Industrial Automation
(Automação de Processos Industriais)

PLC Programming languages
Structured Text

http://users.isr.ist.utl.pt/~jag/courses/api1213/api1213.html

Slides 2010/2011 Prof. Paulo Jorge Oliveira
Rev. 2011-2013 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;
ENDIF;
ELSE
  %M1:=FALSE;
ENDIF;
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:

```plaintext
D := B*B - 4*A*C;
IF D < 0.0 THEN NROOTS := 0;
ELSIF D = 0.0 THEN
    NROOTS := 1;
    X1 := -B/(2.0*A);
ELSE
    NROOTS := 2;
    X1 := (-B + SQRT(D))/(2.0*A);
    X2 := (-B - SQRT(D))/(2.0*A);
END_IF;
```

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**

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>:=</td>
<td>Open contact: contact is active (result is 1) while the control bit is 1.</td>
</tr>
<tr>
<td>:=NOT</td>
<td>Close contact: contact is active (result is 1) while the control bit is 0.</td>
</tr>
<tr>
<td>:=RE</td>
<td>Contact in the rising edge: contact is active during a scan cycle where the control bit has a rising edge.</td>
</tr>
<tr>
<td>:=FE</td>
<td>Contact in the falling edge: contact is active during a scan cycle where the control bit has a falling edge.</td>
</tr>
</tbody>
</table>

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

Basic Instructions

**Store**

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>:=</td>
<td>The result of the logic function activates the coil.</td>
</tr>
<tr>
<td>:=NOT</td>
<td>The inverse result of the logic function activates the coil.</td>
</tr>
<tr>
<td>SET</td>
<td>The result of the logic function energizes the relay (sets the latch).</td>
</tr>
<tr>
<td>RESET</td>
<td>The result of the logic function de-energizes the relay (resets the latch).</td>
</tr>
</tbody>
</table>

Examples:  
```
%M100 := 123; %Q0.4.0 := NOT %M1; %M0 := TRUE; SET(%Q0.4.0);
```
Structured Text

Basic Instructions

**AND**

<table>
<thead>
<tr>
<th>AND</th>
<th>AND(NOT...)</th>
<th>AND(RE...)</th>
<th>AND(FE...)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

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

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

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:

\[
\begin{align*}
\%Q2.3 &: = \%I1.1 \text{ OR } \%M1; \\
\%Q2.2 &: = \%M2 \text{ OR } \text{NOT}\%I1.2; \\
\%Q2.4 &: = \%I1.3 \text{ OR } \text{RE}\%I1.4; \\
\%Q2.5 &: = \%M3 \text{ OR } \text{FE}\%I1.5;
\end{align*}
\]

Unity Pro (Premium PLC):

\[
\begin{align*}
\%Q0.4.3 &: = \%I0.2.1 \text{ OR } \%M1; \\
\%Q0.4.2 &: = \%M2 \text{ OR } \text{NOT}\%I0.2.2; \\
\%Q0.4.4 &: = \%I0.2.3 \text{ OR } \text{RE}\%I0.2.4; \\
\%Q0.4.5 &: = \%M3 \text{ OR } \text{FE}\%I0.2.5;
\end{align*}
\]
Structured Text

Basic Instructions

**XOR**

<table>
<thead>
<tr>
<th>Instruction list</th>
<th>Structured text</th>
<th>Description</th>
<th>Timing diagram</th>
</tr>
</thead>
<tbody>
<tr>
<td>XOR</td>
<td>XOR</td>
<td>OR Exclusive between the operand and the previous instruction’s Boolean result</td>
<td></td>
</tr>
<tr>
<td>XORN</td>
<td>XOR (NOT...)</td>
<td>OR Exclusive between the operand inverse and the previous instruction’s Boolean result</td>
<td></td>
</tr>
<tr>
<td>XORR</td>
<td>XOR (RE...)</td>
<td>OR Exclusive between the operand’s rising edge and the previous instruction’s Boolean result</td>
<td></td>
</tr>
<tr>
<td>XORF</td>
<td>XOR (FE...)</td>
<td>OR Exclusive between the operand’s falling edge and the previous instruction’s Boolean result</td>
<td></td>
</tr>
</tbody>
</table>

Unity Pro (Premium PLC):

\[
\text{XOR: } \%Q0.4.3 := \%I0.2.1 \text{ XOR } \%M1; \\
\text{XOR: } \%Q0.4.4 := \%I0.2.3 \text{ XOR } \%I0.4.2; \\
\text{XOR: } \%Q0.4.5 := \%M3 \text{ XOR } \%F0.2.5; \\
\]
### Structured Text

#### Basic Instructions to Manipulate Bit Tables

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

**Temporized Relays or Timers (pneumatic)**

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**

<table>
<thead>
<tr>
<th>%TMi</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN</td>
</tr>
<tr>
<td>Q</td>
</tr>
</tbody>
</table>

---

**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
Structured Text

Example:

![Diagram](image1)

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)

![Diagram](image2)

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.

(b)

**Fig. 7-3**

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

**Fig. 7-4**

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

NOTE: counters are saturated such that no overflow occurs
Structured Text

Counters in Unity Pro

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

```plaintext
%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**

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>addition of two operands</td>
<td>SQRT</td>
</tr>
<tr>
<td>-</td>
<td>subtraction of two operands</td>
<td>INC</td>
</tr>
<tr>
<td>*</td>
<td>multiplication of two operands</td>
<td>DEC</td>
</tr>
<tr>
<td>/</td>
<td>division of two operands</td>
<td>ABS</td>
</tr>
<tr>
<td>REM</td>
<td>remainder from the division of 2 operands</td>
<td></td>
</tr>
</tbody>
</table>

### Operands

<table>
<thead>
<tr>
<th>Type</th>
<th>Operand 1 (Op1)</th>
<th>Operand 2 (Op2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Indexable words</td>
<td>%MW</td>
<td>%MW,%KW,%Xi.T</td>
</tr>
<tr>
<td>Non-indexable words</td>
<td>%QW,%SW,%NW,%BLK</td>
<td>Imm.Val.,%IW,%QW,%SW,%NW,%BLK, Num.expr.</td>
</tr>
<tr>
<td>Indexable double words</td>
<td>%MD</td>
<td>%MD,%KD</td>
</tr>
<tr>
<td>Non-indexable double words</td>
<td>%QD,%SD</td>
<td>Imm.Val.,%ID,%QD,%SD, Numeric expr.</td>
</tr>
</tbody>
</table>
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*

Example:

Arithmetic functions

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

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

Use of a system variable:
%S18 – flag de overflow
Structured Text

Numerical Processing

Logic Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND</td>
<td>AND (bit by bit) between two operands</td>
</tr>
<tr>
<td>OR</td>
<td>logical OR (bit by bit) between two operands</td>
</tr>
<tr>
<td>XOR</td>
<td>exclusive OR (bit by bit) between two operands</td>
</tr>
<tr>
<td>NOT</td>
<td>logical complement (bit by bit) of an operand</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Type</th>
<th>Operands 1 and 2 (Op1 and Op2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Indexable words</td>
<td><code>%MW,%KW,%Xi.T</code></td>
</tr>
<tr>
<td>Non-indexable words</td>
<td>Imm.val.,%lW,%QW,%SW,%NW,%BLK, Numeric Expr.</td>
</tr>
<tr>
<td>Indexable double words</td>
<td><code>%MD,%KD</code></td>
</tr>
<tr>
<td>Non-indexable double words</td>
<td>Imm.val.,%ID,%QD,%SD,Numeric expr.</td>
</tr>
</tbody>
</table>
Structured Text

Numerical Processing

Example:

Logic functions

Structured text language

\[
\begin{align*}
\%Q2.3 & : = \%MW10 > 100; \\
\%Q2.2 & : = \%M0 \text{ AND } (\%MW20 < \%KW35); \\
\%Q2.4 & : = \%I1.2 \text{ OR } (\%MW30 \geq \%MW40);
\end{align*}
\]
Structured Text

Numerical Processing

Example:

Numeric Tables Manipulation

Structured text language

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

Numerical Processing

Priorities on the execution of the operations

<table>
<thead>
<tr>
<th>Rank</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Instruction to an operand</td>
</tr>
<tr>
<td>2</td>
<td>*, /, REM</td>
</tr>
<tr>
<td>3</td>
<td>+, -</td>
</tr>
<tr>
<td>4</td>
<td>&lt;, &gt;, &lt;=, =&gt;</td>
</tr>
<tr>
<td>5</td>
<td>=, &lt;&gt;</td>
</tr>
<tr>
<td>6</td>
<td>AND</td>
</tr>
<tr>
<td>7</td>
<td>XOR</td>
</tr>
<tr>
<td>8</td>
<td>OR</td>
</tr>
</tbody>
</table>
Structured Text

Structures for Control of Flux

Subroutines

Call and Return

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 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 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;

Jump to label %L10
if %M8=1

%L10:
  %M5:=%M20;
  %Q2.1:=%I1.0 AND %I1.2;

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 ...

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>IF</strong> condition1 <strong>THEN</strong>&lt;br&gt;actions1;&lt;br&gt;<strong>ELSEIF</strong> condition2 <strong>THEN</strong>&lt;br&gt;actions2;&lt;br&gt;<strong>ELSE</strong>&lt;br&gt;actions3;&lt;br&gt;<strong>END_IF</strong>;</td>
<td><strong>Beginning of IF</strong>&lt;br&gt;checked&lt;br&gt;<strong>Condition 1</strong>&lt;br&gt;<strong>not checked</strong>&lt;br&gt;<strong>Actions 1</strong>&lt;br&gt;<strong>Condition 2</strong>&lt;br&gt;<strong>not checked</strong>&lt;br&gt;<strong>Actions 2</strong>&lt;br&gt;<strong>Actions 3</strong>&lt;br&gt;<strong>End of IF</strong></td>
</tr>
</tbody>
</table>
Structured Text

Structures for Control of Flux

**WHILE**

Syntax

```plaintext
WHILE condition DO
    action;
END_WHILE;
```

Operation

![Diagram of WHILE structure]

Example:

```plaintext
(*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:

```plaintext
(* using EXIT to break a loop *)
WHILE %MW1<124 DC
%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

<table>
<thead>
<tr>
<th>%M10</th>
</tr>
</thead>
<tbody>
<tr>
<td>---</td>
</tr>
<tr>
<td>&lt;HALT&gt;</td>
</tr>
</tbody>
</table>

Structured text language
IF %M10 THEN
    HALT;
END_IF;

Stops all processes!

Events masking

<table>
<thead>
<tr>
<th>%M0</th>
</tr>
</thead>
<tbody>
<tr>
<td>---</td>
</tr>
<tr>
<td>MASKEVT()</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>%M8</th>
</tr>
</thead>
<tbody>
<tr>
<td>---</td>
</tr>
<tr>
<td>UNMASKEVT()</td>
</tr>
</tbody>
</table>

Structured text language
IF %M0 THEN
    MASKEVT();
END_IF;
IF %M8 THEN
    UNMASKEVT();
END_IF;
## Structured Text

### Data and time related instructions

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

### Numerical Tables

<table>
<thead>
<tr>
<th>Type</th>
<th>Format</th>
<th>Maximum address</th>
<th>Size</th>
<th>Write access</th>
</tr>
</thead>
<tbody>
<tr>
<td>Internal words</td>
<td>Simple length</td>
<td>%MWi:L</td>
<td>i+L&lt;=Nmax (1)</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>Double length</td>
<td>%MWDi:L</td>
<td>i+L&lt;=Nmax-1 (1)</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>Floating point</td>
<td>%MFi:L</td>
<td>i+L&lt;=Nmax-1 (1)</td>
<td>Yes</td>
</tr>
<tr>
<td>Constant words</td>
<td>Single length</td>
<td>%KWi:L</td>
<td>i+L&lt;=Nmax (1)</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>Double length</td>
<td>%KWDi:L</td>
<td>i+L&lt;=Nmax-1 (1)</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>Floating point</td>
<td>%KFi:L</td>
<td>i+L&lt;=Nmax-1 (1)</td>
<td>No</td>
</tr>
<tr>
<td>System word</td>
<td>Single length</td>
<td>%SW50:4 (2)</td>
<td>-</td>
<td>Yes</td>
</tr>
</tbody>
</table>

### Instruction list language

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

LD %I3.2
[ %MW50:5=%KD0:5+%MD0:5 ]

LD %I3.3
[ %MW0:10=%KW0:10*%MW20 ]
```

Page 38