

# **Industrial Automation**

## **(Automação de Processos Industriais)**

### **PLC Programming languages**

#### ***Structured Text***

<http://users.isr.ist.utl.pt/~jag/courses/api1415/api1415.html>

Slides 2010/2011 Prof. Paulo Jorge Oliveira  
Rev. 2011-2015 Prof. José Gaspar

# Syllabus:

## Chap. 2 – Introduction to PLCs [2 weeks]

...

## Chap. 3 – PLC Programming languages [2 weeks]

Standard languages (IEC-1131-3):

*Ladder Diagram; Instruction List, and Structured Text.*

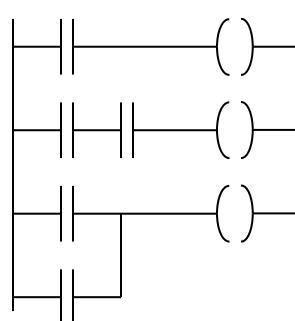
Software development resources.

...

## Chap. 4 - GRAFCET (*Sequential Function Chart*) [1 week]

## PLC Programming Languages (IEC 61131-3)

### *Ladder Diagram*



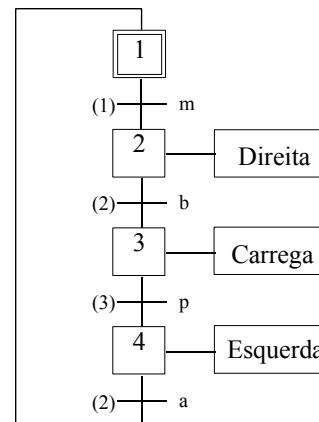
### *Structured Text*

```
If %I1.0 THEN  
    %Q2.1 := TRUE  
ELSE  
    %Q2.2 := FALSE  
END_IF
```

### *Instruction List*

LD	%M12
AND	%I1.0
ANDN	%I1.1
OR	%M10
ST	%Q2.0

### *Sequential Function Chart (GRAFCET)*



## Structured Text

```
(*  
Searching for the first element that is not zero in a  
table of 32 words (table = words %MW100 till %MW131).
```

**Input:**

```
%M0 works as an enable bit (run search iff %M0 is 1)  
%MW100 till %MW131 is the table to search
```

**Output:**

```
%M1 is set to 1/0 if the not zero element was/was-not found  
%MW10 is the non-zero value found  
%MW11 is the location of the non-zero value
```

**Auxiliary:**

```
%MW99 is the table index  
*)
```

```
IF %M0 THEN  
    FOR %MW99:=0 TO 31 DO  
        IF %MW100[%MW99]<>0 THEN  
            %MW10:=%MW100[%MW99];  
            %MW11:=%MW99;  
            %M1:=TRUE;  
            EXIT; (* exit the loop *)  
        ELSE  
            %M1:=FALSE;  
        END_IF;  
    END_FOR;  
ELSE  
    %M1:=FALSE;  
END_IF;
```

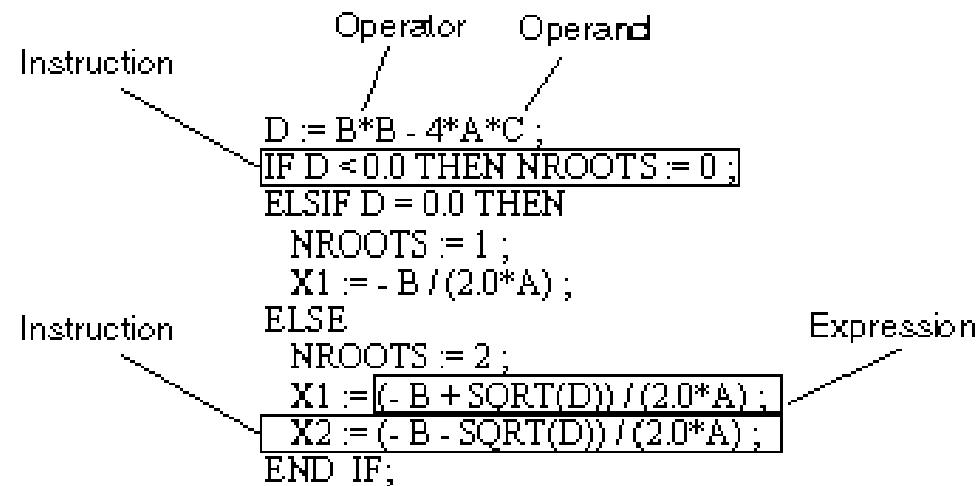
## Structured Text

**PLC Program = {Sections}, Section = {Sequences}**

One sequence is equivalent to one or more rungs in *ladder diagram*.

Each section can be programmed in Ladder, Instruction List, or **Structured Text**

Representation of  
an ST section:



The **length of an instruction line** is limited to 300 characters. The **length of an ST section is not limited** within the programming environment. The length of an ST section is only limited by the size of the PLC memory.

## Structured Text

### Basic Instructions

#### *Load*

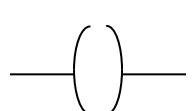
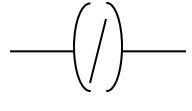
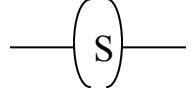
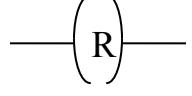
<b>:=</b>		Open contact: contact is active (result is 1) while the control bit is 1.
<b>:=NOT</b>		Close contact: contact is active (result is 1) while the control bit is 0.
<b>:=RE</b>		Contact in the rising edge: contact is active during a scan cycle where the control bit has a rising edge.
<b>:=FE</b>		Contact in the falling edge: contact is active during a scan cycle where the control bit has a falling edge.

Examples: %M0 := %I0.2.0;      %M0 := NOT %I0.2.0;    %M0 := RE (%I0.2.0);

## Structured Text

### Basic Instructions

#### *Store*

<b>:=</b>		The result of the logic function activates the coil.
<b>:=NOT</b>		The inverse result of the logic function activates the coil.
<b>SET</b>		The result of the logic function energizes the relay (sets the latch).
<b>RESET</b>		The result of the logic function de-energizes the relay (resets the latch)..

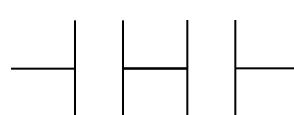
Examples: `%MW100 :=123; %Q0.4.0 :=NOT %M1; %M0 :=TRUE; SET (%Q0.4.0);`

## Structured Text

### Basic Instructions

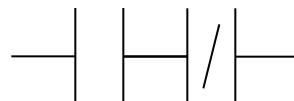
#### *AND*

**AND**



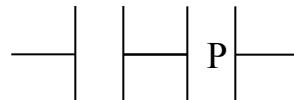
AND of the operand with the result of the previous logical operation.

**AND(NOT...)**



AND of the operand with the inverted result of the previous logical operation.

**AND(RE...)**



AND of the rising edge with the result of the previous logical operation.

**AND(FE...)**



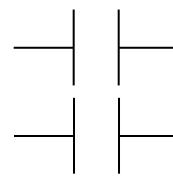
AND of the falling edge with the result of the previous logical operation.

## Structured Text

### Basic Instructions

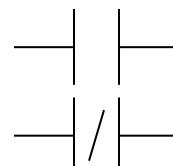
#### *OR*

**OR**



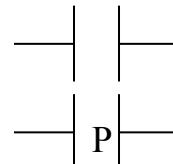
OR of the operand with the result of the previous logical operation.

**OR(NOT...)**



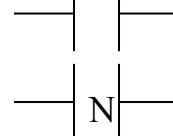
OR of the operand with the inverted result of the previous logical operation.

**OR(RE...)**



OR of the rising edge with the result of the previous logical operation.

**OR(FE...)**



OR of the falling edge with the result of the previous logical operation.

## Structured Text

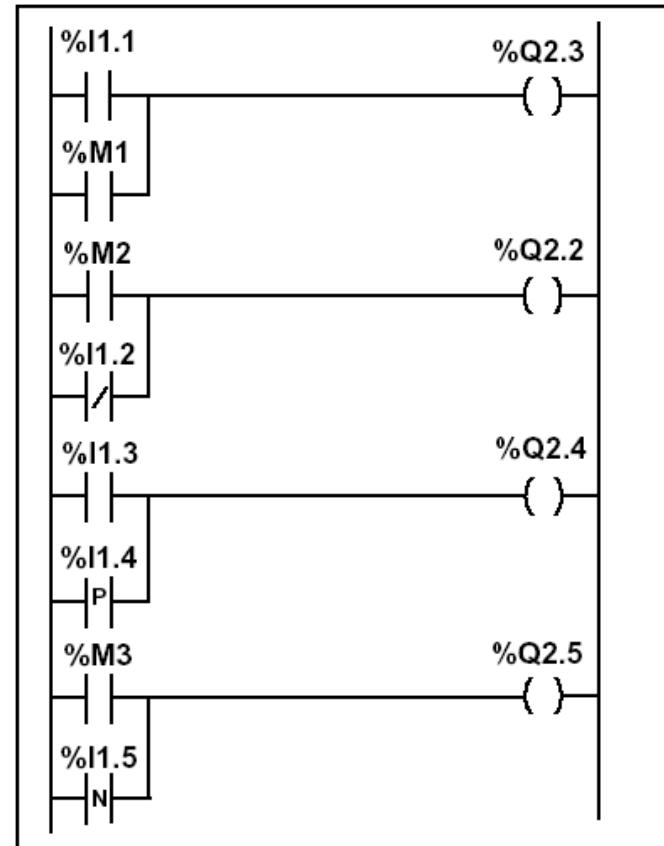
### Example:

PL7 (Micro PLC):

```
%Q2.3 := %I1.1 OR %M1;  
%Q2.2 := %M2 OR (NOT %I1.2);  
%Q2.4 := %I1.3 OR (RE %I1.4);  
%Q2.5 := %M3 OR (FE %I1.5);
```

Unity Pro (Premium PLC):

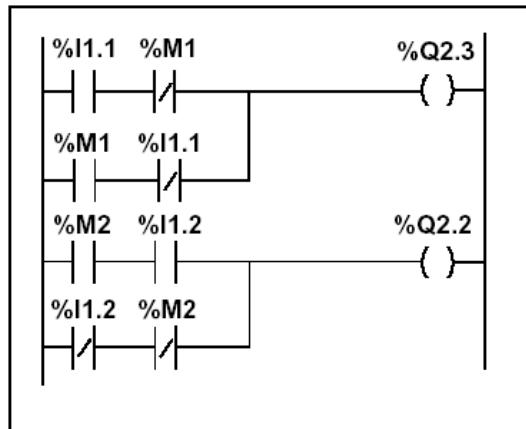
```
%Q0.4.3 := %I0.2.1 OR %M1;  
%Q0.4.2 := %M2 OR (NOT %I0.2.2);  
%Q0.4.4 := %I0.2.3 OR RE (%I0.2.4);  
%Q0.4.5 := %M3 OR FE (%I0.2.5);
```



# Structured Text

## Basic Instructions

### XOR



```
%Q2.3 := %I1.1 XOR %M1;
%Q2.2 := %M2 XOR (NOT %I1.2);
%Q2.4 := %I1.3 XOR (RE %I1.4);
%Q2.5 := %M3 XOR (FE %I1.5);
```

Instruction list	Structured text	Description	Timing diagram
XOR	XOR	OR Exclusive between the operand and the previous instruction's Boolean result	
XORN	XOR (NOT...)	OR Exclusive between the operand inverse and the previous instruction's Boolean result	
XORR	XOR (RE...)	OR Exclusive between the operand's rising edge and the previous instruction's Boolean result	
XORF	XOR (FE...)	OR Exclusive between the operand's falling edge and the previous instruction's Boolean result.	

Unity Pro (Premium PLC):

```
%Q0.4.3 := %I0.2.1 XOR %M1;
%Q0.4.4 := %I0.2.3 XOR RE(%I0.2.4);
```

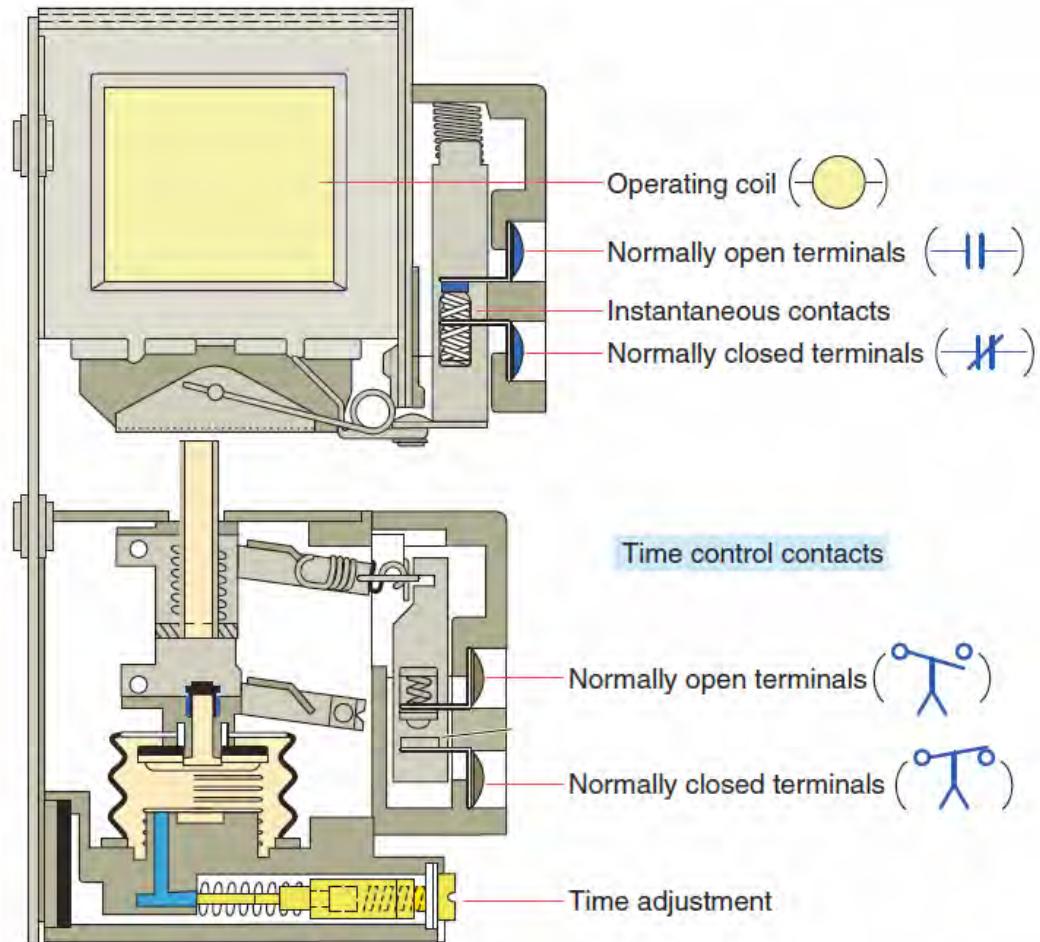
```
%Q0.4.2 := %M2 XOR (NOT %I0.2.2);
%Q0.4.5 := %M3 XOR FE(%I0.2.5);
```

## Structured Text

## *Temporized Relays or Timers (pneumatic)*



Pneumatic timing relay

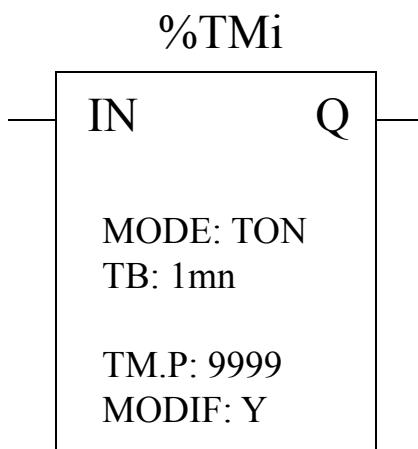


The **instantaneous** contacts change state as soon as the timer coil is powered.

The **delayed** contacts change state at the end of the time delay.

## Structured Text

*Temporized Relays  
or  
Timers*

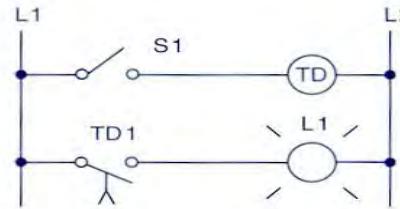


### Characteristics:

Identifier:	%TMi	0..63 in the TSX37
Input:	IN	to activate
Mode:	<b>TON</b>	On delay
	<b>TOFF</b>	Off delay
	<b>TP</b>	Monostable
Time basis:	TB	1mn (def.), 1s, 100ms, 10ms
Programmed value:	%TMi.P	0...9999 (def.) period=TB*TMi.P
Actual value:	%TMi.V	0...TMi.P (can be real or tested)
Modifiable:	Y/N	can be modified from the console

# Structured Text

## Example:



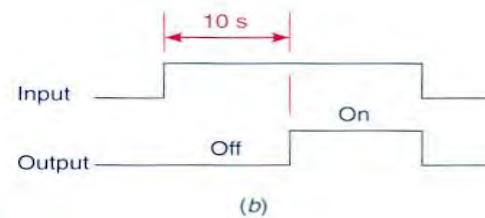
Sequence of operation:  
S1 open, TD de-energized, TD1 open, L1 off.

S1 closes, TD energizes, timing period starts,  
TD1 is still open, L1 is still off.

After 10 s, TD1 closes, L1 is switched on.

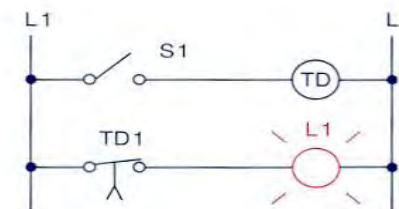
S1 is opened, TD de-energizes, TD1 opens instantly,  
L1 is switched off.

(a)



**Fig. 7-3**

On-delay timer circuit (NOTC contact). (a) Operation.  
(b) Timing diagram.



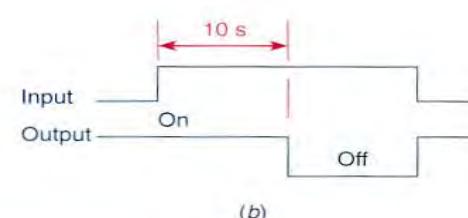
Sequence of operation:  
S1 open, TD de-energized, TD1 closed, L1 on.

S1 closes, TD energizes, timing period starts,  
TD1 is still closed, L1 is still on.

After 10 s, TD1 opens, L1 is switched off.

S1 is opened, TD de-energizes, TD1 closes instantly,  
L1 is switched on.

(a)

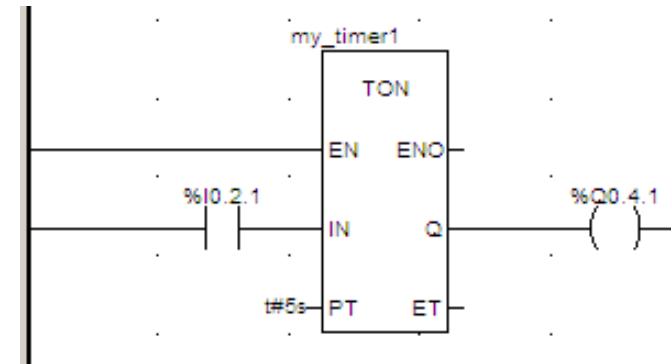


**Fig. 7-4**

On-delay timer circuit (NCTO contact).  
(a) Operation. (b) Timing diagram.

## Structured Text

### *Temporized Relays or Timers*



Screenshot of a PLC programming software interface:

- The toolbar has a red box around the icon for "Function Block Diagram". A red arrow points from this icon to the "Function Input Assistant" dialog.
- The "Function Input Assistant" dialog shows "TON" selected in the FFB type dropdown. The "Prototype" section shows a table with one row. The "Function and Function Block types" section lists various function blocks, with "TON" highlighted in yellow. The "Comment" column for TON states "On delay".
- A code editor window displays the following Structured Text code:
 

```

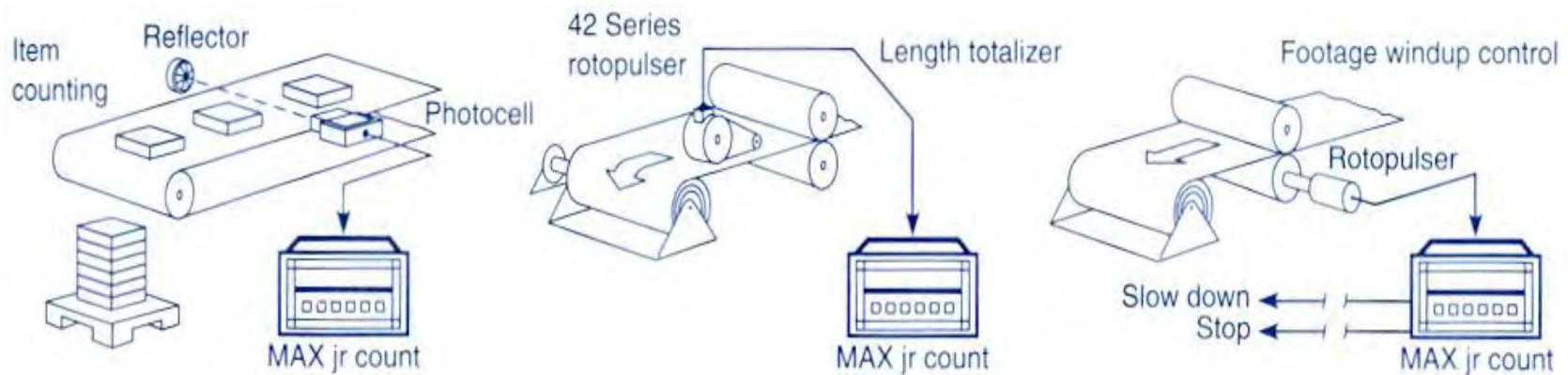
my_timer1 (IN := %IO.2.1 (*BOOL*),
           PT := t#5s (*TIME*),
           Q => %Q0.4.1 (*BOOL*),
           ET => my_var (*TIME*)) ;
      
```

Very similar to IL, notice however the missing CAL and the required “;”.

## Structured Text

### *Counters*

Some applications...

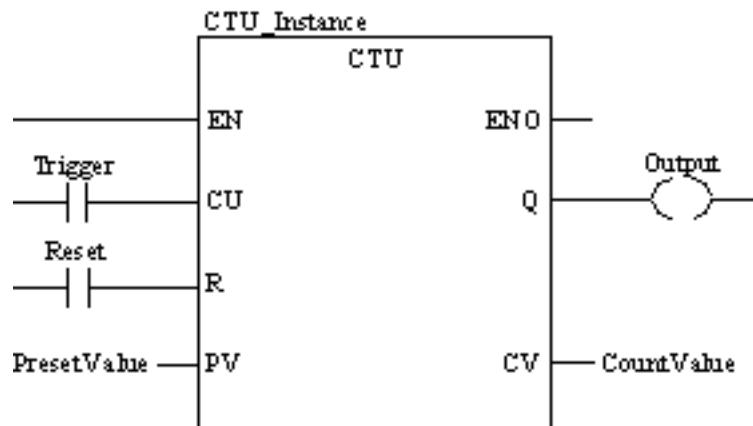


**Fig. 8-3**

Counter applications. (Courtesy of Dynapar Corporation, Gurnee, Illinois.)

## Structured Text

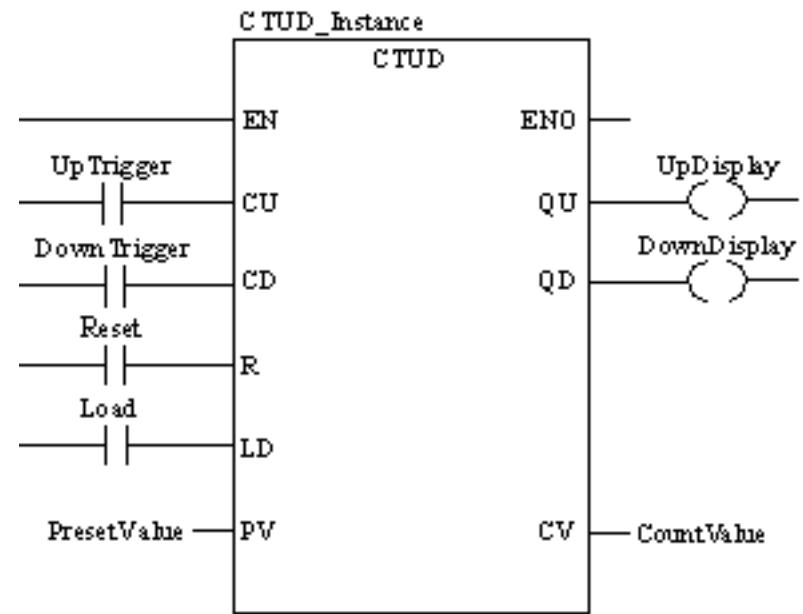
### Counters in Unity Pro



**CU "0" to "1" =>** CV is incremented by 1

**CV ≥ PV =>** Q:=1

**R=1 =>** CV:=0



**CU "0" to "1" =>** CV is incremented by 1

**CD "0" to "1" =>** CV is decremented by 1

**CV ≥ PV =>** QU:=1

**CV ≤ 0 =>** QD:=1

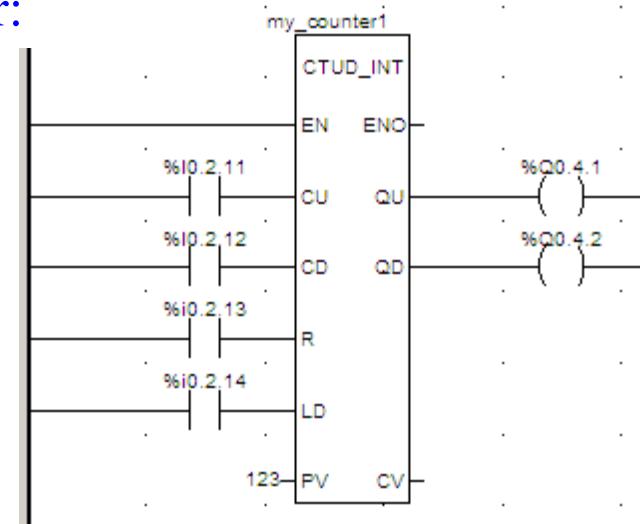
**R=1 =>** CV:=0    **LD=1 =>** CV:=PV  
R has precedence over LD

*NOTE: counters are saturated such that no overflow occurs*

## Structured Text

### *Counters in Unity Pro*

Ladder:



Instruction List:

```
CAL my_counter1 (CU := %I0.2.11 (*BOOL*),
                 CD := %I0.2.12 (*BOOL*),
                 R := %I0.2.13 (*BOOL*),
                 LD := %I0.2.14 (*BOOL*),
                 PV := 123 (*INT*),
                 QU => %Q0.4.1 (*BOOL*),
                 QD => %Q0.4.2 (*BOOL*),
                 CV => %MW100 (*INT*))
```

Structured Text:

```
my_counter1 (CU := %I0.2.11 (*BOOL*),
             CD := %I0.2.12 (*BOOL*),
             R := %I0.2.13 (*BOOL*),
             LD := %I0.2.14 (*BOOL*),
             PV := 123 (*INT*),
             QU => %Q0.4.1 (*BOOL*),
             QD => %Q0.4.2 (*BOOL*),
             CV => %MW100 (*INT*)) ;
```

Again IL and ST are similar, notice however the missing CAL and the required “;”.

## Structured Text

### *Numerical Processing*

### Algebraic and Logic Functions

```
%Q2.2 := %MW50 > 10;  
IF %I1.0 THEN  
    %MW10 := %KW0 + 10;  
END_IF;  
IF FE %I1.2 THEN  
    INC %MW100;  
END_IF;
```

## Structured Text

### Numerical Processing

#### Arithmetic Functions for Words

<b>+</b>	addition of two operands	<b>SQRT</b>	square root of an operand
<b>-</b>	subtraction of two operands	<b>INC</b>	incrementation of an operand
<b>*</b>	multiplication of two operands	<b>DEC</b>	decrementation of an operand
<b>/</b>	division of two operands	<b>ABS</b>	absolute value of an operand
<b>REM</b>	remainder from the division of 2 operands		

#### Operands

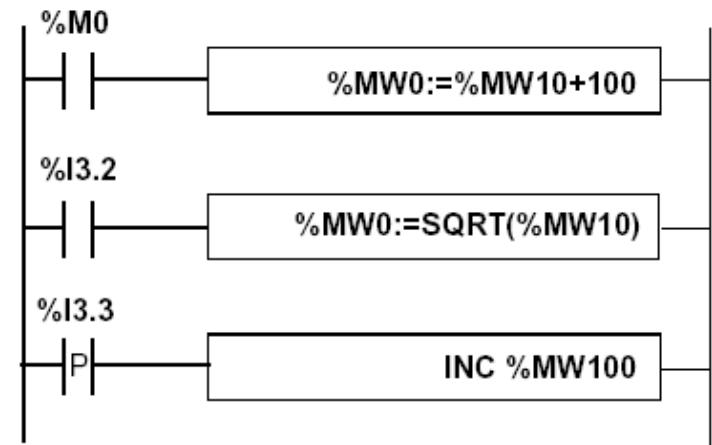
Type	Operand 1 (Op1)	Operand 2 (Op2)
Indexable words	%MW	%MW,%KW,%Xi.T
Non-indexable words	%QW,%SW,%NW,%BLK	Imm.Val.,%IW,%QW,%SW,%NW,%BLK, Num.expr.
Indexable double words	%MD	%MD,%KD
Non-indexable double words	%QD,%SD	Imm.Val.,%ID,%QD,%SD, Numeric expr.

## Structured Text

### Numerical Processing

#### Example:

#### Arithmetic functions



```
IF %M0 THEN
    %MW0 := %MW10 + 100;
END_IF;

IF %I3.2 THEN
    %MW0 := SQRT (%MW10);
END_IF;

IF RE (%I3.3) THEN
    INC (%MW100);
END_IF;
```

# Structured Text

## Numerical Processing

## Logic Functions

<b>AND</b>	AND (bit by bit) between two operands
<b>OR</b>	logical OR (bit by bit) between two operands
<b>XOR</b>	exclusive OR (bit by bit) between two operands
<b>NOT</b>	logical complement (bit by bit) of an operand

Comparison instructions are used to compare two operands.

- >: tests whether operand 1 is greater than operand 2,
- >=: tests whether operand 1 is greater than or equal to operand 2,
- <: tests whether operand 1 is less than operand 2,
- <=: tests whether operand 1 is less than or equal to operand 2,
- =: tests whether operand 1 is different from operand 2.

## Operands

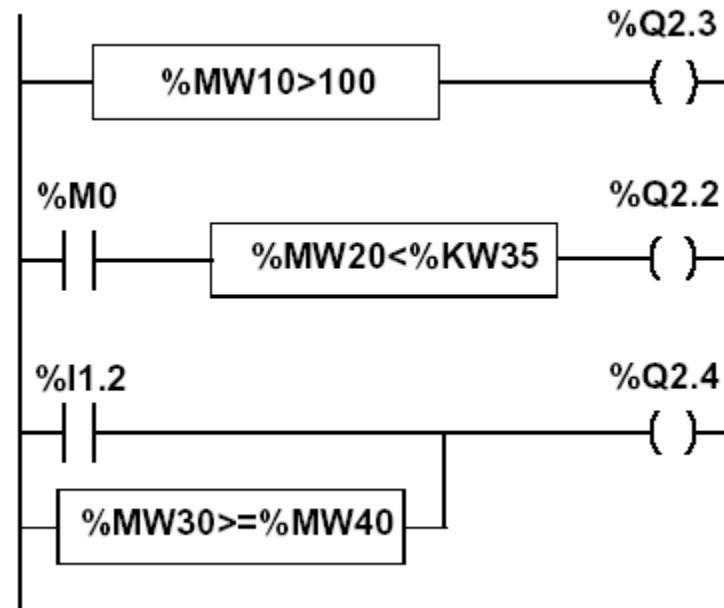
Type	Operands 1 and 2 (Op1 and Op2)
Indexable words	%MW, %KW, %Xi.T
Non-indexable words	Imm.val., %IW, %QW, %SW, %NW, %BLK, Numeric Expr.
Indexable double words	%MD, %KD
Non-indexable double words	Imm.val., %ID, %QD, %SD, Numeric expr.

## Structured Text

### Numerical Processing

Example:

Logic functions



### Structured text language

```
%Q2.3 := %MW10 > 100 ;  
%Q2.2 := %M0 AND (%MW20 < %KW35) ;  
%Q2.4 := %I1.2 OR (%MW30 >= %MW40) ;
```

## Structured Text

### Numerical Processing

#### Priorities on the execution of the operations

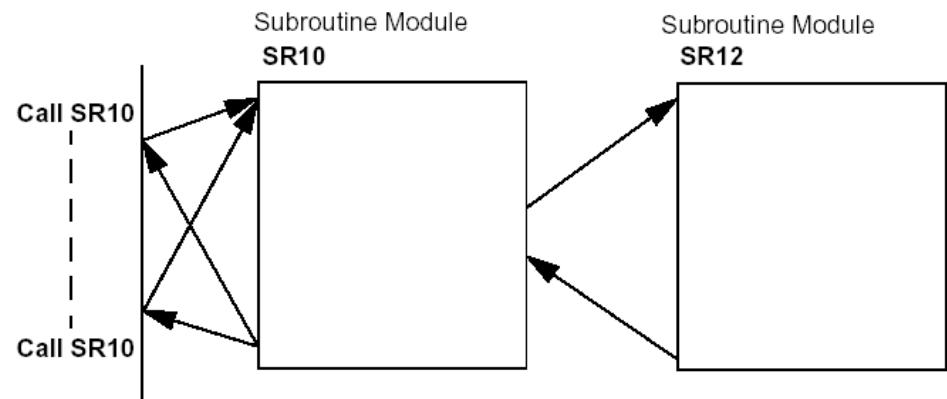
Rank	Instruction
1	Instruction to an operand
2	* , /, REM
3	+,-
4	<,>, <=,>=
5	=,<>
6	AND
7	XOR
8	OR

## Structured Text

### Structures for Control of Flux

#### Subroutines

#### Call and Return



#### Structured text language

```
IF %M8 THEN  
    RETURN;  
END_IF;
```

#### Structured text language

```
IF (%M5>3) THEN  
    RETURN;  
END_IF;  
IF %M8 THEN  
    %MD26 := %MW4 * %KD6 ;  
END_IF;
```

Not executed if %M5  
is larger than 3

## Structured Text

### Structures for Control of Flux

#### JUMP instructions:

#### Instruction List - conditional and unconditional jumps

Jump instructions are used to go to a programming line with an %Li label address:

- **JMP**: unconditional program jump
- **JMPC**: program jump if the instruction's Boolean result from the previous test is set at 1
- **JMPCN**: program jump if the instruction's Boolean result from the previous test is set at 0. %Li is the label of the line to which the jump has been made (address is from 1 to 999 with maximum 256 labels)

**Structured Text – just unconditional jumps as the IF .. THEN .. ELSE provides the conditional clauses.**

Note: by default, **jumps are disabled** in Unity Pro / Structured Text  
(if needed, enable them in the menu Tools -> Project Settings)

## Structured Text

### Structures for Control of Flux

#### Example:

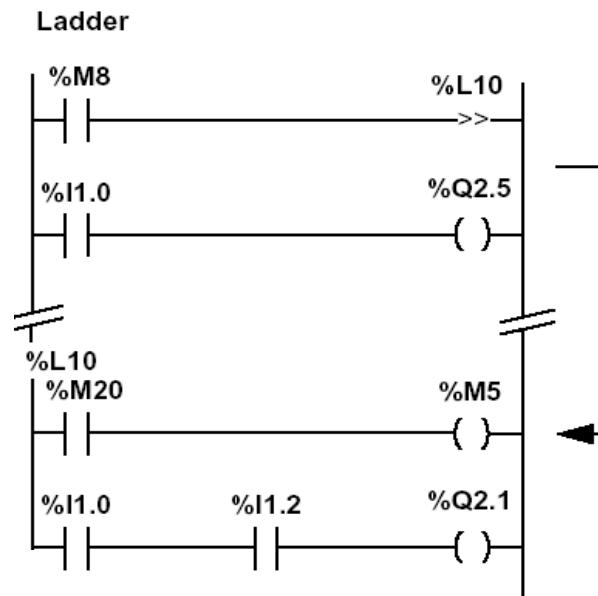
Use of jump instructions

```

IF %M8 THEN
    JUMP %L10;
END_IF;
%Q2.5:=%I1.0;
-----
%L10:
    %M5:=%M20;
    %Q2.1:=%I1.0 AND %I1.2;

```

Jump to label %L10  
if %M8=1



#### Unity Pro:

```

IF %M8 THEN
    JMP my_label_L10;
END_IF;
%Q0.4.5 := %I0.2.0;

(* other code ... *)

```

```

my_label_L10:
    %M5 := %M20;

```

*Notes: It is not a good style of programming. Does not improve the legibility of the proposed solution.  
Attention to INFINITE LOOPS.*

# Structured Text

## Structures for Control of Flux

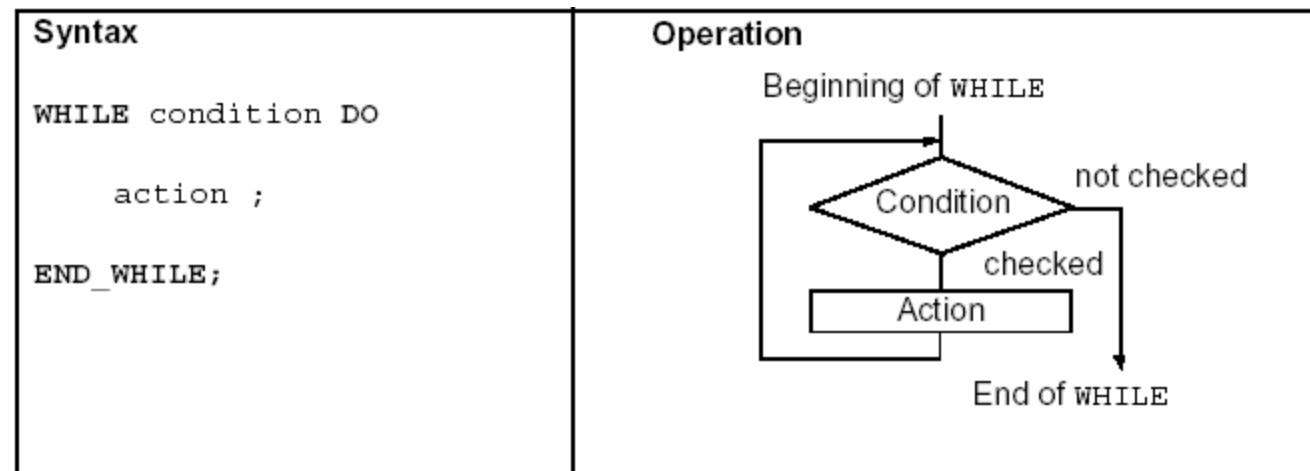
### IF ... THEN ... ELSE ...

Syntax	Operation
<pre>IF condition1 THEN     actions1;  ELSEIF condition2 THEN     actions2;  ELSE     actions3; END_IF;</pre>	<pre>graph TD; Start((Beginning of IF)) --&gt; Cond1{Condition 1}; Cond1 -- checked --&gt; Actions1[Actions 1]; Cond1 -- not checked --&gt; Actions3[Actions 3]; Actions1 --&gt; Cond2{Condition 2}; Cond2 -- checked --&gt; Actions2[Actions 2]; Cond2 -- not checked --&gt; Actions3; Actions2 --&gt; End((End of IF)); Actions3 --&gt; End;</pre>

# Structured Text

## Structures for Control of Flux

### WHILE



Example:

```
(*WHILE conditional repeated action*)
WHILE %MW4<12 DO
    INC(%MW4);
    SET(%M25 [%MW4]);
END WHILE;
```

# Structured Text

## Structures for Control of Flux

**REPEAT ... UNTIL**

**FOR ... DO**

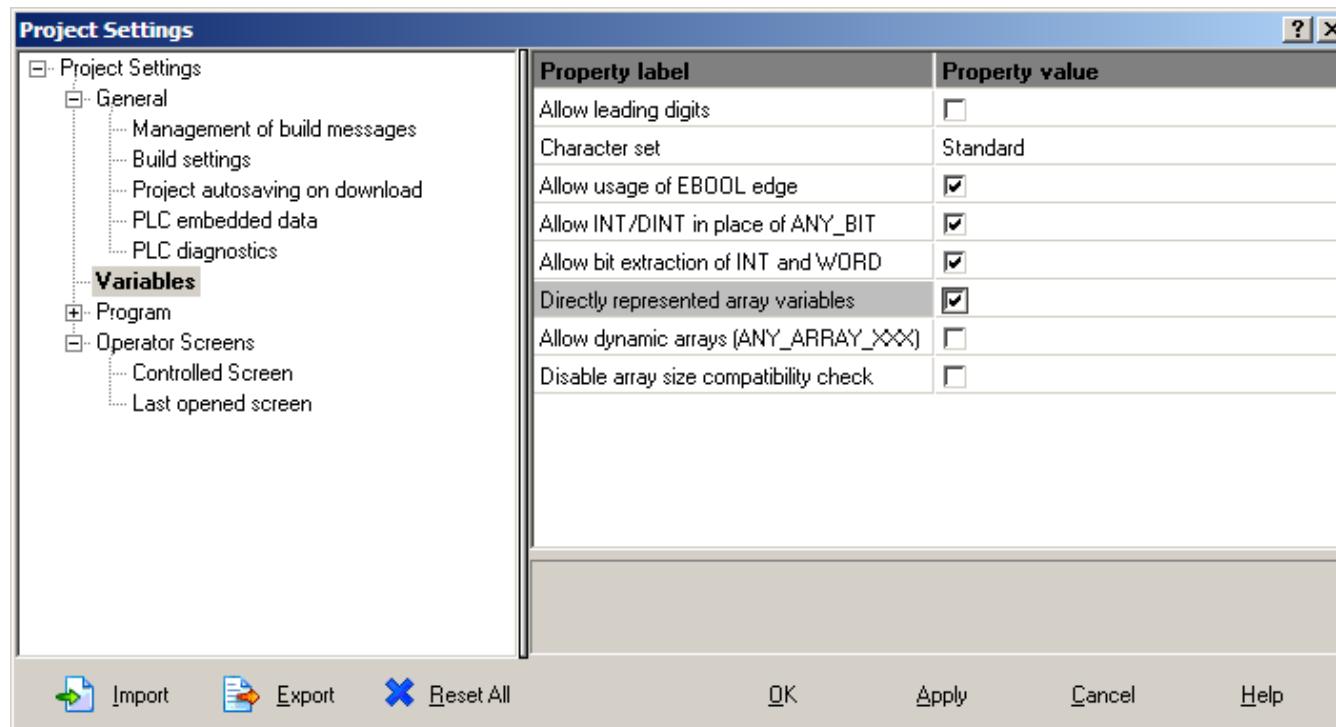
**EXIT** to abort the execution of a structured flux control instruction

Example:

```
(* using EXIT to break a loop *)
WHILE %MW1<124 DO
    %MW2 := 0;
    %MW3 := %MW100[%MW1];
    REPEAT
        %MW500[%MW2] := %MW3 + %MW500[%MW2];
        IF (%MW500[%MW2] > 32700) THEN
            EXIT;
        END_IF;
        INC(%MW2);
    UNTIL %MW2>25 END_REPEAT;
    INC(%MW1);
END_WHILE;
```

## Structured Text

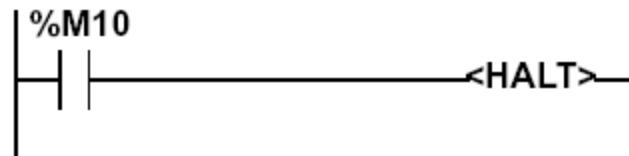
*Note: in Unity Pro, both in Structured Text and Instruction List, the conventional array indexing (e.g. %MW100 [%MW1]) is **disabled by default**. To enable it, go to the project settings, menu Tools -> Project Settings. See the grayed region in the next figure:*



## Structured Text

### Structures for Control of Flux

#### Halt

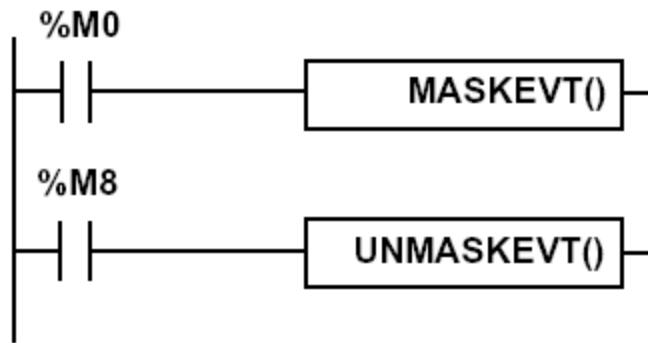


**Structured text language**

```
IF %M10 THEN  
    HALT;  
END_IF;
```

Stops all processes!

#### Events masking



**Structured text language**

```
IF %M0 THEN  
    MASKEVT();  
END_IF;  
IF %M8 THEN  
    UNMASKEVT();  
END_IF;
```

## Structured Text

There are other advanced instructions (see manual)

- Monostable
- Registers of 256 words (LIFO ou FIFO)
- *DRUMs*
- Comparators
- *Shift-registers*
- ...
- Functions to manipulate *floats*
- Functions to convert bases and types

**Structured Text*****Networking (in Unity Pro)***

**Unity Pro Help**

Back Forward Print Options Help

Contents Index Search

Unity Pro Software  
 EF/EFB/DFB Libraries  
 Standard library  
 Control library  
 Communications library  
   Safety Information  
   About the Book  
   General Information  
 Extended  
   ADDM: Address Conversion  
   ADDR: Address Conversion  
   CANCEL: Stopping an Exchange i  
   CREAD\_REG: Continuous Register  
   CWRITE\_REG: Continuous Register  
   DATA\_EXCH: Exchanging Data b  
   INPUT\_BYTE: Receiving Character  
   INPUT\_CHAR: Receiving Character  
   MBP\_MSTR: Modbus Plus Master  
   Modbus+\_ADDR: Modbus Plus Ad  
   OUT\_IN\_CHAR: Sending/Receiving  
   OUT\_IN\_MBUS: Modbus Communica  
   PRINT\_CHAR: Sending character  
   RCV\_TLG: Receiving telegrams  
   READ\_ASYNC: Reading data asyn  
   READ\_GDATA: Reading Modbus  
   READ\_REG: Read Register  
 READ\_VAR: Reading variables  
   Description  
   Assisted entry screen  
   Example of use on a Uni-Telw  
   Example of Reading Bits  
   Example of use in a network  
   Example of Reading Words via  
   Example including execution c  
   SEND\_EMAIL: Sending Email  
   SEND\_REQ: Sending requests  
   SEND\_TLG: Sending telegrams  
   SYMAX\_IP\_ADDR: SY/MAX IP A

## Example including execution check

At a Glance

The following example illustrates the READ\_VAR function with a management parameter check.

Programming the function

Programming in ST:

```
IF NOT %M21 AND %I0.1.2 THEN
  %MW210:4 := 0;
  %MW212 := 50;
  READ_VAR(ADDR('0.3.1.7'), '%MW', 20, 1, %MW210:4, %MW1701:1);
  SET %M21;
END_IF;
```

- the input bit %I0.1.2 controls the function,
- the internet bit %M21 is used to test the activity of the function,
- %MW210:4 := 0; initializes the management table to 0,
- MW212 := 50; initializes the timeout value to 5 seconds.

**NOTE:** READ\_VAR (ADDM ('0.3.1.7'), '%MW', 20, 1, %MW210:4, %MW1701:1); syntax must be used for Modicon M340 PLCs as ADDR function cannot be used by a Modicon M340 PLC.

Programming the exchange check

Programming in ST:

```
IF %M21 AND NOT %M210.0 THEN
  INC %MW214;
  IF %MW211 = 0 THEN
    INC %MW215;
  ELSE
    SET %Q0.2.2;
    INC %MW216;
    %MW217 := %MW211;
  END_IF;
END_IF;
```

- %MW214 counts the number of exchanges,
- %MW215 counts the number of correct exchanges,
- %MW216 counts the number of exchanges generating errors,
- %MW217 stores the error message,
- %Q0.2.2 indicates an exchange failure.

© 2011 Schneider Electric. All rights reserved.