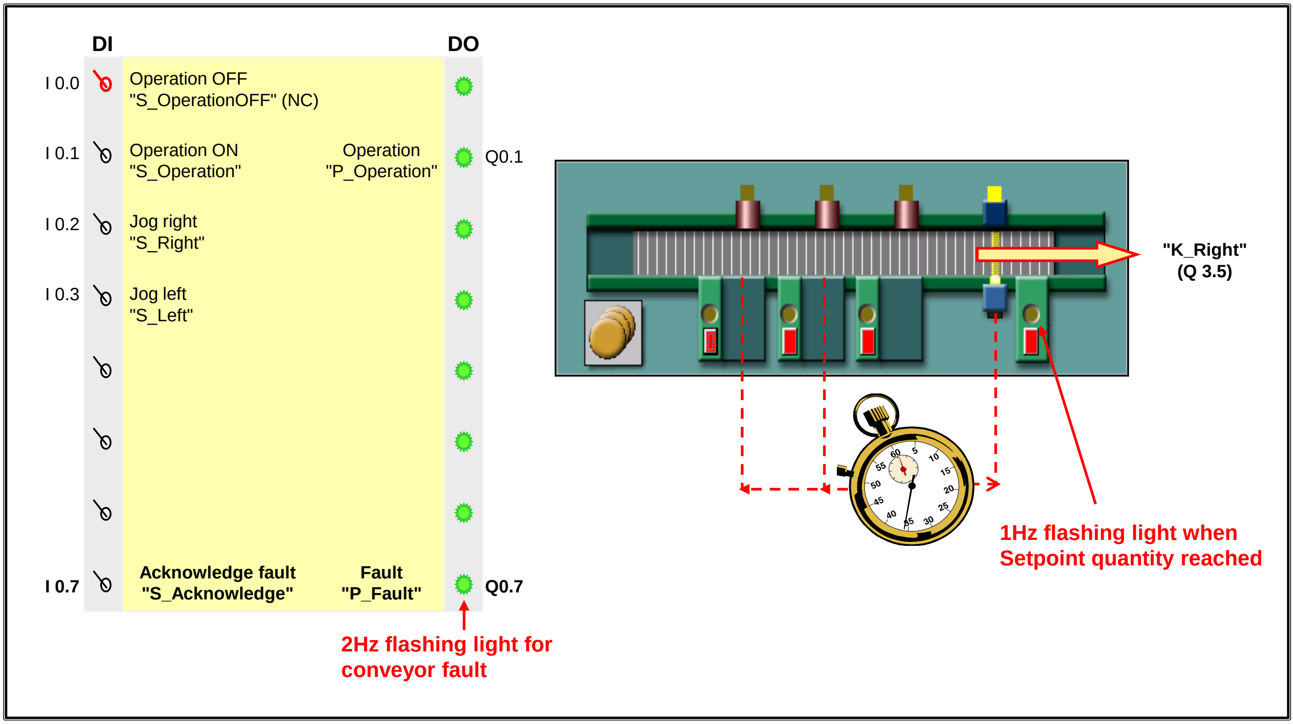
Task

You are to call the "FB_CountADD" block and assign it the relevant parameters.

What to Do:

1. In OB_Cycle, call the new "FB_CountADD" block.
2. Supply the formal parameters with the relevant actual parameters. (see picture)
3. Modify the blocks "FC_Conveyor" and "FC_Signal" in such a way that when the quantity of 3 is reached, it is no longer possible to transport a part, the LEDs of Bays 1 and 2 are switched off and the LED at the light barrier bay flashes with a 1 Hz frequency. For this, use the global variable "M_ActSetp"
4. Compile and save your project and check that it functions correctly.

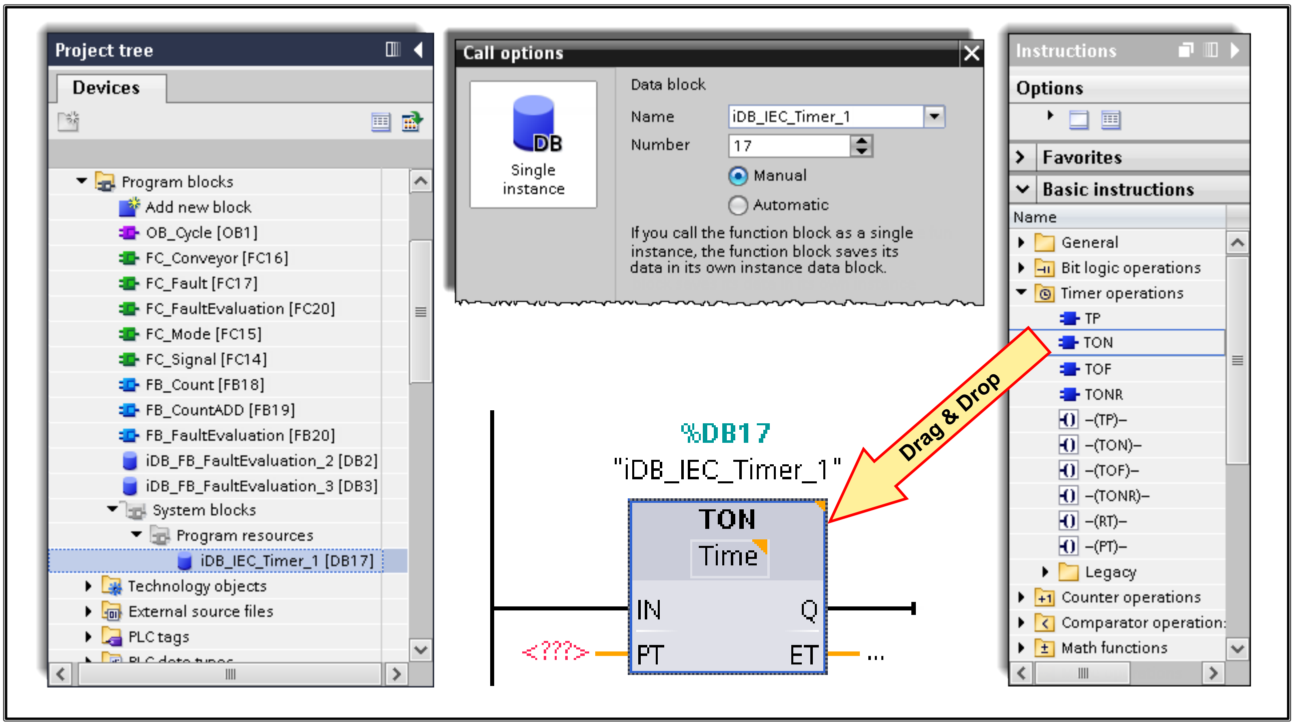
10.3. Task Description: Timed Monitoring of the Transport Sequences and Counting Parts using IEC Functions



Task Description

1. The automatic transport sequences are to be monitored for time with the help of an IEC function. The monitoring is to function as follows:
   - If a transport sequence takes longer than the 6 second monitoring time, there is a fault and the conveyor motor is automatically switched off.
   - A fault is displayed with a 2Hz flashing light on the simulator LED "P_Fault" (Q0.7).
   - A fault can be acknowledged via the simulator switch "S_Acknowledge" (I 0.7).
   - As long as there is an unacknowledged fault, the indicator lights "P_Bay1" (Q3.1) and "P_Bay2" (Q3.2) are dark and no new transport sequence can be started.
2. The counting of the transported parts is to be implemented with an IEC function.

10.3.1. IEC Timer



Call options

Data block

Name: iDB_IEC_Timer_1

Number: 17

● Manual

○ Automatic

If you call the function block as a single instance, the function block saves its data in its own instance data block.

Data Block

In addition to internally required variables, the timer function also stores the current already expired time in a data block which must be specified when programming the timer function. The specified data block is automatically generated by the Editor with exactly the internal structure that the timer function requires. The user has no further programming effort with this data block other than having to download it into the CPU.

Input Variable "PT"

The variable PT of the time function can be of the type Time or LTime.

Data Type TIME

The contents of a variable or constant of the data type TIME is interpreted as an integer number in milliseconds and stored in the memory as a 32-bit integer with sign. The representation contains information for days (d), hours (h), minutes (m), seconds (s) and milliseconds (ms).

Value range: T#-24d20h31m23s648ms to T#+24d20h31m23s647ms

Data Type LTIME

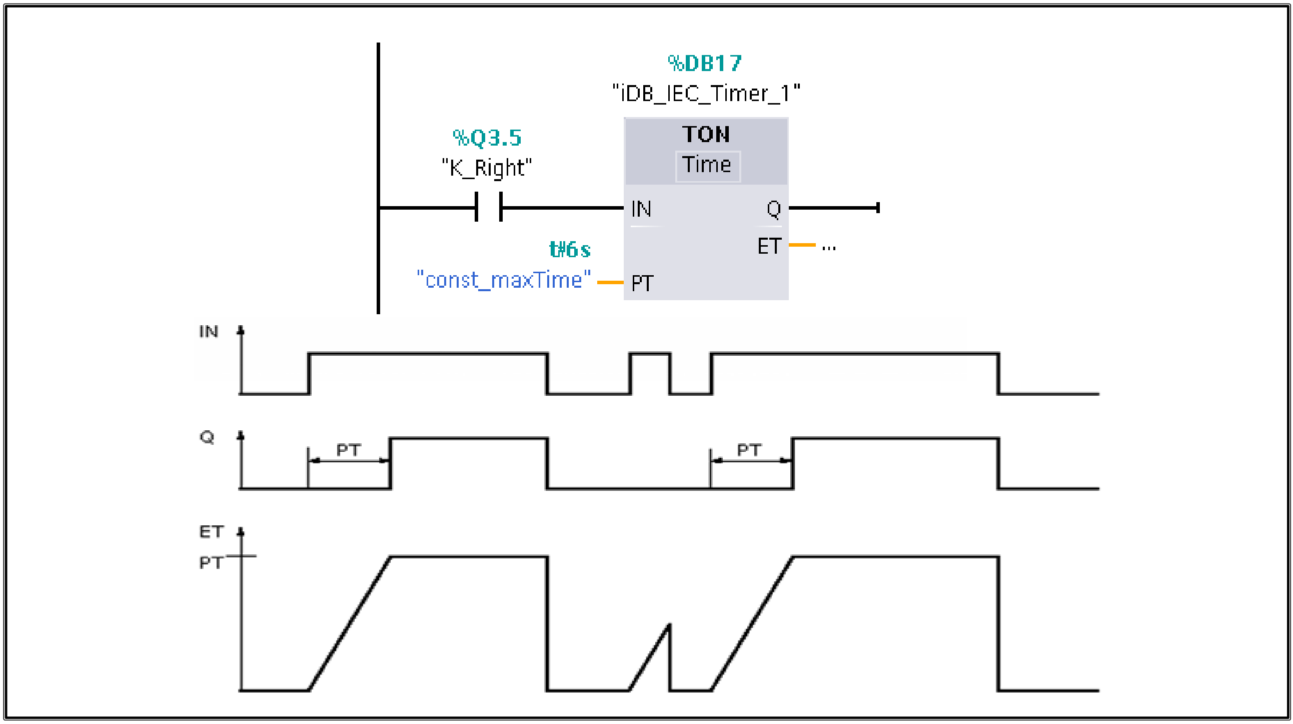
The contents of an operand of the data type LTIME is interpreted as nanoseconds. The representation contains information for days (d), hours (h), minutes (m), seconds (s), milliseconds (ms), microseconds (us) and nanoseconds (ns).

Value range: -106751d 23h 47m 16s 854ms 775us 808ns

to

+106751d 23h 47m 16s 854ms 775us 807ns

10.3.2. IEC Timer TON (ON Delay) Pulse Diagram

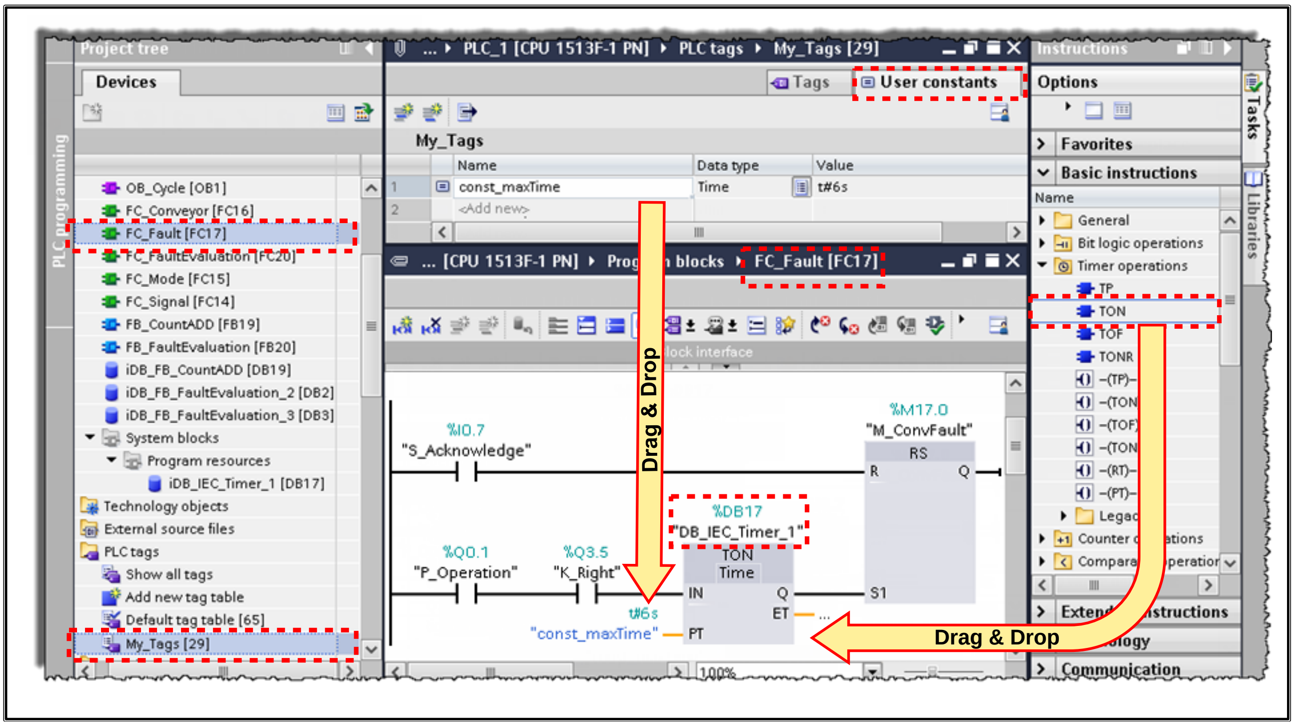


TON

The timer function TON (Timer on delay, "ON Delay") is started with a rising edge at input IN. So that the time expires, RLO must continue to be '1'. The timer function supplies a '1' signal at output Q, as soon as the specified time (variable or constant of data type TIME or LTIME) at input PT has expired and as long as the start signal at input IN still exists. The already expired time can be queried at output ET by passing a variable of data type TIME or LTIME.

Parameter	Data type	Memory area	Description
IN	BOOL	I, Q, M, D, L	Start input
PT	TIME	I, Q, M, D, L or constant	Duration by which the rising edge at input IN is delayed
Q	BOOL	I, Q, M, D, L	Output that is delayed by the time PT
ET	TIME	I, Q, M, D, L	Expired time

10.3.3. Exercise 3: Programming the Time Monitoring of the Transports in "FC_Fault"



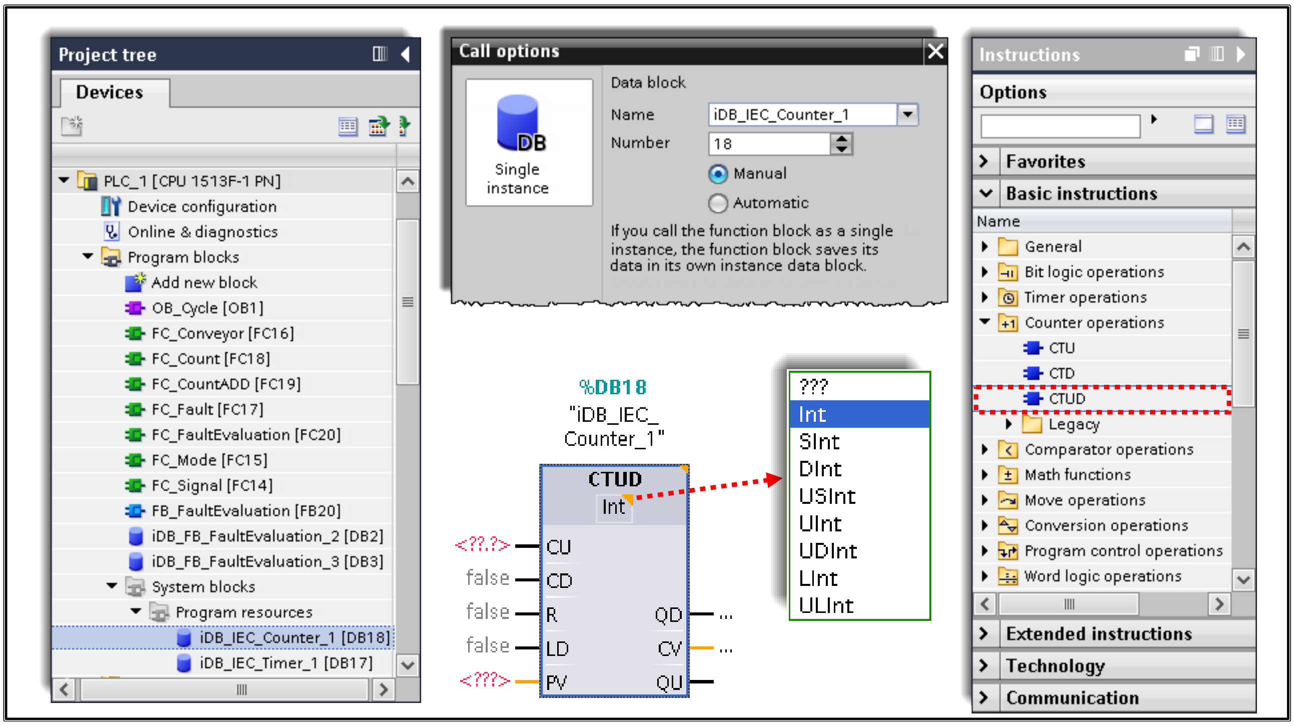
Task

The automatic transport sequences are to be monitored for time as previously described. If a transport sequence takes longer than 6 seconds, the conveyor motor is automatically switched off and the fault is displayed with a 2Hz flashing light on the simulator LED "P_Fault" (Q0.7). As long as a fault is not acknowledged "S_Acknowledge" (I 0.7) no new transport sequence can be started.

What to Do

1. In the PLC tag table "My_Tags", declare the user constant "const_max*Time*" of the data type *Time* with the value *T#6s* as shown in the picture.
2. Expand the "FC_Fault" block with the necessary functions:
   - At input IN, program the relevant start conditions (the timer must be started when an automatic transport sequence starts)
   - Pass the data block "iDB_IEC_Timer_1" as the instance-DB to the IEC timer function TON and as a time duration, the user constant "const_*maxTime*" (see picture).
   - In case the maximum transportation time is exceeded, set the memory bit "M_ConvFault" (M17.0), in order to be able to further logically link it in other blocks later on.
3. In the "FC_Conveyor" block, program the required switching off the conveyor motor when there is a conveyor fault.
4. In "FC_Signal", program the 2Hz flashing of the simulator LED "P_Fault" (Q0.7) and the lock-outs of the indicator lights "P_Bay1" (Q3.1) and "P_Bay2" (Q3.2) when a fault exists.
5. Download all modified blocks into the CPU, check the program function.
6. Save your project.

10.4. IEC Counters: CTU, CTD, CTUD



Call options

Data block

Name: iDB_IEC_Counter_1

Number: 18

● Manual

○ Automatic

If you call the function block as a single instance, the function block saves its data in its own instance data block.

Counters

Counters are used to count events, record quantities, etc. There are up counters and down counters as well as counters that can count in both directions.

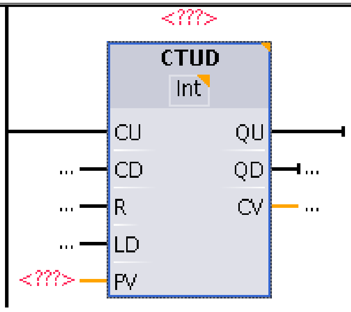
Value Range of a Counter

The count or value range of a counter depends on its data type (see picture) which is always an integer. The various selectable Integer data types merely differentiate themselves in their value range and thus determine the count range of the counter.

Instance Data Block

In addition to internally required variables, the counter also stores its current counter value in a so-called instance data block which must be specified when programming a counter. The specified instance data block is automatically generated by the Editor with exactly the internal structure that the counter requires. The user has no further programming effort with this data block other than having to download it into the CPU.

10.4.1. IEC Counters UP/DOWN: Inputs



Parameter	Declaration	Data type	Memory area	Description
CU	Input	BOOL	I, Q, M, D, L	Count up input
CD	Input	BOOL	I, Q, M, D, L	Count down input
R	Input	BOOL	I, Q, M, D, L	Reset input
LD	Input	BOOL	I, Q, M, D, L	Load input
PV	Input	Integers (INT)	I, Q, M, D, L or constant	Value to which the output QU is set. / Value to which the output CV is set when LD = 1.

Input CU and CD

With a positive edge at input CU, the current count is increased by one; with a positive edge at input CD, the current count is decreased by 1. This means that the user **doesn’t** have to program an edge evaluation.
If a positive edge is detected at both inputs simultaneously or in the same cycle, the current count remains unchanged. If the upper or lower limit of the specified data type is reached, the count is no longer increased or decreased for a positive edge at CU or CD.

Input R

The input R acts statically, that is, as long as RLO '1' is at input R, the count is set to 0 and rising edges or RLO '1' at the inputs CU, CD and LOAD have no effect on the current count.

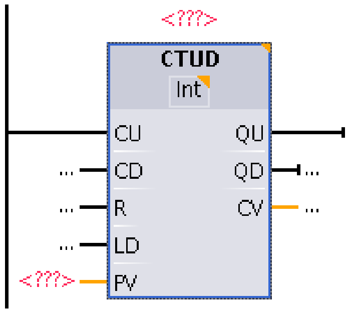
Input LD (Load, only for down counters)

The input LD acts statically, that is, as long as RLO '1' is at input LOAD, the current count is set to the value that is passed to the input PV, and positive edges at the inputs CU and CD have no effect on the count.

Input PV

The value to which the count is to be set must be passed to the input PV as long as RLO '1' is at input LD. The variable or constant passed to the input must be compatible with the data type of the counter.

10.4.2. IEC Counters UP/DOWN: Outputs



Parameter	Declaration	Data type	Memory area	Description
QU	Output	BOOL	I, Q, M, D, L	Current count value greater than or equal to the value at input "PV"
QD	Output	BOOL	I, Q, M, D, L	Current count value less than or equal to "0"
CV	Output	Integers (INT)	I, Q, M, D, L	Current count value

Output QU

The current status of the up counter can be checked at the output QU. As long as the current count is greater than or equal to the value of the parameter PV, the output QU has Status '1', otherwise, Status '0'.

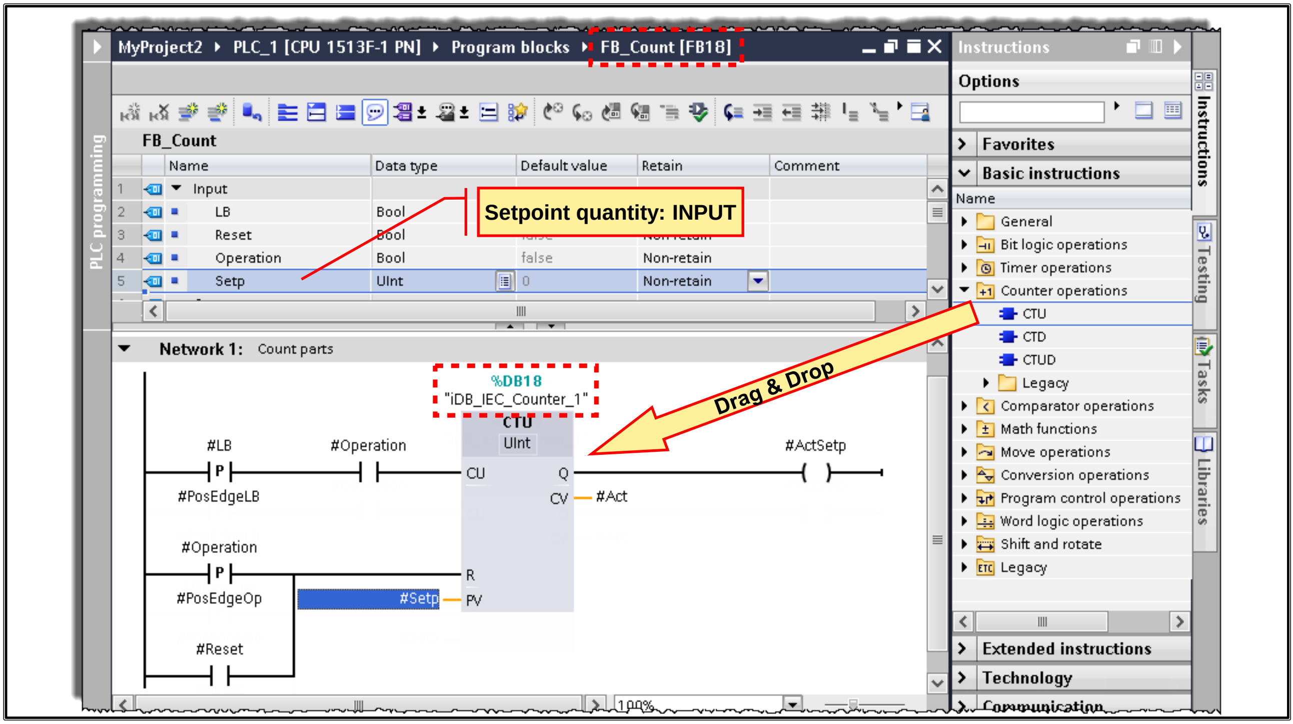
Output QD (Only for down counters)

The current status of the down counter can be checked at the output QD. As long as the current count is less than or equal to zero, the output QD has Status "1", otherwise, Status '0'.

Output CV

The current count is output at output CV. The variable passed to the output must be compatible with the data type of the counter.

10.4.3. Exercise 4: Counting the Transported Parts using an IEC Counter



FB_Count

	Name	Data type	Default value	Retain	Comment
1	▼ Input				
2	LB	Bool			
3	Reset	Bool	false	Non-retain	
4	Operation	Bool	false	Non-retain	
5	Setp	UInt	0	Non-retain	

Task

- When "P_Operation" (Q0.1) is switched on, the transported parts are to be counted as soon as they have passed the light barrier "B_LB" (I 3.0) ("B_LB" 0->1). The number of transported parts (ACTUAL quantity) is to be recorded with a counter and stored in the static variable #ACT.
- When the setpoint quantity (constant value 3) is reached, then no new transport sequence can take place which is also visible on the LEDs "P_Bay1" and "P_Bay2".
- The counter can be acknowledged (reset to 0) at any time via the pushbutton "S_BayLB" (I 3.4). When the operation is switched on, that is, with a positive edge at "P_Operation" (Q0.1), the ACTUAL quantity is also reset to 0.

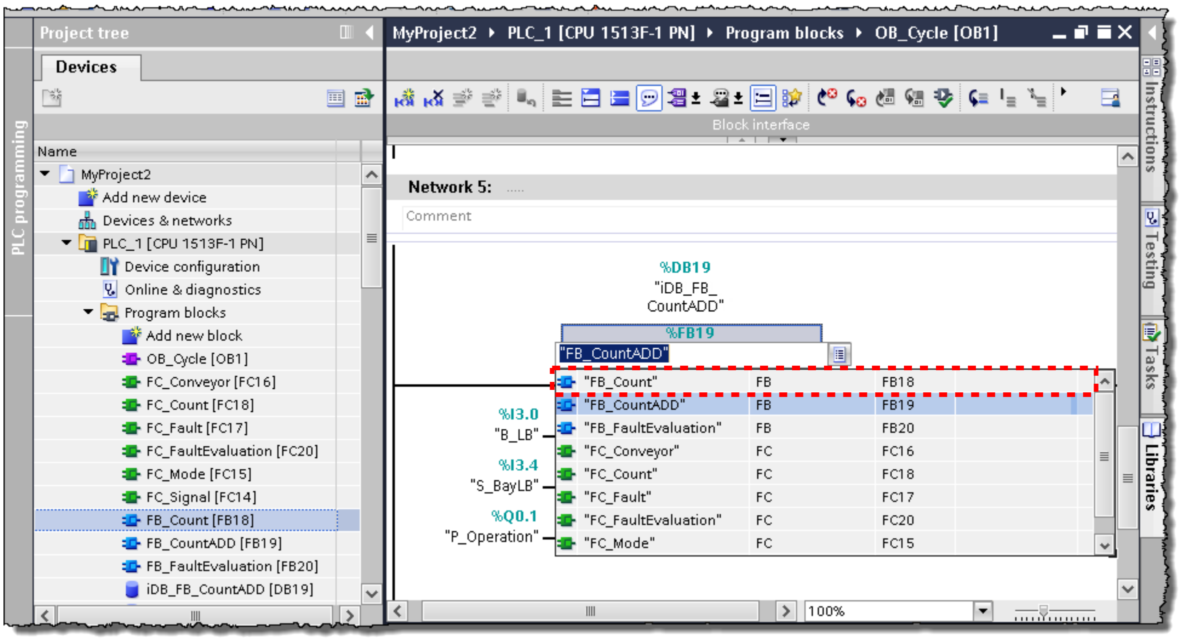
What to Do:

1. Create the new function block "FB_Count".
2. Declare the necessary parameters and variables:

Input:	- LB	BOOL,	Light barrier signal
	- Reset	BOOL,	Reset the actual no. of parts to 0
	- Operation	BOOL,	Operation mode
Output:	- ActSetp	BOOL,	the actual no. of parts has reached the setpoint value
Static:	- PosEdgeLB	BOOL,	auxiliary bit for the edge evaluation of the Light barrier signal
	- PosEdgeOp	BOOL,	auxiliary bit for the edge evaluation of the Operation mode signal
	- Act	UINT,	actual no. of transported parts
Constant:	- const_Setp	UINT,	setpoint no. of parts to be transported

3. Program the new "FB_CountADD" block with the appropriate arithmetic operations and don't use any global variables (tags) to do so. For the edge evaluation of the light barrier, use the static variable #PosEdgeLB and save the current quantity of the transported parts in the static variable #ACT.

10.4.4. Exercise 5: Replacing the "FB_CountADD" Call with the "FB_Count" Call

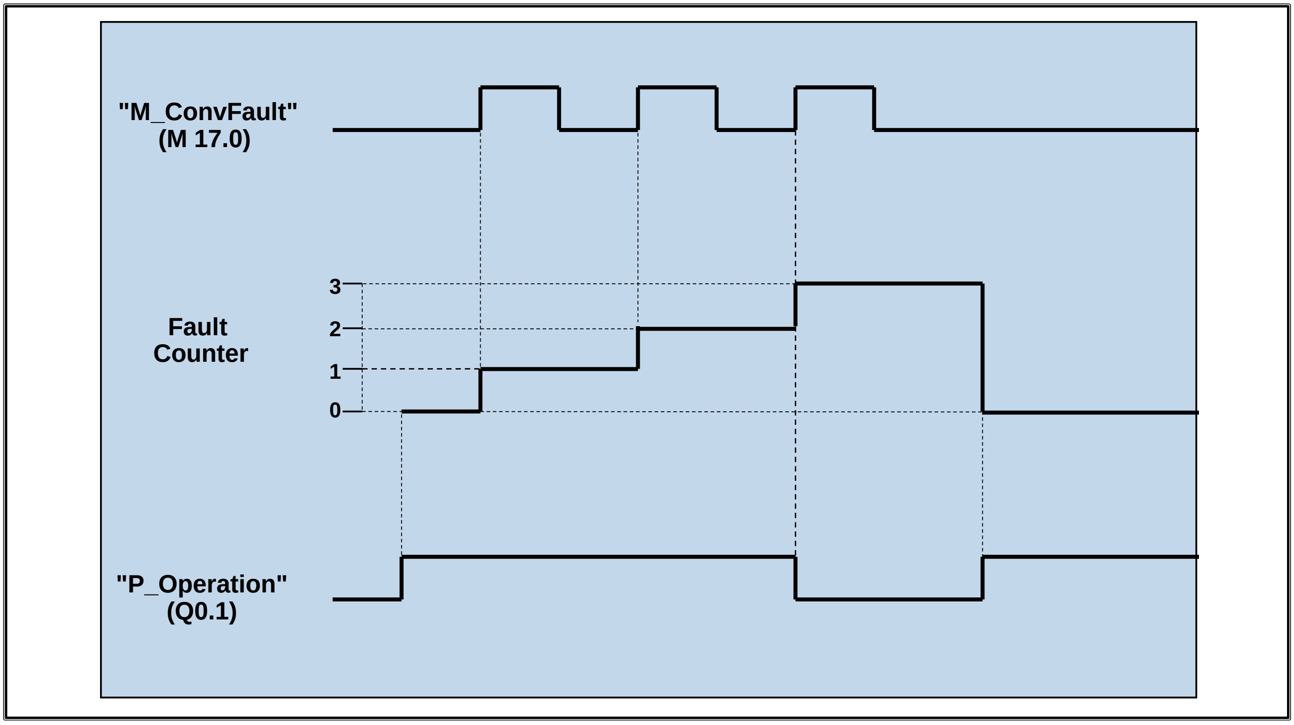


"FB_Count"	FB	FB18
"FB_CountADD"	FB	FB19
"FB_FaultEvaluation"	FB	FB20
"FC_Conveyor"	FC	FC16
"FC_Count"	FC	FC18
"FC_Fault"	FC	FC17
"FC_FaultEvaluation"	FC	FC20
"FC_Mode"	FC	FC15

What to Do

1. In the tag table "My_Tags", create a new tag "const_Setp" of the type UINT and give it the value ‘3’.
2. Replace the call of "FB_CountADD" with the call of "FB_Count".
   For this use the function "Changing the block call" in which the actual parameters are adopted.
3. Create a new instance (iDB_FB_Count) for the block calls by right-clicking on the call of the FB and then selecting the context menu item "Change instance…".
4. Interconnect the new input formal parameter "Setp" with the new global constant "const_Setp"
5. Download all modified blocks into the CPU and check the program function.
6. Save your project.

10.5. Additional Exercise 6: Counting the Conveyor Faults by Expanding "FC_Fault"



Function Up Until Now

When "P_Operation" (Q0.1) is switched on, the transport sequences are monitored for time. If a transport sequence takes longer than the monitoring time of 6 seconds, there is a conveyor fault and the conveyor motor is automatically switched off (logically linked in "FC_Conveyor").

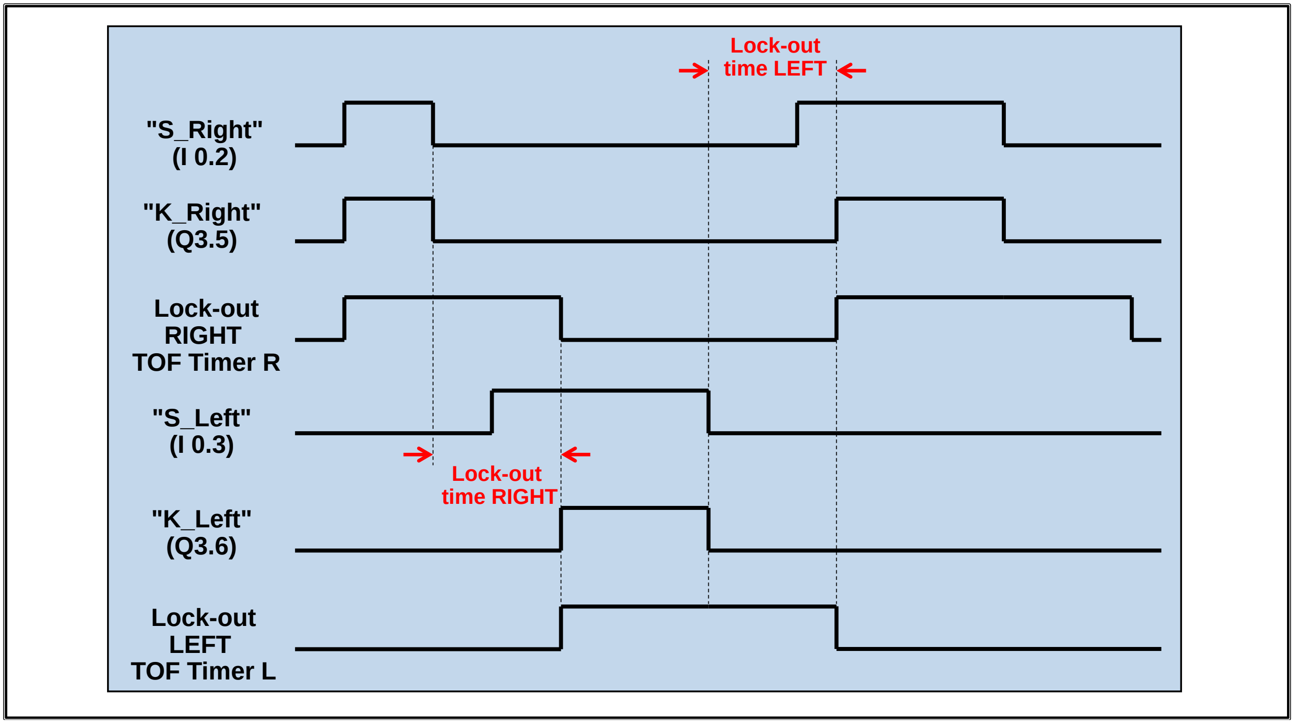
Task:

When "P_Operation" (Q0.1) is switched on, the conveyor faults are to be counted. After 3 conveyor faults have occurred, "P_Operation" (Q0.1) is to be switched off for safety reasons. To start a new transport sequence, the fault (as already programmed) must be acknowledged and "P_Operation" (Q0.1) must be switched on once again.

What to Do:

1. In "FC_Fault in a new network, program the counting of the conveyor faults. The counter counts up ‘1’ every time a conveyor fault occurs ("M_ConvFault" (M17.0) = “1“).
2. In "FC_Fault", pass the value at counter output Q to any memory bit you like.
3. In "FC_Mode", program the switching off (reset) of "P_Operation" (Q0.1) after three conveyor faults. For this, use the memory bit to which you passed the counter output Q in "FC_Fault".
4. Download all modified blocks into the CPU and check the program function.
5. Save your project.

10.5.1.1. Additional Exercise 7: Timely Lock-out of the Conveyor Motor Jogging



Function Up Until Now

When the indicator light of "P_Operation" (Q0.1) is switched off, the conveyor motor can be jogged to the RIGHT and LEFT using the simulator switches "S_Right" (I 0.2) and "S_Left" (I 0.3).

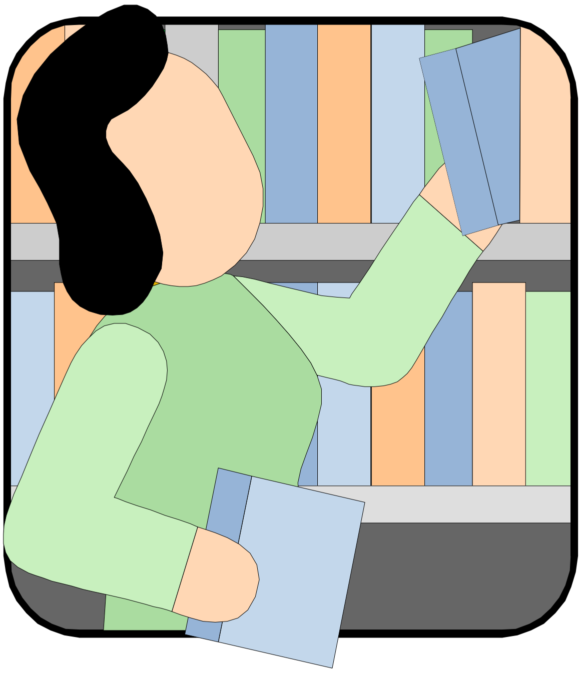
Task:

In order to avoid too great a load change, it should only be possible to jog the conveyor motor in the opposite direction after it has been jogged to the RIGHT or to the LEFT after a lock-out time of 2 seconds (see picture). If, for example, the motor has been jogged to the RIGHT, then it can only be jogged back to the LEFT after the lock-out time of 2 seconds has expired.

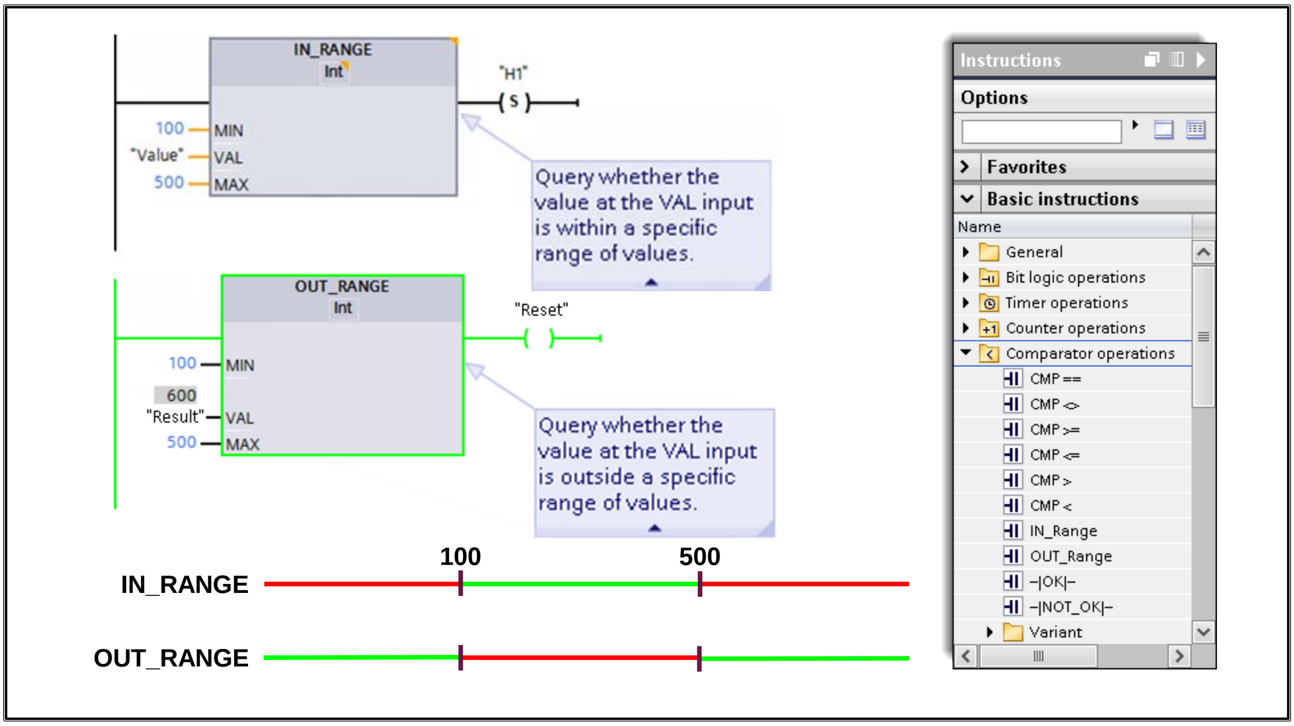
What to Do:

1. In "FC_Conveyor", program two TOF timers (Off Delay) as the lock-out timers RIGHT and LEFT and assign one memory bit each to the Timer result Q.
2. Gate these memory bits to the jog conditions.
3. Download the modified "FC_Conveyor" into the CPU and check the program function.

10.6. Additional Information



10.6.1. Comparator Operations: IN_RANGE, OUT_RANGE



Query whether the value at the VAL input is within a specific range of values.

Query whether the value at the VAL input is outside a specific range of values.

IN_RANGE

You can use the "Value within range" instruction to determine if the value at the VAL input is within a specific value range. You specify the limits of the value range with the MIN and MAX inputs (parameters). If the value at the VAL input fulfills the comparison MIN <= VAL or VAL <= MAX, the box output has the signal status "1". If the comparison is not fulfilled, the box output has the signal status "0".

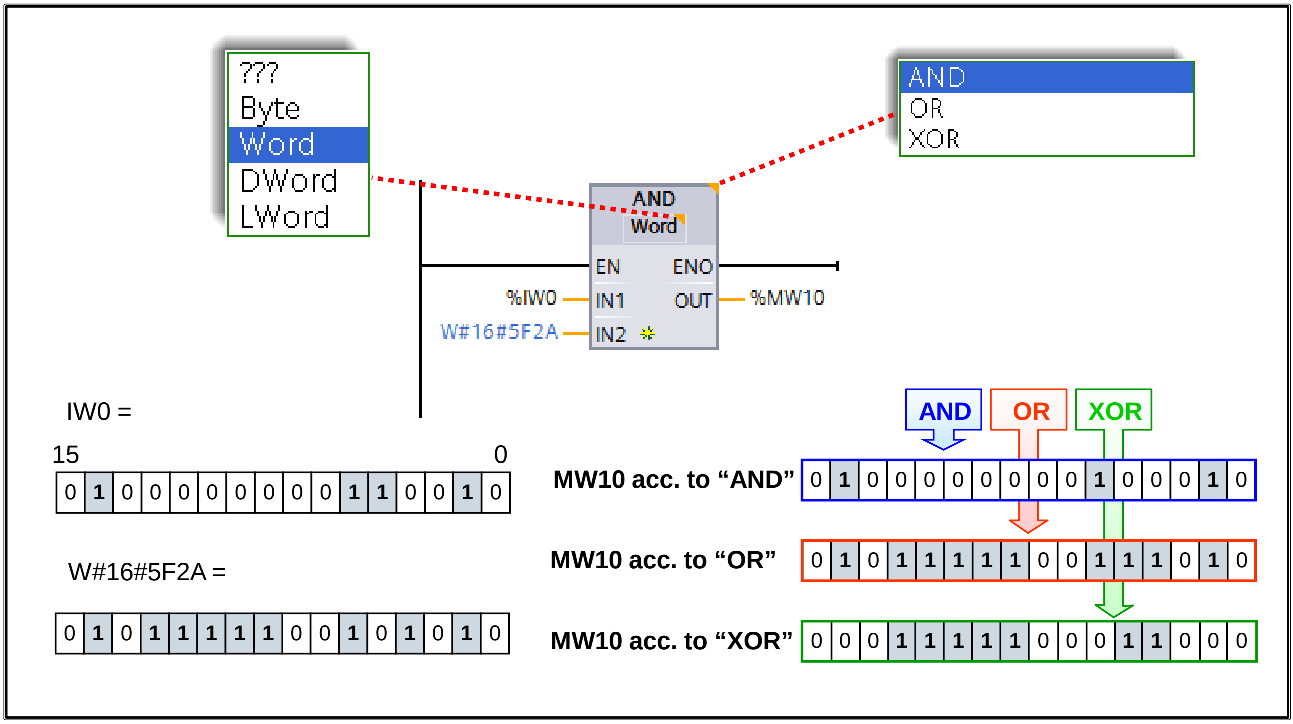
The comparison function can only be executed if the values to be compared are of the same data type.

OUT_RANGE

You can use the "Value outside range" instruction to determine if the value at the VAL input is outside of a specific value range. You specify the limits of the value range with the MIN and MAX inputs (parameters). If the value at the VAL input fulfills the comparison MIN > VAL or VAL > MAX, the box output has signal status "1". If the comparison is not fulfilled, the box output has the signal status "0".

The comparison function can only be executed if the values to be compared are of the same data type.

10.6.2. Digital Logic Operations



IW0 =

15 0

0 1 0 0 0 0 0 0 0 0 1 1 0 0 1 0

W#16#5F2A =

0 1 0 1 1 1 1 1 0 0 1 0 1 0 1 0

MW10 acc. to “AND” 0 1 0 0 0 0 0 0 0 0 1 0 0 0 1 0

MW10 acc. to “OR” 0 1 0 1 1 1 1 1 0 0 1 1 1 0 1 0

MW10 acc. to “XOR” 0 0 0 1 1 1 1 1 0 0 0 1 1 0 0 0

AND

The "AND" instruction gates (combines) the two digital values at inputs IN1 and IN2 bit-by-bit in accordance with the AND truth table. The result of the AND operation is stored at output OUT. The instruction is executed when EN = 1.

Example: Masking out the 4th decade of the thumbwheel buttons:

IW2= = 0100 0100 1100 0100
W#16#0FFF = 0000 1111 1111 1111

MW30 = 0000 0100 1100 0100

OR

The "Or" instruction gates (combines) the two digital values at inputs IN1 and IN2 bit-by-bit in accordance with the OR truth table. The result of the OR operation is stored at output OUT. The instruction is executed when EN = 1.

Example: Setting bit 0 in MW32:

MW32 = 0100 0010 0110 1010
W#16#0001 = 0000 0000 0000 0001

MW32 = 0100 0010 0110 1011

XOR

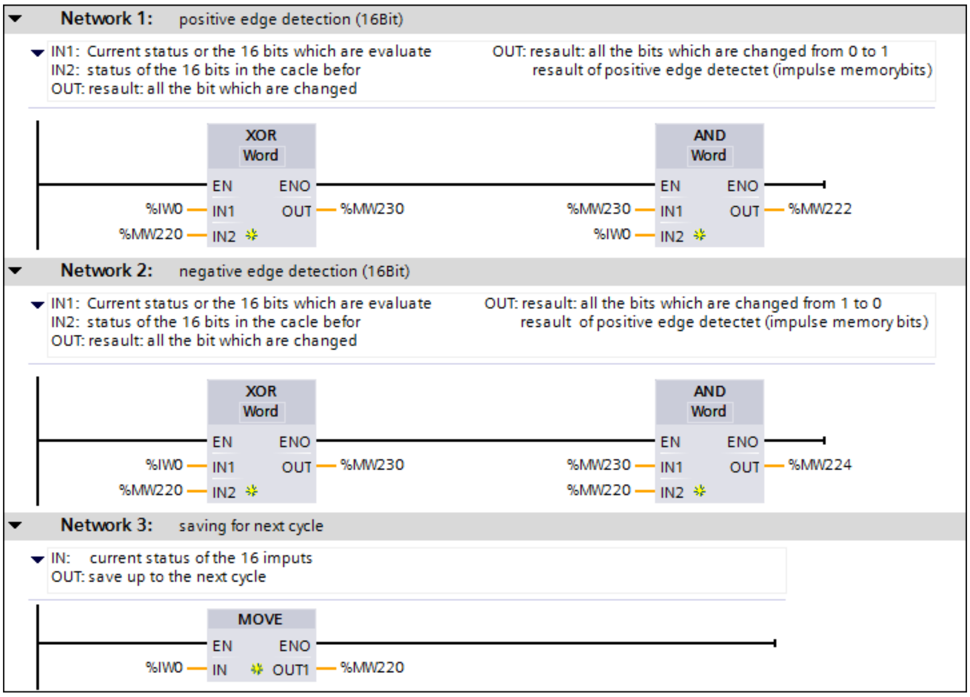
The "Exclusive OR" instruction gates (combines) the two digital values at inputs IN1 and IN2 bit-by-bit in accordance with the XOR truth table. The result of the XOR operation is stored at output OUT. The instruction is executed when EN=1.

Example: Detecting signal changes in IW0:

IW0 = 0100 0100 1100 1010
MW28 = 0110 0010 1011 1001

MW24 = 0010 0110 0111 0011

10.6.2.1. Application Example: Digital Edge Evaluation



Network 1: positive edge detection (16Bit)

IN1: Current status or the 16 bits which are evaluate
IN2: status of the 16 bits in the cacle befor
OUT: resault: all the bit which are changed

OUT: resault: all the bits which are changed from 0 to 1
resault of positive edge detectet (impulse memorybits)

Network 2: negative edge detection (16Bit)

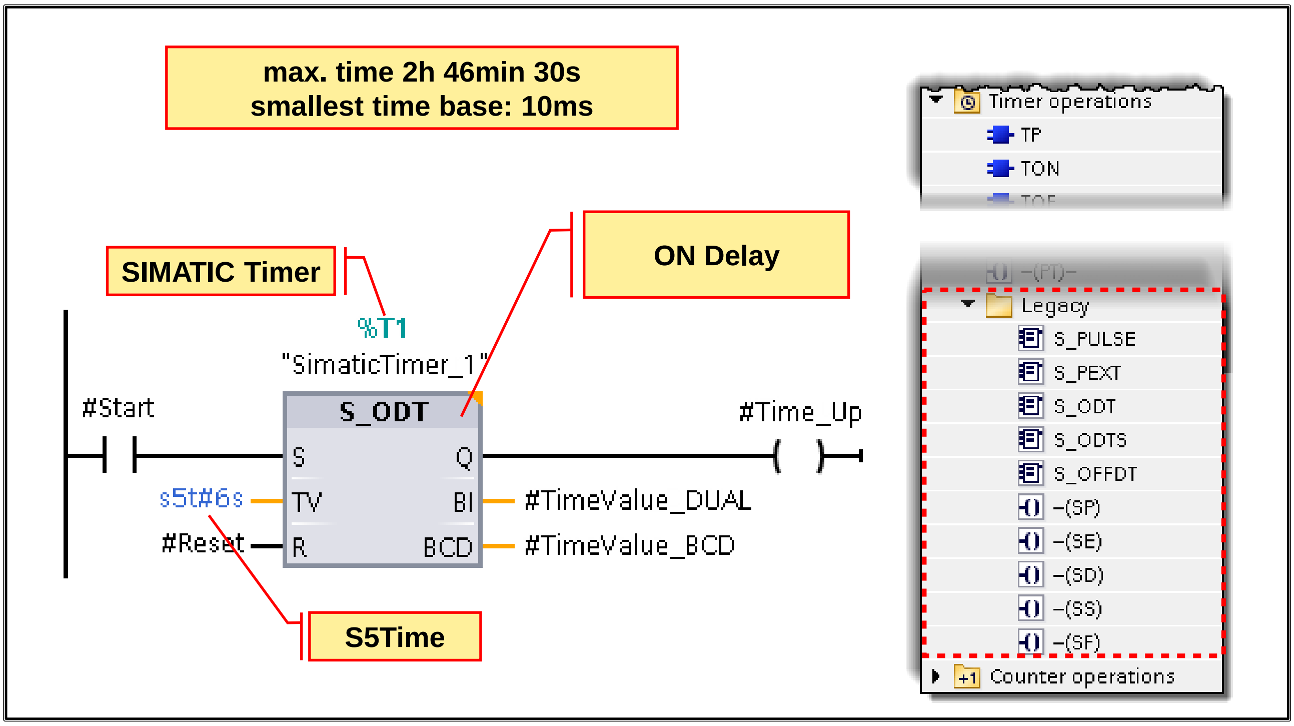
IN1: Current status or the 16 bits which are evaluate
IN2: status of the 16 bits in the cacle befor
OUT: resault: all the bit which are changed

OUT: resault: all the bits which are changed from 1 to 0
resault of positive edge detectet (impulse memory bits)

Network 3: saving for next cycle

IN: current status of the 16 imputs
OUT: save up to the next cycle

10.6.3. SIMATIC-Timer



For reasons of compatibility to STEP5, you have the possibility of using SIMATIC-Timers in an S7-1500. These timers are global. During the Start, the accuracy of the value range and what kind of timer it is (ON Delay, OFF Delay etc.) is defined.

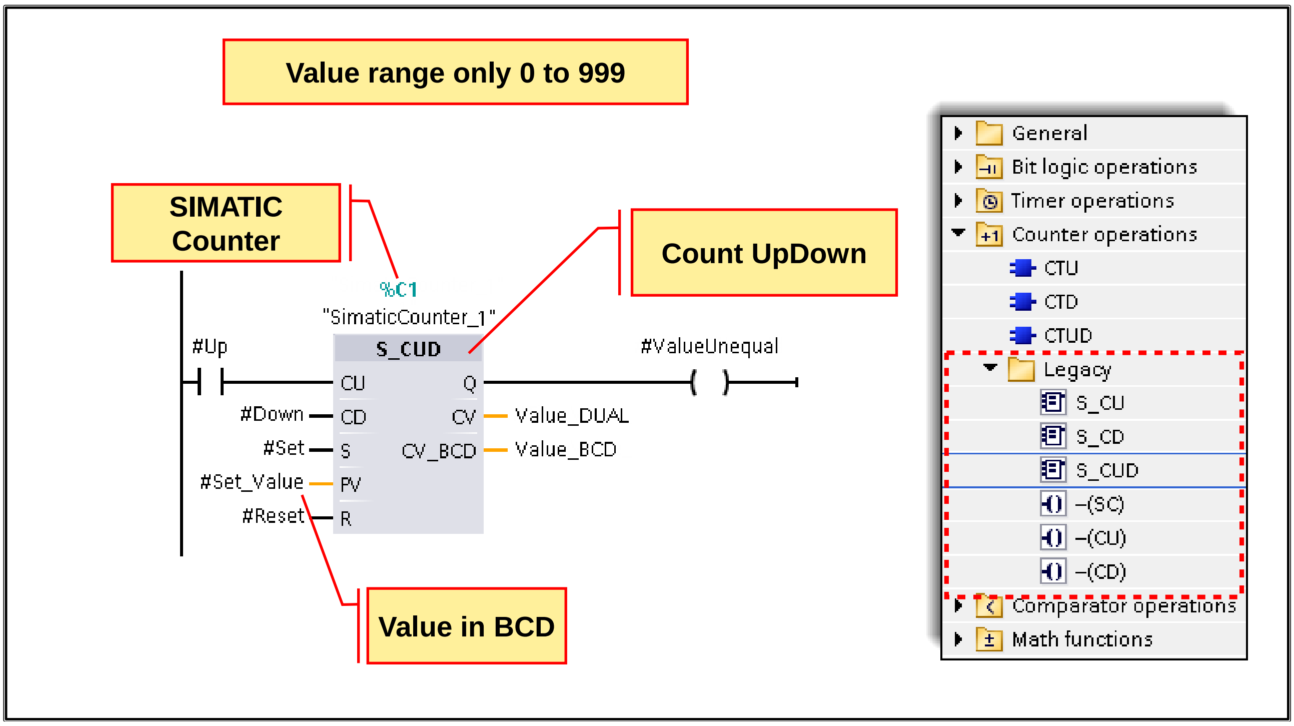
SIMATIC-Timers have the following system properties:

- They are assigned to a fixed number,
- They cannot be used repeatedly,
- The time frame lies between 0.01 and 9.99, 0.1 and 99.9, 1 and 999 or 10 and 9990s
- The accuracy depends on the selected time frame 10ms, 100ms, 1s or 10s

Note

In the time-cell (in the picture T1 of the memory area Timer), the operating system reduces the time value in an interval (time frame) defined by the time base by one unit respectively until the time value equals "0". The reduction occurs asynchronous to the user program. Consequently, the resulting time is maximally up to one time interval of the time base shorter than the desired time value.

10.6.3.1. SIMATIC-Counter



Just as with SIMATIC-Timers, SIMATIC-Counters can also be used for reasons of compatibility to STEP5. The counters are also global.

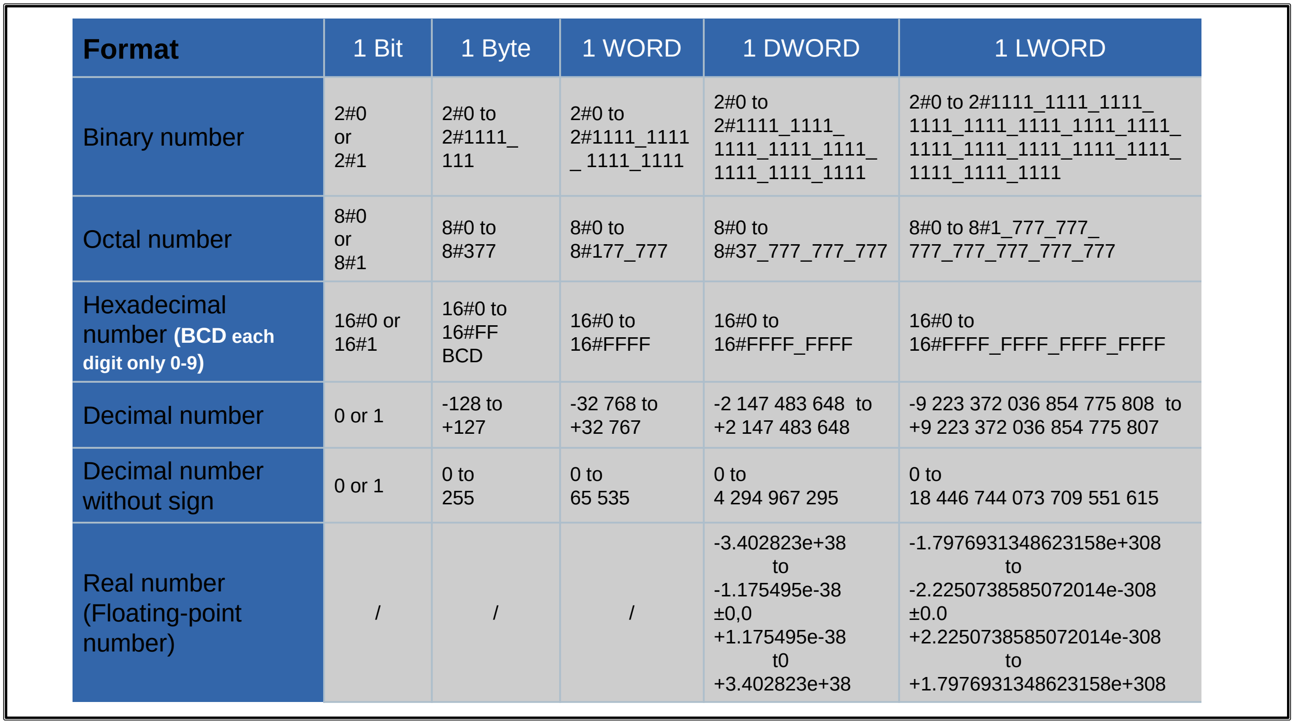
SIMATIC-Counters have the following system properties:

- They are assigned to a fixed number,
- They cannot be used repeatedly,
- The value range lies between 0 and max. 999

Note

If you use a SIMATIC-Counter, then only use it in one location in the program, in order to avoid count errors.

10.6.4. Value Ranges of Various Number Formats



Format	1 Bit	1 Byte	1 WORD	1 DWORD	1 LWORD
Binary number	2#0 or 2#1	2#0 to 2#1111_ 111	2#0 to 2#1111_1111 _ 1111_1111	2#0 to 2#1111_1111_ 1111_1111_1111_ 1111_1111_1111	2#0 to 2#1111_1111_1111_ 1111_1111_1111_1111_1111_ 1111_1111_1111_1111_1111_ 1111_1111_1111
Octal number	8#0 or 8#1	8#0 to 8#377	8#0 to 8#177_777	8#0 to 8#37_777_777_777	8#0 to 8#1_777_777_ 777_777_777_777_777
Hexadecimal number **(BCD each digit only 0-9)**	16#0 or 16#1	16#0 to 16#FF BCD	16#0 to 16#FFFF	16#0 to 16#FFFF_FFFF	16#0 to 16#FFFF_FFFF_FFFF_FFFF
Decimal number	0 or 1	-128 to +127	-32 768 to +32 767	-2 147 483 648 to +2 147 483 648	-9 223 372 036 854 775 808 to +9 223 372 036 854 775 807
Decimal number without sign	0 or 1	0 to 255	0 to 65 535	0 to 4 294 967 295	0 to 18 446 744 073 709 551 615
Real number (Floating-point number)	/	/	/	-3.402823e+38 to -1.175495e-38 ±0,0 +1.175495e-38 t0 +3.402823e+38	-1.7976931348623158e+308 to -2.2250738585072014e-308 ±0.0 +2.2250738585072014e-308 to +1.7976931348623158e+308