Industrial Automation
(Automação de Processos Industriais)

PLC Programming Languages

Instruction List

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-61131-3):
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)
### Instruction list

#### Antique PLC

<table>
<thead>
<tr>
<th>AI1</th>
<th>AI3</th>
<th>LDV50</th>
</tr>
</thead>
<tbody>
<tr>
<td>A(</td>
<td>=P9</td>
<td>=CSW9</td>
</tr>
<tr>
<td>OI2</td>
<td>NO</td>
<td>PE</td>
</tr>
<tr>
<td>O(</td>
<td>OM1</td>
<td></td>
</tr>
<tr>
<td>ANC9</td>
<td>OI4</td>
<td></td>
</tr>
<tr>
<td>AQ9</td>
<td>=Z9</td>
<td></td>
</tr>
<tr>
<td>)</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>)</td>
<td>AC9</td>
<td></td>
</tr>
<tr>
<td>=Q9</td>
<td>=M1</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Instruction list  Reference – see Unity Pro dev. environment
Instruction list

Reference – Unity Pro Help

General Information about the IL Instruction List

Introduction

Using the Instruction list programming language (IL), you can call function blocks and functions conditionally or unconditionally, perform assignments and make jumps conditionally or unconditionally within a section.

Instructions

An instruction list is composed of a series of instructions. Each instruction begins on a new line and consists of:

- an **Operator**, if necessary with a **Modifier** and if necessary one or more **Operands**.

Should several operands be used, they are separated by commas. It is possible for a **Label** to be in front of the instruction. This label is followed by a colon. A **Comment** can follow the instruction.

Example:

```
START:  LD A (* Key 1 *)
ANDW B (* and not key 2 *)
ST C (* Ventilator on *)
```

© 2009 Schneider Electric. All rights reserved.
Instruction list  *Reference – Unity Pro Help*

**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.

**IL is a so-called accumulator orientated language**, i.e. each instruction uses or alters the current content of the accumulator (a form of internal cache). IEC 61131 refers to this accumulator as the "result". For this reason, an instruction list should always begin with the LD operand ("Load in accumulator command").

An **Instruction list (IL)** is composed of a series of instructions. Each instruction begins on a new line and consists of:
- an **Operator**,
- if necessary with a **Modifier** and
- if necessary one or more **Operands**
**Instruction list**

**Basic Instructions**

**Load**

<table>
<thead>
<tr>
<th>LD</th>
<th>Open contact: contact is active (result is 1) while the control bit is 1.</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDN</td>
<td>Close contact: contact is active (result is 1) while the control bit is 0.</td>
</tr>
<tr>
<td>LDR</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>LDF</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>
**Store**

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ST</strong></td>
<td>The result of the logic function activates the coil.</td>
</tr>
<tr>
<td><strong>STN</strong></td>
<td>The inverse result of the logic function activates the coil.</td>
</tr>
<tr>
<td><strong>S</strong></td>
<td>The result of the logic function energizes the relay (sets the latch).</td>
</tr>
<tr>
<td><strong>R</strong></td>
<td>The result of the logic function de-energizes the relay (resets the latch).</td>
</tr>
</tbody>
</table>
Instruction list

Basic Instructions

**AND**

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND</td>
<td>AND of the operand with the result of the previous logical operation.</td>
</tr>
<tr>
<td>ANDN</td>
<td>AND of the operand with the inverted result of the previous logical operation.</td>
</tr>
<tr>
<td>ANDR</td>
<td>AND of the rising edge with the result of the previous logical operation.</td>
</tr>
<tr>
<td>ANDF</td>
<td>AND of the falling edge with the result of the previous logical operation.</td>
</tr>
</tbody>
</table>
### Instruction list

#### Basic Instructions

**OR**

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OR</td>
<td>OR of the operand with the result of the previous logical operation.</td>
</tr>
<tr>
<td>ORN</td>
<td>OR of the operand with the inverted result of the previous logical operation.</td>
</tr>
<tr>
<td>ORR</td>
<td>OR of the rising edge with the result of the previous logical operation.</td>
</tr>
<tr>
<td>ORF</td>
<td>OR of the falling edge with the result of the previous logical operation.</td>
</tr>
</tbody>
</table>
Instruction list

Example:

```
LD %I1.1
OR %M1
ST %Q2.3

LD %M2
ORN %I1.2
ST %Q2.2

LD %I1.3
ORR %I1.4
ST %Q2.4

LD %M3
ORF %I1.5
ST %Q2.5
```
### Instruction list

#### 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>

```plaintext
LD %I1.1
XOR %M1
ST %Q2.3
LD %M2
XOR %I1.2
ST %Q2.2
... 
```
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.
Instruction list

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)

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. 7–3
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.
### Characteristics:

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

#### Temporized Relays or Timers (PL7)

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

**Instruction list**

- IST / DEEC / API
- Chap. 3 - PLC Programming languages

---

**Temporized Relays or Timers (PL7)**

- Identifier: %TMi
- 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
**Instruction list**

*Temporized Relays or Timers (PL7)*

```
LD          %I1.1
IN          %TM1
LD          %TM1.Q
ST          %Q2.3
```

```
%I1.1  IN  Q  %TM1  %Q2.3
```
Instruction list

Temporized Relays
or Timers (Unity)

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

Counters

Some applications...

Fig. 8-3

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

Counters in PL7

Example:

Instruction list language

LD %I1.1
R %C8
LD %I1.2
AND %M0
CU %C8
LD %C8.D
ST %Q2.0
Ladder diagram

Counters in Unity Pro

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

CV ≥ PV => Q:=1

R=1 => CV:=0

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
Ladder diagram

Counters in Unity Pro

```plaintext
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*)
```

```plaintext
CTUD_INT
EN  ENO
%I0.2.11
CU  QU
%Q0.4.1
%I0.2.12
CD  QD
%Q0.4.2
%I0.2.13
R
%I0.2.14
LD
123
PV
CV
```
Instruction list

Numerical Processing

Algebraic and Logic Functions (PL7)

LD [%MW50>10]
ST %Q2.2
LD %I1.0
[=%MW10:=%KW0+10]
LDF %I1.2
[INC%MW100]
### Instruction list

#### Numerical Processing

#### Arithmetic Functions

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>addition of two operands</td>
<td>+</td>
</tr>
<tr>
<td>-</td>
<td>subtraction of two operands</td>
<td>-</td>
</tr>
<tr>
<td>*</td>
<td>multiplication of two operands</td>
<td>*</td>
</tr>
<tr>
<td>/</td>
<td>division of two operands</td>
<td>/</td>
</tr>
<tr>
<td>REM</td>
<td>remainder from the division of 2 operands</td>
<td>REM</td>
</tr>
<tr>
<td>SQRT</td>
<td>square root of an operand</td>
<td>SQRT</td>
</tr>
<tr>
<td>INC</td>
<td>incrementation of an operand</td>
<td>INC</td>
</tr>
<tr>
<td>DEC</td>
<td>decrementation of an operand</td>
<td>DEC</td>
</tr>
<tr>
<td>ABS</td>
<td>absolute value of an operand</td>
<td>ABS</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>
Instruction list

Numerical Processing

Example:

Arithmetic functions

**PL7:**

Instruction list language

LD %M0
[%MW0:=%MW10+100]

LD %I3.2
[ %MW0:=SQRT(%MW10) ]

LD %I3.3
[ INC %MW100 ]
Instruction list

Numerical Processing

Example:

Arithmetic functions

PL7:

Example in instruction list language:

LD %M0
[ %MW0 := %MW1 + %MW2 ]
LDN %S18
[ %MW10 := %MW0 ]
LD %S18
[ %MW10 := 32767 ]
R %S18

Use of a system variable:

%S18 – flag de overflow
Instruction list

Numerical Processing

Logic Functions

<table>
<thead>
<tr>
<th>Instruction</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>%MW, %KW, %Xi.T</td>
</tr>
<tr>
<td>Non-indexable words</td>
<td>Imm.val., %IW, %QW, %SW, %NW, %BLK, Numeric Expr.</td>
</tr>
<tr>
<td>Indexable double words</td>
<td>%MD, %KD</td>
</tr>
<tr>
<td>Non-indexable double words</td>
<td>Imm.val., %ID, %QD, %SD, Numeric expr.</td>
</tr>
</tbody>
</table>
Instruction list

Numerical Processing

Example:

Logic functions

\[
\begin{align*}
\text{PL7:} & \quad [\%MW10>100] \\
\text{LD } & \%M0 \quad [\%MW20<\%KW35] \\
\text{ST } & \%Q2.2 \quad [\%MW30>=\%MW40] \\
\text{LD } & \%I1.2 \quad [\%Q2.3] \\
\text{AND } & \%Q2.4
\end{align*}
\]
Instruction list

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>
Instruction list

Structures for Control of Flux

Subroutines

Call and Return

Ladder language:

Instruction list language:

IST / DEEC / API

Chap. 3 - PLC Programming languages

Page 30
Description of Subroutines

Overview of Subroutines

Subroutines are programmed as separate entities, either in:

- Ladder language (LD),
- Functional block language (FBD),
- Instruction List (IL),
- Structured Text (ST).

The calls to subroutines are carried out in the sections or from another subroutine.

The number of nestings is limited to 8.

A subroutine cannot call itself (non recursive).

Subroutines are also linked to a task. The same subroutine cannot be called from several different tasks.

Example

The following diagram shows a task structured into sections and subroutines.

```
  - MAST
     - Sections
       - SR Sections
         - Control_1
         - Detection
```
Instruction list

Structures for Control of Flux

JUMP instructions:

Conditional and unconditional

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)
Instruction list

Structures for Control of Flux

Example:

Use of jump instructions

```
Instruction list language

LD  %M8
JMP %L10
LD  %I1.0
ST  %Q2.5

----------
%L10:
LD  %M20
ST  %M5
LD  %I1.0
AND %I1.2
ST  %Q2.1
```

Jump to label %L10, if %M8 =1

Ladder

```
%M8
%M20
%M5

%L10

%I1.0 %I1.2

%Q2.1

%L10

%I1.0

%Q2.5

%L10

%L10

%I1.0

%I1.0

%L10

%L10

%L10

%L10

%L10
```
Instruction list

Structures for Control of Flux

Halt

%M10

Stops all processes!

Events masking

%M0

MASKEVT()

%M8

UNMASKEVT()
Instruction list

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
Instruction list

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>%WMDi:L</td>
<td>i+L&lt;=Nmax-1 (1)</td>
<td>Yes</td>
</tr>
<tr>
<td>Floating point</td>
<td>%MFi:L</td>
<td>i+L&lt;=Nmax-1 (1)</td>
<td></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>Floating point</td>
<td>%KFi:L</td>
<td>i+L&lt;=Nmax-1 (1)</td>
<td></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>

**PL7:**

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]
DOLOG80

PLC AEG A020 Plus:

Inputs:
• 20 binary with opto-couplers
• 4 analogs (8 bits, 0-10V)

Outputs:
• 16 binary with relays of 2A
• 1 analogs (8 bits, 0-10V)

Interface for progr.: RS232

Processador:
• 8031
• 2 Kbytes de RAM
• 2 Kbytes EEPROM => 896 instructions
• Average cycle time: 6.5 ms
PLC AEG A020 Plus

DOLOG80

OPERANDS

• I1 to I20 Binary inputs
• Q1 to Q16 Binary outputs
• M1 to M128 Auxiliary memory
• T1 to T8 Timers (base 100ms)
• T9 to T16 Timers (base 25ms)
• C1 to C16 16 bits counters
DOLOG80 (cont.)

Example:

AI1 → AI3 → LDV50 = CSW9
A( OI2 NO OM1 OI4 Z9
O( ANC9 NO AC9 M1
AQ9 NO
)
)
= Q9

Legend:  
Stop = I1  
Start = I2  
Proximity Sensor = I3  
Reset = I4  
Counter = C9  
Internal relay = M1  
Motor = Q9  

Fig. 6-13  
Conveyor motor program.