Industrial Automation

(Automação de Processos Industriais)

PLC Programming languages

Structured Text

http://www.isr.tecnico.ulisboa.pt/~jag/courses/api20b/api2021.html

Prof. Paulo Jorge Oliveira, original slides
Prof. José Gaspar, rev. 2020/2021

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*.

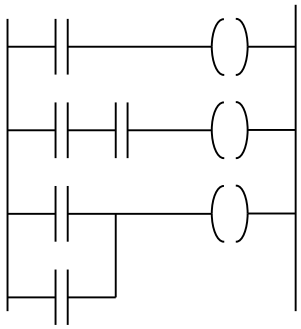
Software development resources.

...

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

PLC Programming Languages (IEC 61131-3)

Ladder Diagram



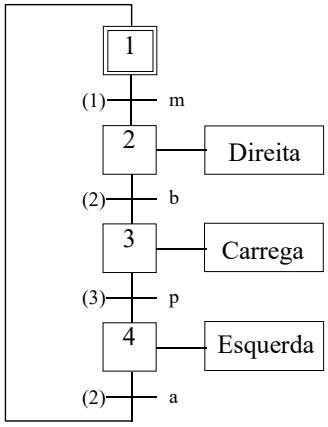
Structured Text

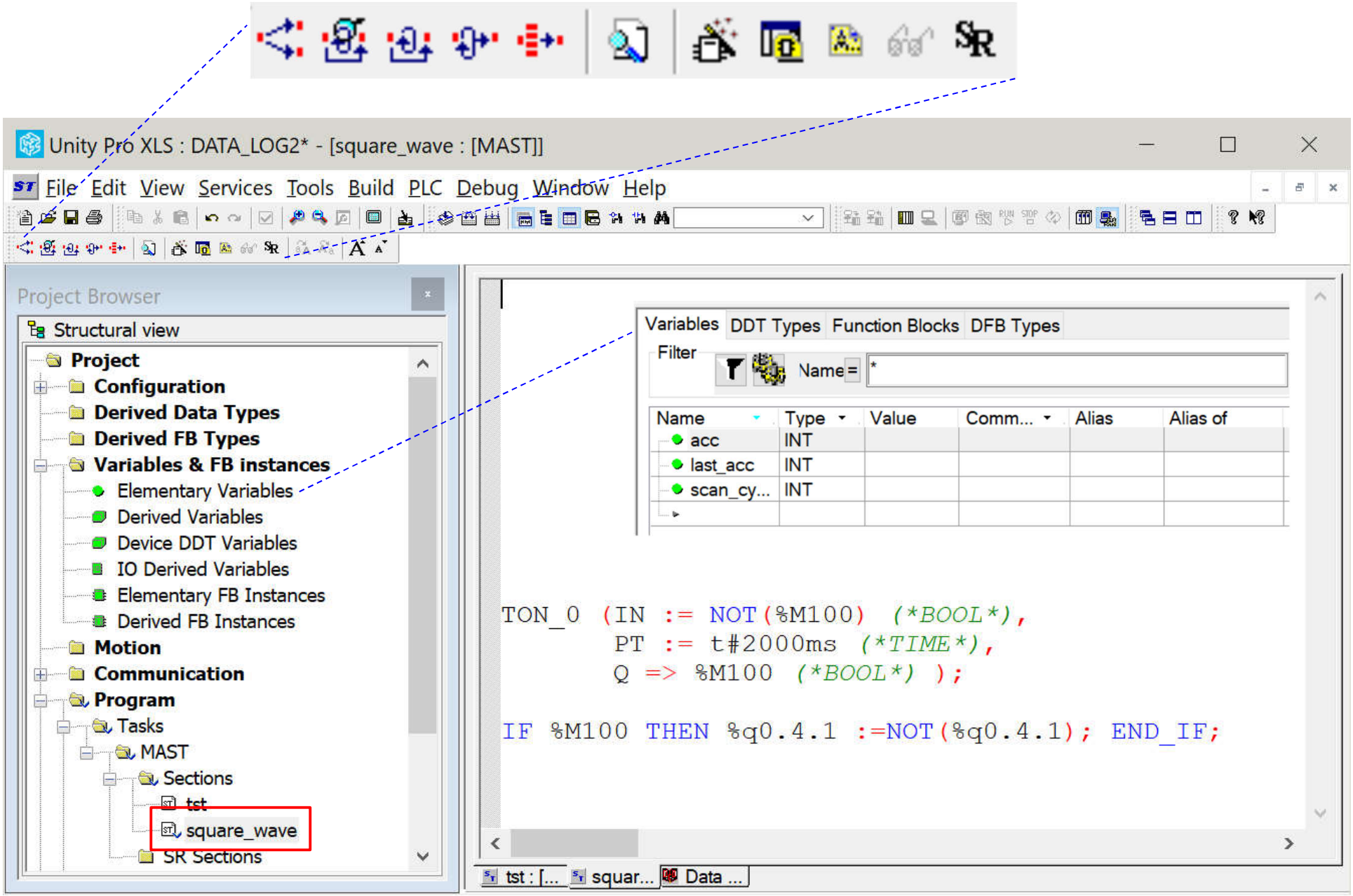
```
If  %I1.0  THEN
  %Q2.1 := TRUE
ELSE
  %Q2.2 := FALSE
END_IF
```

Instruction List

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

Sequential Function Chart (GRAFCET)




Unity Pro XLS : DATA_LOG2* - [square_wave : [MAST]]
File Edit View Services Tools Build PLC Debug Window Help
Project Browser
Structural view
Project
Configuration
Derived Data Types
Derived FB Types
Variables & FB instances
Elementary Variables
Derived Variables
Device DDT Variables
IO Derived Variables
Elementary FB Instances
Derived FB Instances
Motion
Communication
Program
Tasks
MAST
Sections
tst
square_wave
SR Sections
Variables
DDT Types
Function Blocks
DFB Types
Filter
Name =
Name
Type
Value
Comm...
Alias
Alias of
acc
INT
last_acc
INT
scan_cy...
INT
TON_0 (IN := NOT(%M100) (*BOOL*),
PT := t#2000ms (*TIME*),
Q => %M100 (*BOOL*));
IF %M100 THEN %q0.4.1 :=NOT(%q0.4.1); END_IF;
tst : [...
squar...
Data ...

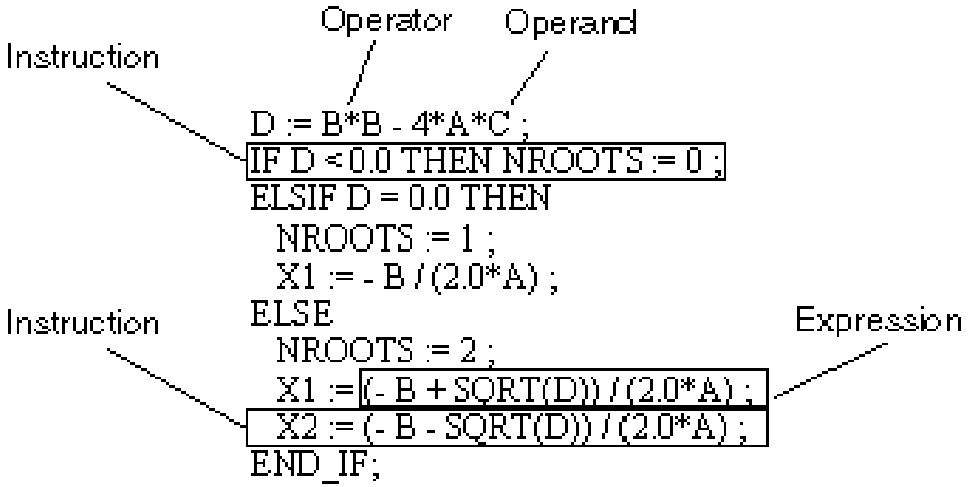
Structured Text

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:

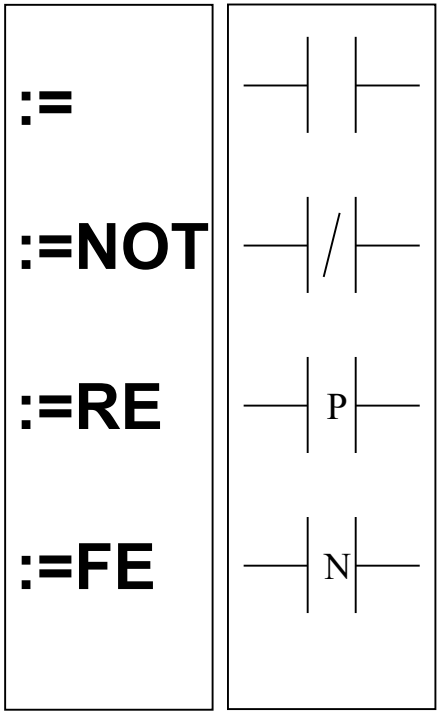


The **length of an instruction line** is limited to 300 characters. The **length of an ST section is not limited** within the programming environment. The length of an ST section is only limited by the size of the PLC memory.

Structured Text

Basic Instructions

Load



Open contact: contact is active (result is 1) while the control bit is 1.

Close contact: contact is active (result is 1) while the control bit is 0.

Contact in the rising edge: contact is active during a scan cycle where the control bit has a rising edge.

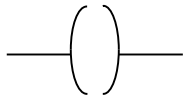
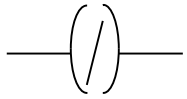
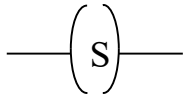
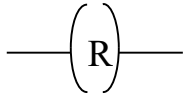
Contact in the falling edge: contact is active during a scan cycle where the control bit has a falling edge.

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

Structured Text

Basic Instructions

Store

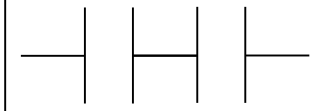
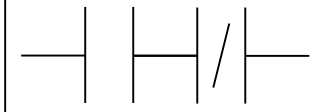
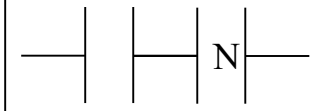
:=		The result of the logic function activates the coil.
:=NOT		The inverse result of the logic function activates the coil.
SET		The result of the logic function energizes the relay (sets the latch).
RESET		The result of the logic function de-energizes the relay (resets the latch)..

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

Structured Text

Basic Instructions

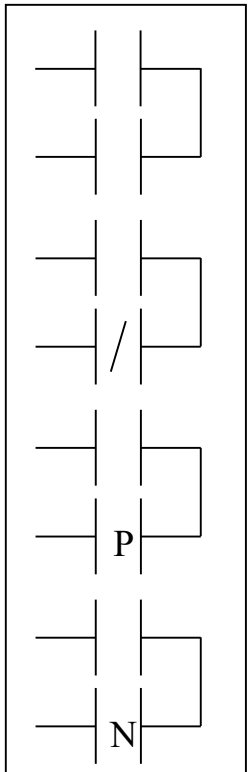
AND

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

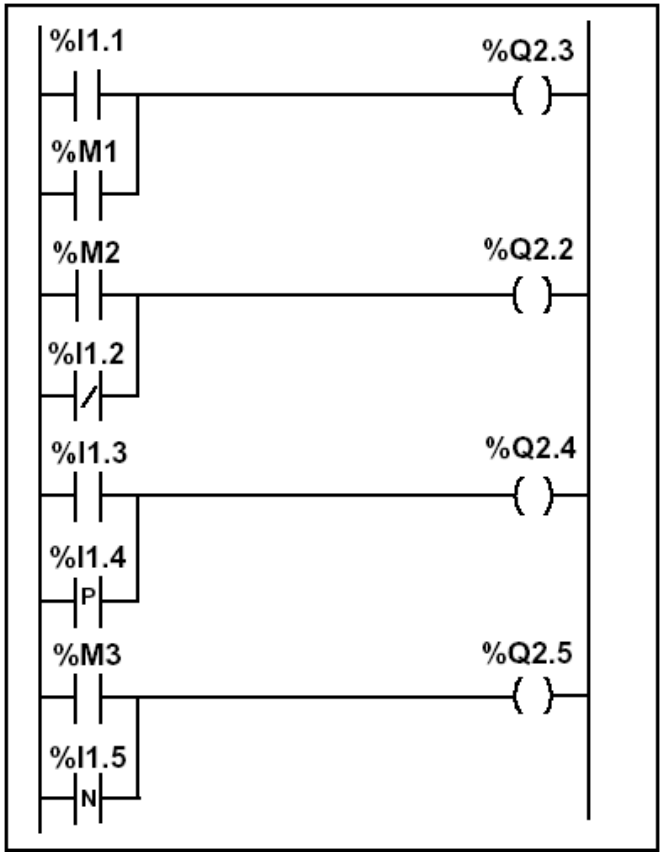
Basic Instructions

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);
```

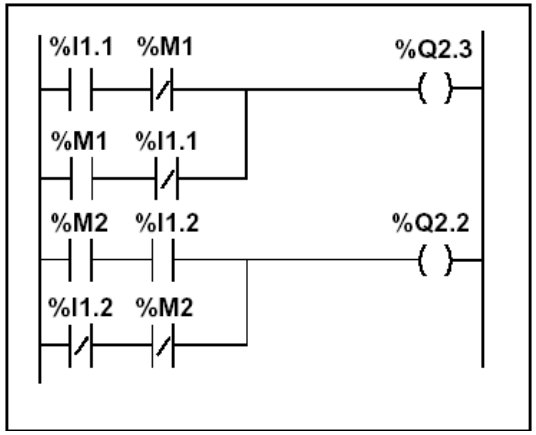
Unity Pro (Premium PLC):

```
%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);
```

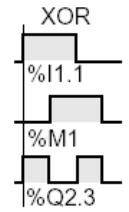
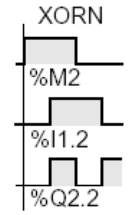
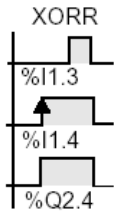
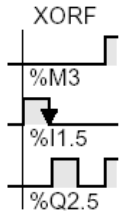
Structured Text

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):

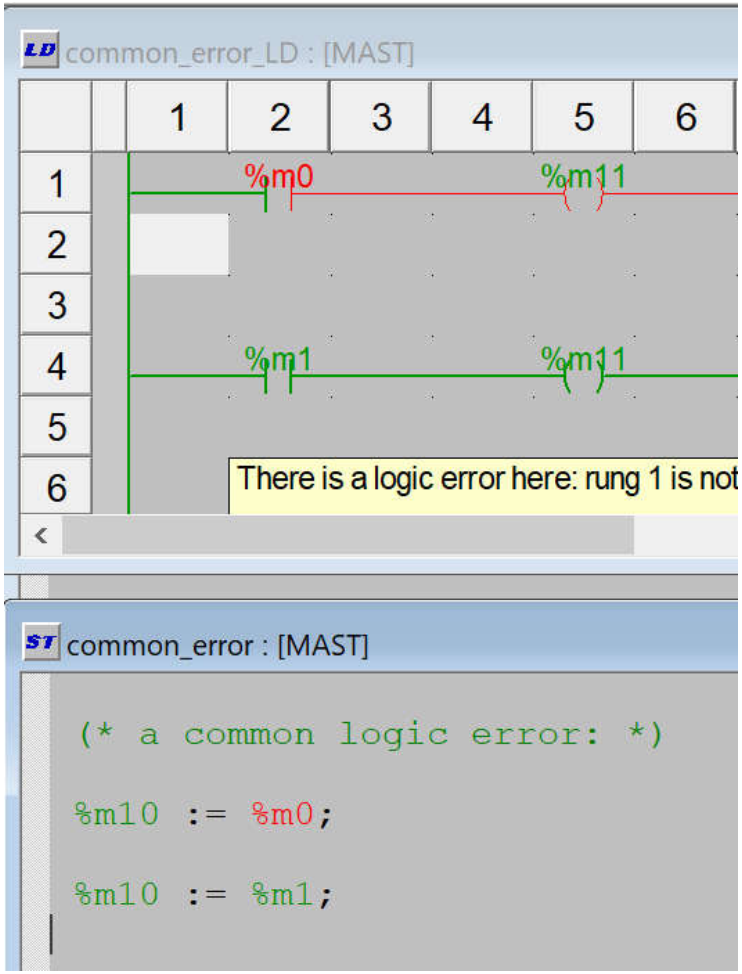
```
%Q0.4.3 := %I0.2.1 XOR %M1;
%Q0.4.4 := %I0.2.3 XOR RE(%I0.2.4);
%Q0.4.2 := %M2 XOR (NOT %I0.2.2);
%Q0.4.5 := %M3 XOR FE(%I0.2.5);
```

Structured Text

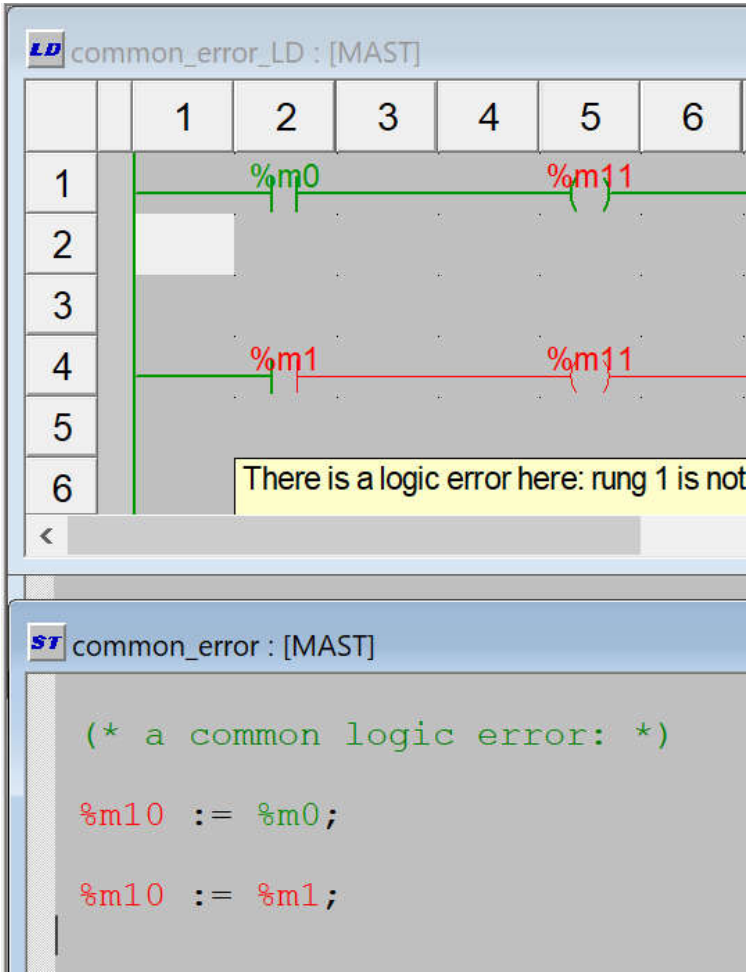
Basic Instructions to Manipulate Bit Tables

Designation	Function
Table:= Table	Assignment between two tables
Table:= Word	Assignment of a word to a table
Word:= Table	Assignment of a table to a word
Table:= Double word	Assignment of a double word to a table
Double word: = Table	Assignment of a table to a double word
COPY_BIT	Copy of a bits table in a bits table
AND_ARX	AND between two tables
OR_ARX	OR between two tables
XOR_ARX	exclusive OR between two tables
NOT_ARX	Negation in a table
BIT_W	Copy of a bits table in a word table
BIT_D	Copy of a bits table in a double word table
W_BIT	Copy of a word table in a bits table
D_BIT	Copy of a double word table in a bits table
LENGHT_ARX	Calculation of the length of a table by the number of elements

A very common programming error:



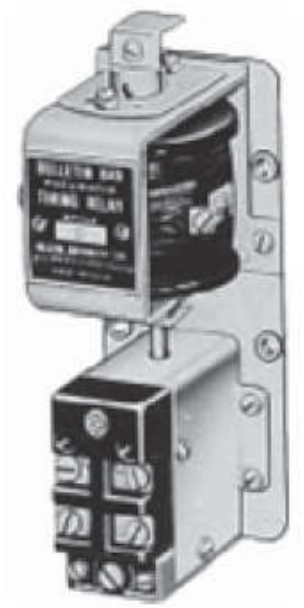
Why are %m10 and %m11 TRUE while %m0 is FALSE?



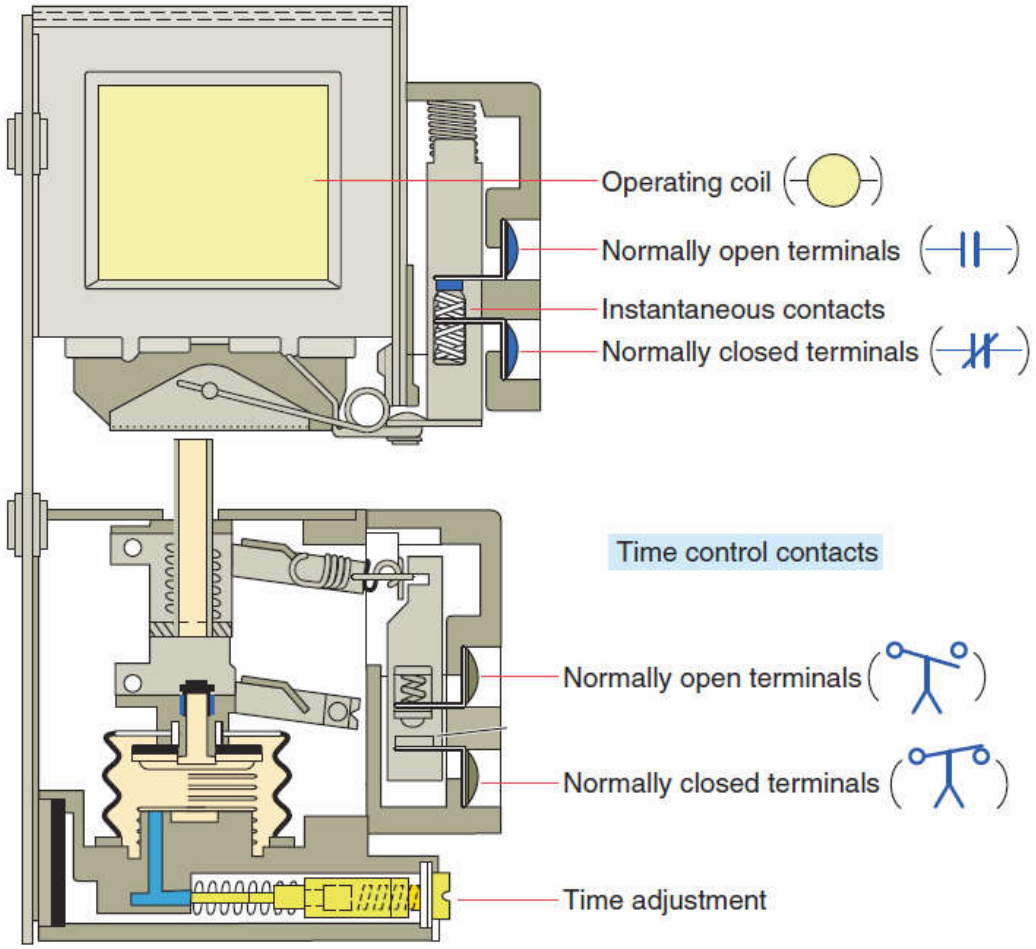
Why are %m10 and %m11 FALSE while %m0 is TRUE?

Structured Text

Temporized Relays or Timers (pneumatic)



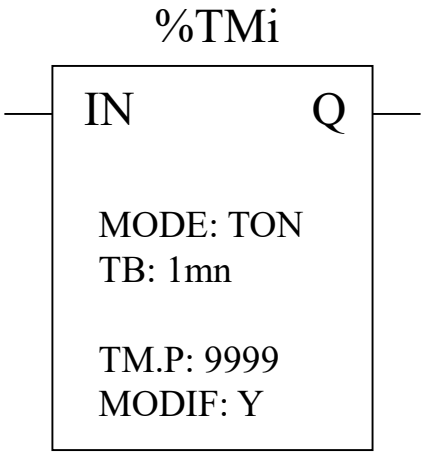
Pneumatic timing relay



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

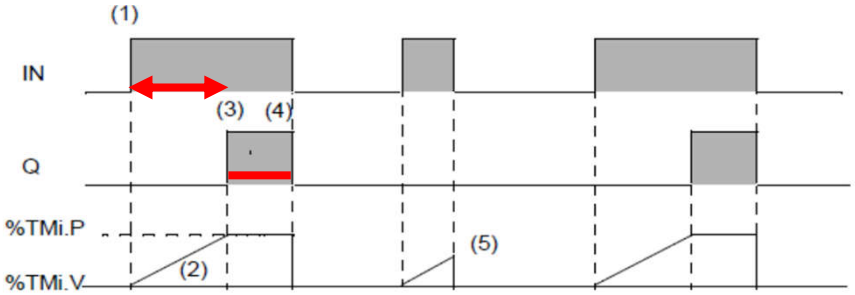


Characteristics:

Identifier:	%TMi	0..63 in the TSX37
Input:	IN	to activate
Mode:	**TON**	On delay
	TOFF	Off delay
	TP	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

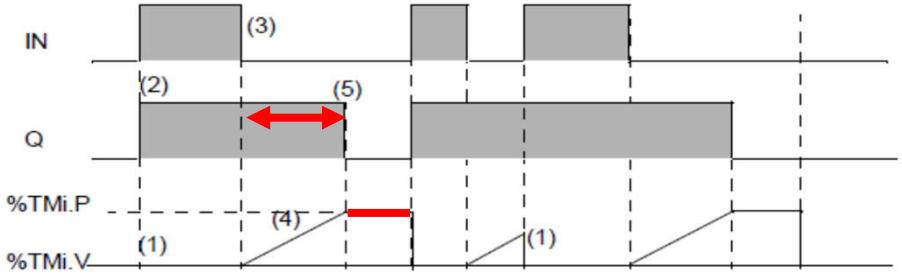
Timers

TON mode



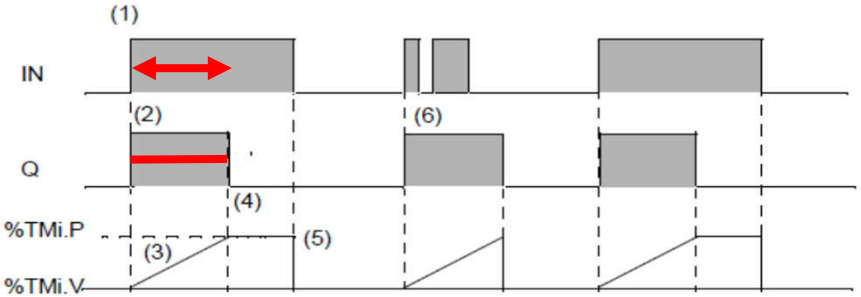
App. example: start ringing the alarm if N sec after door open there is no disarm of the alarm.

TOF mode



App. example: turn off stairways lights after N sec the lights' button has been released.

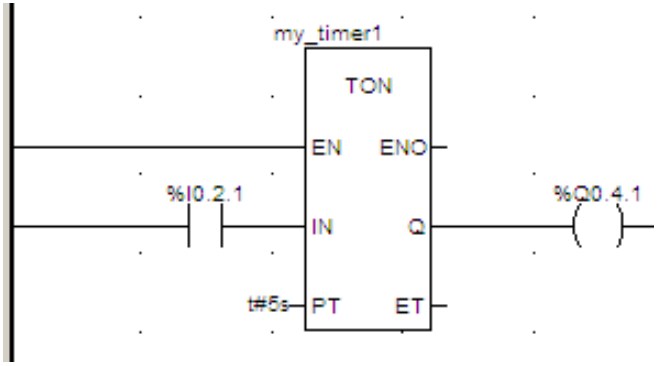
TP mode

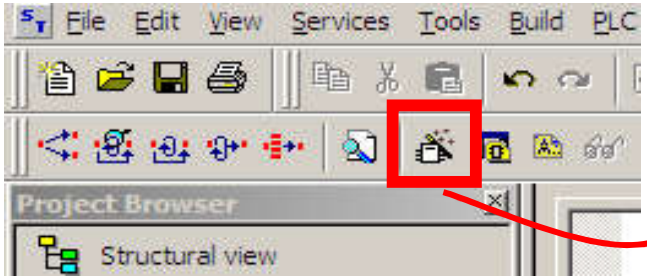


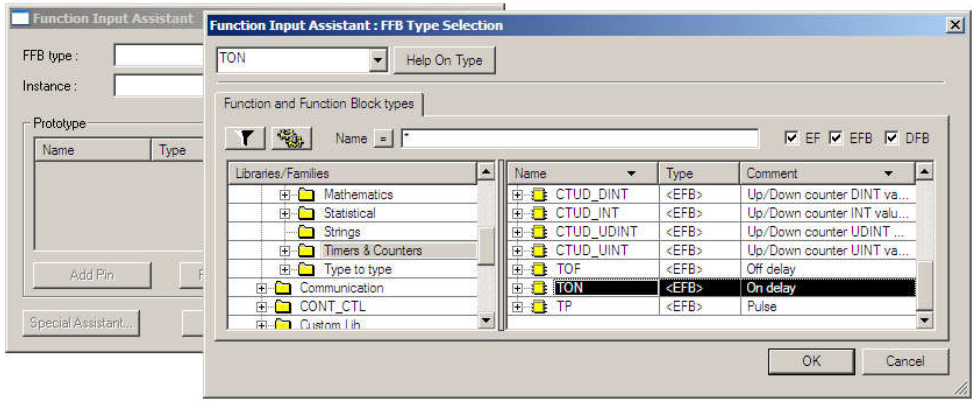
App. example: positive input edge give a controlled (fixed) duration pulse to start a motor.

Structured Text

Temporized Relays or Timers





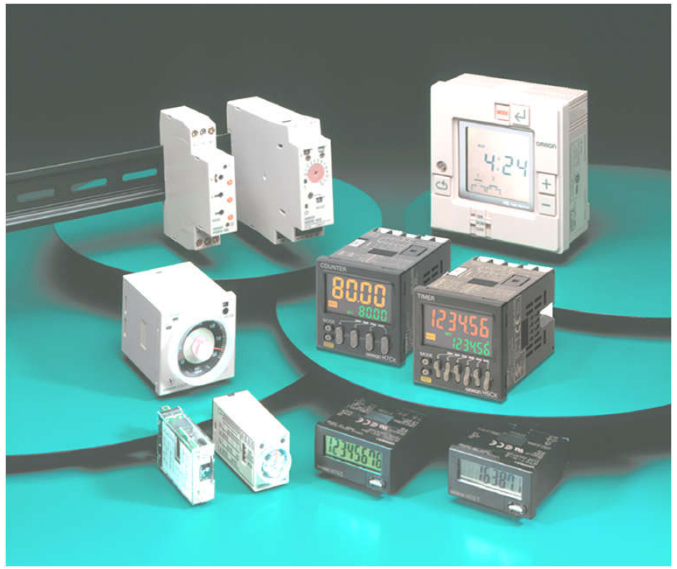



```
my_timer1 (IN := %I0.2.1 (*BOOL*),
           PT := t#5s (*TIME*),
           Q => %Q0.4.1 (*BOOL*),
           ET => my_var (*TIME*));
```

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

Structured Text

Counters



Some applications...

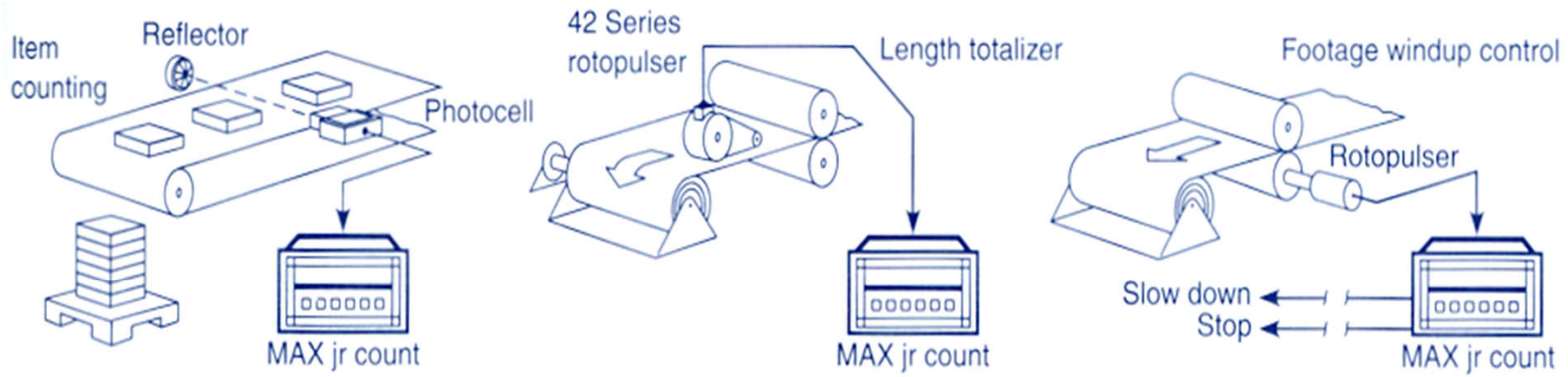
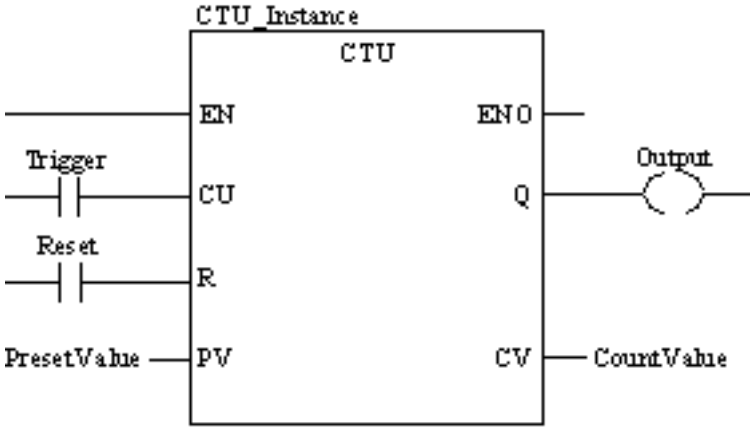


Fig. 8-3

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

Structured Text

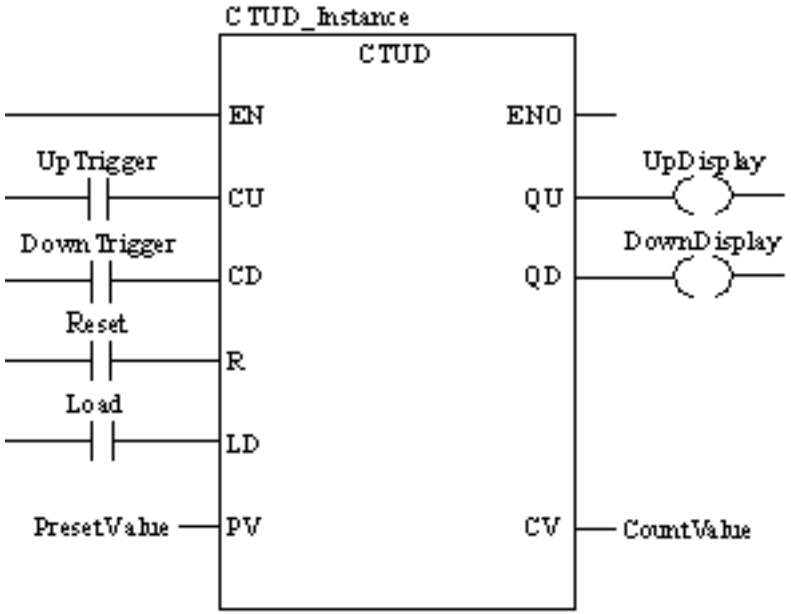
Counters in Unity Pro



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

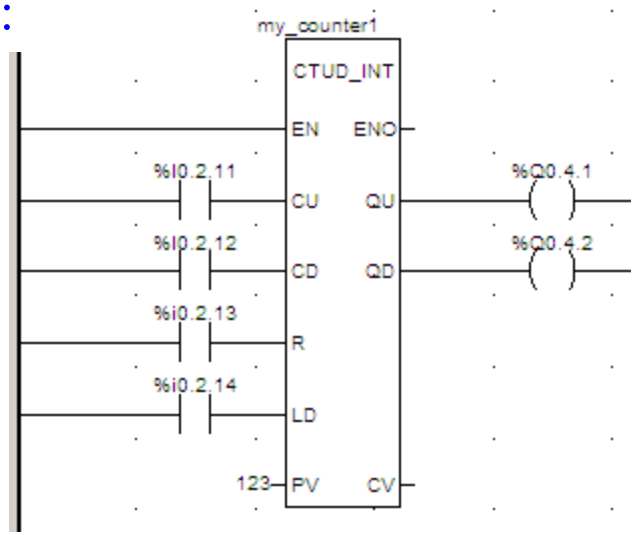
R=1 => CV:=0 **LD=1** => CV:=PV
R has precedence over LD

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

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

Operands

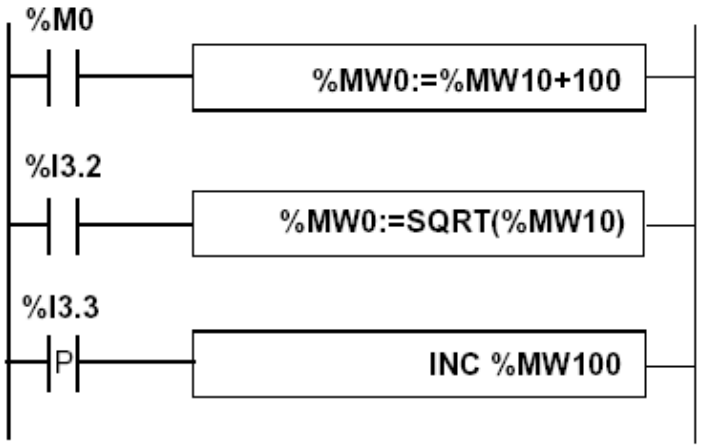
Type	Operand 1 (Op1)	Operand 2 (Op2)
Indexable words	%MW	%MW,%KW,%Xi.T
Non-indexable words	%QW,%SW,%NW,%BLK	Imm.Val.,%IW,%QW,%SW,%NW, %BLK, Num.expr.
Indexable double words	%MD	%MD,%KD
Non-indexable double words	%QD,%SD	Imm.Val.,%ID,%QD,%SD, Numeric expr.

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;
```

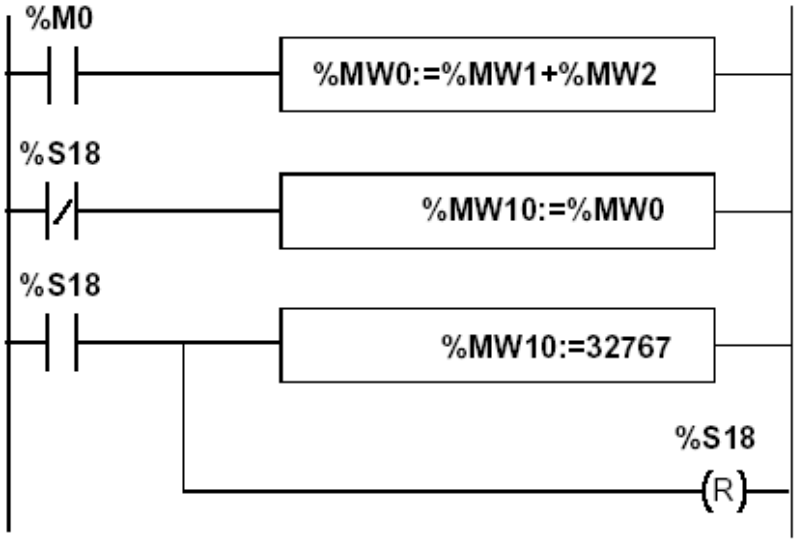
Exercise:
In this page is shown the conversion of LD operate blocks to ST;
Can you give an example of converting an LD compare block?

Structured Text

Numerical Processing

Example:

Arithmetic functions




```
IF %M0 THEN
  %MW0 := %MW1 + %MW2;
END_IF;

IF %S18 THEN
  %MW10 := 32767; RESET %S18;
ELSE
  %MW10 := %MW0;
END_IF;
```

This example contains the usage of a system variable:

%S18 – flag de overflow

Structured Text

Numerical Processing

Logic Functions

AND	AND (bit by bit) between two operands
OR	logical OR (bit by bit) between two operands
XOR	exclusive OR (bit by bit) between two operands
NOT	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

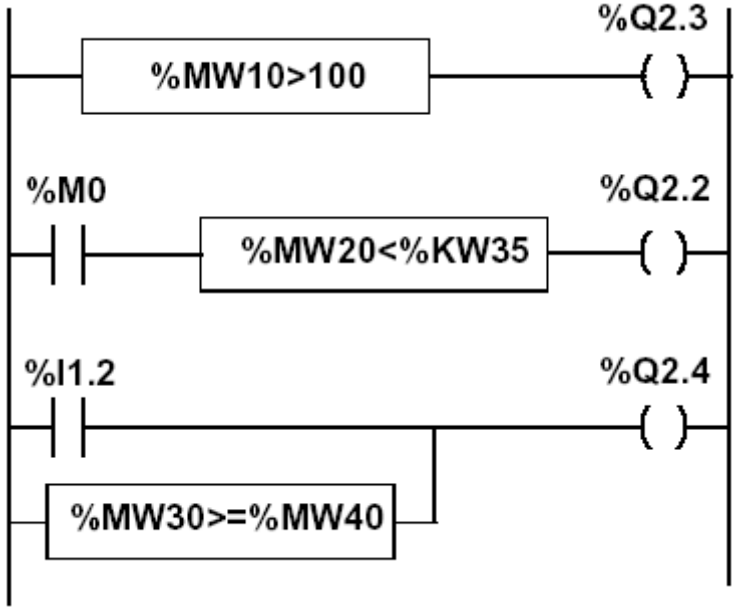
Type	**Operands 1 and 2 (Op1 and Op2)**
Indexable words	%MW,%KW,%Xi.T
Non-indexable words	Imm.val.,%IW,%QW,%SW,%NW,%BLK, Numeric Expr.
Indexable double words	%MD,%KD
Non-indexable double words	Imm.val.,%ID,%QD,%SD,Numeric expr.

Structured Text

Numerical Processing

Example:

Logic functions



Structured text language

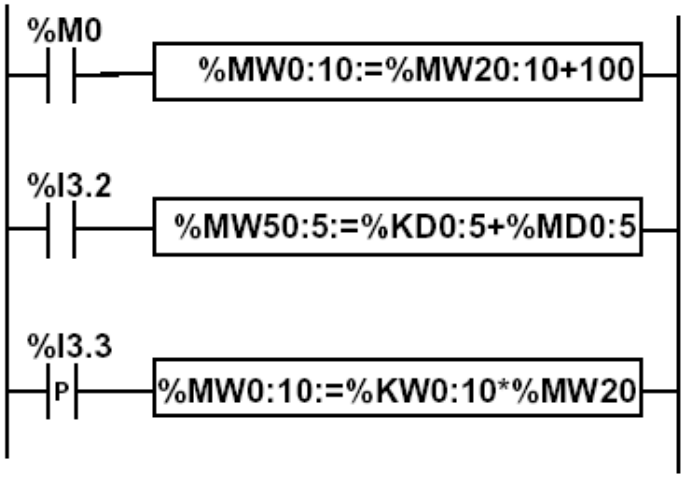
```
%Q2.3:=%MW10>100;
%Q2.2:=%M0 AND (%MW20<%KW35);
%Q2.4:=%I1.2 OR (%MW30>=%MW40);
```

Structured Text

Numerical Processing

Example:

Numeric Tables Manipulation

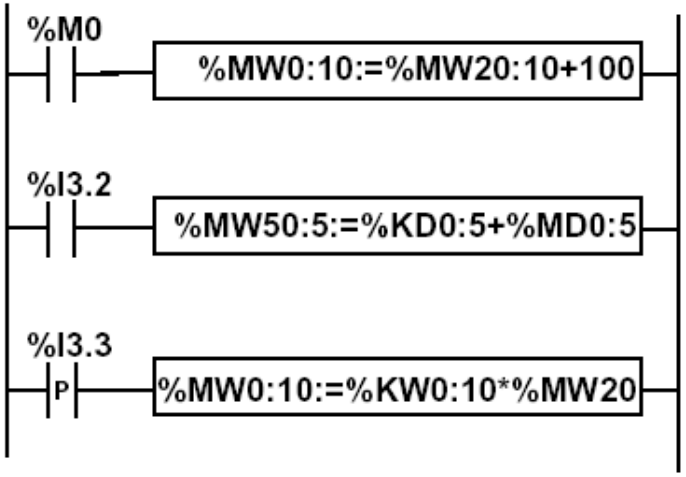


Structured text language

```
IF RE %I3.3 THEN
 %MW0:10:=%KW0:10*%MW20;
END_IF;
```

Structured Text Numerical Tables

Type	Format	Maximum address	Size	Write access
Internal words	Simple length	%MWi:L	i+L<=Nmax (1)	Yes
	Double length	%MWDi:L	i+L<=Nmax-1 (1)	Yes
	Floating point	%MFi:L	i+L<=Nmax-1 (1)	Yes
Constant words	Single length	%KWi:L	i+L<=Nmax (1)	No
	Double length	%KWDi:L	i+L<=Nmax-1 (1)	No
	Floating point	%KFi:L	i+L<=Nmax-1 (1)	No
System word	Single length	%SW50:4 (2)	-	Yes



Instruction list language

```
LD %M0
[%MW0:10:=%MW20:10+100]

LD %I3.2
[%MD50:5:=%KD0:5+%MD0:5]
```

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

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 i from 1 to 999 with maximum 256 labels)

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

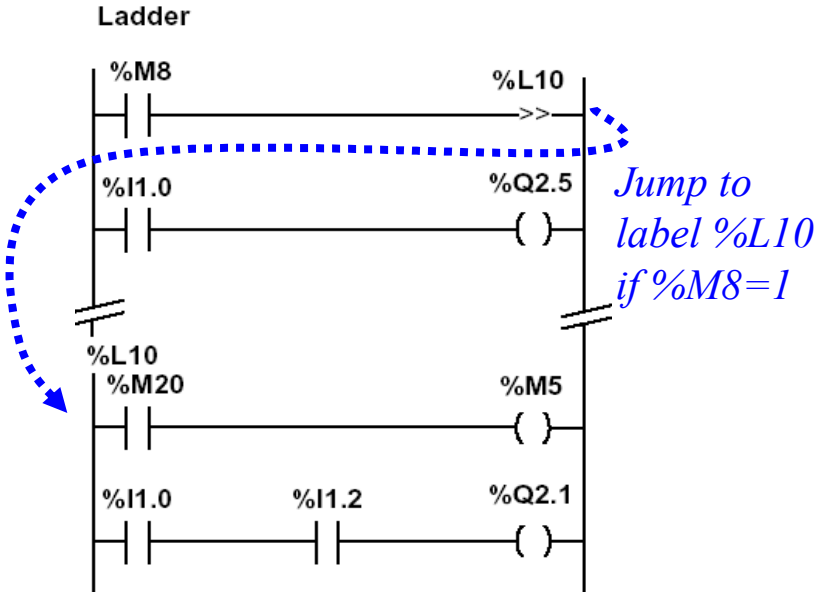
Note: by default, **jumps are disabled** in Unity Pro / Structured Text (if needed, enable them in the menu Tools -> Project Settings)

Structured Text

Structures for Control of Flux

Example:
Use of jump instructions

Ladder



Jump to label %L10 if %M8=1

PL7:

```
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;
```

Jump to label %L10 if %M8=1

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

Structured Text

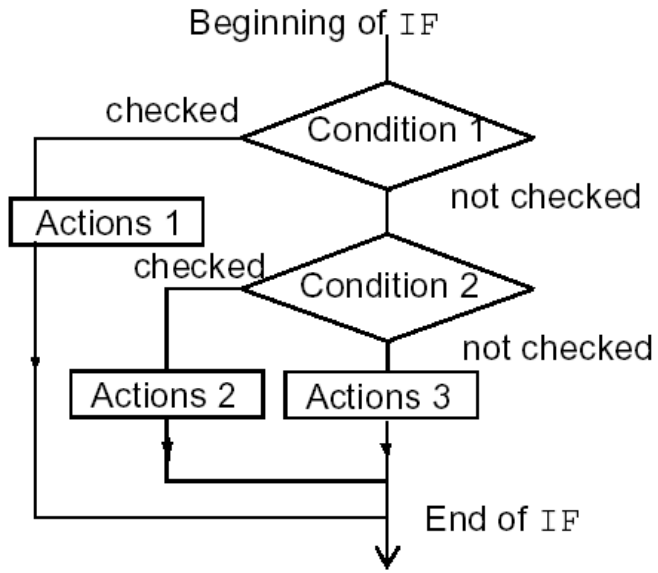
Structures for Control of Flux

IF ... THEN ... ; [[ELSIF ... THEN ;] ELSE ... ;] END_IF; *(* Unity Pro *)*

Syntax *PL7*

```
IF condition1 THEN
    actions1;
ELSEIF condition2 THEN
    actions2;
ELSE
    actions3;
END_IF;
```

Operation



*Note: In PL7 one writes **ELSEIF** while in Unity Pro one writes **ELSIF***

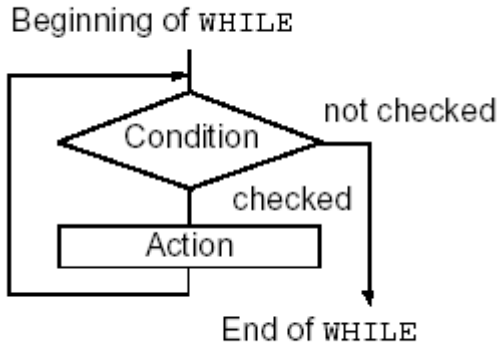
Structured Text

Structures for Control of Flux: WHILE

Syntax

```
WHILE condition DO

    action ;

END_WHILE;
```

Operation



Example:

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

Word of caution: ***do not wait on an input*** *that may take long to happen (e.g. a switch pressed by a person) as the PLC watchdog may timeout.*

Structured Text

Structures for Control of Flux

REPEAT ... UNTIL

FOR ... DO

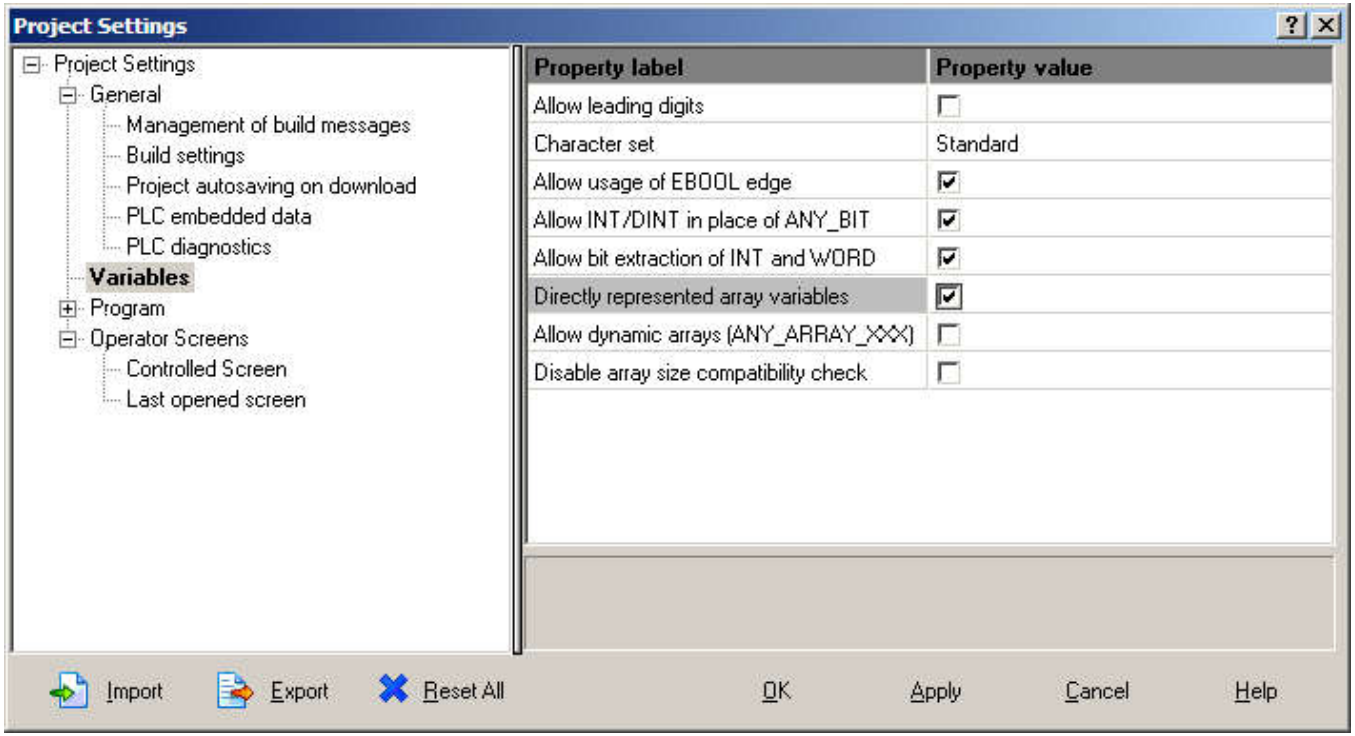
EXIT to abort the execution of a structured flux control instruction

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 Numerical Tables

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 Example

Memory based logging

Uses:
%MW96..%MW99 aux data,
%MW100..%MW139 buffer

Other variables:
acc, last_acc,
scan_cycle_num

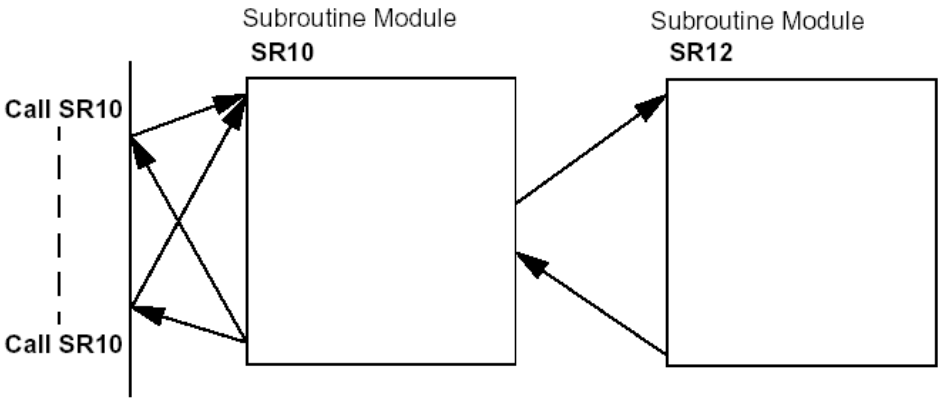
```
(* Mark in memory the data logging will be happening *)
IF %MW97 = 0 THEN
  %MW97 := 12345;
  %MW98 := 12345;
  %MW99 := 2010; (* Matlab matrix 2 x 10 *)
END_IF;

(* Create a word collecting all inputs *)
(* "acc" datatype is INT *)

acc:=0;
IF %i0.2.15 THEN inc(acc); END_IF; acc:= ROL(acc,1);
IF %i0.2.14 THEN inc(acc); END_IF; acc:= ROL(acc,1);
IF %i0.2.13 THEN inc(acc); END_IF; acc:= ROL(acc,1);
IF %i0.2.12 THEN inc(acc); END_IF; acc:= ROL(acc,1);
IF %i0.2.11 THEN inc(acc); END_IF; acc:= ROL(acc,1);
IF %i0.2.10 THEN inc(acc); END_IF; acc:= ROL(acc,1);
IF %i0.2.9 THEN inc(acc); END_IF; acc:= ROL(acc,1);
IF %i0.2.8 THEN inc(acc); END_IF; acc:= ROL(acc,1);
IF %i0.2.7 THEN inc(acc); END_IF; acc:= ROL(acc,1);
IF %i0.2.6 THEN inc(acc); END_IF; acc:= ROL(acc,1);
IF %i0.2.5 THEN inc(acc); END_IF; acc:= ROL(acc,1);
IF %i0.2.4 THEN inc(acc); END_IF; acc:= ROL(acc,1);
IF %i0.2.3 THEN inc(acc); END_IF; acc:= ROL(acc,1);
IF %i0.2.2 THEN inc(acc); END_IF; acc:= ROL(acc,1);
IF %i0.2.1 THEN inc(acc); END_IF; acc:= ROL(acc,1);
IF %i0.2.0 THEN inc(acc); END_IF;

(* Save the word and the scan cycle time, 10x each *)
(* "scan_cycle_num" datatype is INT *)

scan_cycle_num := scan_cycle_num +1;

IF acc <> last_acc AND %MW96<20 THEN
  %MW100[%MW96] := scan_cycle_num;
  %MW96 := %MW96+1;
  %MW100[%MW96] := acc;
  %MW96 := %MW96+1;
  last_acc := acc;
END_IF;
```

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 Subroutines in Unity Pro

Subroutine call example: `SubroutineName() ;`

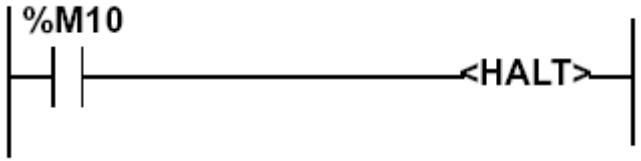
Note name of the subroutine section followed by an empty parameter list. Subroutine calls do not return a value. The subroutine to be called must be located in the same task as the ST section called. Subroutines can also be called from within subroutines. Subroutine calls are a supplement to IEC 61131-3 and must be enabled explicitly. In SFC action sections, subroutine calls are only allowed when Multitoken Operation is enabled.

RETURN instructions can be used in DFBs (derived function blocks) and in SRs (subroutines). Cannot be used in the main program. In a DFB or a SR, a RETURN instruction forces the return to the program which called the DFB or the SR. The rest of the DFB (or SR) section containing the RETURN instruction is not executed. The next sections of the DFB (or SR) are not executed. The program which called the DFB (or SR) will be executed after return from the DFB (or SR). If the DFB (or SR) is called by another DFB (or SR), the calling DFB (or SR) will be executed after return.

Structured Text

Structures for Control of Flux

Halt

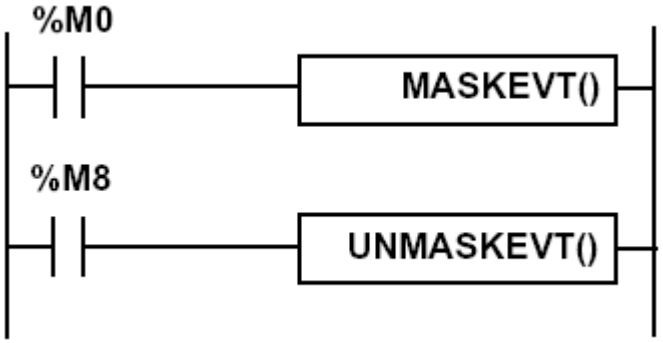


Structured text language

```
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

Data and time related instructions

Name	Function
SCHEDULE	Time function
RRTC	Reading system date
WRTC	Updating system date
PTC	Reading date and stop code
ADD_TOD	Adding a duration to a time of day
ADD_DT	Adding a duration to a date and time
DELTA_TOD	Measuring the gap between times of day
DELTA_D	Measuring the gap between dates (without time).
DELTA_DT	Measuring the gap between dates (with time).
SUB_TOD	Totaling the time to date
SUB_DT	Totaling the time to date and time
DAY_OF_WEEK	Reading the current day of the week
TRANS_TIME	Converting duration into date
DATE_TO_STRING	Converting a date to a character string
TOD_TO_STRING	Converting a time to a character string
DT_TO_STRING	Converting a whole date to a character string
TIME_TO_STRING	Converting a duration to a character string

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

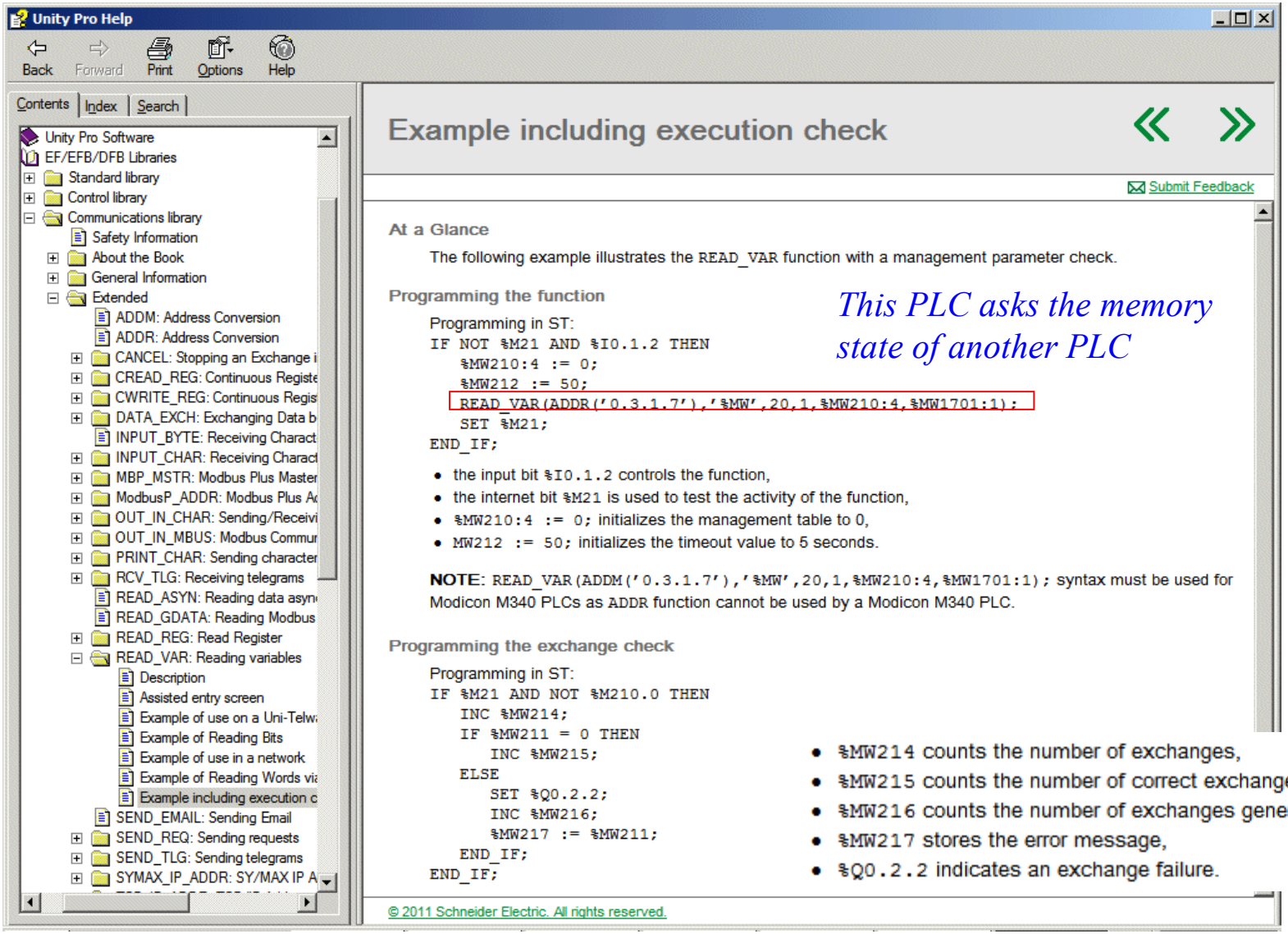
• Casting

```
%MW104 := BOOL_TO_INT( %i0.3.0 AND %i0.3.4 );

IF INT_TO_BOOL(%MW203)
THEN SET(%q0.3.18);
ELSE RESET(%q0.3.18);
END_IF;
```

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 *Networking (in Unity Pro)*



Keywords: MODBUS, READ_VAR, WRITE_VAR