

# **Industrial Automation**

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

### **PLC Programming Languages**

#### ***Instruction List***

<http://users.isr.ist.utl.pt/~jag/courses/api1819/api1819.html>

Prof. Paulo Jorge Oliveira, original slides

Prof. José Gaspar, rev. 2018/2019

## 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.**

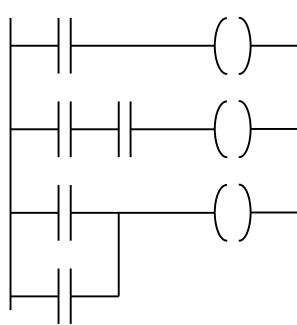
Software development resources.

...

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

## PLC Programming languages (IEC 61131-3)

### *Ladder Diagram*



### *Structured Text*

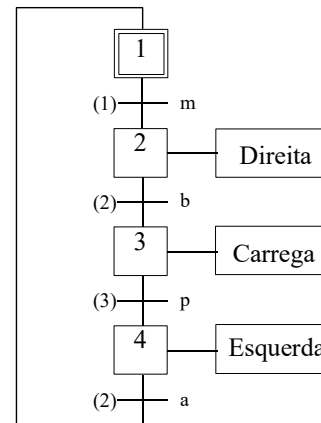
```

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

### *Instruction List*

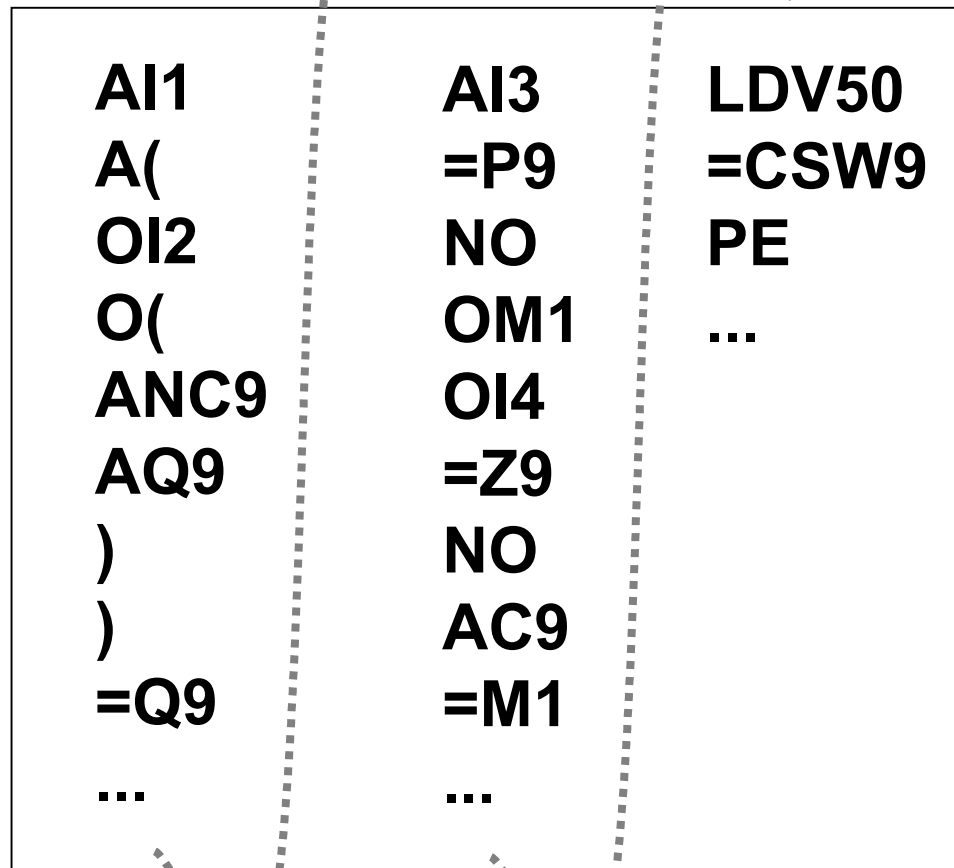
LD	%M12
AND	%I1.0
ANDN	%I1.1
OR	%M10
ST	%Q2.0

### *Sequential Function Chart (GRAFCET)*

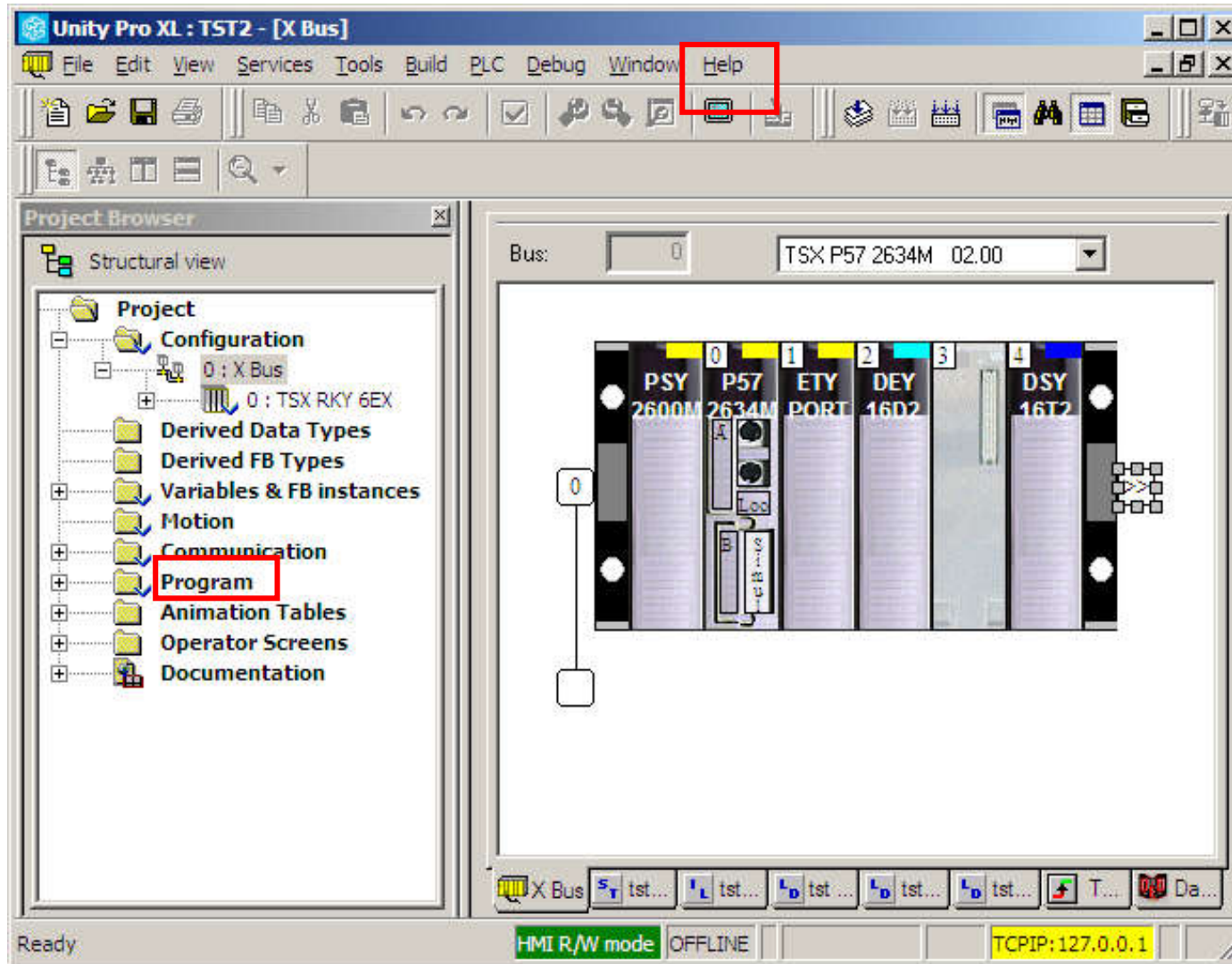


## Instruction list

### *Antique PLC*



Instruction list *Reference – see Unity Pro dev. environment*



# Instruction list

*Reference  
– Unity Pro  
Help*

**Unity Pro Help**

Back Forward Print Options Help

Contents Index Search

Unity

- Whats New
- General Safety Instructions
- Compatibility Rules
- Addendum
- Unity Pro Software
  - Languages Reference
    - Safety Information
    - About the Book
    - General Presentation of Unity Pro
    - Application Structure
    - Data Description
    - Programming Language
      - Function Block Language FBD
      - Ladder Diagram (LD)
      - SFC Sequence Language
      - Instruction List (IL)
        - General Information about the IL Instruction List
          - General Information about the IL Instruction List
          - Operands
          - Modifier
          - Operators
          - Subroutine Call
          - Labels and Jumps
          - Comment
        - Calling Elementary Functions
        - Structured Text (ST)
        - User Function Blocks (DFB)
      - Appendix
      - Glossary
    - Operating Modes
    - OSLoader
    - Unity Loader
    - Concept Converter
    - PL7 converter
    - Communication Drivers
    - EF/EFB/DFB Libraries
    - Communication architectures

**General Information about the IL Instruction List**

See: [Related Topics](#)  [Submit Feedback](#)

**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 *)
      ANDN B (* and not key 2 *)
      ST C (* Ventilator on *)
    
```

Diagram labels in the example:

- Label: START:
- Operator: LD, ANDN, ST
- Modifier: ANDN
- Operands: A, B, C
- Comments: (\* Key 1 \*), (\* and not key 2 \*), (\* Ventilator on \*)

© 2009 Schneider Electric. All rights reserved.

## Instruction list *Reference – Unity Pro Help*

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

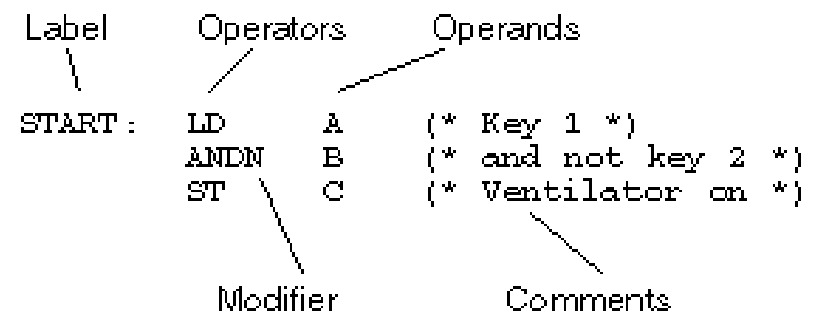
One sequence is equivalent to one or more rungs in *ladder diagram*.

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

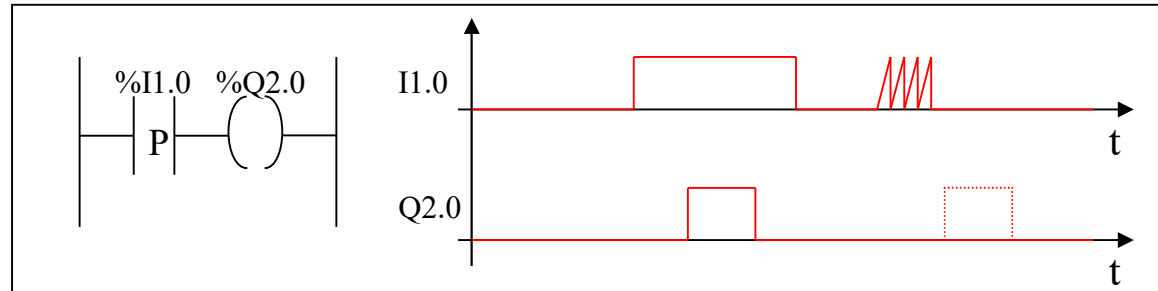
**IL is a so-called accumulator oriented language**, i.e. each instruction uses or alters the current content of the accumulator (a form of internal cache). IEC 61131 refers to this accumulator as the "result". For this reason, an instruction list should always begin with the LD operand ("Load in accumulator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



<b>LD</b>	
<b>LDN</b>	
<b>LDR</b>	
<b>LDF</b>	

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.

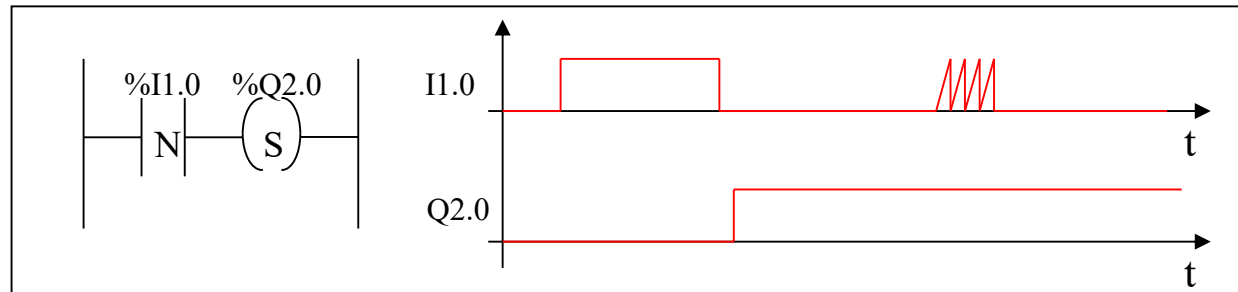
Contact in the rising edge: contact is active during a scan cycle where the control bit has a **rising edge**.

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



**Instruction list**  
**Basic Instructions**

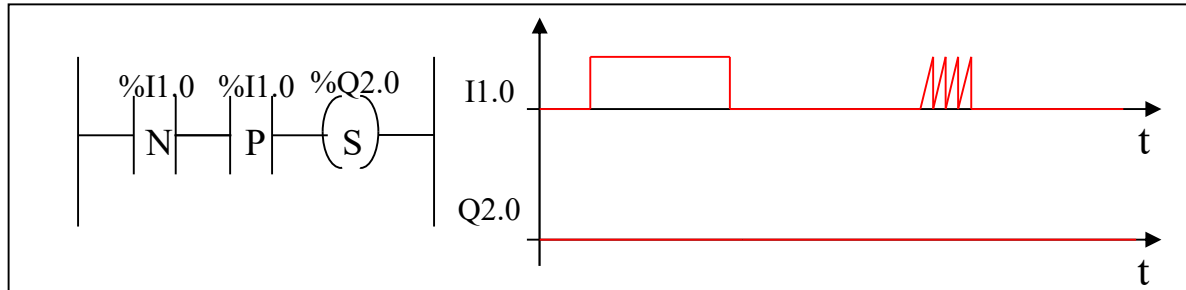
*Store*



<b>ST</b>		The result of the logic function activates the coil.
<b>STN</b>		The inverse result of the logic function activates the coil.
<b>S</b>		The result of the logic function energizes the relay ( <b>sets</b> the latch).
<b>R</b>		The result of the logic function de-energizes the relay ( <b>resets</b> the latch)..

**Instruction list**  
**Basic Instructions**

*AND*



<b>AND</b>	
<b>ANDN</b>	
<b>ANDR</b>	
<b>ANDF</b>	

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.

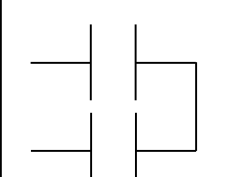
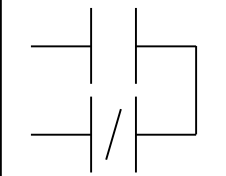
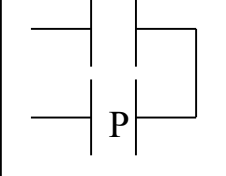
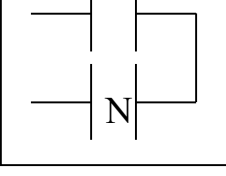
AND of the **rising edge** with the result of the previous logical operation.

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

## Instruction list

### Basic Instructions

# OR

<b>OR</b>	
<b>ORN</b>	
<b>ORR</b>	
<b>ORF</b>	

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

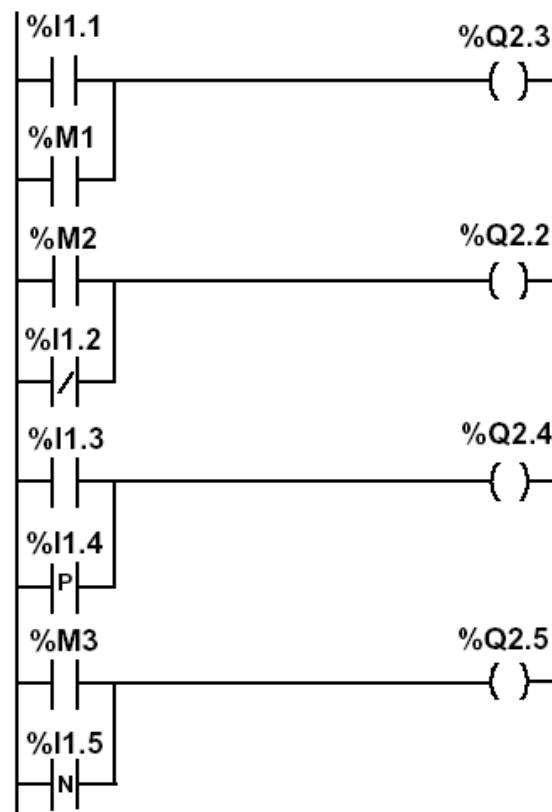
OR of the operand with the **inverted** result of the previous logical operation.

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

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

## Instruction list

### Example:



```

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

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

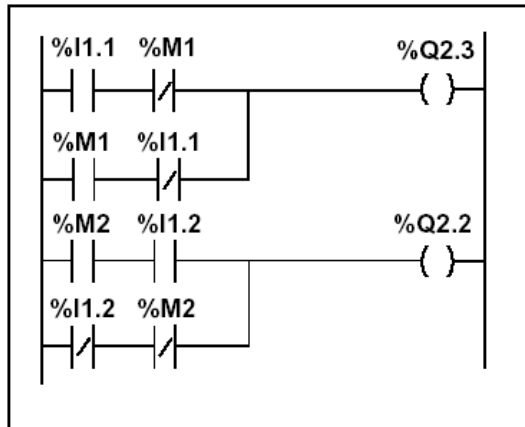
LD %I1.3
ORR %I1.4
ST %Q2.4

LD %M3
ORF %I1.5
ST %Q2.5
  
```

# Instruction list

## Basic Instructions

### XOR



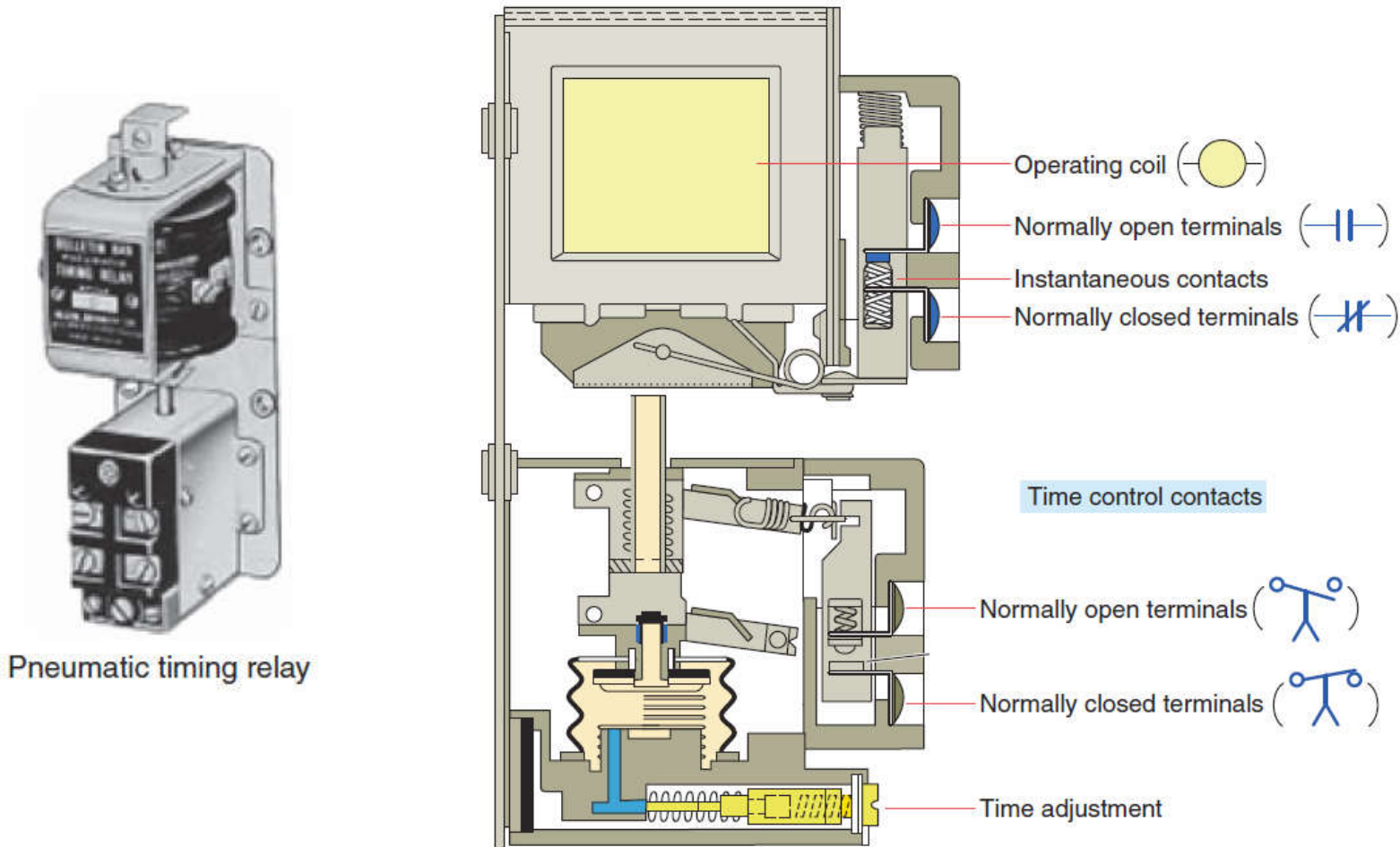
```

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

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.	

Instruction list

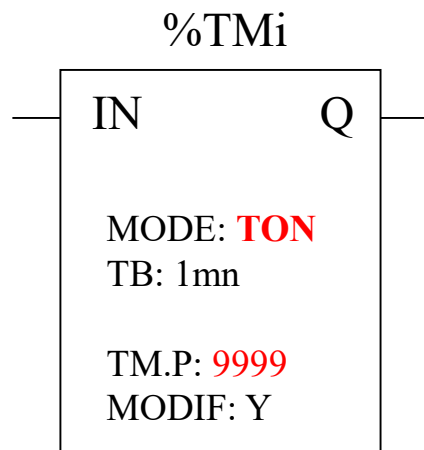
*Temporized Relays or Timers (pneumatic)*



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.

## Instruction list

### *Temporized Relays or Timers (PL7)*

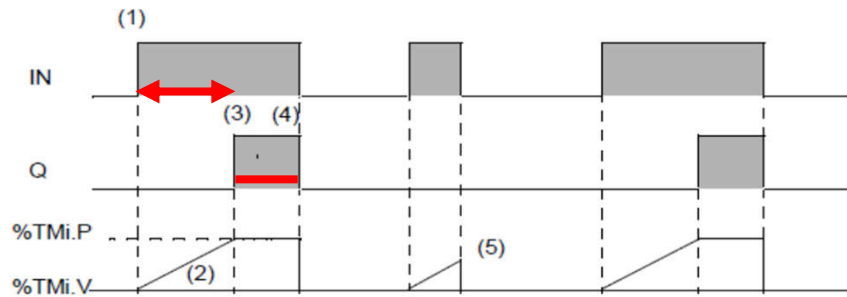


### Characteristics:

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

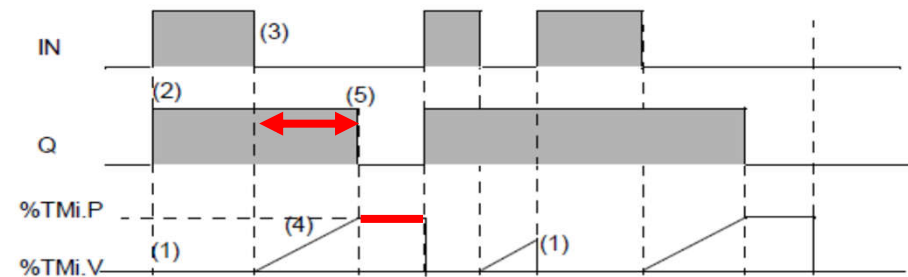
# Timers

## TON mode



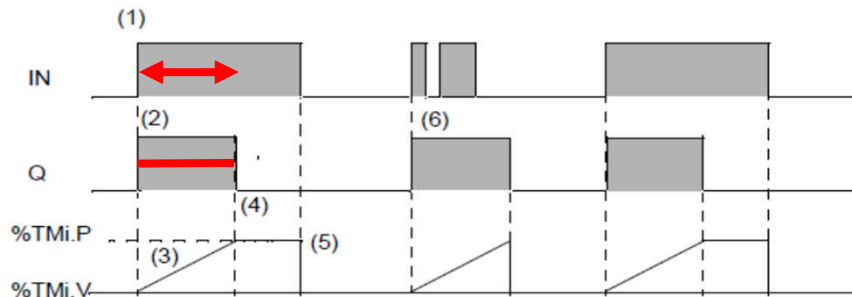
*App. example: start ringing the alarm if N sec after door open there is no disarm of the alarm.*

## TOF mode



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

## TP mode

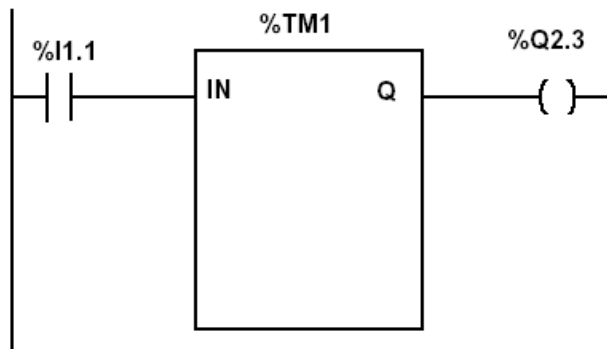


*App. example: positive input edge give a controlled (fixed) duration pulse to start a motor.*



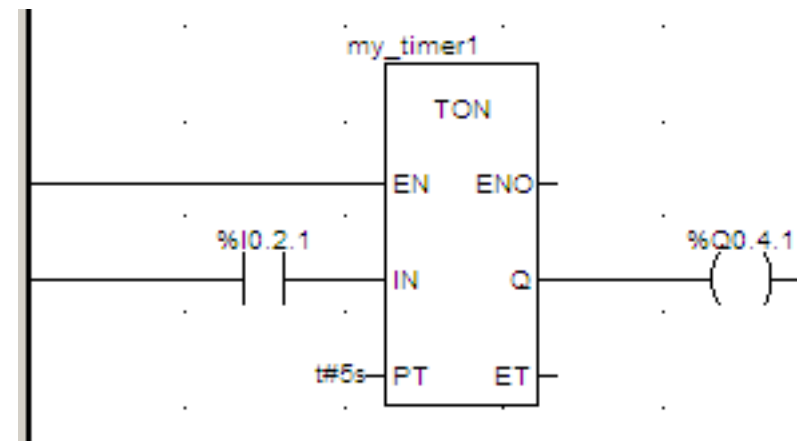
## Instruction list

### Timers (PL7)



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

### Timers (Unity)



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

**Create variable?**

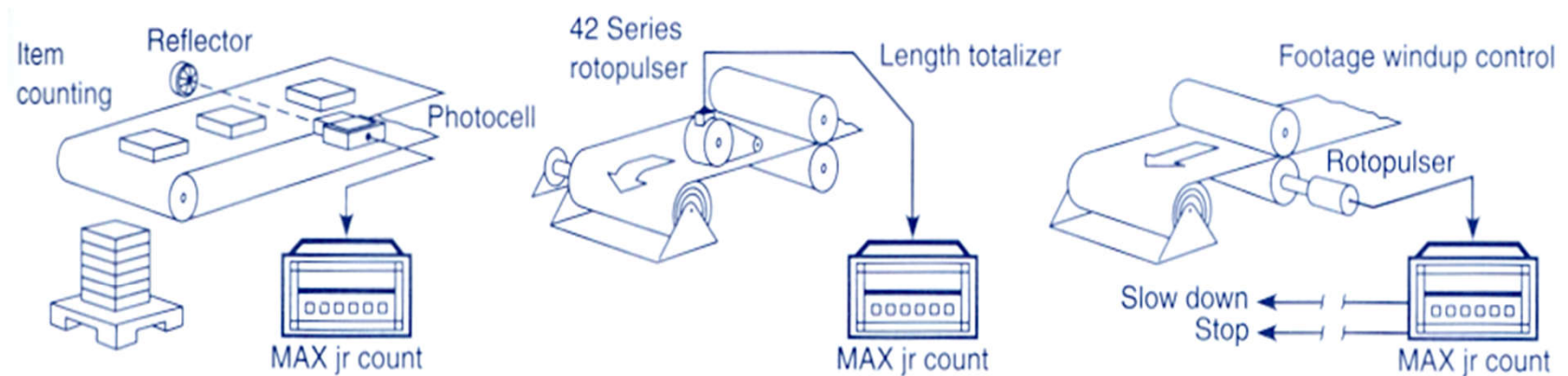
Name:  Type:

Address:  Comment:

# Instruction list

## Counters

Some applications...



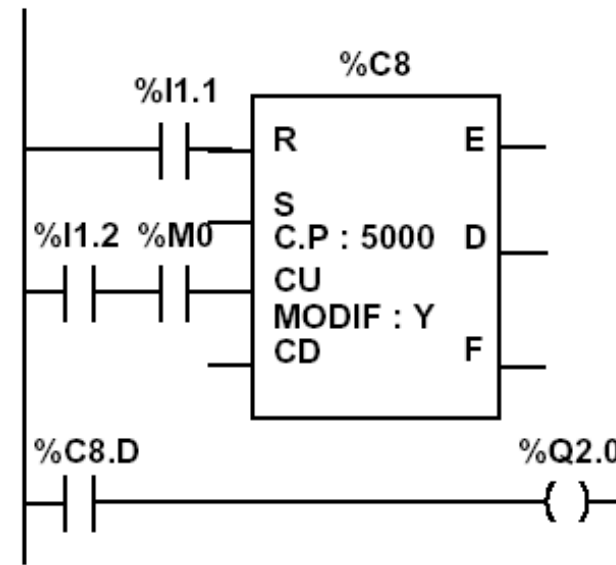
**Fig. 8-3**

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
```

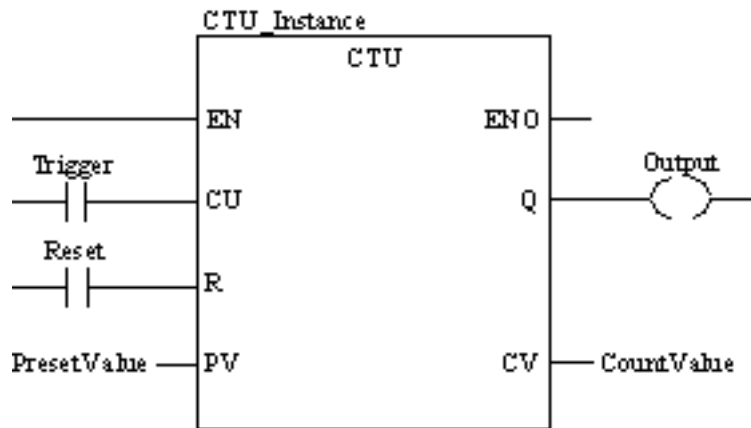
```
CU %C8
```

```
LD %C8.D
```

```
ST %Q2.0
```

# Ladder diagram

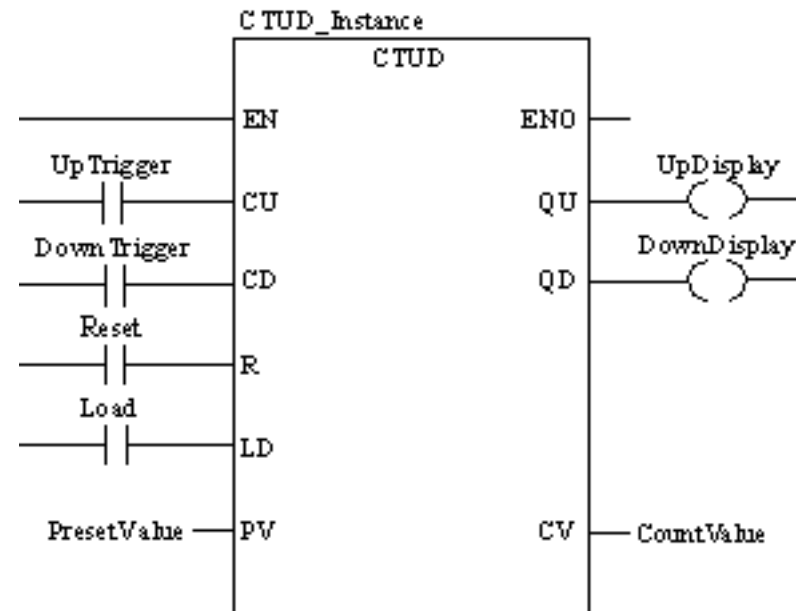
## Counters in Unity Pro



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

**CV ≥ PV** => Q:=1

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



**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

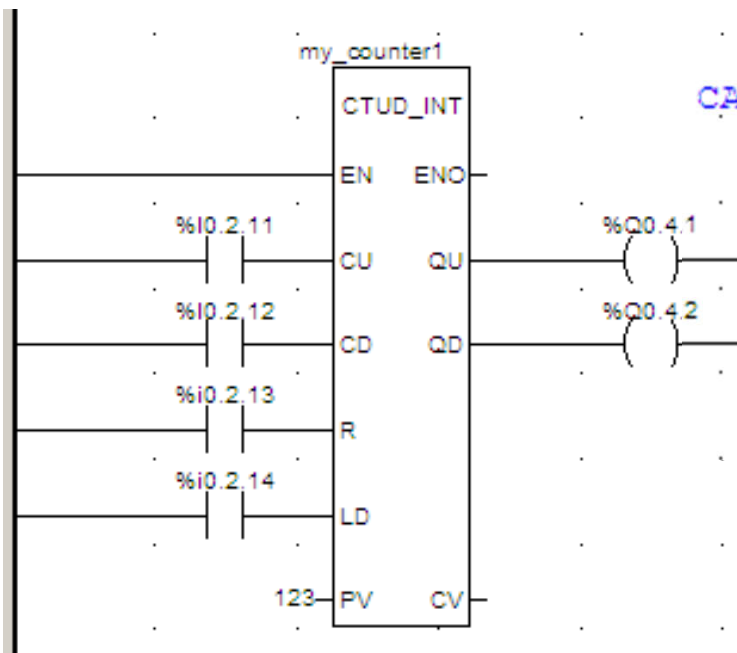
**R=1** => CV:=0    **LD=1** => CV:=PV

R has precedence over LD

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

# Ladder diagram

## Counters in Unity Pro



```

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*))
    
```

## 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

<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

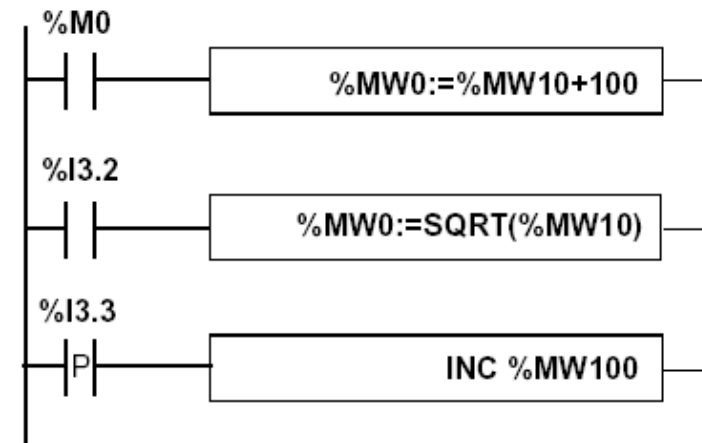
Type	Operand 1 (Op1)	Operand 2 (Op2)
Indexable words	%MW	%MW,%KW,%Xi.T
Non-indexable words	%QW,%SW,%NW,%BLK	Imm.Val.,%IW,%QW,%SW,%NW,%BLK, Num.expr.
Indexable double words	%MD	%MD,%KD
Non-indexable double words	%QD,%SD	Imm.Val.,%ID,%QD,%SD, Numeric expr.

## 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]
```

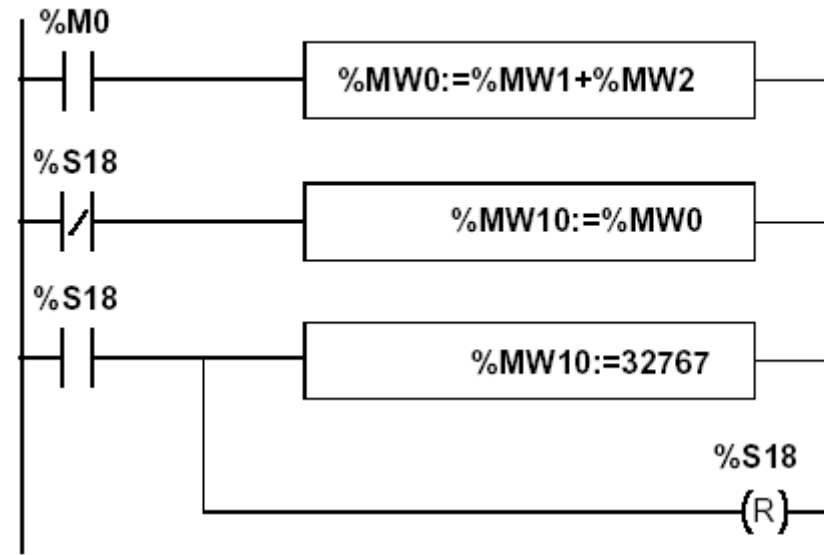


# Instruction list

## 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

<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

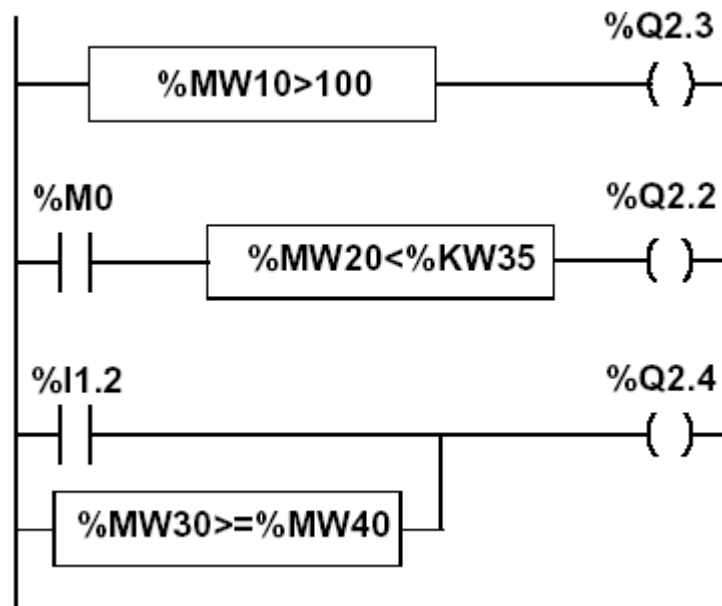
Type	Operands 1 and 2 (Op1 and Op2)
Indexable words	%MW, %KW, %Xi.T
Non-indexable words	Imm.val., %IW, %QW, %SW, %NW, %BLK, Numeric Expr.
Indexable double words	%MD, %KD
Non-indexable double words	Imm.val., %ID, %QD, %SD, Numeric expr.

## Instruction list

### Numerical Processing

#### Example:

#### Logic functions



#### *PL7:*

#### Instruction list language

```

LD    [%MW10>100]
ST    %Q2.3
LD    %M0
AND   [%MW20<%KW35]
ST    %Q2.2
LD    %I1.2
OR    [%MW30>=%MW40]
ST    %Q2.4

```

## Instruction list

### Numerical Processing

#### Priorities on the execution of the operations

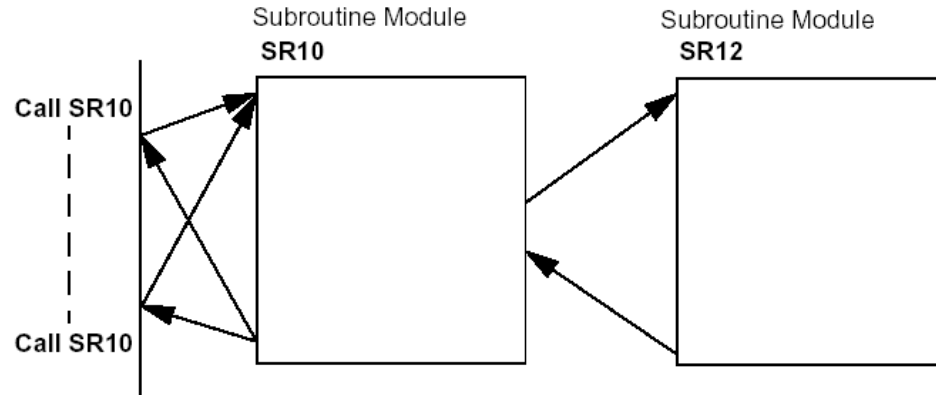
Rank	Instruction
1	Instruction to an operand
2	*,/,REM
3	+,-
4	<,>,<=,>=
5	=,<>
6	AND
7	XOR
8	OR

# Instruction list

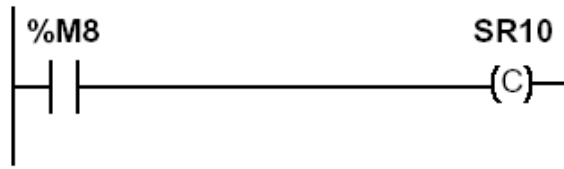
## Structures for Control of Flux

### Subroutines

### Call and Return



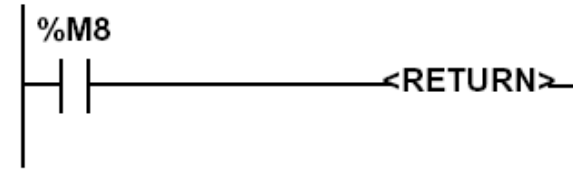
Ladder language:



Instruction list language:

```
LD %M8
CAL SR10
    {
    PL7
    }
Unity Pro
```

Ladder language



Instruction list language

```
LD %M8
RETC
```

Unity Pro Help

Back Forward Print Options Help

Contents Index Search

- Languages Reference
  - Safety Information
  - About the Book
  - General Presentation of Unity Pro
  - Application Structure
    - Description of the Available Functions
    - Application Program Structure
      - Description of Tasks and Procedures
      - Description of Sections and Subroutines
        - Description of Sections
        - Description of SFC sections
        - Description of Subroutines
      - Mono Task Execution
      - Multitasking Execution
    - Application Memory Structure
    - Operating Modes
    - System Objects
  - Data Description
  - Programming Language
  - User Function Blocks (DFB)
  - Appendices
  - Glossary
- Operating Modes
- OSLoader
- Unity Loader
- Concept Converter
- PL7 converter
- LL984 Editor, Reference Manual, LL984 Sp
- Communication Drivers
- EF/EFB/DFB Libraries
- Communication architectures
- Modicon M340 Platform

## Description of Subroutines

See: [Related Topics](#) [Submit Feedback](#)

### 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.

```

graph TD
  MAST[MAST] --- Sections[Sections]
  Sections --- SR_Sections[SR Sections]
  SR_Sections --- Control_1[Control_1]
  SR_Sections --- Detection[Detection]
  
```

© 2011 Schneider Electric. All rights reserved.

## 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:

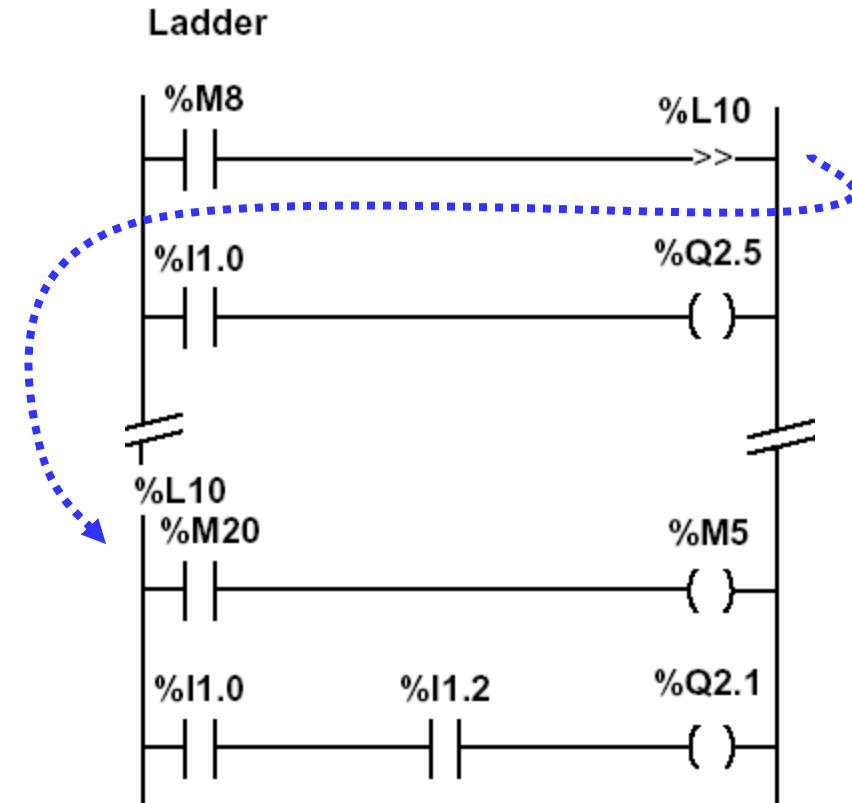
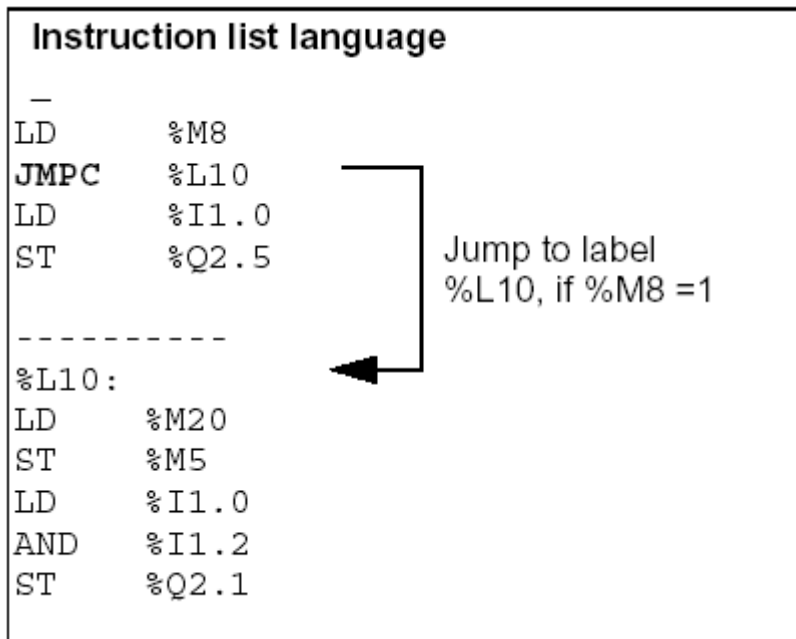
- **JMP**: unconditional program jump
  - **JMPC**: program jump if the instruction's Boolean result from the previous test is set at 1
  - **JMPCN**: program jump if the instruction's Boolean result from the previous test is set at 0. %Li is the label of the line to which the jump has been made (address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

# Instruction list

## Structures for Control of Flux

### Example:

Use of jump instructions

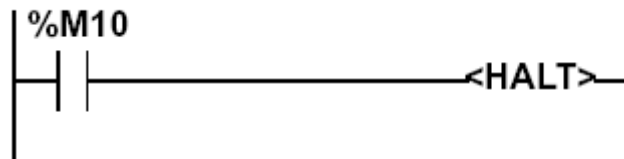




## Instruction list

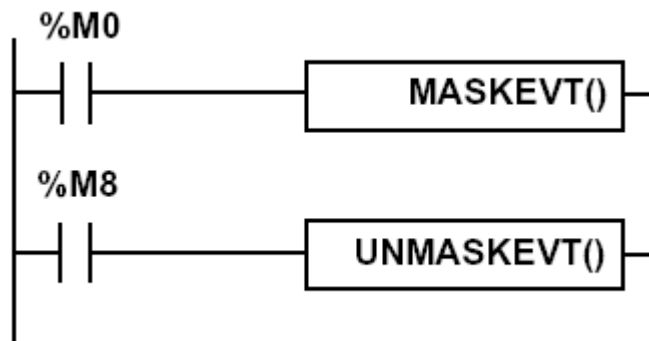
### Structures for Control of Flux

#### Halt



Stops all processes!

#### Events masking



## Instruction list

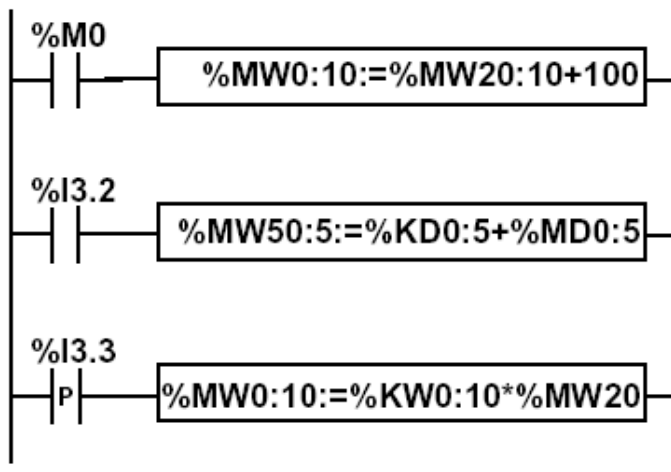
**There are other advanced instructions (see manual)**

- **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

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



*PL7:*

### Instruction list language

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

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

# 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 to program: RS232

### Processor:

- 8031 (no ROM ver. of Intel 8051, ~1980)
- 2 Kbytes de RAM
- 2 Kbytes EEPROM => 896 instructions
- **Average cycle time: 6.5 ms**



# PLC AEG A020 Plus

## DOLOG80

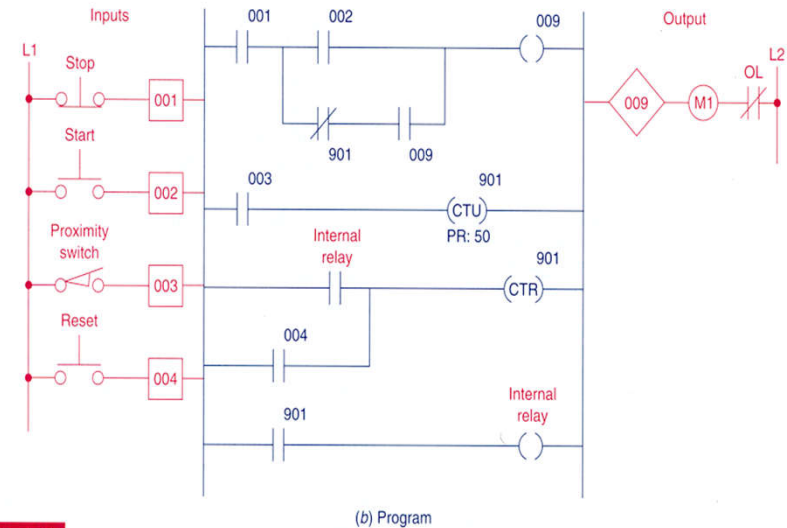
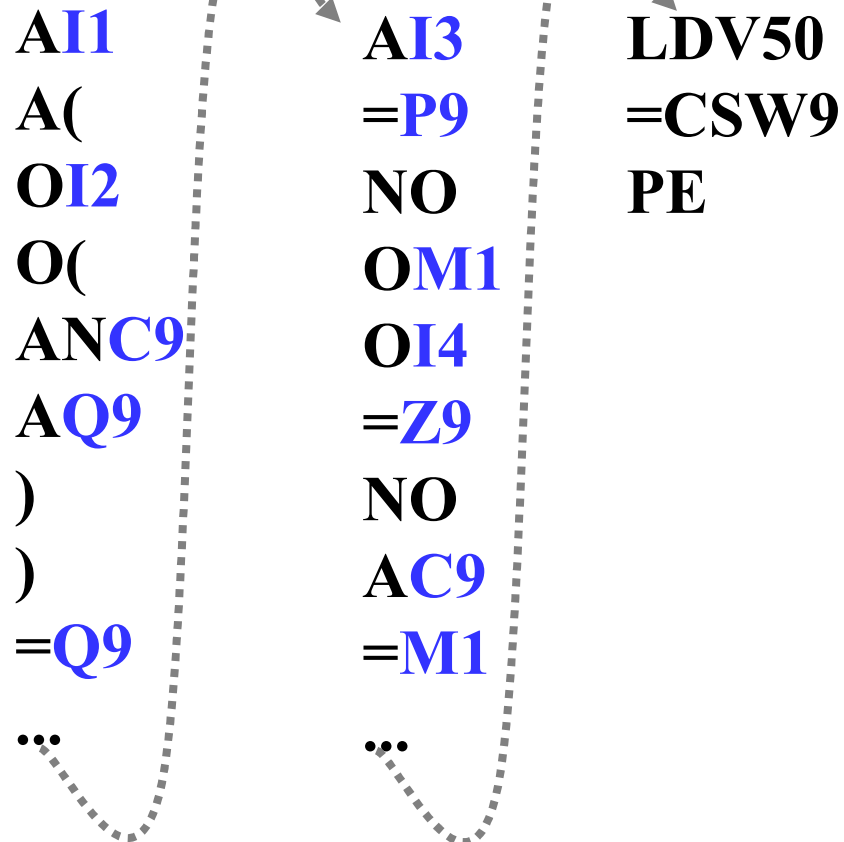
### OPERANDS

- 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:**



**Fig. 8-13**  
Conveyor motor program.

**Legend:**

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